# Deconstructing Models and Methods in Deep Learning

by

Pavel Izmailov

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May, 2023

Professor Andrew Gordon Wilson

# Acknowledgements

I am deeply grateful to a lot of people in my life, and the list is much longer than what I can fit here.

I am very lucky to be advised by Andrew Gordon Wilson, and be a part of Andrew's lab. Andrew's support, insights and guidance were crucial for me to become the researcher that I am, and made a huge impact on my life. I feel very privileged to be a part of Andrew's lab.

I am also very grateful to my undergraduate research advisors, Dmitry Kropotov and Dmitry Vetrov. They are the reason I got excited about machine learning, and why I was able to become a researcher.

I was also incredibly lucky to work with many amazing colleagues during my Ph.D. I would like to especially thank Timur Garipov, Dmitry Podoprikhin, Polina Kirichenko, Marc Finzi, Wesley Maddox and Samuel Stanton who played a major role in forming my research interests. I am also deeply thankful to Gregory Benton, Sanae Lotfi, Nate Gruver, Ben Athiwaratkun, Patrick Nicholson, Micah Goldblum, Sanyam Kapoor, Andres Potapczynski and Shikai Qiu for being amazing collaborators, I learned a lot from each of you. I am also very grateful for my mentors and friends in industry, who made a huge impact on my research interests and career: Alex Alemi, Ben Poole, Lucas Beyer, Simon Kornblith, Matt Hoffman, Bernie Wang and Alex Smola.

I want to thank my parents, who have always supported me and taught me to believe in myself. I would also like to thank my friends. I am grateful to Vadim Bereznyuk, who has been my best friend since high school and has always been there for me, regardless of what happened in my life. I also want to especially thank Polina Kirichenko, who has been my closest friend and collaborator during the Ph.D. A huge thank you to Alexander (Sanya) Pukhov and Marina Zavalina, who let me live on their couch for half a year. I worked on this thesis from that couch! You are truly great friends and amazing people. I am also grateful to Galina (cat), Timur Garipov, Samat Davletshin, Karen Grigorian, Karen Manukyan, Iana Movsesian, Anastasia Erastova, Tatiana Sholokhova, Anastasia Kuptsova, Tamara Makoveeva, Hui Xu, Marc Finzi, Aishwarya Kamath, Nicolas Carion and many others. I cannot imagine going through the Ph.D. without your support.

Back in 2013, I made a bet with three of my high school classmates on who will be the first to get a Ph.D.: me, Vadim Bereznyuk, Stepan Balybin or Petr Maximov. This bet provided great motivation, and I am also happy to report that I won the bet!

Finally, I am thankful to so many of my teachers. In particular, I would like to especially thank Alexey Ponomarev, Vladimir Dubrovsky, and Oleg Mikushonov.

# Abstract

Machine learning models are ultimately used to make decisions in the real world, where mistakes can be incredibly costly. We still understand surprisingly little about neural networks and the procedures that we use to train them, and, as a result, our models are brittle, often rely on spurious features, and generalize poorly under minor distribution shifts. Moreover, these models are often unable to faithfully represent uncertainty in their predictions, further limiting their applicability.

In this dissertation, I present results on neural network loss surfaces, probabilistic deep learning, uncertainty estimation and robustness to distribution shifts. In each of these works, we aim to build foundational understanding of models, training procedures, and their limitations, and then use this understanding to develop practically impactful, interpretable, robust and broadly applicable methods and models.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

Over the last few years, we have seen remarkable progress in many areas of deep learning. We now have deep learning models that can play chess at super-human level [Silver et al. 2016], neural networks that advance the field of protein folding [Jumper et al. 2021], generative models that create realistic images and videos from textual descriptions [Ramesh et al. 2021; Saharia et al. 2022], and language models that show glimpses of general intelligence [OpenAI 2023; Bubeck et al. 2023]. Inevitably, we also see more and more applications of deep learning models in the real world with self-driving cars, medical imaging models for automated diagnosis, and personal assistants and other systems built around language models.

However, these systems, still have major limitations, that make their widespread adoption challenging. In particular, it remains challenging to quantify uncertainty in the predictions of deep learning models [Guo et al. 2017; Kadavath et al. 2022; Minderer et al. 2021]. As a result, it is often hard to know when we can trust these models and when we should defer to a human expert. Another major issue is that neural networks often rely on shortcut features, and generalize poorly when the test data distribution differs from the training distribution [Geirhos et al. 2018; Hendrycks and Dietterich 2019], which is the case in most real-world applications.

Most of my research to date has been on a high level about understanding deep learning. I believe that by deconstructing our models and methods and understanding individual parts, we can build a better intuition and mechanistic understanding of how they work. Ultimately, this understanding often translates into better models, methods, and training procedures. Throughout this dissertation, I present several examples of this type of work.

The rest of this dissertation is organized as follows. In Chapter 2, I present results on the structure of the set of optima in neural network loss surfaces. I also present practical methods for improved training and fast ensembling of deep neural networks motivated by our observations about the loss surfaces. In Chapter 3, I present a broad exposition of Bayesian neural networks and a probabilistic perspective on generalization. I also present practical methods for improved uncertainty estimation in deep neural networks. In Chapter 4, I report the results of a detailed scientific study of the posterior distributions in Bayesian neural networks with many surprising observations challenging conventional wisdom. In particular, I describe the precise mechanism which leads Bayesian neural networks to perform poorly under distribution shift and propose a partial resolution. In Chapter 5, I describe our work on feature learning in neural networks in the presence of shortcut features, and a method for reducing the reliance on these features. Finally, I conclude this dissertation in Chapter 6.

# 2 | Loss Surfaces of Deep Neural Networks

We train neural networks by minimizing a loss function, which captures the goodness of fit to the data as a function of the parameters of the model. During training, the optimization trajectory can be thought of as a descent on the *loss surface*, i.e. the plot of the loss function. Consequently, the properties of the loss surfaces are central to deep learning.

Loss surfaces have a deep connection to optimization and generalization. In particular, whether or not we will be able to train our models with simple optimization methods such as stochastic gradient descent (SGD) will depend on the properties of the loss surfaces. Moreover, different types of optima will provide different generalization performance, and the properties of the loss surfaces will determine which of these solutions we will find in practice. In Chapter 3, we will also discuss the connections of the loss surfaces to Bayesian deep learning and ensembling.

In this chapter, we present our results on the structure of the set of local optima of deep neural networks. In particular, we demonstrate *mode connectivity*: the local optima of neural networks are connected by a curved path of near-constant train loss and test accuracy. We illustrate this phenomenon in Figure 2.1. We propose a simple method for finding the mode-connecting paths and demonstrate that mode connectivity holds very generally in vision and NLP. Inspired by these observations, we propose Stochastic Weight Averaging (SWA), a training procedure for deep neural networks that averages the weights of SGD iterates with high constant or cyclical

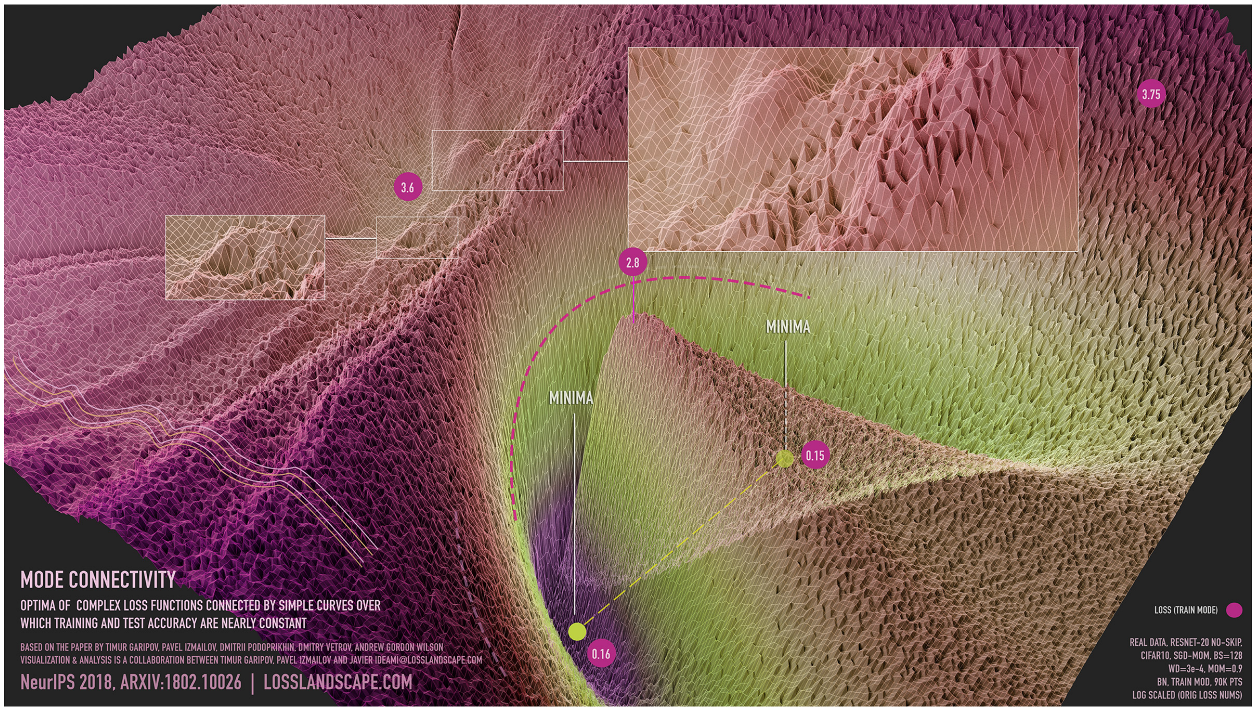

3.6
3.75
2.8
MINIMA
MINIMA
0.15
0.16
MODE CONNECTIVITY
OPTIMA OF COMPLEX LOSS FUNCTIONS CONNECTED BY SIMPLE CURVES OVER WHICH TRAINING AND TEST ACCURACY ARE NEARLY CONSTANT
BASED ON THE PAPER BY TIMUR GARIPOV, PAVEL IZMAILOV, DMITRII PODOPRIKHIN, DMITRY VETROV, ANDREW GORDON WILSON
VISUALIZATION & ANALYSIS IS A COLLABORATION BETWEEN TIMUR GARIPOV, PAVEL IZMAILOV AND JAVIER IDEAMI@LOSSLANDSCAPE.COM
NeurIPS 2018, ARXIV:1802.10026 | LOSSLANDSCAPE.COM
LOSS (TRAIN MODE) ●
REAL DATA, RESNET-20 NO-SKIP, CIFAR10, SGD-MOM, BS=128 WD=3e-4, MOM=0.9 BN, TRAIN MOD, 90K PTS LOG SCALED (ORIG LOSS NUMS)

**Figure 2.1:** Visualization of mode connectivity for ResNet-20 with no skip connections on the CIFAR-10 dataset. The visualization is created in collaboration with Javier Ideami (`https://losslandscape.com/`).

learning rates. SWA can be used as a drop-in replacement for any optimizer in deep learning and consistently leads to improvements in generalization at no cost.

This chapter is adapted from the papers "Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs" [Garipov et al. 2018] which originally appeared at NeurIPS 2018 and "Averaging Weights Leads to Wider Optima and Better Generalization" [Izmailov et al. 2018] which originally appeared at UAI 2018, written jointly with Timur Garipov, Dmitry Podoprikhin, Dmitry Vetrov and Andrew Gordon Wilson.

## 2.1 Mode Connectivity

The loss surfaces of deep neural networks (DNNs) are highly non-convex and can depend on millions of parameters. The geometric properties of these loss surfaces are not well understood. Even for simple networks, the number of local optima and saddle points is large and can grow exponentially in the number of parameters [Auer et al. 1996; Choromanska et al. 2015; Dauphin et al. 2014]. Moreover, the loss is high along a line segment connecting two optima [e.g., Goodfellow et al. 2015; Keskar et al. 2017]. These two observations suggest that the local optima are isolated.

In this chapter, we describe a training procedure that can in fact find paths of near-constant accuracy between the modes of large deep neural networks. Furthermore, we show that for a wide range of architectures, we can find these paths in the form of a simple polygonal chain of two line segments. Consider, for example, Figure 5.1, which illustrates the ResNet-164 $\ell_2$-regularized cross-entropy train loss on CIFAR-100, through three different planes. We form each two-dimensional plane by all affine combinations of three weight vectors.[1]

The left panel shows a plane defined by three independently trained networks. In this plane, all optima are isolated, which corresponds to the standard intuition. However, the middle and right panels show two different paths of near-constant loss between the modes in weight space, discovered by our proposed training procedure. The endpoints of these paths are the two independently trained DNNs corresponding to the two lower modes in the left panel. We show a high-resolution visualization of the path discovered by our method in Figure 2.1.

We release the code for reproducing the results at
https://github.com/timgaripov/dnn-mode-connectivity.

## 2.2 Finding Paths between Modes

We describe a method to minimize the training error along a path that connects two points in the space of DNN weights. Section 2.2.1 introduces this general procedure for arbitrary para-

[1]Suppose we have three weight vectors $w_1, w_2, w_3$. We set $u = (w_2 - w_1)$, $v = (w_3 - w_1) - \langle w_3 - w_1, w_2 - w_1\rangle / \|w_2 - w_1\|^2 \cdot (w_2 - w_1)$. Then the normalized vectors $\hat{u} = u/\|u\|$, $\hat{v} = v/\|v\|$ form an orthonormal basis in the plane containing $w_1, w_2, w_3$. To visualize the loss in this plane, we define a Cartesian grid in the basis $\hat{u}, \hat{v}$ and evaluate the networks corresponding to each of the points in the grid. A point $P$ with coordinates $(x, y)$ in the plane would then be given by $P = w_1 + x \cdot \hat{u} + y \cdot \hat{v}$.

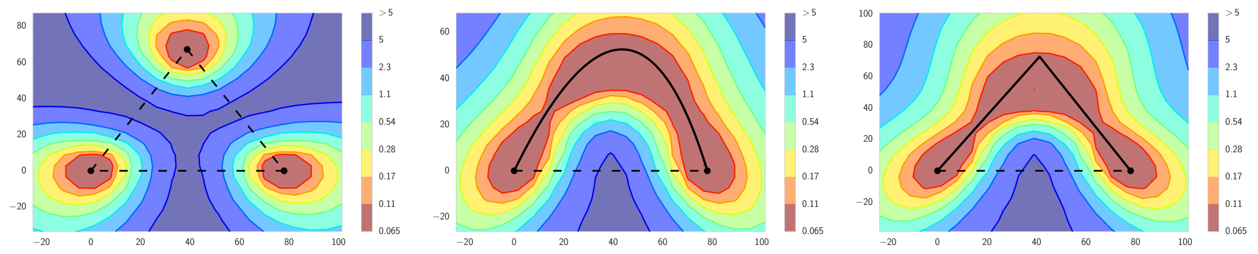


**Figure 2.2:** The $\ell_2$-regularized cross-entropy train loss surface of a ResNet-164 on CIFAR-100, as a function of network weights in a two-dimensional subspace. In each panel, the horizontal axis is fixed and is attached to the optima of two independently trained networks. The vertical axis changes between panels as we change planes (defined in the main text). **Left:** Three optima for independently trained networks. **Middle** and **Right**: A quadratic Bezier curve, and a polygonal chain with one bend, connecting the lower two optima on the left panel along a path of near-constant loss. Notice that in each panel a direct linear path between each mode would incur high loss.

metric curves, and Section 2.2.2 describes polygonal chains and Bezier curves as two example parametrizations of such curves. In Appendix A.2, we discuss the computational complexity of the proposed approach and how to apply batch normalization at test time to points on these curves.

### 2.2.1 Connection Procedure

Let $\hat{w}_1$ and $\hat{w}_2$ in $\mathbb{R}^{|net|}$ be two sets of weights corresponding to two neural networks independently trained by minimizing any user-specified loss $\mathcal{L}(w)$, such as the cross-entropy loss. Here, $|net|$ is the number of weights of the DNN. Moreover, let $\phi_\theta : [0,1] \longrightarrow \mathbb{R}^{|net|}$ be a continuous piecewise smooth parametric curve, with parameters $\theta$, such that $\phi_\theta(0) = \hat{w}_1, \;\; \phi_\theta(1) = \hat{w}_2$.

To find a path of high accuracy between $\hat{w}_1$ and $\hat{w}_2$, we propose to find the parameters $\theta$ that minimize the expectation over a uniform distribution on the curve, $\hat{\ell}(\theta)$:

$$\hat{\ell}(\theta) = \frac{\int \mathcal{L}(\phi_\theta) d\phi_\theta}{\int d\phi_\theta} = \frac{\int\limits_0^1 \mathcal{L}(\phi_\theta(t)) \|\phi'_\theta(t)\| dt}{\int\limits_0^1 \|\phi'_\theta(t)\| dt} = \int\limits_0^1 \mathcal{L}(\phi_\theta(t)) q_\theta(t) dt = \mathbb{E}_{t \sim q_\theta(t)} \Big[ \mathcal{L}(\phi_\theta(t)) \Big], \quad (2.1)$$

where the distribution $q_\theta(t)$ on $t \in [0, 1]$ is defined as: $q_\theta(t) = \|\phi'_\theta(t)\| \cdot \left(\int\limits_0^1 \|\phi'_\theta(t)\| dt\right)^{-1}$. The numerator of (2.1) is the line integral of the loss $\mathcal{L}$ on the curve, and the denominator $\int_0^1 \|\phi'_\theta(t)\| dt$ is the normalizing constant of the uniform distribution on the curve defined by $\phi_\theta(\cdot)$. Stochastic gradients of $\hat{\ell}(\theta)$ in Eq. (2.1) are generally intractable since $q_\theta(t)$ depends on $\theta$. Therefore we also propose a more computationally tractable loss

$$\ell(\theta) = \int_0^1 \mathcal{L}(\phi_\theta(t)) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}(\phi_\theta(t)), \tag{2.2}$$

where $U(0, 1)$ is the uniform distribution on $[0, 1]$. The difference between (2.1) and (2.2) is that the latter is an expectation of the loss $\mathcal{L}(\phi_\theta(t))$ with respect to a uniform distribution on $t \in [0, 1]$, while (2.1) is an expectation with respect to a uniform distribution on the curve. The two losses coincide, for example, when $\phi_\theta(\cdot)$ defines a polygonal chain with two line segments of equal length and the parametrization of each of the two segments is linear in $t$.

To minimize (2.2), at each iteration we sample $\tilde{t}$ from the uniform distribution $U(0, 1)$ and make a gradient step for $\theta$ with respect to the loss $\mathcal{L}(\phi_\theta(\tilde{t}))$. This way we obtain unbiased estimates of the gradients of $\ell(\theta)$, as

$$\nabla_\theta \mathcal{L}(\phi_\theta(\tilde{t})) \simeq \mathbb{E}_{t \sim U(0,1)} \nabla_\theta \mathcal{L}(\phi_\theta(t)) = \nabla_\theta \mathbb{E}_{t \sim U(0,1)} \mathcal{L}(\phi_\theta(t)) = \nabla_\theta \ell(\theta).$$

We repeat these updates until convergence.

### 2.2.2 Example Parametrizations

Polygonal chain. The simplest parametric curve we consider is the polygonal chain (see Figure 5.1, right). The trained networks $\hat{w}_1$ and $\hat{w}_2$ serve as the endpoints of the chain and the bends of the chain are the parameters $\theta$ of the curve parametrization. Consider the simplest case of a

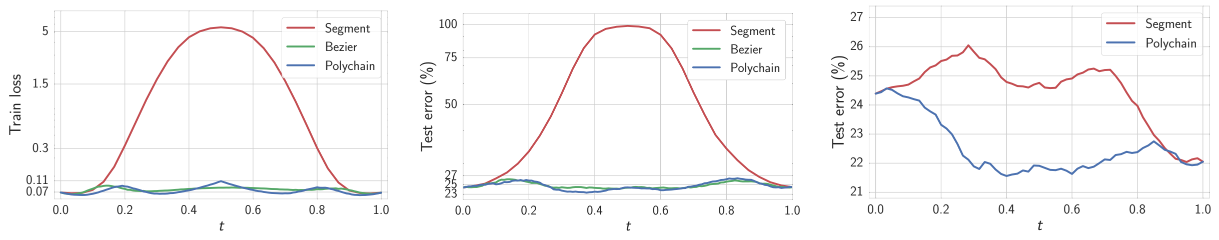


**Figure 2.3:** The $\ell_2$-regularized cross-entropy train loss **(left)** and test error **(middle)** as a function of the point on the curves $\phi_\theta(t)$ found by the proposed method (ResNet-164 on CIFAR-100). **Right:** Error of the two-network ensemble consisting of the endpoint $\phi_\theta(0)$ of the curve and the point $\phi_\theta(t)$ on the curve (CIFAR-100, ResNet-164). "Segment" is a line segment connecting two modes found by SGD. "Polychain" is a polygonal chain connecting the same endpoints.

chain with one bend $\theta$. Then

$$\phi_\theta(t) = \begin{cases} 2\left(t\theta + (0.5 - t)\hat{w}_1\right), & 0 \leqslant t \leqslant 0.5 \\ 2\left((t - 0.5)\hat{w}_2 + (1 - t)\theta\right), & 0.5 \leqslant t \leqslant 1. \end{cases}$$

Bezier curve. A Bezier curve (see Figure 5.1, middle) provides a convenient parametrization of smooth paths with given endpoints. A quadratic Bezier curve $\phi_\theta(t)$ with endpoints $\hat{w}_1$ and $\hat{w}_2$ is given by

$$\phi_\theta(t) = (1 - t)^2\hat{w}_1 + 2t(1 - t)\theta + t^2\hat{w}_2, \quad 0 \leqslant t \leqslant 1.$$

These formulas naturally generalize to $n$ bends $\theta = \{w_1, w_2, \ldots, w_n\}$ (see Appendix A.1.3).

## 2.3 Mode Connectivity Demonstrations

We show that the proposed training procedure in Section 2.2 does indeed find high accuracy paths connecting different modes, across a range of architectures and datasets. Moreover, we further investigate the properties of these curves, showing that they correspond to meaningfully different representations that can be ensembled for improved accuracy. We use these insights to propose a fast ensembling procedure in Section 2.4. In Section 2.5 we describe a training procedure with improved generalization performance also inspired by these observations.

In particular, we test VGG-16 [Simonyan and Zisserman 2014], a 28-layer Wide ResNet with widening factor 10 [Zagoruyko and Komodakis 2016] and a 158-layer ResNet [He et al. 2016] on CIFAR-10, and VGG-16, 164-layer ResNet-bottleneck [He et al. 2016] on CIFAR-100. For CIFAR-10 and CIFAR-100 we use the same standard data augmentation as Huang et al. [2017a]. We provide additional results, including detailed experiments for fully connected and recurrent networks, in the Appendix A.1.4.

For each model and dataset we train two networks with different random initializations to find two modes. Then we use the proposed algorithm of Section 2.2 to find a path connecting these two modes in the weight space with a quadratic Bezier curve and a polygonal chain with one bend. We also connect the two modes with a line segment for comparison. In all experiments we optimize the loss (2.2), as for Bezier curves the gradient of loss (2.1) is intractable, and for polygonal chains we found loss (2.2) to be more stable.

Figures 5.1 and 2.3 show the results of the proposed mode connecting procedure for ResNet-164 on CIFAR-100. Here *loss* refers to $\ell_2$-regularized cross-entropy loss. For both the Bezier curve and polygonal chain, train loss (Figure 2.3, left) and test error (Figure 2.3, middle) are indeed nearly constant. In addition, we provide plots of train error and test loss in Appendix A.1.4. In Appendix A.1.4, we also include a comprehensive table summarizing all path finding experiments on CIFAR-10 and CIFAR-100 for VGGs, ResNets and Wide ResNets, as well as fully connected networks and recurrent neural networks, which follow the same general trends. In the Appendix A.1.7 we also show that the connecting curves can be found consistently as we vary the number of parameters in the network, although the ratio of the arclength for the curves to the length of the line segment connecting the same endpoints decreases with increasing parametrization. In Appendix A.1.4, we also measure the losses (2.1) and (2.2) for all the curves we constructed, and find that the values of the two losses are very close, suggesting that the loss (2.2) is a good practical approximation to the loss (2.1).

The constant-error curves connecting two given networks discovered by the proposed method are not unique. We trained two different polygonal chains with the same endpoints and different random seeds using VGG-16 on CIFAR-10. We then measured the Euclidean distance between the turning points of these curves. For VGG-16 on CIFAR-10 this distance is equal to 29.6 and the distance between the endpoints is 50, showing that the curves are not unique. In this instance, we expect the distance between turning points to be less than the distance between endpoints, since the locations of the turning points were initialized to the same value (the center of the line segment connecting the endpoints).

Although high accuracy connecting curves can often be very simple, such as a polygonal chain with only one bend, we note that line segments *directly* connecting two modes generally incur high error. For VGG-16 on CIFAR-10 the test error goes up to 90% in the center of the segment. For ResNet-158 and Wide ResNet-28-10 the worst errors along direct line segments are still high, but relatively less, at 80% and 66%, respectively. This finding suggests that the loss surfaces of state-of-the-art residual networks are indeed more regular than those of classical models like VGG, in accordance with the observations in Li et al. [2017].

In this paper we focus on connecting pairs of networks trained using the same hyper-parameters, but from different random initializations. Building upon our work, Gotmare et al. [2018] have recently shown that our mode connectivity approach applies to pairs of networks trained with different batch sizes, optimizers, data augmentation strategies, weight decays and learning rate schemes.

To motivate the ensembling procedure proposed in the next section, we now examine how far we need to move along a connecting curve to find a point that produces substantially different, but still useful, predictions. Let $\hat{w}_1$ and $\hat{w}_2$ be two distinct sets of weights corresponding to optima obtained by independently training a DNN two times. We have shown that there exists a path connecting $\hat{w}_1$ and $\hat{w}_2$ with high test accuracy. Let $\phi_\theta(t)$, $t \in [0, 1]$ parametrize this path with $\phi_\theta(0) = \hat{w}_1$, $\phi_\theta(1) = \hat{w}_2$. We investigate the performance of an ensemble of two networks: the endpoint $\phi_\theta(0)$ of the curve and a point $\phi_\theta(t)$ on the curve corresponding to $t \in [0, 1]$. Figure 2.3

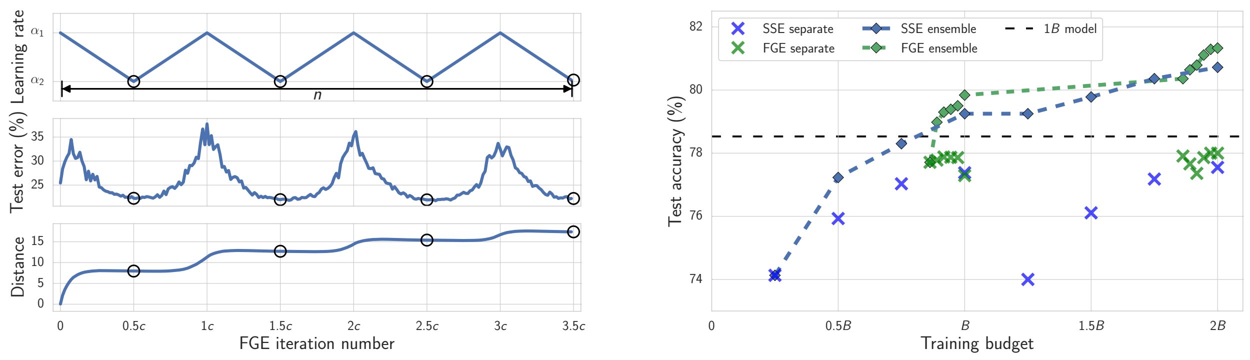


**Figure 2.4: Left:** Plot of the learning rate (**Top**), test error (**Middle**) and distance from the initial value $\hat{w}$ (**Bottom**) as a function of iteration for FGE with Preactivation-ResNet-164 on CIFAR-100. Circles indicate the times when we save models for ensembling. **Right:** Ensemble performance of FGE and SSE (Snapshot Ensembles) as a function of training time, using ResNet-164 on CIFAR-100 ($B$ = 150 epochs). Crosses represent the performance of separate "snapshot" models, and diamonds show the performance of the ensembles constructed of all models available by the given time.

(right) shows the test error of this ensemble as a function of $t$, for a ResNet-164 on CIFAR-100. The test error starts decreasing at $t \approx 0.1$ and for $t \geqslant 0.4$ the error of an ensemble is already as low as the error of an ensemble of the two independently trained networks used as the endpoints of the curve. Thus even by moving away from the endpoint by a relatively small distance along the curve we can find a network that produces meaningfully different predictions from the network at the endpoint. This result also demonstrates that these curves do not exist only due to degenerate parametrizations of the network (such as rescaling on either side of a ReLU); instead, points along the curve correspond to meaningfully different representations of the data that can be ensembled for improved performance. In Appendix A.1.8 we show how to create trivially connecting curves that do not have this property.

## 2.4 Fast Geometric Ensembling

In this section, we introduce a practical ensembling procedure, Fast Geometric Ensembling (FGE), motivated by our observations about mode connectivity.

In the previous section, we considered ensembling along mode connecting curves. Suppose now we instead only have **one** set of weights $\hat{w}$ corresponding to a mode of the loss. We cannot explicitly construct a path $\phi_\theta(\cdot)$ as before, but we know that multiple paths passing through $\hat{w}$ exist, and it is thus possible to move away from $\hat{w}$ in the weight space without increasing the loss. Further, we know that we can find diverse networks providing meaningfully different predictions by making relatively small steps in the weight space (see Figure 2.3, right).

Inspired by these observations, we propose the Fast Geometric Ensembling (FGE) method that aims to find diverse networks with relatively small steps in the weight space, without leaving a region that corresponds to low test error.

While inspired by mode connectivity, FGE **does not** rely on explicitly finding a connecting curve, and thus does not require pre-trained endpoints, and so can be trained in the time required to train a *single* network.

Let us describe Fast Geometric Ensembling. First, we initialize a copy of the network with weights $w$ set equal to the weights of the trained network $\hat{w}$. Now, to force $w$ to move away from $\hat{w}$ without substantially decreasing the prediction accuracy we adopt a cyclical learning rate schedule $\alpha(\cdot)$ (see Figure 2.4, left), with the learning rate at iteration $i = 1, 2, \ldots$ defined as

$$\alpha(i) = \begin{cases} (1 - 2t(i))\alpha_1 + 2t(i)\alpha_2 & 0 < t(i) \leqslant \frac{1}{2} \\ (2 - 2t(i))\alpha_2 + (2t(i) - 1)\alpha_1 & \frac{1}{2} < t(i) \leqslant 1 \end{cases},$$

where $t(i) = \frac{1}{c}(\mathrm{mod}(i - 1, c) + 1)$, the learning rates are $\alpha_1 > \alpha_2$, and the number of iterations in one cycle is given by even number $c$. Here by **iteration** we mean processing one mini-batch of data. We can train the network $w$ using the standard $\ell_2$-regularized cross-entropy loss function (or any other loss that can be used for DNN training) with the proposed learning rate schedule for $n$ iterations. In the middle of each learning rate cycle when the learning rate reaches its minimum value $\alpha(i) = \alpha_2$ (which corresponds to $\mod(i-1, c)+1 = c/2, t(i) = \frac{1}{2}$) we collect the checkpoints of weights $w$. When the training is finished we ensemble the collected models. An outline of the algorithm is provided in Appendix A.1.9.

Figure 2.4 (left) illustrates the adopted learning rate schedule. During the periods when the learning rate is large (close to $\alpha_1$), $w$ is exploring the weight space doing larger steps but sacrificing the test error. When the learning rate is small (close to $\alpha_2$), $w$ is in the exploitation phase in which the steps become smaller and the test error goes down. The cycle length is usually about 2 to 4 epochs, so that the method efficiently balances exploration and exploitation with relatively-small steps in the weight space that are still sufficient to gather diverse and meaningful networks for the ensemble.

To find a good initialization $\hat{w}$ for the proposed procedure, we first train the network with the standard learning rate schedule (the schedule used to train single DNN models) for about 80% of the time required to train a single model. After this pre-training is finished we initialize FGE with $\hat{w}$ and run the proposed fast ensembling algorithm for the remaining computational budget. In order to get more diverse samples, one can run the algorithm described above several times for a smaller number of iterations initializing from different checkpoints saved during training of $\hat{w}$, and then ensemble all of the models gathered across these runs.

Cyclical learning rates have also recently been considered in Smith and Topin [2017] and Huang et al. [2017a]. Our proposed method is perhaps most closely related to Snapshot Ensembles [Huang et al. 2017a], but has several distinctive features, inspired by our geometric insights. In particular, Snapshot Ensembles adopt cyclical learning rates with cycle length on the scale of 20 to 40 epochs from the beginning of the training as they are trying to do large steps in the weight space. However, according to our analysis of the curves it is sufficient to do relatively small steps in the weight space to get diverse networks, so we only employ cyclical learning rates with a small cycle length on the scale of 2 to 4 epochs in the last stage of the training. As illustrated in Figure 2.4 (left), the step sizes made by FGE between saving two models (that is the euclidean distance between sets of weights of corresponding models in the weight space) are on the scale of 7 for Preactivation-ResNet-164 on CIFAR-100. For Snapshot Ensembles for the same model the distance between two snapshots is on the scale of 40. We also use a piecewise linear cyclical

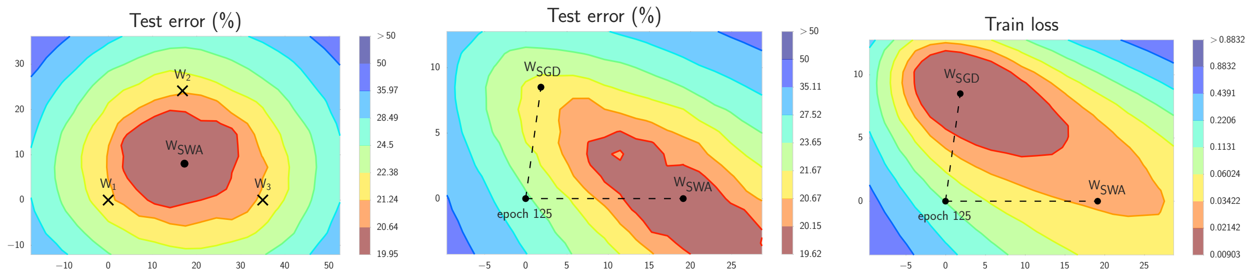


**Figure 2.5:** Illustrations of SWA and SGD with a Preactivation ResNet-164 on CIFAR-100[2]. **Left**: test error surface for three FGE samples and the corresponding SWA solution (averaging in weight space). **Middle** and **Right**: test error and train loss surfaces showing the weights proposed by SGD (at convergence) and SWA, starting from the same initialization of SGD after 125 training epochs.

learning rate schedule following Smith and Topin [2017] as opposed to the cosine schedule in Snapshot Ensembles.

Figure 2.4 (right) illustrates the results for Preactivation-ResNet-164 on CIFAR-100 for one and two training budgets. The training budget $B$ for training the model in the standard way is 150 epochs. Snapshot Ensembles use a cyclical learning rate from the beginning of the training and they gather the models for the ensemble throughout training. To find a good initialization for FGE we run standard independent training for the first 125 epochs before applying FGE. In this case, the whole ensemble is gathered over the following 22 epochs (126-147) to fit in the budget of each of the two runs. During these 22 epochs FGE is able to gather diverse enough networks to outperform Snapshot Ensembles both for $1B$ and $2B$ budgets.

We provide further detailed experiments with FGE in Appendix A.1.9.

## 2.5 Stochastic Weight Averaging

Fast Geometric Ensembling (Section 2.4) uses a high frequency cyclical learning rate with SGD to select networks to ensemble. In Figure 5.1 (left) we see that the weights of the networks ensembled by FGE are on the periphery of the most desirable solutions. This observation suggests it is promising to average these points in *weight space*, and use a network with these averaged weights, instead of forming an ensemble by averaging the outputs of networks in *model space*. Although the general idea of maintaining a running average of weights traversed by SGD dates back to Ruppert [1988], this procedure is not typically used to train neural networks. It is sometimes applied as an exponentially decaying running average in combination with a decaying learning rate (where it is called an exponential moving average), which smooths the trajectory of conventional SGD but does not perform very differently. However, we show that an equally weighted average of the points traversed by SGD with a cyclical or high constant learning rate, which we refer to as *Stochastic Weight Averaging* (SWA), has many surprising and promising features for training deep neural networks, leading to a better understanding of the geometry of their loss surfaces. Indeed, SWA with cyclical or constant learning rates can be used as a drop-in replacement for standard SGD training of multilayer networks — but with improved generalization and essentially no overhead. In particular:

- We show that SGD with cyclical [e.g., Loshchilov and Hutter 2016] and constant learning rates traverses regions of weight space corresponding to high-performing networks. We find that while these models are moving around this optimal set they never reach its central points. We show that we can move into this more desirable space of points by averaging the weights proposed over SGD iterations.

- While FGE ensembles can be trained in the same time as a single model, test predictions for an ensemble of $k$ models requires $k$ times more computation. We show that SWA can be interpreted as an approximation to FGE ensembles but with the test-time, convenience, and interpretability of a single model.

- We demonstrate that SWA leads to solutions that are wider than the optima found by SGD. Keskar et al. [2017] and Hochreiter and Schmidhuber [1997a] conjecture that the width of the optima is critically related to generalization. We illustrate that the loss on the train is shifted with respect to the test error (Figure 5.1, middle and right panels, and sections 3, 4). We show that SGD generally converges to a point near the boundary of the wide flat region of optimal points. SWA on the other hand is able to find a point centered in this region, often with slightly worse train loss but with substantially better test error.

- We show that the loss function is asymmetric in the direction connecting SWA with SGD. In this direction, SGD is near the periphery of sharp ascent. Part of the reason SWA improves generalization is that it finds solutions in flat regions of the training loss in such directions.
- SWA achieves notable improvement for training a broad range of architectures over several consequential benchmarks. In particular, running SWA for just 10 epochs on ImageNet we are able to achieve 0.8% improvement for ResNet-50 and DenseNet-161, and 0.6% improvement for ResNet-150. We achieve improvement of over 1.3% on CIFAR-100 and of over 0.4% on CIFAR-10 with Preactivation ResNet-164, VGG-16 and Wide ResNet-28-10. We also achieve substantial improvement for the recent Shake-Shake Networks and PyramidNets.
- SWA is extremely easy to implement and has virtually no computational overhead compared to the conventional training schemes.

We emphasize that SWA is finding a solution *in the same basin of attraction* as SGD, as can be seen in Figure 5.1, but in a flatter region of the training loss. SGD typically finds points on the *periphery* of a set of good weights. By running SGD with a cyclical or high constant learning rate, we traverse the surface of this set of points, and by averaging we find a more centred solution in a flatter region of the training loss. Further, the training loss for SWA is often slightly worse than for SGD suggesting that SWA solution is not a local optimum of the loss. Throughout this section, *optima* is used in a general sense to mean *solutions* (converged points of a given procedure), rather than different local minima of the same objective.

The rest of this section is organized as follows. In section 2.5.1, we consider trajectories of SGD with a constant and cyclical learning rate, which helps understand the geometry of SGD training

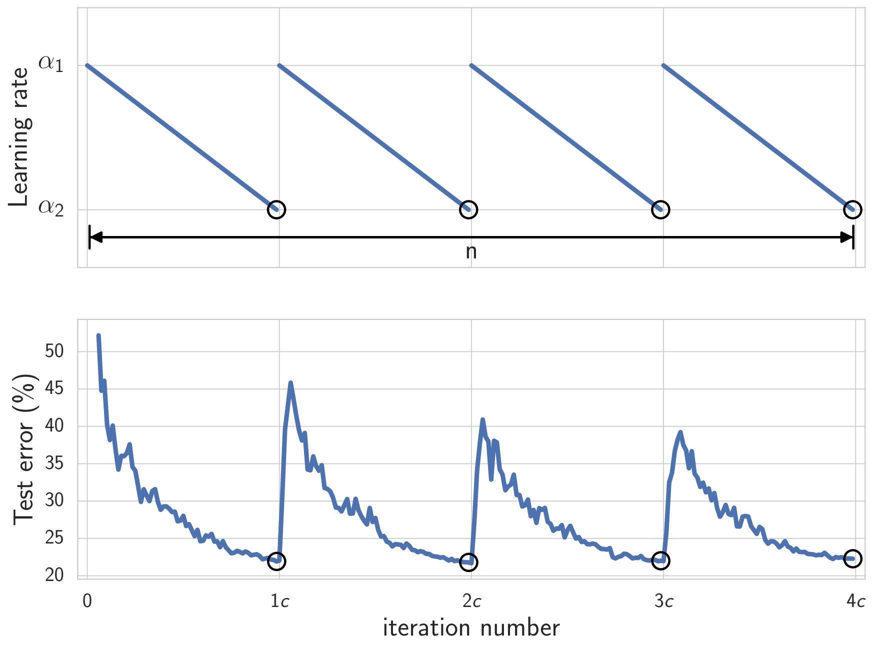


**Figure 2.6: Top**: cyclical learning rate as a function of iteration. **Bottom**: test error as a function of iteration for cyclical learning rate schedule with Preactivation-ResNet-164 on CIFAR-100. Circles indicate iterations corresponding to the minimum learning rates.

for neural networks, and motivates the SWA procedure. Then in section 5.4 we present the SWA algorithm in detail, in section 2.5.3 we derive its complexity, and in section 2.5.4 we analyze the width of solutions found by SWA versus conventional SGD training. In section 2.5.5 we then examine the relationship between SWA and Fast Geometric Ensembling. Finally, in section 3.4 we consider SWA from the perspective of stochastic convex optimization.

### 2.5.1 Analysis of SGD Trajectories

SWA is based on averaging the samples proposed by SGD using a learning rate schedule that allows exploration of the region of weight space corresponding to high-performing networks. In particular we consider cyclical and constant learning rate schedules.

The cyclical learning rate schedule that we adopt is inspired by Garipov et al. [2018] and Smith and Topin [2017]. In each cycle we linearly decrease the learning rate from $\alpha_1$ to $\alpha_2$. The formula

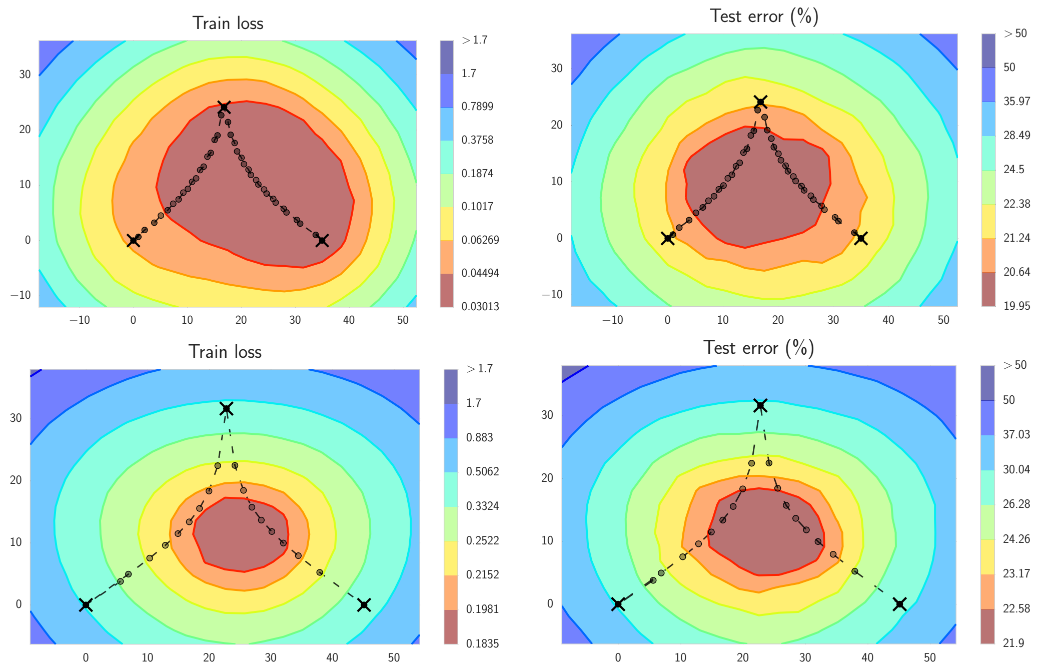


**Figure 2.7:** The $L_2$-regularized cross-entropy train loss and test error surfaces of a Preactivation ResNet-164 on CIFAR-100 in the plane containing the first, middle and last points (indicated by black crosses) in the trajectories with (**top**) cyclical and (**bottom**) constant learning rate schedules.

for the learning rate at iteration $i$ is given by

$$\begin{aligned} \alpha(i) &= (1 - t(i))\alpha_1 + t(i)\alpha_2, \\ t(i) &= \frac{1}{c}\left(\mathrm{mod}(i-1, c) + 1\right). \end{aligned}$$

The base learning rates $\alpha_1 \geqslant \alpha_2$ and the cycle length $c$ are the hyper-parameters of the method. Here by iteration we assume the processing of one batch of data. Figure 2.6 illustrates the cyclical learning rate schedule and the test error of the corresponding points. Note that unlike the cyclical learning rate schedule of Garipov et al. [2018] and Smith and Topin [2017], here we propose to use a discontinuous schedule that jumps directly from the minimum to maximum learning rates, and does *not* steadily increase the learning rate as part of the cycle. We use this more abrupt cycle because for our purposes exploration is more important than the accuracy of individual proposals. For even greater exploration, we also consider constant learning rates $\alpha(i) = \alpha_1$.

We run SGD with cyclical and constant learning rate schedules starting from a pretrained point for a Preactivation ResNet-164 on CIFAR-100. We then use the first, middle and last point of each of the trajectories to define a 2-dimensional plane in the weight space containing all affine combinations of these points. In Figure 2.7 we plot the loss on train and error on test for points in these planes. We then project the other points of the trajectory to the plane of the plot. Note that the trajectories do not generally lie in the plane of the plot, except for the first, last and middle points, showed by black crosses in the figure. Therefore for other points of the trajectories it is not possible to tell the value of train loss and test error from the plots.

The key insight from Figure 2.7 is that both methods explore points close to the periphery of the set of high-performing networks. The visualizations suggest that both methods are doing exploration in the region of space corresponding to DNNs with high accuracy. The main difference between the two approaches is that the individual proposals of SGD with a cyclical learning rate schedule are in general much more accurate than the proposals of a fixed-learning rate SGD. After making a large step, SGD with a cyclical learning rate spends several epochs fine-tuning the resulting point with a decreasing learning rate. SGD with a fixed learning rate on the other hand is always making steps of relatively large sizes, exploring more efficiently than with a cyclical learning rate, but the individual proposals are worse.

Another important insight we can get from Figure 2.7 is that while the train loss and test error surfaces are qualitatively similar, they are not perfectly aligned. The shift between train and test suggests that more robust central points in the set of high-performing networks can lead to better generalization. Indeed, if we average several proposals from the optimization trajectories, we get a more robust point that has a substantially higher test performance than the individual proposals of SGD, and is essentially centered on the shifted mode for test error. We further discuss the reasons for this behaviour in Sections 2.5.4, 2.5.5, 3.4.

**Algorithm 1** Stochastic Weight Averaging

```
Require:
  weights $\hat{w}$, LR bounds $\alpha_1, \alpha_2$,
  cycle length $c$ (for constant learning rate $c = 1$), number of iterations $n$
Ensure: $w_{\text{SWA}}$
  $w \leftarrow \hat{w}$ {Initialize weights with $\hat{w}$}
  $w_{\text{SWA}} \leftarrow w$
  for $i \leftarrow 1, 2, \ldots, n$ do
    $\alpha \leftarrow \alpha(i)$ {Calculate LR for the iteration}
    $w \leftarrow w - \alpha \nabla \mathcal{L}_i(w)$ {Stochastic gradient update}
    if $\text{mod}(i, c) = 0$ then
      $n_{\text{models}} \leftarrow i/c$ {Number of models}
      $w_{\text{SWA}} \leftarrow \frac{w_{\text{SWA}} \cdot n_{\text{models}} + w}{n_{\text{models}} + 1}$ {Update average}
    end if
  end for
  {Compute BatchNorm statistics for $w_{\text{SWA}}$ weights}
```

### 2.5.2 SWA Algorithm

We now present the details of the Stochastic Weight Averaging algorithm, a simple but effective modification for training neural networks, motivated by our observations in section 2.5.1.

Following Garipov et al. [2018], we start with a pretrained model $\hat{w}$. We will refer to the number of epochs required to train a given DNN with the conventional training procedure as its training budget and will denote it by $B$. The pretrained model $\hat{w}$ can be trained with the conventional training procedure for full training budget or reduced number of epochs (e.g. $0.75B$). In the latter case we just stop the training early without modifying the learning rate schedule. Starting from $\hat{w}$ we continue training, using a cyclical or constant learning rate schedule. When using a cyclical learning rate we capture the models $w_i$ that correspond to the minimum values of the learning rate (see Figure 2.6), following Garipov et al. [2018]. For constant learning rates we capture models at each epoch. Next, we average the weights of all the captured networks $w_i$ to get our final model $w_{\text{SWA}}$.

Note that for cyclical learning rate schedule, the SWA algorithm is related to FGE (Section 2.4), except that instead of averaging the predictions of the models, we average their weights, and we use a different type of learning rate cycle. In section 2.5.5 we show how SWA can approximate FGE, but with a single model.

Batch Normalization. If the DNN uses batch normalization [Ioffe and Szegedy 2015], we run one additional pass over the data, as described in Appendix A.1.2, to compute the running mean and standard deviation of the activations for each layer of the network with $w_{\text{SWA}}$ weights after the training is finished, since these statistics are not collected during training. For most deep learning libraries, such as PyTorch or Tensorflow, one can typically collect these statistics by making a forward pass over the data in training mode.

The SWA procedure is summarized in Algorithm 1.

### 2.5.3 Computational Complexity

The time and memory overhead of SWA compared to conventional training is negligible. During training, we need to maintain a copy of the running average of DNN weights. Note however that the memory consumption in storing a DNN is dominated by its activations rather than its weights, and thus is only slightly increased by the SWA procedure, even for large DNNs (e.g., on the order of 10%). After the training is complete we only need to store the model that aggregates the average, leading to the same memory requirements as standard training.

During training extra time is only spent to update the aggregated weight average. This operation is of the form

$$w_{\text{SWA}} \leftarrow \frac{w_{\text{SWA}} \cdot n_{\text{models}} + w}{n_{\text{models}} + 1},$$

and it only requires computing a weighted sum of the weights of two DNNs. As we apply this operation at most once per epoch, SWA and SGD require practically the same amount of compu-

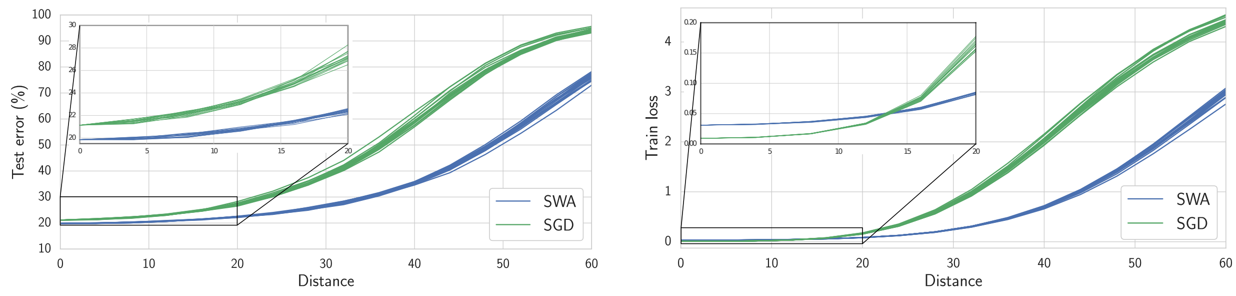


**Figure 2.8:** (**Left**) Test error and (**Right**) $L_2$-regularized cross-entropy train loss as a function of a point on a random ray starting at SWA (blue) and SGD (green) solutions for Preactivation ResNet-164 on CIFAR-100. Each line corresponds to a different random ray.

tation. Indeed, a similar operation is performed as a part of each gradient step, and each epoch consists of hundreds of gradient steps.

### 2.5.4 Solution Width

Keskar et al. [2017] and Chaudhari et al. [2016] conjecture that the width of a local optimum is related to generalization. The general explanation for the importance of width is that the surfaces of train loss and test error are shifted with respect to each other and it is thus desirable to converge to the modes of broad optima, which stay approximately optimal under small perturbations. In this section we compare the solutions found by SWA and SGD and show that SWA generally leads to much wider solutions.

Let $w_{\text{SWA}}$ and $w_{\text{SGD}}$ denote the weights of DNNs trained using SWA and conventional SGD, respectively. Consider the rays

$$w_{\text{SWA}}(t, d) = w_{\text{SWA}} + t \cdot d,$$
$$w_{\text{SGD}}(t, d) = w_{\text{SGD}} + t \cdot d,$$

which follow a direction vector $d$ on the unit sphere, starting at $w_{\text{SWA}}$ and $w_{\text{SGD}}$, respectively. In Figure 2.8 we plot train loss and test error of $w_{\text{SWA}}(t, d_i)$ and $w_{\text{SGD}}(t, d_i)$ as a function of $t$ for 10 random directions $d_i$, $i = 1, 2, \dots, 10$ drawn from a uniform distribution on the unit sphere. For

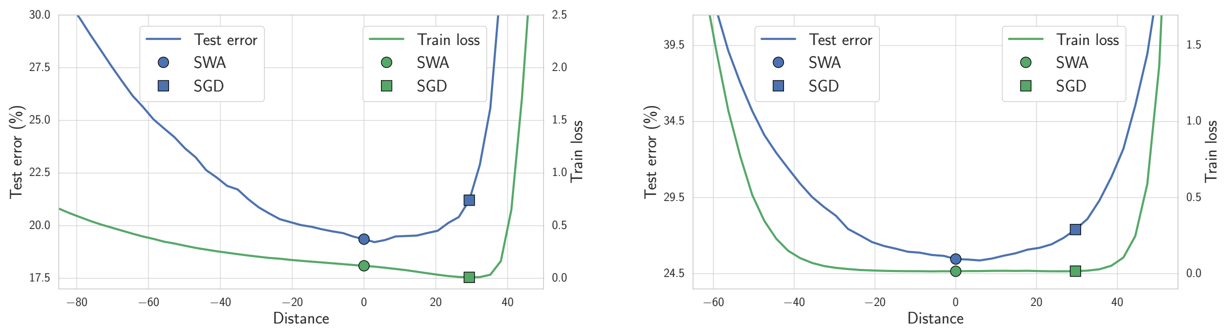


**Figure 2.9:** $L_2$-regularized cross-entropy train loss and test error as a function of a point on the line connecting SWA and SGD solutions on CIFAR-100. **Left**: Preactivation ResNet-164. **Right**: VGG-16.

this visualization we use a Preactivation ResNet-164 on CIFAR-100.

First, while the loss values on train for $w_{\text{SGD}}$ and $w_{\text{SWA}}$ are quite similar (and in fact $w_{\text{SGD}}$ has a slightly lower train loss), the test error for $w_{\text{SGD}}$ is lower by 1.5% (at the converged value corresponding to $t = 0$). Further, the shapes of both train loss and test error curves are considerably wider for $w_{\text{SWA}}$ than for $w_{\text{SGD}}$, suggesting that SWA indeed converges to a wider solution: we have to step much further away from $w_{\text{SWA}}$ to increase error by a given amount. We even see the error curve for SGD has an inflection point that is not present for these distances with SWA.

Notice that in Figure 2.8 any of the random directions from $w_{\text{SGD}}$ increase test error. However, we know that the direction from $w_{\text{SGD}}$ to $w_{\text{SWA}}$ would decrease test error, since $w_{\text{SWA}}$ has considerably lower test error than $w_{\text{SGD}}$. In other words, the path from $w_{\text{SGD}}$ to $w_{\text{SWA}}$ is qualitatively different from all directions shown in Figure 2.8, because along this direction $w_{\text{SGD}}$ is far from optimal. We therefore consider the line segment connecting $w_{\text{SGD}}$ and $w_{\text{SWA}}$:

$$w(t) = t \cdot w_{\text{SGD}} + (1 - t) \cdot w_{\text{SWA}} \,.$$

In Figure 2.9 we plot the train loss and test error of $w(t)$ as a function of signed distance from $w_{\text{SWA}}$ for Preactivation ResNet-164 and VGG-16 on CIFAR-100.

We can extract several key insights about $w_{\text{SWA}}$ and $w_{\text{SGD}}$ from Figure 2.9. First, the train loss and test error plots are indeed substantially shifted, and the point obtained by minimizing the train loss is far from optimal on test. Second, $w_{\text{SGD}}$ lies near the boundary of a wide flat region of the train loss. Further, the loss is very steep near $w_{\text{SGD}}$.

Keskar et al. [2017] argue that the loss near sharp optima found by SGD with very large batches are actually flat in most directions, but there exist directions in which the optima are extremely steep. They conjecture that because of this sharpness the generalization performance of large batch optimization is substantially worse than that of solutions found by small batch SGD. Remarkably, in our experiments in this section we observe that there exist directions of steep ascent even for small batch optima, and that SWA provides even wider solutions (at least along random directions) with better generalization. Indeed, we can see clearly in Figure 2.9 that SWA is not finding a different basin than SGD, but rather a flatter region in the same basin of attraction. We can also see clearly that the significant asymmetry of the loss function in certain directions, such as the direction SWA to SGD, has a role in understanding why SWA provides better generalization than SGD. In these directions SWA finds a much flatter solution than SGD, which can be near the periphery of sharp ascent.

### 2.5.5 Connection to Ensembling

In Section 2.4 we described the Fast Geometric Ensembling (FGE) procedure for training ensembles in the time required to train a single model. Using a cyclical learning rate, FGE generates a sequence of points that are close to each other in the weight space, but produce diverse predictions. In SWA instead of averaging the predictions of the models we average their weights. However, the predictions proposed by FGE ensembles and SWA models have similar properties.

Let $f(\cdot)$ denote the predictions of a neural network parametrized by weights $w$. We will assume that $f$ is a scalar (e.g. the probability for a particular class) twice continuously differentiable function with respect to $w$.

Consider points $w_i$ proposed by FGE. These points are close in the weight space by design, and concentrated around their average $w_{\text{SWA}} = \frac{1}{n}\sum_{i=1}^{n} w_i$. We denote $\Delta_i = w_i - w_{\text{SWA}}$. Note

$\sum_{i=1}^{n} \Delta_i = 0$. Ensembling the networks corresponds to averaging the function values

$$\bar{f} = \frac{1}{n} \sum_{i=1}^{n} f(w_i).$$

Consider the linearization of $f$ at $w_{\text{SWA}}$.

$$f(w_j) = f(w_{\text{SWA}}) + \left\langle \nabla f(w_{\text{SWA}}), \Delta_j \right\rangle + O(\|\Delta_j\|^2),$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product. Thus, the difference between averaging the weights and averaging the predictions

$$\begin{aligned} \bar{f} - f(w_{\text{SWA}}) &= \frac{1}{n} \sum_{i=1}^{n} \left( \langle \nabla f(w_{\text{SWA}}), \Delta_i \rangle + O(\|\Delta_i\|^2) \right) \\ &= \left\langle \nabla f(w_{\text{SWA}}), \frac{1}{n} \sum_{i=1}^{n} \Delta_i \right\rangle + O(\Delta^2) = O(\Delta^2), \end{aligned}$$

where $\Delta = \max_{i=1}^{n} \|\Delta_i\|$. Note that the difference between the predictions of different perturbed networks is

$$f(w_i) - f(w_j) = \langle \nabla f(w_{\text{SWA}}), \Delta_i - \Delta_j \rangle + O(\Delta^2),$$

and is thus of the first order of smallness, while the difference between averaging predictions and averaging weights is of the second order of smallness. Note that for the points proposed by FGE the distances between proposals are relatively small by design, which justifies the local analysis.

To analyze the difference between ensembling and averaging the weights of FGE proposals in practice, we run FGE for 20 epochs and compare the predictions of different models on the test dataset with a Preactivation ResNet-164 [He et al. 2016] on CIFAR-100. The norm of the difference between the class probabilities of consecutive FGE proposals averaged over the test dataset is 0.126. We then average the weights of the proposals and compute the class probabilities on the test dataset. The norm of difference of the probabilities for the SWA model and the FGE ensemble is 0.079, which is substantially smaller than the difference between the probabilities of consecutive FGE proposals. Further, the fraction of objects for which consecutive FGE proposals output the same labels is not greater than 87.33%. For FGE and SWA the fraction of identically labeled objects is 95.26%.

The theoretical considerations and empirical results presented in this section suggest that SWA can approximate the FGE ensemble with a single model.

### 2.5.6 Connection to Convex Minimization

Mandt et al. [2017b] showed that under strong simplifying assumptions SGD with a fixed learning rate approximately samples from a Gaussian distribution centered at the minimum of the loss. Suppose this is the case when we run SGD with a fixed learning rate for training a DNN.

Let us denote the dimensionality of the weight space of the neural network by $d$. Denote the samples produced by SGD by $w_i,\ i = 1, 2, \dots, k$. Assume the points $w_i$ are concentrated around the local optimum $\hat{w}$. The SWA solution is given by $w_{\text{SWA}} = \frac{1}{n}\sum_{i=1}^{k} w_i$. The points $w_i$ are samples from a multidimensional Gaussian $\mathcal{N}(\hat{w}, \Sigma)$ for some covariance matrix $\Sigma$ defined by the curvature of the loss, batch size and the learning rate. Note that the samples from a multidimensional Gaussian are concentrated on the ellipsoid

$$\left\{ z \in \mathbb{R}^d \middle| \ \|\Sigma^{-\frac{1}{2}}(z - \hat{w})\| = \sqrt{d} \right\},$$

and the probability mass for a sample to end up inside the ellipsoid near $\hat{w}$ is negligible. On the other hand, $w_{\text{SWA}}$ is guaranteed to converge to $\hat{w}$ as $k \to \infty$.

Moreover, Polyak and Juditsky [1992] showed that averaging SGD proposals achieves the best possible convergence rate among all stochastic gradient algorithms. The proof relies on the convexity of the underlying problem and in general there are no convergence guarantees if the loss function is non-convex [see e.g. Ghadimi and Lan 2013]. While DNN loss functions are known to be non-convex [e.g. Choromanska et al. 2015], over the trajectory of SGD these loss surfaces are approximately convex [e.g. Goodfellow et al. 2015]. However, even when the loss is locally non-convex, SWA can improve *generalization*. For example, in Figure 2.9 we see that SWA converges to a central point of the training loss.

In other words, there are a set of points that all achieve low training loss. By running SGD with a high constant or cyclical schedule, we traverse over the surface of this set. Then by averaging the corresponding iterates, we get to move inside the set. This observation explains both convergence rates and generalization. In deep learning we mostly observe benefits in generalization from averaging. Averaging can move to a more central point, which means one has to move further from this point to increase the loss by a given amount, in virtually any direction. By contrast, conventional SGD with a decaying schedule will converge to a point on the periphery of this set. With different initializations conventional SGD will find different points on the boundary, of solutions with low training loss, but it will not move inside.

## 2.6 Experiments

We compare SWA against conventional SGD training and FGE on CIFAR-10, CIFAR-100 and ImageNet ILSVRC-2012 [Russakovsky et al. 2012]. We note that FGE is an ensemble whereas SWA corresponds to a single model. Conventional SGD training uses a standard decaying learning rate schedule (details in the Appendix) until convergence. We found an exponentially decaying average of SGD to perform comparably to conventional SGD at convergence. The code for reproducing the results in this section is available at `https://github.com/timgaripov/swa`.

| | | | SWA | | |
|---|---|---|---|---|---|
| DNN (Budget) | SGD | FGE (1 Budget) | 1 Budget | 1.25 Budgets | 1.5 Budgets |
| | | CIFAR-100 | | | |
| VGG-16 (200) | 72.55 ± 0.10 | 74.26 | 73.91 ± 0.12 | 74.17 ± 0.15 | 74.27 ± 0.25 |
| ResNet-164 (150) | 78.49 ± 0.36 | 79.84 | 79.77 ± 0.17 | 80.18 ± 0.23 | 80.35 ± 0.16 |
| WRN-28-10 (200) | 80.82 ± 0.23 | 82.27 | 81.46 ± 0.23 | 81.91 ± 0.27 | 82.15 ± 0.27 |
| PyramidNet-272 (300) | 83.41 ± 0.21 | – | – | 83.93 ± 0.18 | 84.16 ± 0.15 |
| | | CIFAR-10 | | | |
| VGG-16 (200) | 93.25 ± 0.16 | 93.52 | 93.59 ± 0.16 | 93.70 ± 0.22 | 93.64 ± 0.18 |
| ResNet-164 (150) | 95.28 ± 0.10 | 95.45 | 95.56 ± 0.11 | 95.77 ± 0.04 | 95.83 ± 0.03 |
| WRN-28-10 (200) | 96.18 ± 0.11 | 96.36 | 96.45 ± 0.11 | 96.64 ± 0.08 | 96.79 ± 0.05 |
| ShakeShake-2x64d (1800) | 96.93 ± 0.10 | – | – | 97.16 ± 0.10 | 97.12 ± 0.06 |

**Table 2.1:** Accuracies (%) of SWA, SGD and FGE methods on CIFAR-100 and CIFAR-10 datasets for different training budgets. Accuracies for the FGE ensemble are from Garipov et al. [2018].

### 2.6.1 CIFAR Datasets

For the experiments on CIFAR datasets we use VGG-16 [Simonyan and Zisserman 2014], a 164-layer Preactivation-ResNet [He et al. 2016] and Wide ResNet-28-10 [Zagoruyko and Komodakis 2016] models. Additionally, we experiment with the recent Shake-Shake-2x64d [Gastaldi 2017] on CIFAR-10 and PyramidNet-272 (bottleneck, $\alpha = 200$) [Han et al. 2016] on CIFAR-100. All models are trained using $L_2$-regularization, and VGG-16 also uses dropout.

For each model we define *budget* as the number of epochs required to train the model until convergence with conventional SGD training, such that we do not see improvement with SGD beyond this budget. We use the same budgets for VGG, Preactivation ResNet and Wide ResNet models as Garipov et al. [2018]. For Shake-Shake and PyramidNets we use the budgets indicated by the papers that proposed these models [Gastaldi 2017; Han et al. 2016]. We report the results of SWA training within 1, 1.25 and 1.5 budgets of epochs.

For VGG, Wide ResNet and Preactivation-ResNet models we first run standard SGD training for $\approx 75\%$ of the training budget, and then use the weights at the last epoch as an initialization for SWA with a fixed learning rate schedule. We ran SWA for 0.25, 0.5 and 0.75 budget to complete the training within 1, 1.25 and 1.5 budgets respectively.

For Shake-Shake and PyramidNet architectures we do not report the results in one budget. For these models we use a full budget to get an initialization for the procedure, and then train with a cyclical learning rate schedule for 0.25 and 0.5 budgets. We used long cycles of small learning rates for Shake-Shake, because this architecture already involves many stochastic components.

We present the details of the learning rate schedules for each of these models in the Appendix.

For each model we also report the results of conventional SGD training, which we denote by **SGD**. For VGG, Preactivation ResNet and Wide ResNet we also provide the results of the **FGE** method with one budget. Note that for FGE we report the accuracy of an ensemble of 6 to 12 networks, while for SWA we report the accuracy of a single model.

We summarize the experimental results in Table 2.1. For all models we report the mean and standard deviation of test accuracy over 3 runs. In all conducted experiments SWA substantially outperforms SGD in one budget, and improves further, as we allow more training epochs. Across different architectures we see consistent improvement by $\approx 0.5\%$ on CIFAR-10 (excluding Shake-Shake, for which SGD performance is already extremely high) and by 0.75-1.5% on CIFAR-100. Amazingly, SWA is able to achieve comparable or better performance than FGE ensembles with just one model. On CIFAR-100 SWA usually needs more than one budget to get results comparable with FGE ensembles, but on CIFAR-10 even with 1 budget SWA outperforms FGE.

## 2.6.2 ImageNet

On ImageNet we experimented with ResNet-50, ResNet-152 [He et al. 2016] and DenseNet-161 [Huang et al. 2017b]. For these architectures we used pretrained models from `PyTorch.torchvision`. For each of the models we ran SWA for 10 epochs with a cyclical learning rate schedule with the same parameters for all models (the details can be found in Appendix A.2.1), and report the mean and standard deviation of test error averaged over 3 runs. The results are shown in Table 2.2. For all 3 architectures SWA provides consistent improvement by 0.6-0.9% over the pretrained models.

| DNN | SGD | SWA 5 epochs | SWA 10 epochs |
|---|---|---|---|
| ResNet-50 | 76.15 | 76.83 ± 0.01 | 76.97 ± 0.05 |
| ResNet-152 | 78.31 | 78.82 ± 0.01 | 78.94 ± 0.07 |
| DenseNet-161 | 77.65 | 78.26 ± 0.09 | 78.44 ± 0.06 |

**Table 2.2:** Top-1 accuracies (%) on ImageNet for SWA and SGD with different architectures.

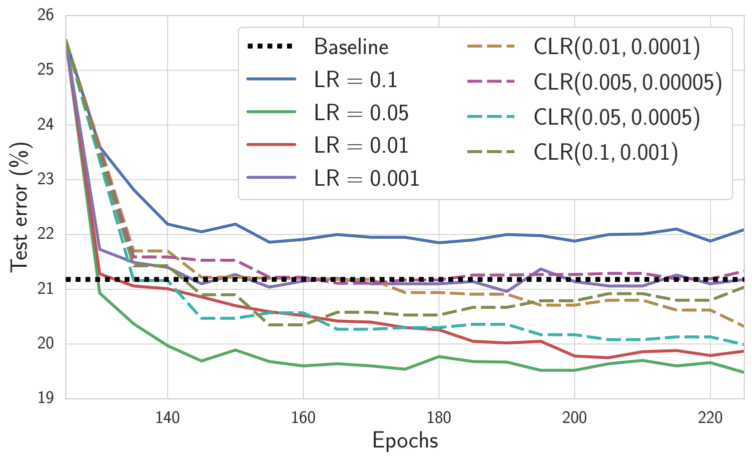


**Figure 2.10:** Test error as a function of training epoch for SWA with different learning rate schedules with a Preactivation ResNet-164 on CIFAR-100.

### 2.6.3 Effect of the Learning Rate Schedule

In this section we explore how the learning rate schedule affects the performance of SWA. We run experiments on Preactivation ResNet-164 on CIFAR-100. For all schedules we use the same initialization from a model trained for 125 epochs using the conventional SGD training. As a baseline we use a fully-trained model trained with conventional SGD for 150 epochs.

We consider a range of constant and cyclical learning rate schedules. For cyclical learning rates we fix the cycle length to 5, and consider the pairs of base learning rate parameters $(\alpha_1, \alpha_2) \in \{(10^{-1}, 10^{-3}), (5\cdot 10^{-2}, 5\cdot 10^{-4}), (10^{-2}, 10^{-4}), (5\cdot 10^{-3}, 5\cdot 10^{-5})\}$. Among the constant learning rates we consider $\alpha_1 \in \{10^{-1}, 5\cdot 10^{-2}, 10^{-2}, 10^{-3}\}$.

We plot the test error of the SWA procedure for different learning rate schedules as a function of the number of training epochs in Figure 2.10.

We find that in general the more aggressive constant learning rate schedule leads to faster convergence of SWA. In our experiments we found that setting the learning rate to some intermediate value between the largest and the smallest learning rate used in the annealing scheme in conventional training usually gave us the best results. The approach is however universal and can work well with different learning rate schedules tailored for particular tasks.

## 2.7 Related Work

Despite the success of deep learning across many application domains, the loss surfaces of deep neural networks are not well understood. These loss surfaces are an active area of research, which falls into two distinct categories.

The first category explores the local structure of minima found by SGD and its modifications. Researchers typically distinguish sharp and wide local minima, which are respectively found by using large and small mini-batch sizes during training. Hochreiter and Schmidhuber [1997a] and Keskar et al. [2017], for example, claim that flat minima lead to strong generalization, while sharp minima deliver poor results on the test dataset. However, recently Dinh et al. [2017a] argue that most existing notions of flatness cannot directly explain generalization. To better understand the local structure of DNN loss minima, Li et al. [2017] proposed a new visualization method for the loss surface near the minima found by SGD. Applying the method for a variety of different architectures, they showed that the loss surfaces of residual networks are seemingly smoother than those of VGG-like models.

The other major category of research considers global loss structure. One of the main questions in this area is how neural networks are able to overcome poor local optima. Choromanska et al. [2015] investigated the link between the loss function of a simple fully-connected network and the Hamiltonian of the spherical spin-glass model. Under strong simplifying assumptions they showed that the values of the investigated loss function at local optima are within a well-defined bound. In other research, Lee et al. [2016a] showed that under mild conditions gradient descent almost surely converges to a local minimizer and not a saddle point, starting from a random initialization.

Freeman and Bruna [2017] theoretically show that local minima of a neural network with one hidden layer and ReLU activations can be connected with a curve along which the loss is upper-bounded by a constant that depends on the number of parameters of the network and the "smoothness of the data". Their theoretical results do not readily generalize to multilayer networks. Using a dynamic programming approach they empirically construct a polygonal chain for a CNN on MNIST and an RNN on PTB next word prediction. However, in more difficult settings such as AlexNet on CIFAR-10 their approach struggles to achieve even the modest test accuracy of 80%. Moreover, they do not consider ensembling.

By contrast, in this chapter we describe a much simpler training procedure that can find near-constant accuracy polygonal chains with only one bend between optima, on a range of architectures. Inspired by properties of the loss function discovered by our procedure, we also propose a new state-of-the-art ensembling method that can be trained in the time required to train a single DNN, with compelling performance on many key benchmarks (e.g., 96.4% accuracy on CIFAR-10).

Fast Ensembling Methods. Xie et al. [2013] proposed a related ensembling approach that gathers outputs of neural networks from different epochs at the end of training to stabilize final predictions. More recently, Huang et al. [2017a] proposed *snapshot ensembles*, which use a cosine cyclical learning rate [Loshchilov and Hutter 2017b] to save "snapshots" of the model during training at times when the learning rate achieves its minimum. In our experiments, we compare our geometrically inspired approach to Huang et al. [2017a], showing improved performance.

Extensions of Mode Connectivity. Draxler et al. [2018a] simultaneously and independently discovered the existence of curves connecting local optima in DNN loss landscapes. To find these curves they used a different approach inspired by the *Nudged Elastic Band* method [Jonsson et al. 1998] from quantum chemistry. Skorokhodov and Burtsev [2019] use a method inspired by mode-connectivity to show that it is possible to find arbitrary shapes in the loss surfaces of neural networks. Benton et al. [2021] generalize our observations to construct mode-connecting simplicial complexes that form multi-dimensional manifolds of low loss. Fort and Jastrzebski [2019] present a large-scale model of neural network loss surfaces that agrees with mode connectivity. Kuditipudi et al. [2019] provide a theoretical explanation of mode connectivity. Recently, Ainsworth et al. [2022] showed that it is often possible to achieve *linear mode connectivity* if we account for permutation invariances in the parameterization of neural networks.

Weight Averaging. The SWA method is based on averaging multiple points along the trajectory of SGD with cyclical or constant learning rates. The general idea of maintaining a running average of weights proposed by SGD was first considered in convex optimization by Ruppert [1988] and later by Polyak and Juditsky [1992]. However, this procedure is not typically used to train neural networks. Practitioners instead sometimes use an exponentially decaying running average of the weights found by SGD with a decaying learning rate, which smooths the trajectory of SGD but performs comparably.

SWA is making use of multiple samples gathered through exploration of the set of points corresponding to high performing networks. To enforce exploration we run SGD with constant or cyclical learning rates. Mandt et al. [2017b] show that under several simplifying assumptions running SGD with a constant learning rate is equivalent to sampling from a Gaussian distribution centered at the minimum of the loss, and the covariance of this Gaussian is controlled by the learning rate. Following this explanation from [Mandt et al. 2017b], we can interpret points proposed by SGD as being constrained to the surface of a sphere, since they come from a high dimensional Gaussian distribution. SWA effectively allows us to go inside the sphere to find higher density solutions.

Dropout [Srivastava et al. 2014] is an extremely popular approach to regularizing DNNs. Across each mini-batch used for SGD, a different architecture is created by randomly dropping out neurons. The authors make analogies between dropout, ensembling, and Bayesian model averaging. At test time, an ensemble approach is proposed, but then approximated with similar results by multiplying each connection by the dropout rate. At a high level, SWA and Dropout are both at once regularizers and training procedures, motivated to approximate an ensemble. Each approach implements these high level ideas quite differently, and as we show in our experiments, can be combined for improved performance.

## 2.8 Discussion

We have shown that the optima of deep neural networks are connected by simple pathways, such as a polygonal chain with a single bend, with near constant accuracy. We introduced a training procedure to find these pathways, with a user-specific curve of choice. We were inspired by these insights to propose a practical new training method, Stochastic Weight Averaging, which achieves strong results on CIFAR-10, CIFAR-100, and ImageNet. Stochastic Weight Averaging is extremely easy to implement, architecture-agnostic, and improves generalization performance at virtually no additional cost over conventional training.

At a high level we have shown that even though the loss surfaces of deep neural networks are very complex, there is relatively simple structure connecting different optima. Indeed, we can now move towards thinking about valleys of low loss, rather than isolated modes. A better understanding of the loss surfaces for multilayer networks will help continue to unlock the potential of these rich models. In Chapters 3 and 4 we show how mode connectivity connects to Bayesian deep learning and probabilistic perspective on deep learning.

In this chapter, we also showed that SGD typically converges to the boundary of a vast low-loss region, and by averaging the SGD iterates with SWA, we can move towards the center of this region. In subsequent works, we developed extensions of SWA for reinforcement learning [Nikishin et al. 2018], semi-supervised learning [Athiwaratkun et al. 2019] and uncertainty estimation [Chapter 3; Maddox et al. 2019]. SWA has been applied to numerous domains from skin lesion segmentation [Tang et al. 2019] and cosmic microwave background inference [Lemos et al. 2022] to low-precision training [Yang et al. 2019a] and domain generalization [Cha et al. 2021]. Due to its high popularity, SWA was added to the core PyTorch library, among a small selection of optimizers [Izmailov et al. 2020], and discussed in several modern deep learning textbooks [Zhang et al. 2021; Murphy 2023]. SWA continues to inspire new methods, such as efficient training of language models [Li et al. 2022] and the current state-of-the-art on ImageNet [Wortsman et al. 2022].

# 3 | Probabilistic Models and Bayesian Deep Learning

Deep neural networks are highly flexible models that are typically not fully determined by the training data. Indeed, in Chapter 2 we explored in detail the optima of neural networks and the diversity of predictions that they make on the test set. We have also seen how these optima can be combined for improved predictions by ensembling.

Bayesian methods provide a natural and principled way to account for the *epistemic uncertainty* — uncertainty over the parameters of our model due to limited data. In particular, true Bayesian neural networks would automatically consider and ensemble *all* the possible settings of the parameters that are consistent with the training data. As a result, these models should provide better predictions and uncertainty estimates compared to standard training.

In this chapter, we describe the Bayesian approach to deep learning, and the probabilistic perspective on generalization, providing new insights into several phenomena in deep learning. We also propose practical methods for improved uncertainty estimation in neural networks. Throughout the chapter, we heavily rely on our observations about the loss surfaces of neural networks discussed in Chapter 2.

This chapter is adapted from the papers "Bayesian Deep Learning and a Probabilistic Perspective of Generalization" [Wilson and Izmailov 2020] which appeared at NeurIPS 2020 and "A Simple Baseline for Bayesian Uncertainty in Deep Learning" [Maddox et al. 2019] which appeared at NeurIPS 2019. These papers are written jointly with Wesley Maddox, Timur Garipov, Dmitry Vetrov and Andrew Gordon Wilson.

## 3.1 Bayesian Perspective on Generalization

Imagine fitting the airline passenger data in Figure 3.1. Which model would you choose: (1) $f_1(x) = w_0 + w_1 x$, (2) $f_2(x) = \sum_{j=0}^{3} w_j x^j$, or (3) $f_3(x) = \sum_{j=0}^{10^4} w_j x^j$?

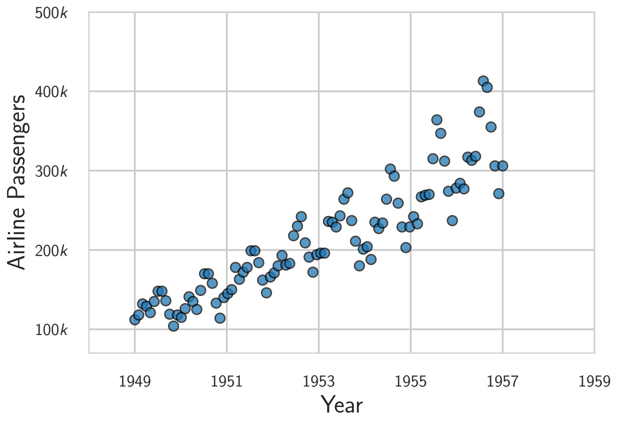
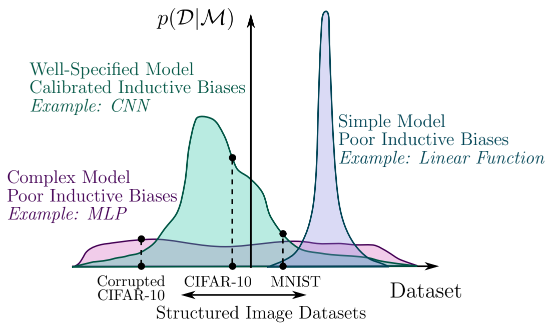


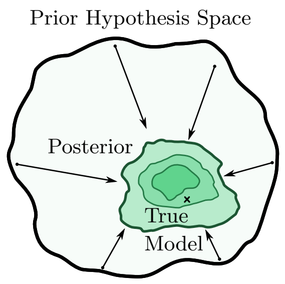
**Figure 3.1:** Airline passenger numbers recorded monthly.

Put this way, most audiences overwhelmingly favour choices (1) and (2), for fear of overfitting. But of these options, choice (3) most honestly represents our beliefs. Indeed, it is likely that the ground truth explanation for the data is out of class for any of these choices, but there is some setting of the coefficients $\{w_j\}$ in choice (3) which provides a better description of reality than could be managed by choices (1) and (2), which are special cases of choice (3). Moreover, our beliefs about the generative processes for our observations, which are often very sophisticated, typically ought to be independent of how many data points we happen to observe.

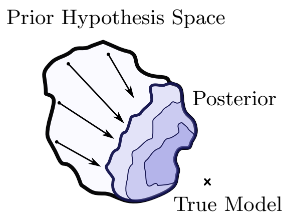
And in modern practice, we are implicitly favouring choice (3): we often use neural networks with millions of parameters to fit datasets with thousands of points. Furthermore, non-parametric methods such as Gaussian processes often involve infinitely many parameters, enabling the flexibility for universal approximation [Rasmussen and Williams 2006], yet in many cases provide



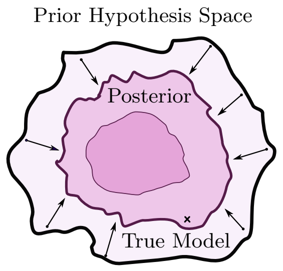
(a)



(b)



(c)



(d)

**Figure 3.2: A probabilistic perspective of generalization.** (a) Ideally, a model supports a wide range of datasets, but with inductive biases that provide high prior probability to a particular class of problems being considered. Here, the CNN is preferred over the linear model and the fully-connected MLP for CIFAR-10 (while we do not consider MLP models to in general have poor inductive biases, here we are considering a hypothetical example involving images and a very large MLP). (b) By representing a large hypothesis space, a model can contract around a true solution, which in the real-world is often very sophisticated. (c) With truncated support, a model will converge to an erroneous solution. (d) Even if the hypothesis space contains the truth, a model will not efficiently contract unless it also has reasonable inductive biases.

very simple predictive distributions. Indeed, parameter counting is a poor proxy for understanding generalization behaviour.

From a probabilistic perspective, we argue that generalization depends largely on *two* properties, the *support* and the *inductive biases* of a model. Consider Figure 3.2(a), where on the horizontal axis we have a conceptualization of all possible datasets, and on the vertical axis the Bayesian *evidence* for a model. The evidence, or marginal likelihood, $p(\mathcal{D}|\mathcal{M}) = \int p(\mathcal{D}|\mathcal{M}, w)p(w)dw$, is the probability we would generate a dataset if we were to randomly sample from the prior over functions $p(f(x))$ induced by a prior over parameters $p(w)$. We define the support as the range of datasets for which $p(\mathcal{D}|\mathcal{M}) > 0$. We define the inductive biases as the relative prior probabilities of different datasets — the *distribution of support* given by $p(\mathcal{D}|\mathcal{M})$. A similar schematic to Figure 3.2(a) was used by MacKay [1992a] to understand an Occam's razor effect in using the evidence for model selection; we believe it can also be used to reason about model construction and generalization.

From this perspective, we want the support of the model to be large so that we can represent any hypothesis we believe to be possible, even if it is unlikely. We would even want the model to be able to represent pure noise, such as noisy CIFAR [Zhang et al. 2017a], as long as we honestly believe there is some non-zero, but potentially arbitrarily small, probability that the data are simply noise. Crucially, we also need the inductive biases to carefully represent which hypotheses we believe to be a priori likely for a particular problem class. If we are modelling images, then our model should have statistical properties, such as convolutional structure, which are good descriptions of images.

Figure 3.2(a) illustrates three models. We can imagine the blue curve as a simple linear function, $f(x) = w_0+w_1x$, combined with a distribution over parameters $p(w_0, w_1)$, e.g., $\mathcal{N}(0, I)$, which induces a distribution over functions $p(f(x))$. Parameters we sample from our prior $p(w_0, w_1)$ give rise to functions $f(x)$ that correspond to straight lines with different slopes and intercepts. This model thus has truncated support: it cannot even represent a quadratic function. But because the marginal likelihood must normalize over datasets $\mathcal{D}$, this model assigns much mass to the datasets it does support. The red curve could represent a large fully-connected MLP. This model is highly flexible, but distributes its support across datasets too evenly to be particularly compelling for many image datasets. The green curve could represent a convolutional neural network, which represents a compelling specification of support and inductive biases for image recognition: this model has the flexibility to represent many solutions, but its structural properties provide particularly good support for many image problems.

With large support, we cast a wide enough net that the posterior can contract around the true solution to a given problem as in Figure 3.2(b), which in reality we often believe to be very sophisticated. On the other hand, the simple model will have a posterior that contracts around an erroneous solution if it is not contained in the hypothesis space as in Figure 3.2(c). Moreover, in Figure 3.2(d), the model has wide support, but does not contract around a good solution because its support is too evenly distributed.

Returning to the opening example, we can justify the high order polynomial by wanting large support. But we would still have to carefully choose the prior on the coefficients to induce a distribution over functions that would have reasonable inductive biases. Indeed, this Bayesian notion of generalization is not based on a single number, but is a two dimensional concept. From this probabilistic perspective, it is crucial not to conflate the *flexibility* of a model with the *complexity* of a model class. Indeed Gaussian processes with RBF kernels have large support, and are thus flexible, but have inductive biases towards very simple solutions. We also see that *parameter counting* has no significance in this perspective of generalization: what matters is how a distribution over parameters combines with a functional form of a model, to induce a distribution over solutions. Rademacher complexity [Mohri and Rostamizadeh 2009], VC dimension [Vapnik 1998], and many conventional metrics, are by contrast *one dimensional notions*, corresponding roughly to the support of the model, which is why they have been found to provide an incomplete picture of generalization in deep learning [Zhang et al. 2017a].

The key distinguishing property of a Bayesian approach is marginalization instead of optimization, where we represent solutions given by all settings of parameters weighted by their posterior probabilities, rather than bet everything on a single setting of parameters. Neural networks are typically underspecified by the data, and can represent many different but high performing models corresponding to different settings of parameters, which is exactly when marginalization will make the biggest difference for accuracy and calibration. Moreover, we clarify that the recent deep ensembles [Lakshminarayanan et al. 2017] are not a competing approach to

Bayesian inference, but can be viewed as a compelling mechanism for Bayesian marginalization. Indeed, we empirically demonstrate that deep ensembles can provide a *better* approximation to the Bayesian predictive distribution than standard Bayesian approaches. We further propose a new method, MultiSWAG, inspired by deep ensembles, which marginalizes within basins of attraction — achieving significantly improved performance, with a similar training time.

We then investigate the properties of priors over functions induced by priors over the weights of neural networks, showing that they have reasonable inductive biases. We also show that the mysterious generalization properties recently presented in Zhang et al. [2017a] can be understood by reasoning about prior distributions over functions, and are not specific to neural networks. Indeed, we show Gaussian processes can also perfectly fit images with random labels, yet generalize on the noise-free problem. These results are a consequence of large support but reasonable inductive biases for common problem settings. We further show that while Bayesian neural networks can fit the noisy datasets, the marginal likelihood has much better support for the noise free datasets, in line with Figure 3.2. We additionally show that the multimodal marginalization in MultiSWAG alleviates double descent, so as to achieve monotonic improvements in performance with model flexibility, in line with our perspective of generalization. MultiSWAG also provides significant improvements in both accuracy and NLL over SGD training and unimodal marginalization. Finally we provide several perspectives on tempering in Bayesian deep learning.

In the Appendix we provide several additional experiments and results. We also provide code at `https://github.com/izmailovpavel/understandingbdl`.

## 3.2 Bayesian Marginalization

Often the predictive distribution we want to compute is given by

$$p(y|x, \mathcal{D}) = \int p(y|x, w)p(w|\mathcal{D})dw\,. \tag{3.1}$$

The outputs are $y$ (e.g., regression values, class labels, ...), indexed by inputs $x$ (e.g. spatial locations, images, ...), the weights (or parameters) of the neural network $f(x; w)$ are $w$, and $\mathcal{D}$ are the data. Eq. (3.1) represents a *Bayesian model average* (BMA). Rather than bet everything on one hypothesis — with a single setting of parameters $w$ — we want to use all settings of parameters, weighted by their posterior probabilities. This procedure is called *marginalization* of the parameters $w$, as the predictive distribution of interest no longer conditions on $w$. This is not a controversial equation, but simply the sum and product rules of probability.

### 3.2.1 Importance of Marginalization in Deep Learning

In general, we can view classical training as performing approximate Bayesian inference, using the approximate posterior $p(w|\mathcal{D}) \approx \delta(w = \hat{w})$ to compute Eq. (3.1), where $\delta$ is a Dirac delta function that is zero everywhere except at $\hat{w} = \text{argmax}_w p(w|\mathcal{D})$. In this case, we recover the standard predictive distribution $p(y|x, \hat{w})$. From this perspective, many alternatives, albeit imperfect, will be preferable — including impoverished Gaussian posterior approximations for $p(w|\mathcal{D})$, even if the posterior or likelihood are actually highly non-Gaussian and multimodal.

The difference between a classical and Bayesian approach will depend on how sharp the posterior $p(w|\mathcal{D})$ becomes. If the posterior is sharply peaked, and the conditional predictive distribution $p(y|x, w)$ does not vary significantly where the posterior has mass, there may be almost no difference, since a delta function may then be a reasonable approximation of the posterior for the purpose of BMA. However, modern neural networks are usually highly underspecified by the available data, and therefore have diffuse likelihoods $p(\mathcal{D}|w)$, not strongly favouring any one setting of parameters. Not only are the likelihoods diffuse, but different settings of the parameters correspond to a diverse variety of compelling hypotheses for the data [Garipov et al. 2018; Izmailov et al. 2019]. This is exactly the setting when we *most* want to perform a Bayesian model average, which will lead to an ensemble containing many different but high performing models, for better calibration *and* accuracy than classical training.

Loss Valleys. Flat regions of low loss (negative log posterior density $-\log p(w|\mathcal{D})$) are associated with good generalization [e.g., Hochreiter and Schmidhuber 1997a; Hinton and Van Camp 1993; Dziugaite and Roy 2017; Izmailov et al. 2018; Keskar et al. 2016]. While flat solutions that generalize poorly can be contrived through reparametrization [Dinh et al. 2017b], the flat regions that lead to good generalization contain a *diversity* of high performing models on test data [Izmailov et al. 2018], corresponding to different parameter settings in those regions. And indeed, as we discussed in Chapter 2, there are large contiguous regions of low loss that contain such solutions, even connecting together different SGD solutions [Garipov et al. 2018; Izmailov et al. 2019] (see also Figure A.15, Appendix).

Since these regions of the loss represent a large volume in a high-dimensional space [Huang et al. 2019], and provide a diversity of solutions, they will dominate in forming the predictive distribution in a Bayesian model average. By contrast, if the parameters in these regions provided similar functions, as would be the case in flatness obtained through reparametrization, these functions would be redundant in the model average. That is, although the solutions of high posterior density can provide poor generalization, it is the solutions that generalize well that will have greatest posterior *mass*, and thus be automatically favoured by the BMA.

Calibration by Epistemic Uncertainty Representation. It has been noticed that modern neural networks are often *miscalibrated* in the sense that their predictions are typically *overconfident* [Guo et al. 2017]. For example, in classification the highest softmax output of a convolutional neural network is typically much larger than the probability of the associated class label. The fundamental reason for miscalibration is ignoring epistemic uncertainty. A neural network can represent many models that are consistent with our observations. By selecting only one, in a classical procedure, we lose uncertainty when the models disagree for a test point. In regression, we can visualize epistemic uncertainty by looking at the spread of the predictive distribution; as we move away from the data, there are a greater variety of consistent solutions, leading to larger uncertainty, as in Figure 3.7. We can further calibrate the model with tempering, which we discuss in the Appendix Section 3.10.

Accuracy. An often overlooked benefit of Bayesian model averaging in *modern* deep learning is improved *accuracy*. If we average the predictions of many high performing models that disagree in some cases, we should see significantly improved accuracy. This benefit is now starting to be observed in practice [e.g., Izmailov et al. 2019]. Improvements in accuracy are very convincingly exemplified by *deep ensembles* [Lakshminarayanan et al. 2017], which have been perceived as a competing approach to Bayesian methods, but in fact provides a compelling mechanism for approximate Bayesian model averaging, as we show in Section 3.2.3. We also demonstrate significant accuracy benefits for multimodal Bayesian marginalization in Section 3.9.

### 3.2.2 Beyond Monte Carlo

Nearly all approaches to estimating the integral in Eq. (3.1), when it cannot be computed in closed form, involve a *simple Monte Carlo* approximation: $p(y|x,\mathcal{D}) \approx \frac{1}{J}\sum_{j=1}^{J} p(y|x,w_j)\,, w_j \sim p(w|\mathcal{D})$. In practice, the samples from the posterior $p(w|\mathcal{D})$ are also approximate, and found through MCMC or deterministic methods. The deterministic methods approximate $p(w|\mathcal{D})$ with a different more convenient density $q(w|\mathcal{D},\theta)$ from which we can sample, often chosen to be Gaussian. The parameters $\theta$ are selected to make $q$ close to $p$ in some sense; for example, variational approximations [e.g., Beal 2003], which have emerged as a popular deterministic approach, find $\text{argmin}_\theta \mathcal{KL}(q||p)$. Other standard deterministic approximations include Laplace [e.g., MacKay 1995], EP [Minka 2001], and INLA [Rue et al. 2009].

From the perspective of estimating the predictive distribution in Eq. (3.1), we can view simple Monte Carlo as approximating the posterior with a set of point masses, with locations given by samples from another approximate posterior $q$, even if $q$ is a continuous distribution. That is, $p(w|\mathcal{D}) \approx \sum_{j=1}^{J} \delta(w=w_j)\,, w_j \sim q(w|\mathcal{D})$.

Ultimately, the goal is to accurately compute the predictive distribution in Eq. (3.1), rather than find a generally accurate representation of the posterior. In particular, we must carefully represent the posterior in regions that will make the greatest contributions to the BMA integral. In terms of efficiently computing the predictive distribution, we do not necessarily want to place point masses at locations given by samples from the posterior. For example, functional diversity is important for a good approximation to the BMA integral, because we are summing together terms of the form $p(y|x,w)$; if two settings of the weights $w_i$ and $w_j$ each provide high likelihood (and consequently high posterior density), but give rise to similar functions $f(x;w_i)$, $f(x;w_j)$, then they will be largely redundant in the model average, and the second setting of parameters will not contribute much to estimating the BMA integral for the unconditional predictive distribution. In Sections 3.2.3 and 3.6, we consider how various approaches approximate the predictive distribution.

### 3.2.3 Deep Ensembles are BMA

*Deep ensembles* [Lakshminarayanan et al. 2017] is fast becoming a gold standard for accurate and well-calibrated predictive distributions. Recent reports [e.g., Ovadia et al. 2019; Ashukha et al. 2020] show that deep ensembles appear to outperform some particular approaches to Bayesian neural networks for uncertainty representation, leading to the confusion that deep ensembles and Bayesian methods are competing approaches. These methods are often explicitly referred to as non-Bayesian [e.g., Lakshminarayanan et al. 2017; Ovadia et al. 2019; Wenzel et al. 2020]. To the contrary, we argue that deep ensembles are actually a compelling approach to Bayesian model averaging, in the vein of Section 3.2.2.

There is a fundamental difference between a Bayesian model average and some approaches to ensembling. The Bayesian model average assumes that *one* hypothesis (one parameter setting) is correct, and averages over models due to an inability to distinguish between hypotheses given limited information [Minka 2000b]. As we observe more data, the posterior collapses onto a single hypothesis. If the true explanation for the data is a combination of hypotheses, then the Bayesian model average may appear to perform worse as we observe more data. Some ensembling methods work by enriching the hypothesis space, and therefore do not collapse in this way. Deep ensembles, however, are formed by MAP or maximum likelihood retraining of the same architecture multiple times, leading to different basins of attraction. The deep ensemble will therefore collapse in the same way as a Bayesian model average, as the posterior concentrates. Since the hypotheses space (support) for a modern neural network is large, containing many different possible explanations for the data, posterior collapse will often be desirable.

Furthermore, by representing multiple basins of attraction, deep ensembles can provide a *better* approximation to the BMA than the Bayesian approaches in Ovadia et al. [2019]. Indeed, the functional diversity is important for a good approximation to the BMA integral, as per Section 3.2.2. The approaches referred to as Bayesian in Ovadia et al. [2019] instead focus their approximation on a single basin, which may contain a lot of redundancy in function space, making a relatively minimal contribution to computing the Bayesian predictive distribution. On the other hand, retraining a neural network multiple times for deep ensembles incurs a significant computational expense. The single basin approaches may be preferred if we are to control for computation. We explore these questions in Section 3.6.

## 3.3 SWA-Gaussian

Bayesian methods provide a natural probabilistic representation of uncertainty in deep learning [e.g., Blundell et al. 2015; Kingma et al. 2015; Chen et al. 2014], and previously had been a gold standard for inference with neural networks [Neal 1996]. However, existing approaches are often highly sensitive to hyperparameter choices, and hard to scale to modern datasets and architectures, which limits their general applicability in modern deep learning.

In this section we propose a different approach to Bayesian deep learning: we use the information contained in the SGD trajectory to efficiently approximate the posterior distribution over the weights of the neural network. We find that the Gaussian distribution fitted to the first two moments of SGD iterates, with a modified learning rate schedule, captures the local geometry of the posterior surprisingly well. Using this Gaussian distribution we are able to obtain convenient, efficient, accurate and well-calibrated predictions in a broad range of tasks in computer vision.

We propose SWA-Gaussian in Sections 3.3.2 and 3.3.3 to estimate the covariance of the stationary distribution, forming a Gaussian approximation to the posterior over weight parameters. With SWAG, uncertainty in weight space is captured with minimal modifications to the SWA training procedure. We then present further theoretical and empirical analysis for SWAG in Section 3.4.

### 3.3.1 Stochastic Gradient Descent (SGD)

Standard training of deep neural networks (DNNs) proceeds by applying stochastic gradient descent on the model weights $\theta$ with the following update rule:

$$\Delta\theta_t = -\eta_t \left( \frac{1}{B} \sum_{i=1}^{B} \nabla_\theta \log p(y_i | f_\theta(x_i)) - \frac{\nabla_\theta \log p(\theta)}{N} \right),$$

where the learning rate is $\eta$, the $i$th input (e.g. image) and label are $\{x_i, y_i\}$, the size of the whole training set is $N$, the size of the batch is $B$, and the DNN, $f$, has weight parameters $\theta$.[1] The loss function is a negative log likelihood $-\sum_i \log p(y_i | f_\theta(x_i))$, combined with a regularizer $\log p(\theta)$. This type of maximum likelihood training does not represent uncertainty in the predictions or parameters $\theta$.

[1]We ignore momentum for simplicity in this update; however we utilized momentum in the resulting experiments and it is covered theoretically [Mandt et al. 2017a].

### 3.3.2 SWAG-Diagonal

We first consider a simple diagonal format for the covariance matrix. In order to fit a diagonal covariance approximation, we maintain a running average of the second uncentered moment for each weight, and then compute the covariance using the following standard identity at the end of training: $\overline{\theta^2} = \frac{1}{T}\sum_{i=1}^{T}\theta_i^2$, $\Sigma_{\text{diag}} = \text{diag}(\overline{\theta^2} - \theta_{\text{SWA}}^2)$; here the squares in $\theta_{\text{SWA}}^2$ and $\theta_i^2$ are applied elementwise. The resulting approximate posterior distribution is then $\mathcal{N}(\theta_{\text{SWA}}, \Sigma_{\text{Diag}})$. In our experiments, we term this method SWAG-Diagonal.

Constructing the SWAG-Diagonal posterior approximation requires storing two additional copies of DNN weights: $\theta_{\text{SWA}}$ and $\overline{\theta^2}$. Note that these models do not have to be stored on the GPU. The additional computational complexity of constructing SWAG-Diagonal compared to standard training is negligible, as it only requires updating the running averages of weights once per epoch.

### 3.3.3 SWAG: Low Rank plus Diagonal Covariance Structure

We now describe the full SWAG algorithm. While the diagonal covariance approximation is standard in Bayesian deep learning [Blundell et al. 2015; Kirkpatrick et al. 2017], it can be too restrictive. We extend the idea of diagonal covariance approximations to utilize a more flexible low-rank plus diagonal posterior approximation. SWAG approximates the sample covariance $\Sigma$ of the SGD iterates along with the mean $\theta_{\text{SWA}}$.[2]

Note that the sample covariance matrix of the SGD iterates can be written as the sum of outer products, $\Sigma = \frac{1}{T-1}\sum_{i=1}^{T}(\theta_i - \theta_{\text{SWA}})(\theta_i - \theta_{\text{SWA}})^\top$, and is of rank $T$. As we do not have access to the value of $\theta_{\text{SWA}}$ during training, we approximate the sample covariance with $\Sigma \approx \frac{1}{T-1}\sum_{i=1}^{T}(\theta_i - \bar{\theta}_i)(\theta_i - \bar{\theta}_i)^\top = \frac{1}{T-1}DD^\top$, where $D$ is the deviation matrix comprised of columns

[2]We note that stochastic gradient Monte Carlo methods [Chen et al. 2014; Welling and Teh 2011] also use the SGD trajectory to construct samples from the approximate posterior. However, these methods are principally different from SWAG in that they (1) require adding Gaussian noise to the gradients, (2) decay learning rate to zero and (3) do not construct a closed-form approximation to the posterior distribution, which for instance enables SWAG to draw new samples with minimal overhead. We include comparisons to SGLD [Welling and Teh 2011] in the Appendix.

$D_i = (\theta_i - \bar{\theta}_i)$, and $\bar{\theta}_i$ is the running estimate of the parameters' mean obtained from the first $i$ samples. To limit the rank of the estimated covariance matrix we only use the last $K$ of $D_i$ vectors corresponding to the last $K$ epochs of training. Here $K$ is the rank of the resulting approximation and is a hyperparameter of the method. We define $\widehat{D}$ to be the matrix with columns equal to $D_i$ for $i = T - K + 1, \dots, T$.

We then combine the resulting low-rank approximation $\Sigma_{\text{low-rank}} = \frac{1}{K-1} \cdot \widehat{D}\widehat{D}^\top$ with the diagonal approximation $\Sigma_{\text{diag}}$ of Section 3.3.2. The resulting approximate posterior distribution is a Gaussian with the SWA mean $\theta_{\text{SWA}}$ and summed covariance: $\mathcal{N}(\theta_{\text{SWA}}, \frac{1}{2} \cdot (\Sigma_{\text{diag}} + \Sigma_{\text{low-rank}}))$.[3] In our experiments, we term this method SWAG. Computing this approximate posterior distribution requires storing $K$ vectors $D_i$ of the same size as the model as well as the vectors $\theta_{\text{SWA}}$ and $\overline{\theta^2}$. These models do not have to be stored on a GPU.

To sample from SWAG we use the following identity

$$\widetilde{\theta} = \theta_{\text{SWA}} + \frac{1}{\sqrt{2}} \cdot \Sigma_{\text{diag}}^{\frac{1}{2}} z_1 + \frac{1}{\sqrt{2(K-1)}} \widehat{D} z_2, \quad \text{where } z_1 \sim \mathcal{N}(0, I_d),\ z_2 \sim \mathcal{N}(0, I_K). \tag{3.2}$$

Here $d$ is the number of parameters in the network. Note that $\Sigma_{\text{diag}}$ is diagonal, and the product $\Sigma_{\text{diag}}^{\frac{1}{2}} z_1$ can be computed in $\mathcal{O}(d)$ time. The product $\widehat{D} z_2$ can be computed in $\mathcal{O}(Kd)$ time.

Related methods for estimating the covariance of SGD iterates were considered in Mandt et al. [2017a] and Chen et al. [2016], but store full-rank covariance $\Sigma$ and thus scale quadratically in the number of parameters, which is prohibitively expensive for deep learning applications. We additionally note that using the deviation matrix for online covariance matrix estimation comes from viewing the online updates used in Dasgupta and Hsu [2007] in matrix fashion.

The full Bayesian model averaging procedure is given in Algorithm 2. As in Izmailov et al. [2018] (SWA) we update the batch normalization statistics after sampling weights for models that use batch normalization [Ioffe and Szegedy 2015]; we investigate the necessity of this update in

[3] We use one half as the scale here because both the diagonal and low rank terms include the variance of the weights. We tested several other scales in Appendix A.3.4.

**Algorithm 2** Bayesian Model Averaging with SWAG

```
$\theta_0$: pretrained weights; $\eta$: learning rate; $T$: number of steps; $c$: moment update frequency; $K$: maximum number of columns in deviation matrix; $S$: number of samples in Bayesian model averaging
Train SWAG
  $\overline{\theta} \leftarrow \theta_0$,  $\overline{\theta^2} \leftarrow \theta_0^2$                                {Initialize moments}
  for $i \leftarrow 1, 2, ..., T$ do
    $\theta_i \leftarrow \theta_{i-1} - \eta\nabla_\theta\mathcal{L}(\theta_{i-1})${Perform SGD update}
    if $\texttt{MOD}(i, c) = 0$ then
      $n \leftarrow i/c$                                                {Number of models}
      $\overline{\theta} \leftarrow \frac{n\overline{\theta} + \theta_i}{n+1}$,  $\overline{\theta^2} \leftarrow \frac{n\overline{\theta^2} + \theta_i^2}{n+1}$                        {Moments}
      if $\texttt{NUM\_COLS}(\widehat{D}) = K$ then
        $\texttt{REMOVE\_COL}(\widehat{D}[:, 1])$
      $\texttt{APPEND\_COL}(\widehat{D}, \theta_i - \overline{\theta})$                              {Store deviation}
  return $\theta_{\text{SWA}} = \overline{\theta}$,  $\Sigma_{\text{diag}} = \overline{\theta^2} - \overline{\theta}^2$,  $\widehat{D}$
Test  Bayesian Model Averaging
  for $i \leftarrow 1, 2, ..., S$ do
    Draw $\widetilde{\theta}_i \sim \mathcal{N}\left(\theta_{\text{SWA}}, \frac{1}{2}\Sigma_{\text{diag}} + \frac{\widehat{D}\widehat{D}^\top}{2(K-1)}\right)$ (3.2)
    Update batch norm statistics with new sample.
    $p(y^*|\text{Data}) \mathrel{+}= \frac{1}{S}p(y^*|\widetilde{\theta}_i)$
  return $p(y^*|\text{Data})$
```

Appendix A.3.5.3.

### 3.3.4 Bayesian Model Averaging with SWAG

Maximum a-posteriori (MAP) optimization is a procedure whereby one maximizes the (log) posterior with respect to parameters $\theta$: $\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}|\theta) + \log p(\theta)$. Here, the prior $p(\theta)$ is viewed as a regularizer in optimization. However, MAP is *not* Bayesian inference, since one only considers a single setting of the parameters $\hat{\theta}_{\text{MAP}} = \text{argmax}_\theta p(\theta|\mathcal{D})$ in making predictions, forming $p(y_*|\hat{\theta}_{\text{MAP}}, x_*)$, where $x_*$ and $y_*$ are test inputs and outputs.

A Bayesian procedure instead *marginalizes* the posterior distribution over $\theta$, in a Bayesian model average, for the unconditional predictive distribution: $p(y_*|\mathcal{D}, x_*) = \int p(y_*|\theta, x_*)p(\theta|\mathcal{D})d\theta$. In practice, this integral is computed through a Monte Carlo sampling procedure:
$p(y_*|\mathcal{D}, x_*) \approx \frac{1}{T}\sum_{t=1}^{T} p(y_*|\theta_t, x_*)\,, \quad \theta_t \sim p(\theta|\mathcal{D})$.

We emphasize that in this paper we are approximating *fully Bayesian inference*, rather than

MAP optimization. We develop a Gaussian approximation to the posterior from SGD iterates, $p(\theta|\mathcal{D}) \approx \mathcal{N}(\theta; \mu, \Sigma)$, and then sample from this posterior distribution to perform a Bayesian model average. In our procedure, optimization with different regularizers, to characterize the Gaussian posterior approximation, corresponds to approximate Bayesian inference with different priors $p(\theta)$.

Prior Choice. Typically, weight decay is used to regularize DNNs, corresponding to explicit L2 regularization when SGD without momentum is used to train the model. When SGD is used *with* momentum, as is typically the case, implicit regularization still occurs, producing a vague prior on the weights of the DNN in our procedure. This regularizer can be given an explicit Gaussian-like form (see Proposition 3 of Loshchilov and Hutter [2017a]), corresponding to a prior distribution on the weights.

Thus, SWAG is an approximate Bayesian inference algorithm in our experiments (see Section 3.5) and can be applied to most DNNs without any modifications of the training procedure (as long as SGD is used with weight decay or explicit L2 regularization). Alternative regularization techniques could also be used, producing different priors on the weights. It may also be possible to similarly utilize Adam and other stochastic first-order methods, which view as a promising direction for future work.

## 3.4 Does the SGD Trajectory Capture Loss Geometry?

To analyze the quality of the SWAG approximation, we study the posterior density along the directions corresponding to the eigenvectors of the SWAG covariance matrix for PreResNet-164 on CIFAR-100. In order to find these eigenvectors we use randomized SVD [Halko et al. 2011].[4] In the left panel of Figure 3.3 we visualize the $\ell_2$-regularized cross-entropy loss $L(\cdot)$ (equivalent to the joint density of the weights and the loss with a Gaussian prior) as a function of distance

[4]From `sklearn.decomposition.TruncatedSVD`.

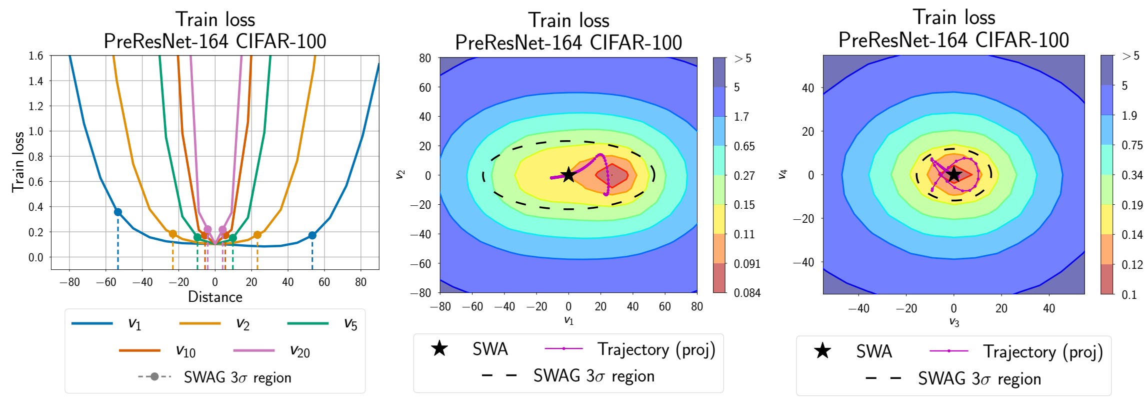


**Figure 3.3: Left:** Posterior joint density cross-sections along the rays corresponding to different eigenvectors of SWAG covariance matrix. **Middle:** Posterior joint density surface in the plane spanned by eigenvectors of SWAG covariance matrix corresponding to the first and second largest eigenvalues and (**Right:**) the third and fourth largest eigenvalues. All plots are produced using PreResNet-164 on CIFAR-100. The SWAG distribution projected onto these directions fits the geometry of the posterior density remarkably well.

$t$ from the SWA solution $\theta_{\text{SWA}}$ along the $i$-th eigenvector $v_i$ of the SWAG covariance: $\phi(t) = L(\theta_{\text{SWA}} + t \cdot \frac{v_i}{\|v_i\|})$. Figure 3.3 (left) shows a clear correlation between the variance of the SWAG approximation and the width of the posterior along the directions $v_i$. The SGD iterates indeed contain useful information about the shape of the posterior distribution, and SWAG is able to capture this information. We repeated the same experiment for SWAG-Diagonal, finding that there was almost no variance in these eigen-directions. Next, in Figure 3.3 (middle) we plot the posterior density surface in the 2-dimensional plane in the weight space spanning the two top eigenvectors $v_1$ and $v_2$ of the SWAG covariance: $\psi(t_1, t_2) = L(\theta_{\text{SWA}} + t_1 \cdot \frac{v_1}{\|v_1\|} + t_2 \cdot \frac{v_2}{\|v_2\|})$. Again, SWAG is able to capture the geometry of the posterior. The contours of constant posterior density appear remarkably well aligned with the eigenvalues of the SWAG covariance. We also present the analogous plot for the third and fourth top eigenvectors in Figure 3.3 (right). In Appendix A.3.3, we additionally present similar results for PreResNet-164 on CIFAR-10 and VGG-16 on CIFAR-100.

As we can see, SWAG is able to capture the geometry of the posterior in the subspace spanned by SGD iterates. However, the dimensionality of this subspace is very low compared to the dimensionality of the weight space, and we can not guarantee that SWAG variance estimates are adequate along all directions in weight space. In particular, we would expect SWAG to underestimate the variances along random directions, as the SGD trajectory is in a low-dimensional subspace of the weight space, and a random vector has a close-to-zero projection on this subspace with high probability. In Appendix A.3.1 we visualize the trajectory of SGD applied to a quadratic function, and further discuss the relation between the geometry of objective and SGD trajectory. In Appendices A.3.1 and A.3.2, we also empirically test the assumptions behind theory relating the SGD stationary distribution to the true posterior for neural networks.

## 3.5 SWAG Experiments

We conduct a thorough empirical evaluation of SWAG, comparing to a range of high performing baselines, including MC dropout [Gal and Ghahramani 2016], temperature scaling [Guo et al. 2017], SGLD [Welling and Teh 2011], Laplace approximations [Ritter et al. 2018b], deep ensembles [Lakshminarayanan et al. 2017], and ensembles of SGD iterates that were used to construct the SWAG approximation. In Section 3.5.1 we evaluate SWAG predictions and uncertainty estimates on image classification tasks. We also evaluate SWAG for transfer learning and out-of-domain data detection. We investigate the effect of hyperparameter choices and practical limitations in SWAG, such as the effect of learning rate on the scale of uncertainty, in Appendix A.3.4.

### 3.5.1 Calibration and Uncertainty Estimation on Image Classification Tasks

In this section we evaluate the quality of uncertainty estimates as well as predictive accuracy for SWAG and SWAG-Diagonal on CIFAR-10, CIFAR-100 and ImageNet ILSVRC-2012 [Russakovsky et al. 2012].

For all methods we analyze test negative log-likelihood, which reflects both the accuracy

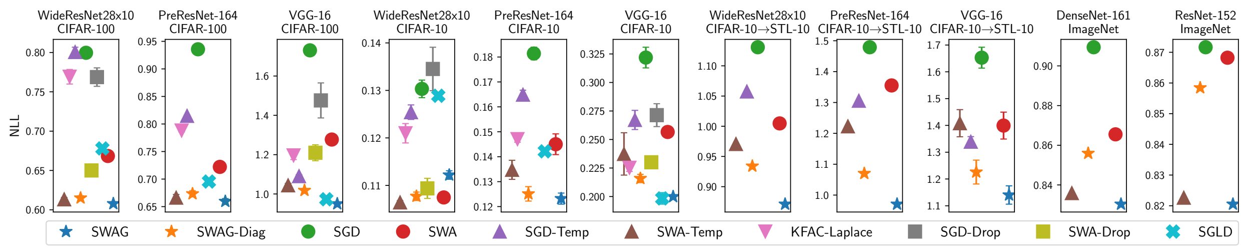


**Figure 3.4:** Negative log likelihoods for SWAG and baselines. Mean and standard deviation (shown with error-bars) over 3 runs are reported for each experiment on CIFAR datasets. SWAG (blue star) consistently outperforms alternatives, with lower negative log likelihood, with the largest improvements on transfer learning. Temperature scaling applied on top of SWA (SWA-Temp) often performs close to as well on the non-transfer learning tasks, but requires a validation set.

and the quality of predictive uncertainty. Following Guo et al. [2017] we also consider a variant of *reliability diagrams* to evaluate the calibration of uncertainty estimates (see Figure 3.5) and to show the difference between a method's confidence in its predictions and its accuracy. To produce this plot for a given method we split the test data into 20 bins uniformly based on the confidence of a method (maximum predicted probability). We then evaluate the accuracy and mean confidence of the method on the images from each bin, and plot the difference between confidence and accuracy. For a well-calibrated model, this difference should be close to zero for each bin. We found that this procedure gives a more effective visualization of the actual confidence distribution of DNN predictions than the standard reliability diagrams used in Guo et al. [2017] and Niculescu-Mizil and Caruana [2005].

We provide tables containing the test accuracy, negative log likelihood and expected calibration error for all methods and datasets in Appendix A.3.6.3.

CIFAR DATASETS. On CIFAR datasets we run experiments with VGG-16, PreResNet-164 and WideResNet-28x10 networks. In order to compare SWAG with existing alternatives we report the results for standard SGD and SWA [Izmailov et al. 2018] solutions (single models), MC-Dropout [Gal and Ghahramani 2016], temperature scaling [Guo et al. 2017] applied to SWA and SGD solutions, SGLD [Welling and Teh 2011], and K-FAC Laplace [Ritter et al. 2018b] methods. For all

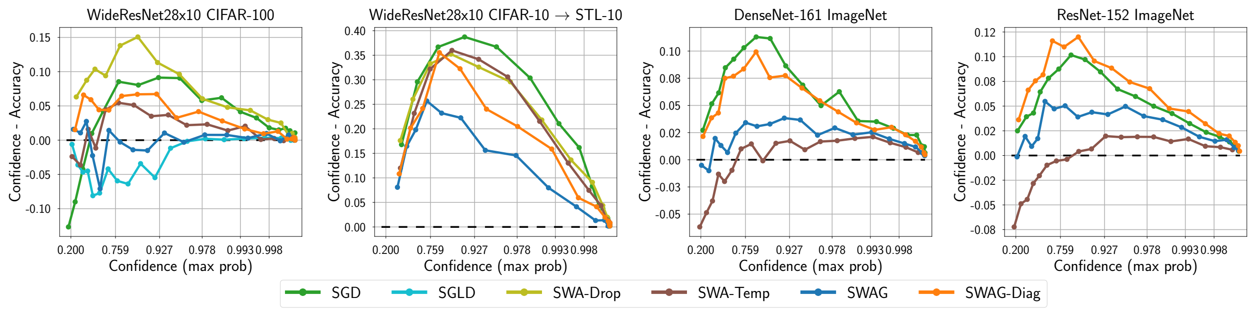


**Figure 3.5:** Reliability diagrams for WideResNet28x10 on CIFAR-100 and transfer task; ResNet-152 and DenseNet-161 on ImageNet. Confidence is the value of the max softmax output. A perfectly calibrated network has no difference between confidence and accuracy, represented by a dashed black line. Points below this line correspond to under-confident predictions, whereas points above the line are overconfident predictions. SWAG is able to substantially improve calibration over standard training (SGD), as well as SWA. Additionally, SWAG significantly outperforms temperature scaling for transfer learning (CIFAR-10 to STL), where the target data are not from the same distribution as the training data.

the methods we use our implementations in PyTorch (see Appendix A.3.9). We train all networks for 300 epochs, starting to collect models for SWA and SWAG approximations once per epoch after epoch 160. For SWAG, K-FAC Laplace, and Dropout we use 30 samples at test time.

ImageNet. On ImageNet we report our results for SWAG, SWAG-Diagonal, SWA and SGD. We run experiments with DenseNet-161 [Huang et al. 2017b] and Resnet-152 [He et al. 2016]. For each model we start from a pre-trained model available in the `torchvision` package, and run SGD with a constant learning rate for 10 epochs. We collect models for the SWAG versions and SWA 4 times per epoch. For SWAG we use 30 samples from the posterior over network weights at test-time, and use randomly sampled 10% of the training data to update batch-normalization statistics for each of the samples. For SGD with temperature scaling, we use the results reported in Guo et al. [2017].

Transfer from CIFAR-10 to STL-10. We use the models trained on CIFAR-10 and evaluate them on STL-10 [Coates et al. 2011]. STL-10 has a similar set of classes as CIFAR-10, but the image distribution is different, so adapting the model from CIFAR-10 to STL-10 is a commonly used transfer learning benchmark. We provide further details on the architectures and hyperparameters in Appendix A.3.9.

Results. We visualize the negative log-likelihood for all methods and datasets in Figure 3.4. On all considered tasks SWAG and SWAG diagonal perform comparably or better than all the considered alternatives, SWAG being best overall. We note that the combination of SWA and temperature scaling presents a competitive baseline. However, unlike SWAG it requires using a validation set to tune the temperature; further, temperature scaling is not effective when the test data distribution differs from train, as we observe in experiments on transfer learning from CIFAR-10 to STL-10.

Next, we analyze the calibration of uncertainty estimates provided by different methods. In Figure 3.5 we present reliability plots for WideResNet on CIFAR-100, DenseNet-161 and ResNet-152 on ImageNet. The reliability diagrams for all other datasets and architectures are presented in the Appendix A.3.6.1. As we can see, SWAG and SWAG-Diagonal both achieve good calibration across the board. The low-rank plus diagonal version of SWAG is generally better calibrated than SWAG-Diagonal. We also present the expected calibration error for each of the methods, architectures and datasets in Tables A.5, A.6. Finally, in Tables A.11, A.12 we present the predictive accuracy for all of the methods, where SWAG is comparable with SWA and generally outperforms the other approaches.

## 3.5.2 Comparison to ensembling SGD solutions

We evaluated ensembles of independently trained SGD solutions (Deep Ensembles, [Lakshminarayanan et al. 2017]) on PreResNet-164 on CIFAR-100. We found that an ensemble of 3 SGD solutions has high accuracy (82.1%), but only achieves NLL 0.6922, which is *worse than a single SWAG solution* (0.6595 NLL). While the accuracy of this ensemble is high, SWAG solutions are much better calibrated. An ensemble of 5 SGD solutions achieves NLL 0.6478, which is *competitive* *with a single SWAG solution, that requires* 5× *less computation to train.* Moreover, we can similarly ensemble independently trained SWAG models; an ensemble of 3 SWAG models achieves NLL of 0.6178.

We also evaluated ensembles of SGD iterates that were used to construct the SWAG approximation (SGD-Ens) for all of our CIFAR models. SWAG has higher NLL than SGD-Ens on VGG-16, but much lower NLL on the larger PreResNet-164 and WideResNet28x10; the results for accuracy and ECE are analogous.

### 3.5.3 Out-of-Domain Image Detection

To evaluate SWAG on out-of-domain data detection we train a WideResNet as described in section 3.5.1 on the data from five classes of the CIFAR-10 dataset, and then analyze predictions of SWAG variants along with the baselines on the full test set. We expect the outputted class probabilities on objects that belong to classes that were not present in the training data to have high-entropy reflecting the model's high uncertainty in its predictions, and considerably lower entropy on the images that are similar to those on which the network was trained. We plot the histograms of predictive entropies on the in-domain and out-of-domain in Figure A.14 for a qualitative comparison and report the symmetrized KL divergence between the binned in and out of sample distributions in Table A.4, finding that SWAG and Dropout perform best on this measure. Additional details are in Appendix A.3.6.2.

### 3.5.4 Language Modeling with LSTMs

We next apply SWAG to an LSTM network on language modeling tasks on Penn Treebank and WikiText-2 datasets. In Appendix A.3.7 we demonstrate that SWAG easily outperforms both SWA and NT-ASGD [Merity et al. 2017], a strong baseline for LSTM training, in terms of test and validation perplexities.

We compare SWAG to SWA and the NT-ASGD method [Merity et al. 2017], which is a strong baseline for training LSTM models. The main difference between SWA and NT-ASGD, which is also based on weight averaging, is that NT-ASGD starts weight averaging much earlier than SWA: NT-ASGD switches to ASGD (averaged SGD) typically around epoch 100 while with SWA we start averaging after pre-training for 500 epochs. We report test and validation perplexities for different methods and datasets in Table 3.1.

As we can see, SWA substantially improves perplexities on both datasets over NT-ASGD. Further, we observe that SWAG is able to substantially improve test perplexities over the SWA solution.

**Table 3.1:** Validation and Test perplexities for NT-ASGD, SWA and SWAG on Penn Treebank and WikiText-2 datasets.

| Method | PTB val | PTB test | WikiText-2 val | WikiText-2 test |
|---|---|---|---|---|
| NT-ASGD | 61.2 | 58.8 | 68.7 | 65.6 |
| SWA | 59.1 | 56.7 | 68.1 | 65.0 |
| SWAG | **58.6** | **56.26** | **67.2** | **64.1** |

### 3.5.5 Regression

Finally, while the empirical focus of our paper is classification calibration, we also compare to additional approximate BNN inference methods which perform well on smaller architectures, including deterministic variational inference (DVI) [Wu et al. 2018], single-layer deep GPs (DGP) with expectation propagation [Bui et al. 2016], SGLD [Welling and Teh 2011], and re-parameterization VI [Kingma and Welling 2013] on a set of UCI regression tasks. We report test log-likelihoods, RMSEs and test calibration results in Appendix Tables A.14 and A.15 where it is possible to see that SWAG is competitive with these methods. Additional details are in Appendix A.3.8.

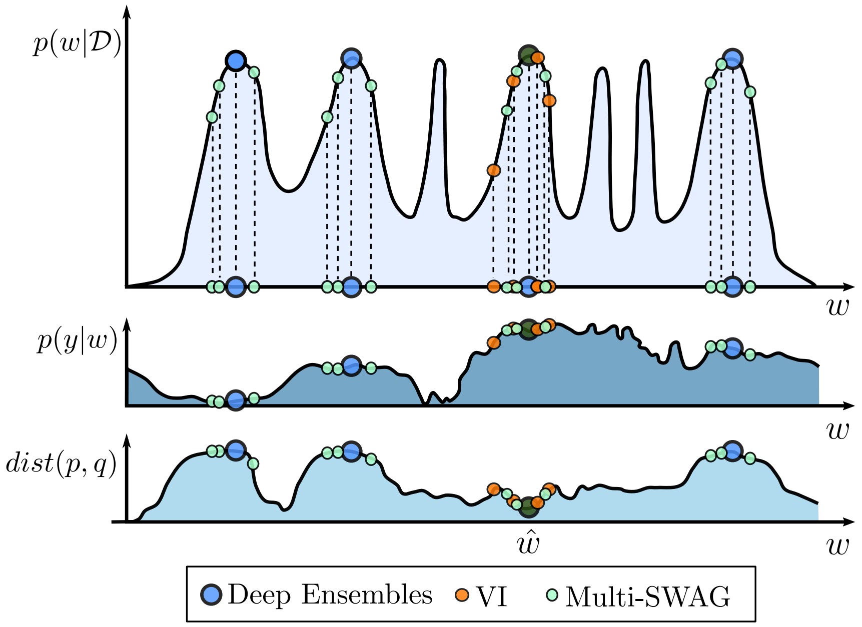
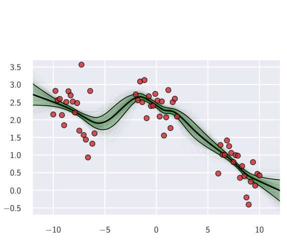
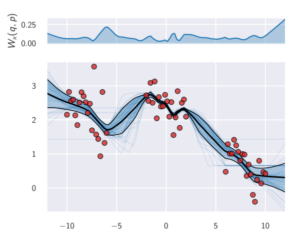


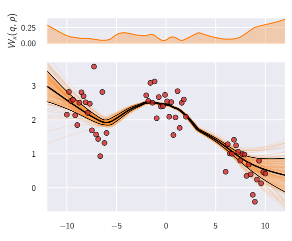
**Figure 3.6: Approximating the BMA.** $p(y|x, \mathcal{D}) = \int p(y|x, w)p(w|\mathcal{D})dw$. **Top:** $p(w|\mathcal{D})$, with representations from VI (orange) deep ensembles (blue), MultiSWAG (red). **Middle:** $p(y|x, w)$ as a function of $w$ for a test input $x$. This function does not vary much within modes, but changes significantly between modes. **Bottom:** Distance between the true predictive distribution and the approximation, as a function of representing a posterior at an additional point $w$, assuming we have sampled the mode in dark green. There is more to be gained by exploring new basins, than continuing to explore the same basin.

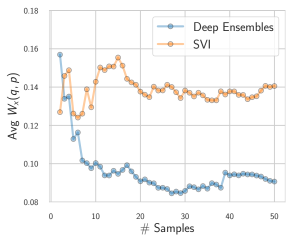
## 3.6 An Empirical Study of Marginalization

We have shown that deep ensembles can be interpreted as an approximate approach to Bayesian marginalization, which selects for functional diversity by representing multiple basins of attraction in the posterior. Most Bayesian deep learning methods instead focus on faithfully approximating a posterior within a single basin of attraction. We propose a new method, MultiSWAG, which combines these two types of approaches. MultiSWAG combines multiple independently trained SWAG approximations [Maddox et al. 2019], to create a mixture of Gaussians approximation to the posterior, with each Gaussian centred on a different basin of attraction. We note that MultiSWAG does not require *any* additional training time over standard deep ensembles.


(a)


(b)


(c)



(d)

**Figure 3.7: Approximating the true predictive distribution. (a)**: A close approximation of the true predictive distribution obtained by combining 200 HMC chains. **(b)**: Deep ensembles predictive distribution using 50 independently trained networks. **(c)**: Predictive distribution for factorized variational inference (VI). **(d)**: Convergence of the predictive distributions for deep ensembles and variational inference as a function of the number of samples; we measure the average Wasserstein distance between the marginals in the range of input positions. The multi-basin deep ensembles approach provides a more faithful approximation of the Bayesian predictive distribution than the conventional single-basin VI approach, which is overconfident between data clusters. The top panels show the Wasserstein distance between the true predictive distribution and the deep ensemble and VI approximations, as a function of inputs $x$.

We illustrate the conceptual difference between deep ensembles, a standard variational single basin approach, and MultiSWAG, in Figure 3.6. In the top panel, we have a conceptualization of a multimodal posterior. Variational Inference (VI), which is often considered as a representative Bayesian method [e.g. Ovadia et al. 2019], approximates the posterior with multiple samples within a single basin. But we see in the middle panel that the conditional predictive distribution $p(y|x, w)$ does not vary significantly within the basin, and thus each additional sample contributes minimally to computing the marginal predictive distribution $p(y|x, \mathcal{D})$. On the other hand, $p(y|x, w)$ varies significantly between basins, and thus each point mass for deep ensembles contributes significantly to the marginal predictive distribution. By sampling within the basins, MultiSWAG provides additional contributions to the predictive distribution. In the bottom panel, we have the gain in approximating the predictive distribution when adding a point mass to the representation of the posterior, as a function of its location, assuming we have already sampled the mode in dark green. Including samples from different modes provides significant gain over continuing to sample from the same mode, and including weights in wide basins provide relatively more gain than the narrow ones.

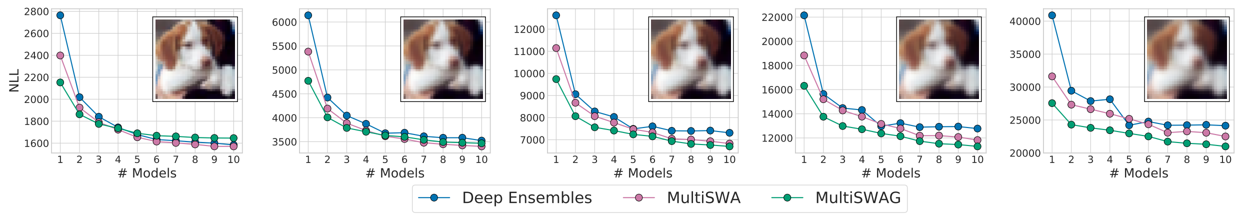


**Figure 3.8:** Negative log likelihood for Deep Ensembles, MultiSWAG and MultiSWA using a PreResNet-20 on CIFAR-10 with varying intensity of the *Gaussian blur* corruption. The image in each plot shows the intensity of corruption. For all levels of intensity, MultiSWAG and MultiSWA outperform Deep Ensembles for a small number of independent models. For high levels of corruption MultiSWAG significantly outperforms other methods even for many independent models. We present results for other corruptions in the Appendix.

In Figure 3.7 we evaluate single basin and multi-basin approaches in a case where we can near-exactly compute the predictive distribution. We provide details for generating the data and training the models in Appendix A.4.4.1. We see that the predictive distribution given by deep ensembles is qualitatively closer to the true distribution, compared to the single basin variational method: between data clusters, the deep ensemble approach provides a similar representation of epistemic uncertainty, whereas the variational method is extremely overconfident in these regions. Moreover, we see that the Wasserstein distance between the true predictive distribution and these two approximations quickly shrinks with number of samples for deep ensembles, but is roughly independent of number of samples for the variational approach. Thus the deep ensemble is providing a better approximation of the Bayesian model average in Eq. (3.1) than the single basin variational approach, which has traditionally been labelled as the Bayesian alternative.

Next, we evaluate MultiSWAG under distribution shift on the CIFAR-10 dataset [Krizhevsky et al. 2014], replicating the setup in Ovadia et al. [2019]. We consider 16 data corruptions, each at 5 different levels of severity, introduced by Hendrycks and Dietterich [2019]. For each corruption, we evaluate the performance of deep ensembles and MultiSWAG varying the training budget. For deep ensembles we show performance as a function of the number of independently trained models in the ensemble. For MultiSWAG we show performance as a function of the number of independent SWAG approximations that we construct; we then sample 20 models from each of these approximations to construct the final ensemble.

While the training time for MultiSWAG is the same as for deep ensembles, at test time MultiSWAG is more expensive, as the corresponding ensemble consists of a larger number of models. To account for situations when test time is constrained, we also propose MultiSWA, a method that ensembles independently trained SWA solutions [Izmailov et al. 2018], discussed in Section 2.5. SWA solutions are the means of the corresponding Gaussian SWAG approximations. In Section 2.5.5, we argue that SWA solutions approximate the local ensembles represented by SWAG with a single model.

In Figure 3.8 we show the negative log-likelihood as a function of the number of independently trained models for a Preactivation ResNet-20 on CIFAR-10 corrupted with Gaussian blur with varying levels of intensity (increasing from left to right) in Figure 3.8. MultiSWAG outperforms deep ensembles significantly on highly corrupted data. For lower levels of corruption, MultiSWAG works particularly well when only a small number of independently trained models are available. We note that MultiSWA also outperforms deep ensembles, and has the same computational requirements at training and test time as deep ensembles. We present results for other types of corruption in Appendix Figures A.18, A.19, A.20, A.21, showing similar trends. In general, there is an extensive evaluation of MultiSWAG in the Appendix.

Our perspective of generalization is deeply connected with Bayesian marginalization. In order to best realize the benefits of marginalization in deep learning, we need to consider as many hypotheses as possible through multimodal posterior approximations, such as MultiSWAG. In Section 3.9 we return to MultiSWAG, showing how it can entirely alleviate prominent double descent behaviour, and lead to striking improvements in generalization over SGD and single basin marginalization, for both accuracy and NLL.

## 3.7 Neural Network Priors

A prior over parameters $p(w)$ combines with the functional form of a model $f(x;w)$ to induce a distribution over functions $p(f(x;w))$. It is this distribution over functions that controls the generalization properties of the model; the prior over parameters, in isolation, has no meaning. Neural networks are imbued with structural properties that provide good inductive biases, such as translation equivariance, hierarchical representations, and sparsity. In the sense of Figure 3.2, the prior will have large support, due to the flexibility of neural networks, but its inductive biases provide the most mass to datasets which are representative of problem settings where neural networks are often applied. In this section, we study the properties of the induced distribution over functions. We directly continue the discussion of priors in Section 3.8, with a focus on examining the noisy CIFAR results in Zhang et al. [2017a], from a probabilistic perspective of generalization. These sections are best read together.

We also provide several additional experiments in the Appendix. In Section A.4.5, we present analytic results on the dependence of the prior distribution in function space on the variance of the prior over parameters, considering also layer-wise parameter priors with ReLU activations. As part of a discussion on tempering, in Section 3.10.4 we study the effect of $\alpha$ in $p(w) = \mathcal{N}(0, \alpha^2 I)$ on prior class probabilities for individual sample functions $p(f(x;w))$, the predictive distribution, and posterior samples as we observe varying amounts of data. In Section A.4.6, we further study the correlation structure over images induced by neural network priors, subject to perturbations of the images. In Section A.4.4.3 we provide additional experimental details.

### 3.7.1 Deep Image Prior and Random Network Features

Two recent results provide strong evidence that vague Gaussian priors over parameters, when combined with a neural network architecture, induce a distribution over functions with useful inductive biases. In the *deep image prior*, Ulyanov et al. [2018] show that *randomly initialized* con-

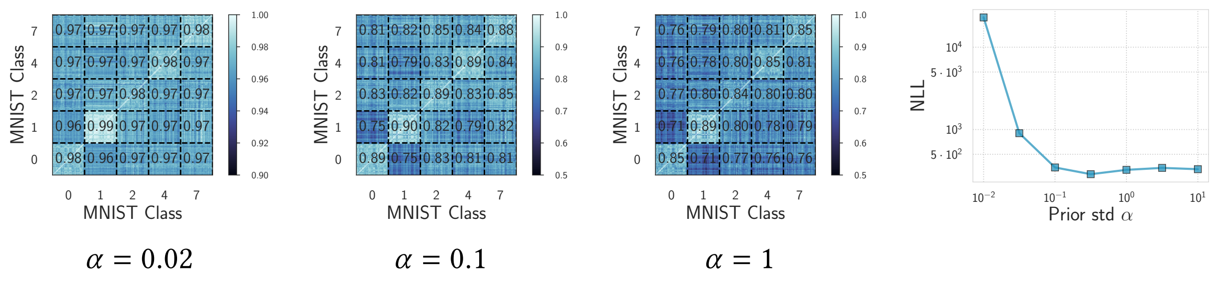


**Figure 3.9: Induced prior correlation function.** Average pairwise prior correlations for pairs of objects in classes $\{0, 1, 2, 4, 7\}$ of MNIST induced by LeNet-5 for $p(f(x; w))$ when $p(w) = \mathcal{N}(0, \alpha^2 I)$. Images in the same class have higher prior correlations than images from different classes, suggesting that $p(f(x; w))$ has desirable inductive biases. The correlations slightly decrease with increases in $\alpha$. **(d)**: NLL of an ensemble of 20 SWAG samples on MNIST as a function of $\alpha$ using a LeNet-5.

volutional neural networks *without training* provide excellent performance for image denoising, super-resolution, and inpainting. This result demonstrates the ability for a sample function from a random prior over neural networks $p(f(x; w))$ to capture low-level image statistics, before any training. Similarly, Zhang et al. [2017a] shows that pre-processing CIFAR-10 with a *randomly initialized untrained* convolutional neural network dramatically improves the test performance of a simple Gaussian kernel on pixels from 54% accuracy to 71%. Adding $\ell_2$ regularization only improves the accuracy by an additional 2%. These results again indicate that *broad* Gaussian priors over parameters induce reasonable priors over networks, with a minor additional gain from decreasing the variance of the prior in parameter space, which corresponds to $\ell_2$ regularization.

### 3.7.2 Prior Class Correlations

In Figure 3.9 we study the prior correlations in the outputs of the LeNet-5 convolutional network [LeCun et al. 1998] on objects of different MNIST classes. We sample networks with weights $p(w) = \mathcal{N}(0, \alpha^2 I)$, and compute the values of logits corresponding to the first class for all pairs of images and compute correlations of these logits. For all levels of $\alpha$ the correlations between objects corresponding to the same class are consistently higher than the correlation between objects of different classes, showing that the network induces a reasonable prior similarity metric over these images. Additionally, we observe that the prior correlations somewhat decrease as we increase $\alpha$, showing that bounding the norm of the weights has some minor utility, in accordance with Section 3.7.1. Similarly, in panel (d) we see that the NLL significantly decreases as $\alpha$ increases in $[0, 0.5]$, and then slightly increases, but is relatively constant thereafter.

In the Appendix A.4.5, we further describe analytic results and illustrate the effect of $\alpha$ on sample functions.

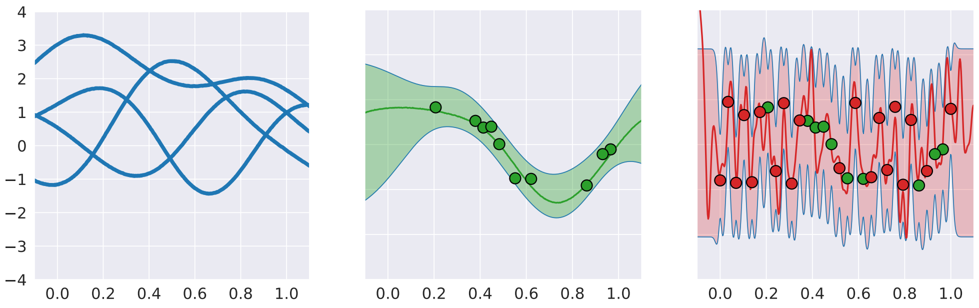
### 3.7.3 Effect of Prior Variance on CIFAR-10

We further study the effect of the parameter prior standard deviation $\alpha$, measuring performance of approximate Bayesian inference for CIFAR-10 with a Preactivation ResNet-20 [He et al. 2016] and VGG-16 [Simonyan and Zisserman 2014]. For each of these architectures we run SWAG [Maddox et al. 2019] with fixed hyper-parameters and varying $\alpha$. We report the results in Figure A.16(d), (h). For both architectures, the performance is near-optimal in the range $\alpha \in [10^{-2}, 10^{-1}]$. Smaller $\alpha$ constrains the weights too much. Performance is reasonable and becomes mostly insensitive to $\alpha$ as it continues to increase, due to the inductive biases of the functional form of the neural network.

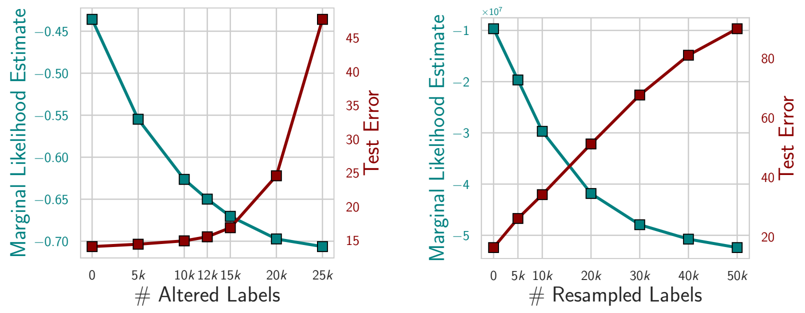
## 3.8 Rethinking Generalization

Zhang et al. [2017a] demonstrated that deep neural networks have sufficient capacity to fit randomized labels on popular image classification tasks, and suggest this result requires re-thinking generalization to understand deep learning.

We argue, however, that this behaviour is not puzzling from a probabilistic perspective, is not unique to neural networks, and cannot be used as evidence against Bayesian neural networks (BNNs) with vague parameter priors. Fundamentally, the resolution is the view presented in Section 3.1: from a probabilistic perspective, generalization is at least a *two-dimensional* con-



(a) Prior Draws (b) True Labels (c) Corrupted Labels



(d) Gaussian Process (e) PreResNet-20

**Figure 3.10: Rethinking generalization. (a)**: Sample functions from a Gaussian process prior. **(b)**: GP fit (with 95% credible region) to structured data generated as $y_{\text{green}}(x) = \sin(x \cdot 2\pi) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, 0.2^2)$. **(c)**: GP fit, with no training error, after a significant addition of corrupted data in red, drawn from Uniform$[0.5, 1]$. **(d)**: Variational GP marginal likelihood with RBF kernel for two classes of CIFAR-10. **(e)**: Laplace BNN marginal likelihood for a PreResNet-20 on CIFAR-10 with different fractions of random labels. The marginal likelihood for both the GP and BNN decreases as we increase the level of corruption in the labels, suggesting reasonable inductive biases in the prior over functions. Moreover, both the GP and BNN have 100% training accuracy on images with fully corrupted labels.

cept, related to support (flexibility), which should be as large as possible, supporting even noisy solutions, and inductive biases that represent relative prior probabilities of solutions.

Indeed, we demonstrate that the behaviour in Zhang et al. [2017a] that was treated as mysterious and specific to neural networks can be exactly reproduced by Gaussian processes (GPs). Gaussian processes are an ideal choice for this experiment, because they are popular Bayesian non-parametric models, and they assign a prior directly in function space. Moreover, GPs have remarkable flexibility, providing universal approximation with popular covariance functions such as the RBF kernel. Yet the functions that are a priori *likely* under a GP with an RBF kernel are relatively simple. We describe GPs further in Appendix A.4.2, and Rasmussen and Williams [2006] provides an extensive introduction.

We start with a simple example to illustrate the ability for a GP with an RBF kernel to easily fit a corrupted dataset, yet generalize well on a non-corrupted dataset, in Figure 3.10. In Fig 3.10(a), we have sample functions from a GP prior over functions $p(f(x))$, showing that likely functions under the prior are smooth and well-behaved. In Fig 3.10(b) we see the GP is able to reasonably fit data from a structured function. And in Fig 3.10(c) the GP is also able to fit highly corrupted data, with essentially no structure; although these data are not a likely draw from the prior, the GP has support for a wide range of solutions, including noise.

We next show that GPs can replicate the generalization behaviour described in Zhang et al. [2017a] (experimental details in the Appendix). When applied to CIFAR-10 images with random labels, *Gaussian processes achieve 100% train accuracy*, and 10.4% test accuracy (at the level of random guessing). However, the same model trained on the true labels achieves a training accuracy of 72.8% and a test accuracy of 54.3%. Thus, the generalization behaviour described in Zhang et al. [2017a] is not unique to neural networks, and can be described by separately understanding the support and the inductive biases of a model.

Indeed, although Gaussian processes support CIFAR-10 images with random labels, they are not likely under the GP prior. In Fig 3.10(d), we compute the approximate GP marginal likelihood on a binary CIFAR-10 classification problem, with labels of varying levels of corruption. We see as the noise in the data increases, the approximate marginal likelihood, and thus the prior support for these data, decreases. In Fig 3.10(e), we see a similar trend for a Bayesian neural network. Again, as the fraction of corrupted labels increases, the approximate marginal likelihood decreases, showing that the prior over functions given by the Bayesian neural network has less support for these noisy datasets. We provide further experimental details in the Appendix.

Dziugaite and Roy [2017] and Smith and Le [2018] provide complementary perspectives on Zhang et al. [2017a], for MNIST; Dziugaite and Roy [2017] show non-vacuous PAC-Bayes bounds for the noise-free binary MNIST but not noisy MNIST, and Smith and Le [2018] show that logistic regression can fit noisy labels on subsampled MNIST, interpreting the results from an Occam factor perspective.

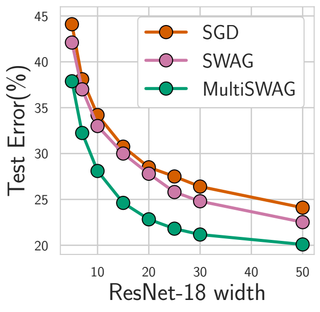
## 3.9 Double Descent

*Double descent* [e.g., Belkin et al. 2019] describes generalization error that decreases, increases, and then again decreases, with increases in model flexibility. The first decrease and then increase is referred to as the *classical regime*: models with increasing flexibility are increasingly able to capture structure and perform better, until they begin to overfit. The next regime is referred to as the *modern interpolating regime*. The existence of the interpolation regime has been presented as mysterious generalization behaviour in deep learning.

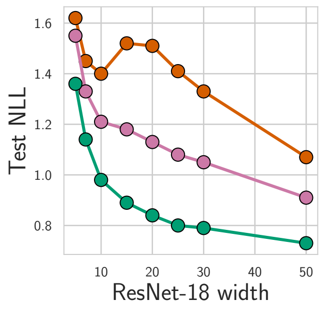
However, our perspective of generalization suggests that performance should monotonically improve as we increase model flexibility when we use Bayesian model averaging with a reasonable prior. Indeed, in the opening example of Figure 3.1, we would in principle want to use the most flexible possible model. Our results in Section 3.7 show that standard BNN priors induce structured and useful priors in the function space, so we should not expect double descent in Bayesian deep learning models that perform reasonable marginalization.

To test this hypothesis, we evaluate MultiSWAG, SWAG and standard SGD with ResNet-18 models of varying width, following Nakkiran et al. [2019], measuring both error and negative log likelihood (NLL). For the details, see Appendix A.4.4. We present the results in Figure 3.11 and Appendix Figure A.17.

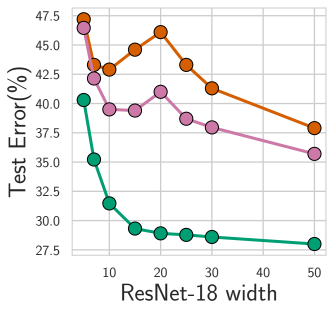
First, we observe that models trained with SGD indeed suffer from double descent, especially when the train labels are partially corrupted (see panels (c), (d) in Figure 3.11). We also see that SWAG, a unimodal posterior approximation, reduces the extent of double descent. Moreover, MultiSWAG, which performs a more exhaustive *multimodal* Bayesian model average *completely*



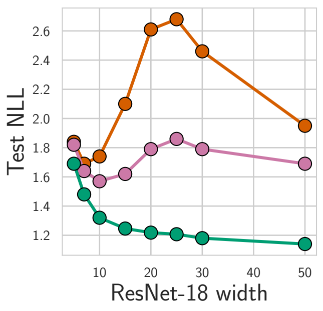
(a) True Labels (Err)



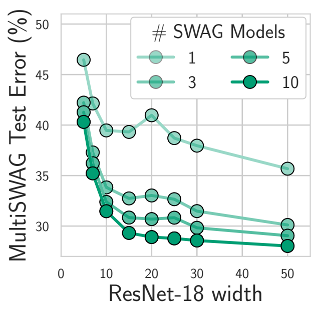
(b) True Labels (NLL)



(c) Corrupted (Err)



(d) Corrupted (NLL)



(e) Corrupted (# Models)

**Figure 3.11: Bayesian model averaging alleviates double descent. (a)**: Test error and **(b)**: NLL loss for ResNet-18 with varying width on CIFAR-100 for SGD, SWAG and MultiSWAG. **(c)**: Test error and **(d)**: NLL loss when 20% of the labels are randomly reshuffled. SWAG reduces double descent, and MultiSWAG, which marginalizes over multiple modes, entirely alleviates double descent both on the original labels and under label noise, both in accuracy and NLL. **(e)**: Test errors for MultiSWAG with varying number of independent SWAG models; error monotonically decreases with increased number of independent models, alleviating double descent. We also note that MultiSWAG provides significant improvements in accuracy and NLL over SGD and SWAG models. See Appendix Figure A.17 for additional results.

*mitigates double descent*: the performance of MultiSWAG solutions increases monotonically with the size of the model, showing no double descent even under significant label corruption, for both accuracy and NLL. We also found that deep ensembles follow a similar pattern to MultiSWAG in Figure 3.11(c), also mitigating double descent, with slightly worse accuracy (about 1-2%). This result is in line with our perspective of Section 3.2.3 of deep ensembles providing a better approximation to the Bayesian predictive distribution than conventional single-basin Bayesian marginalization procedures.

Our results highlight the importance of marginalization over multiple modes of the posterior:

under 20% label corruption SWAG clearly suffers from double descent while MultiSWAG does not. In Figure 3.11(e) we show how the double descent is alleviated with increased number of independent modes marginalized in MultiSWAG.

These results also clearly show that MultiSWAG provides significant improvements in *accuracy* over both SGD and SWAG models, in addition to NLL, an often overlooked advantage of Bayesian model averaging we discuss in Section 3.2.1.

Recently, Nakkiran et al. [2020] show that carefully tuned $l_2$ regularization can help mitigate double descent. Alternatively, we show that Bayesian model averaging, particularly based on multimodal marginalization, can mitigate prominent double descent behaviour. The perspective in Sections 3.1 and 3.2 predicts this result: models with reasonable priors and effective Bayesian model averaging should monotonically improve with increases in flexibility.

## 3.10 Temperature Scaling

The standard Bayesian posterior distribution is given by

$$p(w|\mathcal{D}) = \frac{1}{Z} p(\mathcal{D}|w) p(w), \tag{3.3}$$

where $p(\mathcal{D}|w)$ is a likelihood, $p(w)$ is a prior, and $Z$ is a normalizing constant.

In Bayesian deep learning it is typical to consider the *tempered* posterior

$$p_T(w|\mathcal{D}) = \frac{1}{Z(T)} p(\mathcal{D}|w)^{1/T} p(w), \tag{3.4}$$

where $T$ is a *temperature* parameter, and $Z(T)$ is the normalizing constant corresponding to temperature $T$. The temperature parameter controls how the prior and likelihood interact in the posterior:

- $T < 1$ corresponds to *cold posteriors*, where the posterior distribution is more concentrated around solutions with high likelihood.

- $T = 1$ corresponds to the standard Bayesian posterior distribution.
- $T > 1$ corresponds to *warm posteriors*, where the prior effect is stronger and the posterior collapse is slower.

Tempering posteriors is a well-known practice in statistics, where it goes by the names *Safe Bayes*, *generalized Bayesian inference*, and *fractional Bayesian inference* [e.g., de Heide et al. 2019; Grünwald et al. 2017; Barron and Cover 1991; Walker and Hjort 2001; Zhang 2006; Bissiri et al. 2016; Grünwald 2012]. Safe Bayes has been shown to be natural from a variety of perspectives, including from prequential, learning theory, and minimum description length frameworks [e.g., Grünwald et al. 2017].

Concurrently with our work, Wenzel et al. [2020] noticed that successful Bayesian deep learning methods tend to use cold posteriors. They provide an empirical study that shows that Bayesian neural networks (BNNs) with cold posteriors outperform models with SGD based maximum likelihood training, while BNNs with $T = 1$ can perform worse than the maximum likelihood solution. They claim that cold posteriors sharply deviate from the Bayesian paradigm, and consider possible reasons for why tempering is helpful in Bayesian deep learning.

In this section, we provide an alternative view and argue that tempering is not at odds with Bayesian principles. Moreover, for virtually any realistic model class and dataset, it would be highly surprising if $T = 1$ *were* in fact the best setting of this hyperparameter. Indeed, as long as it is practically convenient, we would advocate tempering for essentially *any* model, especially parametric models that do not scale their capacity automatically with the amount of available information. Our position is that at a high level Bayesian methods are trying to combine honest beliefs with data to form a posterior. By reflecting the belief that the model is misspecified, the tempered posterior is often more of a *true posterior* than the posterior that results from ignoring our belief that the model misspecified.

Finding that $T < 1$ helps for Bayesian neural networks is neither surprising nor discouraging. And the actual results of the experiments in Wenzel et al. [2020], which show great improvements over standard SGD training, are in fact very encouraging of deriving inspiration from Bayesian procedures in deep learning.

We consider (1) tempering under misspecification (Section 3.10.1); (2) tempering in terms of overcounting data (Section 3.10.2); (3) how tempering compares to changing the observation model (Section 3.10.3); (4) the effect of the prior in relation to the experiments of Wenzel et al. [2020] (Section 3.10.4); (5) the effect of approximate inference, including how tempering can help in efficiently estimating parameters even for the untempered posterior (Section 3.10.5).

This section shows how tempering can be a reasonable procedure, and addresses several of the points in Wenzel et al. [2020], particularly on prior misspecification.

Since the original publication of our paper, there have been many papers discussing the cold posterior effect. In Section 4.8 (also, Izmailov et al. [2021b]), we show that there is no cold posterior effect in any of the examples of Wenzel et al. [2020] if we remove data augmentation. In Kapoor et al. [2022], we show precisely how data augmentation leads to underconfidence in Bayesian classification, and how posterior tempering can more naturally reflect our beliefs about aleatoric uncertainty than using $T = 1$. We also show that the cold posterior effect can be removed in the presence of data augmentation by using a Dirichlet observation model, which explicitly enables one to represent aleatoric uncertainty.

### 3.10.1 Tempering Helps with Misspecified Models

Many works explain how tempered posteriors help under model misspecification [e.g., de Heide et al. 2019; Grünwald et al. 2017; Barron and Cover 1991; Walker and Hjort 2001; Zhang 2006; Bissiri et al. 2016; Grünwald 2012]. In fact, de Heide et al. [2019] and Grünwald et al. [2017] provide several simple examples where Bayesian inference fails to provide good convergence behaviour for untempered posteriors. While it is easier to show theoretical results for $T > 1$, several of these works also show that $T < 1$ can be preferred, even in well-specified settings, and indeed recommend learning $T$ from data, for example by cross-validation [e.g., Grünwald 2012; de Heide et al. 2019].

Are we in a misspecified setting for Bayesian neural networks? Of course. And it would be irrational to proceed as if it were otherwise. Every model is misspecified. In the context of Bayesian neural networks specifically, the mass of solutions expressed by the prior outside of the datasets we typically consider is likely much larger than desired for most applications. We can calibrate for this discrepancy through tempering. The resulting tempered posterior will be more in line with our beliefs than pretending the model is not misspecified and finding the untempered posterior.

*Non-parametric models*, such as Gaussian processes, attempt to side-step model misspecification by growing the number of free parameters (information capacity) automatically with the amount of available data. In parametric models, we take much more of a manual guess about the model capacity. In the case of deep neural networks, this choice is not even close to a *best guess*; it was once the case that architectural design was a large component of works involving neural networks, but now it is more standard practice to choose an off-the-shelf architecture, without much consideration of model capacity. We do not believe that knowingly using a misspecified model to find a posterior is more reasonable (or Bayesian) than honestly reflecting the belief that the model is misspecified and then using a tempered posterior. For parametric models such as neural networks, it is to be expected that the capacity is particularly misspecified.

### 3.10.2 Overcounting Data with Cold Posteriors

The criticism of cold posteriors raised by Wenzel et al. [2020] is largely based on the fact that decreasing temperature leads to overcounting data in the posterior distribution.

However, a similar argument can be made against marginal likelihood maximization (also known as *empirical Bayes* or *type 2 maximum likelihood*). Indeed, here, the prior will depend on the same data as the likelihood, which can lead to miscalibrated predictive distributions [Darnieder 2011].

Nonetheless, empirical Bayes has been embraced and widely adopted in Bayesian machine learning [e.g., Bishop and Nasrabadi 2006; Rasmussen and Williams 2006; MacKay 2003; Minka 2000a], as embodying several Bayesian principles. Empirical Bayes has been particularly embraced in seminal work on Bayesian neural networks [e.g., MacKay 1992a, 1995], where it has been proposed as a principled approach to learning hyperparameters, such as the scale of the variance for the prior over weights, automatically embodying Occam's razor. While there is in this case some deviation from the fully Bayesian paradigm, the procedure, which depends on marginalization, is nonetheless clearly inspired by Bayesian thinking — and it is thus helpful to reflect this inspiration and provide understanding of how it works from a Bayesian perspective.

There is also work showing the marginal likelihood can lead to miscalibrated Bayes factors under model misspecification. Attempts to calibrate these factors [Xu et al. 2019], as part of the Bayesian paradigm, are highly reminiscent of work on safe Bayes.

### 3.10.3 Tempered Posterior or Different Likelihood?

In some cases, the tempered posterior for one model is an untempered posterior using a different likelihood function. Specifically, consider regression with a Gaussian likelihood and noise variance $\sigma^2$:

$$\begin{aligned} p(y|x,w) &= \mathcal{N}(y|f(x,w),\sigma^2) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(y-f(x,w))^2}{2\sigma^2}\right), \end{aligned}$$

where $f(x, w)$ is the prediction of the model $f$ with parameters $w$ on the input $x$. Then, tempering the likelihood, we achieve

$$\begin{aligned} p(y|x,w)^{1/T} &= \frac{1}{\sqrt{2\pi\sigma^2}^{1/T}} \cdot \exp\left(-\frac{(y - f(x,w))^2}{2T\sigma^2}\right) \\ &= \mathcal{N}(y|f(x,w), T\sigma^2) \cdot \sqrt{\frac{(2\pi T\sigma^2)}{(2\pi\sigma^2)^{1/T}}} \\ &= \mathcal{N}(y|f(x,w), T\sigma^2) \cdot C, \end{aligned}$$

where $C$ is a renormalization constant that does not depend on the parameters $w$ of the model. In this case, the standard Bayesian posterior in the model with noise variance $T\sigma^2$ is equal to the posterior of temperature $T$ in the original model with noise variance $\sigma^2$. Section 4.1 of Grünwald et al. [2017] considers a related construction.

The predictive distribution differs for the two models; even though the posteriors coincide, the likelihoods for a new datapoint $y^*$ are different:

$$\int p(y^*|w)p(w)dw \neq \int p_T(y^*|w)p(w)dw\,. \tag{3.5}$$

For the Gaussian model described above, the predictions of the tempered model and model with modified likelihood will have the same mean but different predictive variance.

Wenzel et al. [2020] provide a construction of a likelihood function that is equivalent to tempering for classification problems. For a general likelihood function $p(\mathcal{D}|w)$, we can consider a modified likelihood of the form

$$\hat{p}(\mathcal{D}|w) = p(\mathcal{D}|w)^{1/T} \cdot C(w), \tag{3.6}$$

where $C(w)$ is a renormalization constant that in general depends on the parameters $w$ and inputs $x$, but not the target values $y$. The standard posterior $\hat{p}(w|\mathcal{D})$ in the model with the modified likelihood will then coincide with the tempered posterior $p_T(w|\mathcal{D})$ in the original model up to $C(w)$:

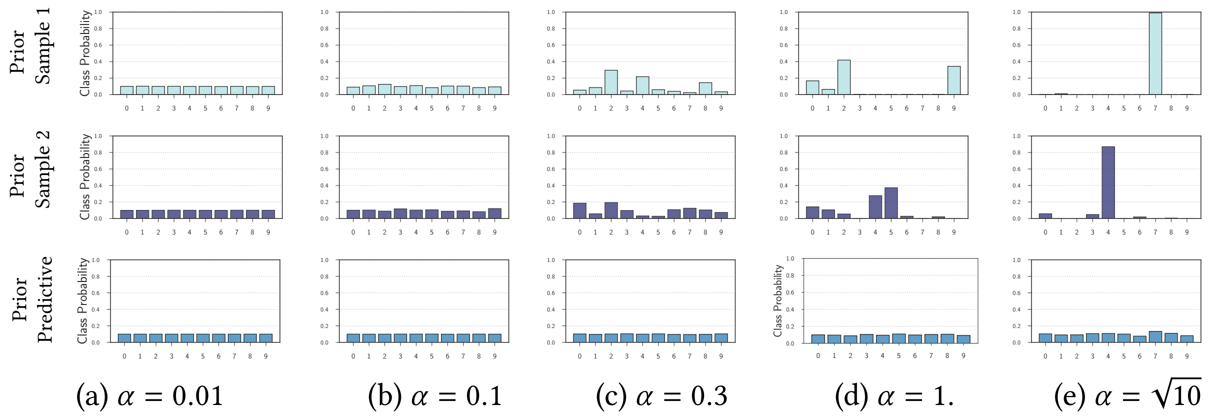
$$\frac{\hat{p}(w|\mathcal{D})}{p_T(w|\mathcal{D})} = C(w). \tag{3.7}$$

Furthermore, as Wenzel et al. [2020] show, the tempered softmax likelihood with $T < 1$ can be viewed as a valid likelihood if we introduce a new class, not observed in the training data. While they discard that particular interpretation, it is not unreasonable to include an unobserved class, since our observation model may want to recognize that we have not observed all possible classes, and therefore retain an additional label. This extra class can, for example, correspond to all the possible images that do not belong to any of the classes in our dataset. Finally, new work [Kapoor et al. 2022] shows that we can naturally interpret the tempered likelihood as using the multinomial observation model, assuming $1/T$ counts of the label are observed for each of the training datapoints, which is perfectly valid.

We present other key considerations in the discussion of tempering for Bayes posteriors in other parts of this section.

### 3.10.4 Effect of the Prior

While a somewhat misspecified prior will certainly interact with the utility of tempering, we do not believe the experiments in Wenzel et al. [2020] provide evidence that even the prior $p(w) = \mathcal{N}(0, I)$ is misspecified to any serious extent. For a relatively wide range of distributions over $w$, the functional form of the network $f(x; w)$ can produce a generally reasonable distribution over functions $p(f(x; w))$. In Figure 3.13, we reproduce the findings in Wenzel et al. [2020] showing sample functions $p(f(x; w))$ corresponding to the prior $p(w) = \mathcal{N}(0, 10 \cdot I)$ strongly favour a single class over the dataset. While this behaviour appears superficially dramatic, we note it is simply an artifact a miscalibrated signal variance. A miscalibrated signal variance interacts



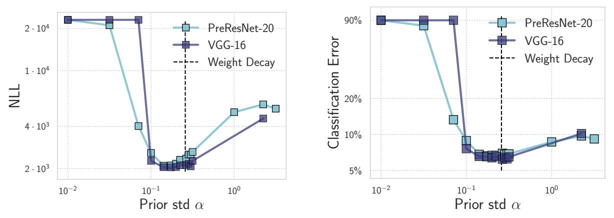


**Figure 3.12: Effects of the prior variance $\alpha^2$. (a)–(e)**: Average class probabilities over all of CIFAR-10 for two sample prior functions $p(f(x;w))$ (two top rows) and predictive distribution (average over 200 samples of weights, bottom row) for varying settings of $\alpha$ in $p(w) = \mathcal{N}(0, \alpha^2 I)$. **(f)**: NLL and **(g)** classification error of an ensemble of 20 SWAG samples on CIFAR-10 as a function of $\alpha$ using a Preactivation ResNet-20 and VGG-16. The NLL is high for overly small $\alpha$ and near-optimal in the range of $[0.1, 0.3]$. The NLL remains relatively low for vague priors corresponding to large values of $\alpha$.

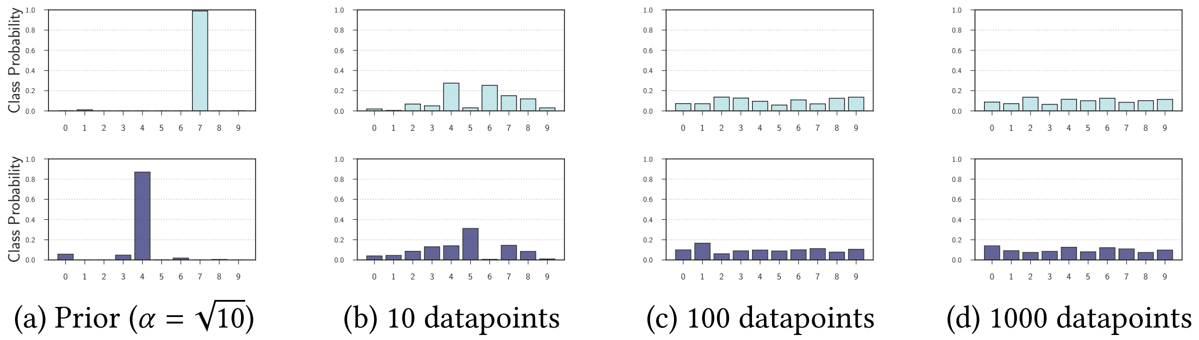


**Figure 3.13: Adaptivity of posterior variance with data**. We sample two functions $f(x;w)$ from the distribution over functions induced by a distribution over weights, starting with the prior $p(w) = \mathcal{N}(0, 10 \cdot I)$, in combination with a PreResNet-20. We measure class probabilities averaged across the CIFAR-10 test set, as we vary the amount of available training data. Although the prior variance is too large, such that the softmax saturates for logits sampled from the prior, leading to one class being favoured, we see that the posterior quickly adapts to correct the scale of the logits in the presence of data. In Figure 3.12 we also show that the prior variance can easily be calibrated such that the prior predictive distribution, even before observing data, is high entropy.

with a quickly saturating soft-max link function to provide a seemingly dramatic preference to a given class. If we instead use $p(w) = \mathcal{N}(0, \alpha^2 I)$, for quite a range of $\alpha$, then sample functions provide reasonably high entropy across labels averaged over the dataset, as in Figure 3.12. For individual points the posterior samples have different particular class preferences. $\alpha$ can be easily determined through cross-validation, as in Figure 3.12, or specified as a standard value used for $L_2$ regularization ($\alpha = 0.24$ in this case).

However, even with the inappropriate prior scale, we see in panels (a)–(e) of Figure 3.12 that the unconditional predictive distribution *is* completely reasonable. Moreover, the prior variance represents a *soft* prior bias, and will quickly update with data. In Figure 3.13 we show posterior samples after observing 10, 100, and 1000 data points.

Other aspects of the prior, outside of the prior signal variance, will have a much greater effect on the inductive biases of the model. For example, the induced covariance function $\mathrm{cov}(f(x_i, w), f(x_j, w))$ reflects the induced similarity metric over data instances; through the covariance function we can answer, for instance, whether the model believes a priori that a translated image is similar to the original. Unlike the signal variance of the prior, the prior covariance function will continue to have a significant effect on posterior inference for even very large datasets, and strongly reflects the structural properties of the neural network. We explore these structures of the prior in Figure A.16.

### 3.10.5 The Effect of Inexact Inference

We have to keep in mind what we ultimately use posterior samples to compute. Ultimately, we wish to estimate the predictive distribution given by the integral in Equation (3.1). With a finite number of samples, the tempered posterior could be used to provide a better approximation to the expectation of the predictive distribution associated with untempered posterior.

Consider a simple example, where we wish to estimate the mean of a high-dimensional Gaussian distribution $\mathcal{N}(0, I)$. Suppose we use $J$ independent samples. The mean of these samples is also Gaussian distributed, $\mu \sim \mathcal{N}\left(0, \frac{1}{J}I\right)$. In Bayesian deep learning, the dimension $d$ is typically on the order $10^7$, and $J$ would be on the order of 10. The norm of $\mu$ would be highly concentrated around $\frac{\sqrt{10^7}}{\sqrt{10}} = 1000$. In this case, sampling from a tempered posterior with $T < 1$ would lead to a better approximation of the Bayesian model average associated with an untempered posterior.

Furthermore, no current sampling procedure will be providing samples that are close to independent samples from the true posterior of a Bayesian neural network. The posterior landscape is far too multimodal and complex for there to be any reasonable coverage. The approximations we have are practically useful, and often preferable to conventional training, but we cannot realistically proceed with analysis assuming that we have obtained true samples from a posterior. While we would expect that some value of $T \neq 1$ would be preferred for any finite dataset in practice, it is conceivable that some of the results in Wenzel et al. [2020] may be affected by the specifics of the approximate inference technique being used.

We should be wary not to view Bayesian model averaging purely through the prism of simple Monte Carlo, as advised in Section 3.2.2. Given a finite computational budget, our goal in effectively approximating a Bayesian model average is *not* equivalent to obtaining good samples from the posterior.

## 3.11 Related Work

Notable early works on Bayesian neural networks include MacKay [1992a], MacKay [1995], and Neal [1996]. These works generally argue in favour of making the model class for Bayesian approaches as flexible as possible, in line with Box and Tiao [1973]. Accordingly, Neal [1996] pursued the limits of large Bayesian neural networks, showing that as the number of hidden units approached infinity, these models become Gaussian processes with particular kernel functions. This work harmonizes with recent work describing the neural tangent kernel [e.g., Jacot et al. 2018].

The marginal likelihood is often used for Bayesian hypothesis testing, model comparison, and hyperparameter tuning, with *Bayes factors* used to select between models [Kass and Raftery 1995]. MacKay [2003, Ch. 28] uses a diagram similar to Fig 3.2 to show the marginal likelihood has an *Occam's razor* property, favouring the simplest model consistent with a given dataset, even if the prior assigns equal probability to the various models. Rasmussen and Ghahramani [2001] reasons about how the marginal likelihood can favour large flexible models, as long as such models correspond to a reasonable distribution over functions.

There has been much recent interest in developing Bayesian approaches for modern deep learning, with new challenges and architectures quite different from what had been considered in early work. Recent work has largely focused on scalable inference [e.g., Blundell et al. 2015; Gal and Ghahramani 2016; Kendall and Gal 2017; Ritter et al. 2018b; Khan et al. 2018], function-space inspired priors [e.g., Yang et al. 2019b; Louizos et al. 2019; Sun et al. 2019; Hafner et al. 2018], and developing flat objective priors in parameter space, directly leveraging the biases of the neural network functional form [e.g, Nalisnick 2018]. Wilson [2020] provides a note motivating Bayesian deep learning.

Early works tend to provide a connection between loss geometry and generalization using minimum description length frameworks [e.g., Hinton and Van Camp 1993; Hochreiter and Schmidhuber 1997a; MacKay 1995]. Empirically, Keskar et al. [2016] argue that smaller batch SGD provides better generalization than large batch SGD, by finding flatter minima. Chaudhari et al. [2016] and Izmailov et al. [2018] (see Section 2.5) design optimization procedures to specifically find flat minima.

By connecting flat solutions with ensemble approximations, Izmailov et al. [2018] also suggest that functions associated with parameters in flat regions ought to provide different predictions on test data, for flatness to be helpful in generalization, which is distinct from the flatness in Dinh et al. [2017b]. Garipov et al. [2018] (see Section 2.1) also show that there are mode connecting curves, forming loss valleys, which contain a variety of distinct solutions. We argue that flat regions of the loss containing a diversity of solutions are particularly relevant for Bayesian model averaging, since the model average will then contain many compelling and complementary explanations for the data. Additionally, Huang et al. [2019] describes neural networks as having a *blessing of dimensionality*, since flat regions will occupy much greater volume in a high dimensional space, which we argue means that flat solutions will dominate in the Bayesian model average.

Smith and Le [2018] and MacKay [2003, Chapter 28] additionally connect the width of the posterior with Occam factors; from a Bayesian perspective, larger width corresponds to a smaller Occam factor, and thus ought to provide better generalization. Dziugaite and Roy [2017] and Smith and Le [2018] also provide different perspectives on the results in Zhang et al. [2017a], which shows that deep convolutional neural networks can fit CIFAR-10 with random labels and no training error. The PAC-Bayes bound of Dziugaite and Roy [2017] becomes vacuous when applied to randomly-labelled binary MNIST. Smith and Le [2018] show that logistic regression can fit noisy labels on sub-sampled MNIST, interpreting the result from an Occam factor perspective.

In general, PAC-Bayes provides a compelling framework for deriving explicit non-asymptotic generalization bounds for stochastic networks with distributions over parameters [McAllester 1999; Langford and Caruana 2002; Dziugaite and Roy 2017; Neyshabur et al. 2017, 2018; Masegosa 2019; Jiang et al. 2019; Guedj 2019; Alquier 2021]. Langford and Caruana [2002] devised a PAC-Bayes generalization bound for small stochastic neural networks (two layer with two hidden units) achieving an improvement over the existing deterministic generalization bounds. Dziugaite and Roy [2017] extended this approach, optimizing a PAC-Bayes bound with respect to a parametric distribution over the weights of the network, exploiting the flatness of solutions discovered by SGD, for non-vacuous bounds with an overparametrized network on binary MNIST. Neyshabur et al. [2017] also discuss the connection between PAC-Bayes bounds and sharpness, and Neyshabur et al. [2018] devises PAC-Bayes bounds based on spectral norms of the layers and the Frobenius norm of the weights of the network. Achille and Soatto [2018] additionally combine PAC-Bayes and information theoretic approaches to argue that flat minima have low information content. Masegosa [2019] also proposes variational and ensemble learning methods based on PAC-Bayes analysis under model misspecification. Jiang et al. [2019] provide a review and comparison of several generalization bounds, including PAC-Bayes.

Our contributions are largely orthogonal and complementary to PAC-Bayes. PAC-Bayes bounds can be be improved by, e.g. fewer parameters, and very compact priors, which can be different from what provides optimal generalization. From our perspective, model flexibility and priors with *large* support, rather than compactness, are desirable. Moreover, we show the great significance of multi-basin marginalization for generalization, whereas multi-basin marginalization has a minimal logarithmic effect on PAC-Bayes bounds. Indeed, *marginalization* — a posterior weighted model average — is our key focus, whereas PAC-Bayes bounds are typically bounding the empirical risk of a single sample. In general, our focus is complementary to PAC-Bayes, aiming to provide *prescriptive* intuitions on model construction, inference, and neural network priors, as well as new connections between Bayesian model averaging and deep ensembles, benefits of Bayesian model averaging in the context of modern deep neural networks, views of marginalization that contrast with simple Monte Carlo, and new methods for Bayesian marginalization in deep learning.

In other work, Pearce et al. [2018] propose a modification of deep ensembles and argue that it performs approximate Bayesian inference, and Gustafsson et al. [2019] briefly mention how deep ensembles can be viewed as samples from an approximate posterior. In the context of deep ensembles, we believe it is natural to consider the BMA integral separately from the simple Monte Carlo approximation that is often used to approximate this integral; to compute an accurate predictive distribution, we do not need samples from a posterior, or even a faithful approximation to the posterior.

Fort et al. [2019] considered the diversity of predictions produced by models from multiple independent SGD runs, and suggested to ensemble averages of SGD iterates. Although MultiSWA (one of the methods considered in Section 3.6) is related to this idea, the crucial practical difference is that MultiSWA uses a learning rate schedule that selects for flat regions of the loss, the key to the success of the SWA method [Izmailov et al. 2018]. Section 3.6 also shows that MultiSWAG, which we propose for multimodal Bayesian marginalization, outperforms MultiSWA.

Double descent, which describes generalization error that decreases, increases, and then again decreases with model flexibility, was demonstrated early by Opper et al. [1990]. Recently, Belkin et al. [2019] extensively demonstrated double descent, leading to a surge of modern interest, with Nakkiran et al. [2019] showing double descent in deep learning. Nakkiran et al. [2020] shows that tuned $l_2$ regularization can mitigate double descent. Alternatively, we show that Bayesian model averaging, particularly based on multimodal marginalization, can alleviate even prominent double descent behaviour.

Tempering in Bayesian modelling has been considered under the names *Safe Bayes*, *generalized Bayesian inference*, and *fractional Bayesian inference* [e.g., de Heide et al. 2019; Grünwald et al. 2017; Barron and Cover 1991; Walker and Hjort 2001; Zhang 2006; Bissiri et al. 2016; Grünwald 2012]. We provide several perspectives of tempering in Bayesian deep learning, and analyze the results in a recent paper by Wenzel et al. [2020] that questions tempering for Bayesian neural networks.

### 3.11.1 Methods for Uncertainty Estimation

Markov chain Monte Carlo (MCMC) was at one time a gold standard for inference with neural networks, through the Hamiltonian Monte Carlo (HMC) work of Neal [1996]. However, HMC requires full gradients, which is computationally intractable for modern neural networks. To extend the HMC framework, stochastic gradient HMC (SGHMC) was introduced by Chen et al. [2014] and allows for stochastic gradients to be used in Bayesian inference, crucial for both scalability and exploring a space of solutions that provide good generalization. Alternatively, stochastic gradient Langevin dynamics (SGLD) [Welling and Teh 2011] uses first order Langevin dynamics in the stochastic gradient setting. Theoretically, both SGHMC and SGLD asymptotically sample from the posterior in the limit of infinitely small step sizes. In practice, using finite learning rates introduces approximation errors (see e.g. [Mandt et al. 2017a]), and tuning stochastic gradient MCMC methods can be quite difficult.

Variational Inference: Graves [2011] suggested fitting a Gaussian variational posterior approximation over the weights of neural networks. This technique was generalized by Kingma and Welling [2013] which proposed the *reparameterization trick* for training deep latent variable models; multiple variational inference methods based on the reparameterization trick were proposed for DNNs [e.g., Kingma et al. 2015; Blundell et al. 2015; Molchanov et al. 2017; Louizos and Welling 2017]. While variational methods achieve strong performance for moderately sized networks, they are empirically noted to be difficult to train on larger architectures such as deep residual networks [He et al. 2016]; Blier and Ollivier [2018] argue that the difficulty of training is explained by variational methods providing inusfficient data compression for DNNs despite being designed for data compression (minimum description length). Recent key advances [Louizos and Welling 2017; Wu et al. 2018] in variational inference for deep learning typically focus on smaller-scale datasets and architectures. An alternative line of work re-interprets noisy versions of optimization algorithms: for example, noisy Adam [Khan et al. 2018] and noisy KFAC [Zhang et al. 2017b], as approximate variational inference.

Dropout Variational Inference: Gal and Ghahramani [2016] used a spike and slab variational distribution to view dropout at test time as approximate variational Bayesian inference. Concrete dropout [Gal et al. 2017] extends this idea to optimize the dropout probabilities as well. From a practical perspective, these approaches are quite appealing as they only require ensembling dropout predictions at test time, and they were succesfully applied to several downstream tasks [Kendall and Gal 2017; Mukhoti and Gal 2018].

Laplace Approximations assume a Gaussian posterior, $\mathcal{N}(\theta^*, \mathcal{I}(\theta^*)^{-1})$, where $\theta^*$ is a MAP estimate and $\mathcal{I}(\theta^*)^{-1}$ is the inverse of the Fisher information matrix (expected value of the Hessian evaluated at $\theta^*$). It was notably used for Bayesian neural networks in MacKay [1992b], where a diagonal approximation to the inverse of the Hessian was utilized for computational reasons. More recently, Kirkpatrick et al. [2017] proposed using diagonal Laplace approximations to overcome catastrophic forgetting in deep learning. Ritter et al. [2018b] proposed the use of either a diagonal or block Kronecker factored (KFAC) approximation to the Hessian matrix for Laplace approximations, and Ritter et al. [2018a] successfully applied the KFAC approach to online learning scenarios.

SGD Based Approximations. Mandt et al. [2017a] proposed to use the iterates of averaged SGD as an MCMC sampler, after analyzing the dynamics of SGD using tools from stochastic calculus. From a frequentist perspective, Chen et al. [2016] showed that under certain conditions a batch means estimator of the sample covariance matrix of the SGD iterates converges to $A = \mathcal{H}(\theta)^{-1}C(\theta)\mathcal{H}(\theta)^{-1}$, where $\mathcal{H}(\theta)^{-1}$ is the inverse of the Hessian of the log likelihood and $C(\theta) = \mathbb{E}(\nabla \log p(\theta) \nabla \log p(\theta)^T)$ is the covariance of the gradients of the log likelihood. Chen et al. [2016] then show that using $A$ and the sample average of the iterates for a Gaussian approximation produces well calibrated confidence intervals of the parameters and that the variance of these estimators achieves the Cramer Rao lower bound (the minimum possible variance). A description of the asymptotic covariance of the SGD iterates dates back to Ruppert [1988] and Polyak and Juditsky [1992], who show asymptotic convergence of Polyak-Ruppert averaging.

Methods for Calibration of DNNs. Lakshminarayanan et al. [2017] proposed using ensembles of several networks for enhanced calibration, and incorporated an adversarial loss function to be used when possible as well. Outside of probabilistic neural networks, Guo et al. [2017] proposed temperature scaling, a procedure which uses a validation set and a single hyperparameter to rescale the logits of DNN outputs for enhanced calibration. Kuleshov et al. [2018] propose calibrated regression using a similar rescaling technique.

## 3.12 Discussion

*"It is now common practice for Bayesians to fit models that have more parameters than the number of data points… Incorporate every imaginable possibility into the model space: for example, if it is conceivable that a very simple model might be able to explain the data, one should include simple models; if the noise might have a long-tailed distribution, one should include a hyperparameter which controls the heaviness of the tails of the distribution; if an input variable might be irrelevant to a regression, include it in the regression anyway."* MacKay [1995]

We have presented a probabilistic perspective of generalization, which depends on the support and inductive biases of the model. The support should be as large possible, but the inductive biases must be well-calibrated to a given problem class. We argue that Bayesian neural networks embody these properties — and through the lens of probabilistic inference, explain generalization behaviour that has previously been viewed as mysterious. Moreover, we argue that Bayesian marginalization is particularly compelling for neural networks, show how deep ensembles provide a practical mechanism for marginalization, and propose a new approach that generalizes deep ensembles to marginalize within basins of attraction. We show that this multimodal approach to Bayesian model averaging, MultiSWAG, can entirely alleviate double descent, to enable monotonic performance improvements with increases in model flexibility, as well significant improvements in generalization accuracy and log-likelihood over SGD and single basin marginalization.

In the next Chapter 4, we present a detailed analysis of Bayesian neural networks, where we aim to approximate the posterior as precisely as possible. We find that the intuitions that we built in this chapter hold for BNNs with precise inference, and also identify several new surprising phenomena.

# 4 | Detailed Study of Bayesian Neural Network Posteriors

In the previous Chapter 3, we presented a broad argument favouring a Bayesian approach to deep learning [see also, MacKay 1995; Neal 1996; Blundell et al. 2015; Gal 2016]. Bayesian inference for neural networks promises improved predictions, reliable uncertainty estimates, and principled model comparison, naturally supporting active learning, continual learning, and decision-making under uncertainty. The Bayesian deep learning community has designed multiple successful practical methods inspired by the Bayesian approach [Blundell et al. 2015; Gal and Ghahramani 2016; Welling and Teh 2011; Kirkpatrick et al. 2017; Maddox et al. 2019; Izmailov et al. 2019; Daxberger et al. 2020] with applications ranging from astrophysics [Cranmer et al. 2021] to automatic diagnosis of Diabetic Retinopathy [Filos et al. 2019], click-through rate prediction in advertising [Liu et al. 2017] and modeling of fluid dynamics [Geneva and Zabaras 2020].

However, inference with modern neural networks is distinctly challenging. We wish to compute a Bayesian model average corresponding to an integral over a multi-million dimensional multi-modal posterior, with unusual topological properties like mode-connectivity (see Chapter 2), under severe computational constraints.

There are therefore many unresolved questions about Bayesian deep learning practice. Variational procedures typically provide unimodal Gaussian approximations to the multimodal posterior. Practically successful methods such as deep ensembles [Lakshminarayanan et al. 2017; Fort et al. 2019] have a natural Bayesian interpretation [Wilson and Izmailov 2020], but only represent modes of the posterior. While Stochastic MCMC [Welling and Teh 2011; Chen et al. 2014; Zhang et al. 2020c] is computationally convenient, it could be providing heavily biased estimates of posterior expectations. Moreover, Wenzel et al. [2020] question the quality of standard Bayes posteriors, citing results where "cold posteriors", raised to a power $1/T$ with $T < 1$, improve performance.

Additionally, Bayesian deep learning methods are typically evaluated on their ability to generate useful, well-calibrated predictions on held-out or out-of-distribution data. However, strong performance on benchmark problems does not imply that the algorithm accurately approximates the true Bayesian model average (BMA).

In this chapter, we investigate fundamental open questions in Bayesian deep learning, using multi-chain full-batch Hamiltonian Monte Carlo [HMC, Neal et al. 2011]. HMC is a highly-efficient and well-studied Markov Chain Monte Carlo (MCMC) method that is guaranteed to asymptotically produce samples from the true posterior. However it is enormously challenging to apply HMC to modern neural networks due to its extreme computational requirements: HMC can take *tens of thousands of training epochs* to produce a single sample from the posterior. To address this computational challenge, we parallelize the computation over hundreds of Tensor processing unit (TPU) devices.

We argue that full-batch HMC provides the most precise tool for studying the BNN posterior to date. We are not proposing HMC as a computationally efficient method for practical applications. Rather, using our implementation of HMC we are able to explore fundamental questions about posterior geometry, the performance of BNNs, approximate inference, effect of priors and posterior temperature.

In particular, we show: (1) BNNs can achieve significant performance gains over standard training and deep ensembles; (2) a single long HMC chain can provide a comparable representation of the posterior to multiple shorter chains; (3) in contrast to recent studies, we find posterior tempering is not needed for near-optimal performance, with little evidence for a "cold posterior" effect, which we show is largely an artifact of data augmentation; (4) BMA performance is robust to the choice of prior scale, and relatively similar for diagonal Gaussian, mixture of Gaussian, and logistic priors over weights. This result highlights the importance of architecture relative to parameter priors in specifying the prior over functions. (5) While Bayesian neural networks have good performance for OOD detection, they show surprisingly poor generalization under domain shift; (6) while cheaper alternatives such as deep ensembles and SGMCMC can provide good generalization, they provide distinct predictive distributions from HMC. Notably, deep ensemble predictive distributions are similarly close to HMC as standard SGLD, and closer than standard variational inference.

We additionally show how to effectively deploy full batch HMC on modern neural networks, including insights about how to tune crucial hyperparameters for good performance, and parallelize sampling over hundreds of TPUs. Our HMC samples and implementation are a public resource.

This chapter is adapted from the papers "What Are Bayesian Neural Network Posteriors Really Like?" [Izmailov et al. 2021b] which originally appeared at ICML 2021 and "Dangers of Bayesian Model Averaging under Covariate Shift" [Izmailov et al. 2021a] which originally appeared at NeurIPS 2021, written jointly with Sharad Vikram, Matthew D. Hoffman, Patrick Nicholson, Sanae Lotfi and Andrew Gordon Wilson.

## 4.1 Background

See Chapter 3 for a detailed discussion of Bayesian deep learning. Here, we introduce the concepts and notation used throughout this chapter.

Bayesian neural networks. The goal of classical learning is to find a single best setting of the parameters for the model, typically through maximum-likelihood optimization. In the Bayesian framework, the learner instead infers a *posterior* distribution $p(w|\mathcal{D})$ over the parameters $w$ of the model after observing the data $\mathcal{D}$. The posterior distribution is given by Bayes' rule: $p(w|\mathcal{D}) \propto p(\mathcal{D}|w)p(w)$, where $p(\mathcal{D}|w)$ is the likelihood of $\mathcal{D}$ given by the model with parameters $w$, and $p(w)$ is the prior distribution over the parameters. The predictions of the model on a new test example $x$ are then given by the *Bayesian model average* (BMA)

$$p(y|x, \mathcal{D}) = \int_w p(y|x, w)p(w|\mathcal{D})dw, \tag{4.1}$$

where $p(y|x, w)$ is the predictive distribution for a given value of the parameters $w$. This BMA is particularly compelling in Bayesian deep learning, because the posterior over parameters for a modern neural network can represent many complementary solutions to a given problem, corresponding to different settings of parameters [Wilson and Izmailov 2020]. Unfortunately, the integral in Eq. (4.1) cannot be evaluated in closed form for neural networks, so one must resort to approximate inference. Moreover, approximating Eq. (4.1) is challenging due to a high dimensional and sophisticated posterior $p(w|\mathcal{D})$.

Markov Chain Monte Carlo. The integral in Eq. (4.1) can be approximated by sampling: $p(y|x, \mathcal{D}) \approx \frac{1}{M}\sum_{i=1}^{M} p(y|x, w_i)$, where $w_i \sim p(w|\mathcal{D})$ are samples drawn from the posterior. MCMC methods construct a Markov chain that, if simulated for long enough, generates approximate samples from the posterior. In this work, we focus on Hamiltonian Monte Carlo [Neal et al. 2011], a method that produces asymptotically exact samples assuming access to the unnormalized posterior density $p(\mathcal{D}|w)p(w)$ and its gradient.

Maximum a-posteriori (MAP) estimation. In contrast with Bayesian model averaging, a MAP estimator uses the single setting of weights (hypothesis) that maximizes the posterior density $w_{\text{MAP}} = \underset{w}{\operatorname{argmax}}\, p(w|\mathcal{D}) = \underset{w}{\operatorname{argmax}}(\log p(\mathcal{D}|w) + \log p(w))$, where the log prior can be viewed as a regularizer. For example, if we use a Gaussian prior on $w$, then $\log p(w)$ will penalize the $\ell_2$ norm of the parameters, driving parameters that do not improve the log likelihood $\log p(\mathcal{D}|w)$ to zero. MAP is the standard approach to training neural networks and our baseline for *classical training* throughout the paper. We perform MAP estimation with SGD unless otherwise specified.

## 4.2 HMC for deep neural networks

We use full-batch Hamiltonian Monte Carlo (HMC) to sample from the parameter posteriors for Bayesian neural networks. In this section, we show how to make HMC effective for modern Bayesian neural networks, discussing important details such as hyper-parameter specification. In the next sections, we use the HMC samples to explore fundamental questions about approximate inference in modern deep learning. We summarize HMC in Appendix Figure 4 and Figure 5. Intuitively, HMC is simulating the dynamics of a particle sliding on the plot of the density function that we are trying to sample from.[1]

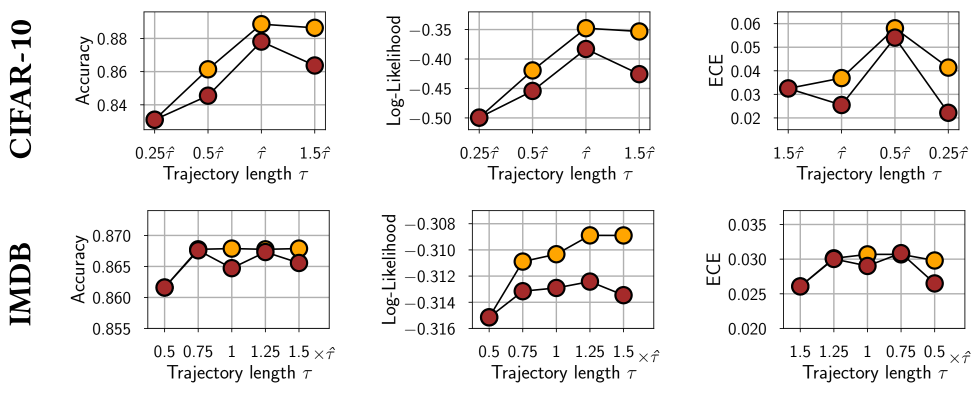
Implementation. To scale HMC to modern neural network architectures and for datasets like CIFAR-10 and IMDB, we parallelize the computation over 512 TPUv3 devices[2] [Jouppi et al. 2020]. We execute HMC in a single-program multiple-data (SPMD) configuration, wherein a dataset is sharded evenly over each of the devices and an identical HMC implementation is run on each device. Each device maintains a synchronized copy of the Markov chain state, where the full-batch gradients needed for leapfrog integration are computed using cross-device collectives. We release our JAX [Bradbury et al. 2018] implementation.

[1]For a detailed introduction to HMC please see Neal et al. [2011]. See also interactive visualization here: `http://chi-feng.github.io/mcmc-demo/`.

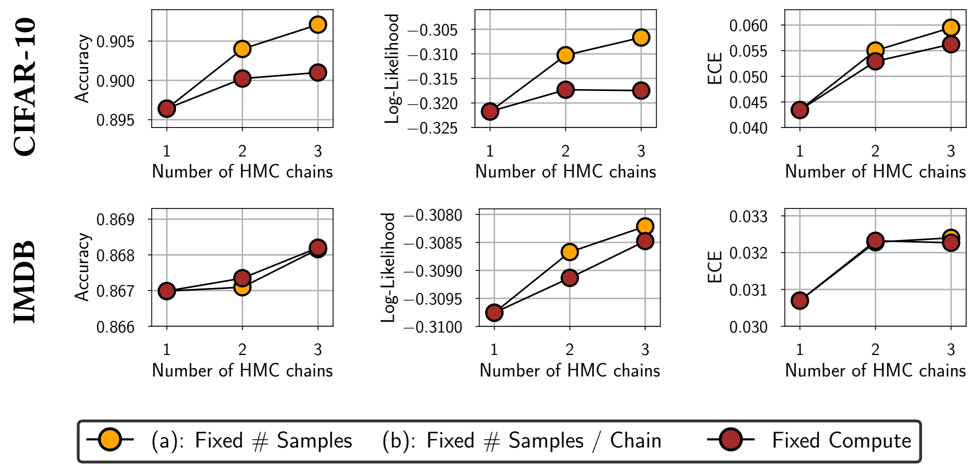
[2]We use other hardware configurations in several experiments. We state the hardware that we used in the corresponding sections.

Neural network architectures. In our evaluation, following Wenzel et al. [2020], we primarily focus on two architectures: ResNet-20-FRN and CNN-LSTM. ResNet-20-FRN is a residual architecture [He et al. 2016] of depth 20 with batch normalization layers [Ioffe and Szegedy 2015] replaced with filter response normalization [FRN; Singh and Krishnan 2020]. Batch normalization makes the likelihood harder to interpret by creating dependencies between training examples, whereas the outputs of FRN layers are independent across inputs. We use Swish (SiLU) activations [Hendrycks and Gimpel 2016; Elfwing et al. 2018; Ramachandran et al. 2017] instead of ReLUs to ensure smoothness of the posterior density surface, which we found improves acceptance rates of HMC proposals without hurting the overall performance. The CNN-LSTM is a long short-term memory network [Hochreiter and Schmidhuber 1997b] adapted from Wenzel et al. [2020] without modifications.

Datasets and Data Augmentation. In our main evaluations we use the CIFAR image classification datasets [Krizhevsky et al. 2014] and the IMDB dataset [Maas et al. 2011] for sentiment analysis. We do not use any data augmentation, both because the random augmentations introduce stochasticity into the evaluation of the posterior log-density and its gradient, and because the expected randomly perturbed log-likelihood does not have a clean interpretation as a valid likelihood function [Wenzel et al. 2020]. In this section we perform ablations using ResNet-20-FRN on CIFAR-10 and CNN-LSTM on IMDB.



(a) Trajectory length ablation



(b) Number of chains ablation

**Figure 4.1: Effect of HMC hyper-parameters.** BMA accuracy, log-likelihood and expected calibration error (ECE) as a function of **(a)**: the trajectory length $\tau$ and **(b)**: number of HMC chains. The orange curve shows the results for a fixed number of samples in (a) and for a fixed number of samples per chain in (b); the brown curve shows the results for a fixed amount of compute. All experiments are done on CIFAR-10 using the ResNet-20-FRN architecture on IMDB using CNN-LSTM. Longer trajectory lengths decrease correlation between subsequent samples improving accuracy and log-likelihood. For a given amount of computation, increasing the number of chains from one to two modestly improves the accuracy and log-likelihood.

### 4.2.1 Trajectory length $\tau$

The trajectory length parameter $\tau$ determines the length of the dynamics simulation on each HMC iteration. Effectively, it determines the correlation of subsequent samples produced by HMC. To suppress random-walk behavior and speed up mixing, we want the length of the trajectory to be relatively high. But increasing the length of the trajectory also leads to an increased computational cost: the number of evaluations of the gradient of the target density (evaluations of the gradient of the loss on the full dataset) is equal to the ratio $\tau/\Delta$ of the trajectory length to the step size.

We suggest the following value of the trajectory length $\tau$:

$$\hat{\tau} = \frac{\pi \alpha_{\text{prior}}}{2}, \tag{4.2}$$

where $\alpha_{\text{prior}}$ is the standard deviation of the prior distribution over the parameters. If applied to a spherical Gaussian distribution, HMC with a small step size and this trajectory length will generate exact samples[3]. While we are interested in sampling from the posterior rather than from the spherical Gaussian prior, we argue that in large BNNs the prior tends to determine the scale of the posterior. We provide more detail and confirm this intuition empirically in the Section A.5.3.

In order to test the validity of our recommended trajectory length, we perform an ablation and report the results in Figure 4.1(a). As expected, longer trajectory lengths provide better performance in terms of accuracy and log-likelihood. Expected calibration error is generally low across the board. The trajectory length $\hat{\tau}$ provides good performance in all three metrics. This result confirms that, despite the expense, when applying HMC to BNNs it is actually helpful to use tens of thousands of gradient evaluations per iteration.

[3]Since the Hamiltonian defines a set of independent harmonic oscillators with period $2\pi\alpha$, $\tau = \pi\alpha/2$ applies a quarter-turn in phase space, swapping the positions and momenta.

### 4.2.2 Step size $\Delta$

The step size parameter $\Delta$ determines the discretization step size of the Hamiltonian dynamics and consequently the number of leapfrog integrator steps. Lower step sizes lead to a better approximation of the dynamics and higher rates of proposal acceptance at the Metropolis-Hastings correction step. However, lower step sizes require more gradient evaluations per iteration to hold the trajectory length $\tau$ constant.

Using ResNet-20-FRN on CIFAR-10, we run HMC for 50 iterations with step sizes of $1 \cdot 10^{-5}$, $5 \cdot 10^{-5}$, $1 \cdot 10^{-4}$, and $5 \cdot 10^{-4}$ respectively, ignoring the Metropolis-Hastings correction. We find the chains achieve average accept probabilities of 72.2%, 46.3%, 22.2%, and 12.5%, reflecting large drops in accept probability as step size is increased. We also observe BMA log-likelihoods of $-0.331$, $-0.3406$, $-0.3407$, and $-0.895$, indicating that higher accept rates result in higher likelihoods.

### 4.2.3 Number of HMC chains

We can improve the coverage of the posterior distribution by running multiple independent chains of HMC. Effectively, each chain is an independent run of the procedure using a different random initialization. Then, we combine the samples from the different chains. The computational requirements of running multiple chains are hence proportional to the number of chains.

We report the Bayesian model average performance as a function of the number of chains in Figure 4.1(b). Holding compute budget fixed, using two or three chains is only slightly better than using one chain. This result notably shows that HMC is relatively unobstructed by energy barriers in the posterior surface that would otherwise require multiple chains to overcome. We explore this result further in Section 4.3.

## 4.3 How well does HMC mix?

The primary goal of our work is to construct accurate samples from the posterior, and use them to understand the properties of Bayesian neural networks better. In this section we consider several diagnostics to evaluate whether our HMC sampler has converged, and discuss their implications to the posterior geometry. We consider mixing in both *weight space* and *function space*. A distribution over weights $w$ combined with a neural network architecture $f(x, w)$ induces a distribution over functions $f(x)$. Ultimately, since we are using functions to make predictions, we care mostly about mixing in function space.

**Summary**: HMC is able to mix surprisingly well in function space, and better than in parameter space. Geometrically, HMC is able to explore connected basins of the posterior with high functional diversity.

### 4.3.1 $\hat{R}$ diagnostics

We apply the classic Gelman et al. [1992] "$\hat{R}$" potential-scale-reduction diagnostic to our HMC runs. Given two or more chains, $\hat{R}$ estimates the ratio between the between-chain variance (i.e., the variance estimated by pooling samples from all chains) and the average within-chain variance (i.e., the variances estimated from each chain independently). The intuition is that, if the chains are stuck in isolated regions, then combining samples from multiple chains will yield greater diversity than taking samples from a single chain. For the precise mathematical definition of $\hat{R}$, please see the Section A.5.2.

We compute $\hat{R}$ using TensorFlow Probability's implementation[4] [Lao et al. 2020] for both the weights and the test-set softmax predictions on CIFAR-10 with ResNet-20-FRN and on IMDB with

[4]tfp.mcmc.potential_scale_reduction

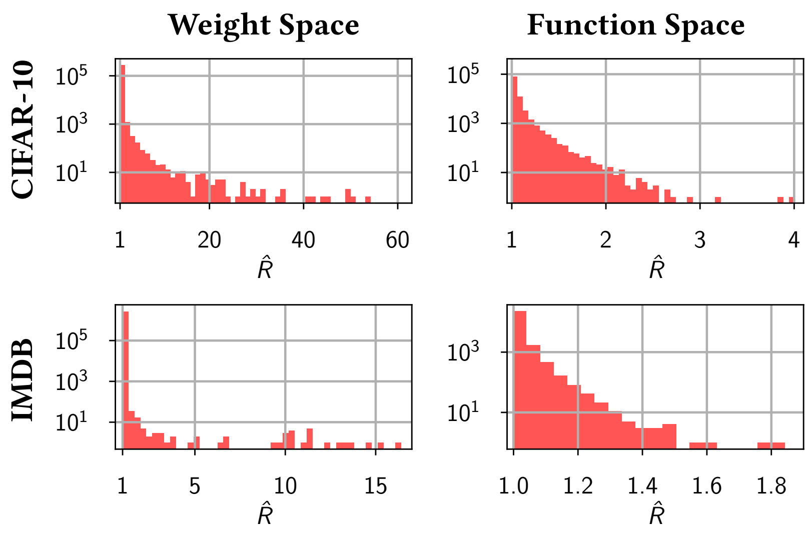
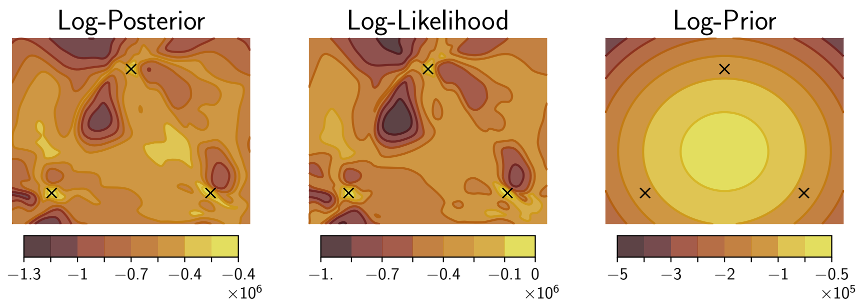


**Figure 4.2: Log-scale histograms of $\hat{R}$ convergence diagnostics.** Function-space $\hat{R}$s are computed on the test-set softmax predictions of the classifiers and weight-space $\hat{R}$s are computed on the raw weights. About 91% of CIFAR-10 and 98% of IMDB posterior-predictive probabilities get an $\hat{R}$ less than 1.1. Most weight-space $\hat{R}$ values are quite small, but enough parameters have very large $\hat{R}$s to make it clear that the chains are sampling from different distributions in weight space.

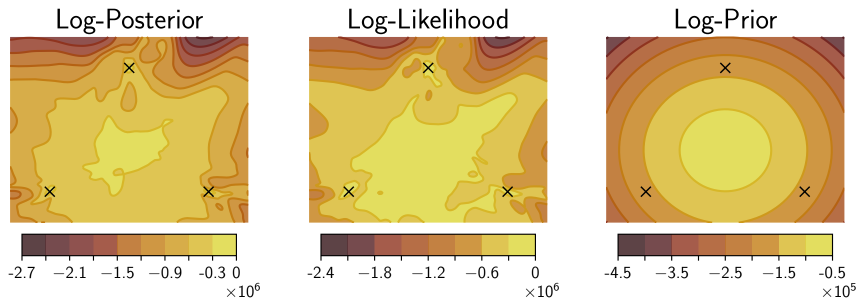
CNN-LSTM. We report the results in Figure 4.2. We observe that on both IMDB and CIFAR, the bulk of the function-space $\hat{R}$ values is concentrated near 1, meaning intuitively that a single chain can capture the diversity of predictions on most of the test data points nearly as well as multiple chains. The mixing is especially good on the IMDB dataset, where only 2% of inputs correspond to $\hat{R}$ larger than 1.1. In Section A.5.5 we apply HMC to a synthetic regression problem and show that HMC can indeed mix in the prediction space: different HMC chains provide very similar predictions.

In weight space, although most parameters show no evidence of poor mixing, some have very large $\hat{R}$s, indicating that there are directions in which the chains fail to mix.

Implications for the Posterior Geometry. The fact that a single HMC chain is able to mix well in function space (aka prediction space) suggests that the posterior contains connected regions which correspond to high functional diversity. Indeed, a single HMC chain is extremely unlikely to jump between isolated modes, but appears able to produce samples with diverse pre-



(a) Same chain



(b) Independent chains

**Figure 4.3: Posterior density visualization.** Visualizations of posterior log-density, log-likelihood and log-prior in the two-dimensional subspace of the parameter space spanned by three HMC samples from **(a)** the same chain and **(b)** three independent chains. Each HMC chain explores a region of high posterior density of a complex non-convex shape, that appears multi-modal in the presented cross-sections.

dictions. Moreover, HMC is able to navigate these regions effeciently. Prior work on *mode connectivity* [Garipov et al. 2018; Draxler et al. 2018b] has shown that there exist paths of high density connecting different modes of the posterior. Our observations suggest a stronger version of mode connectivity: not only do mode-connecting paths exist between functionally diverse modes, but also at least some of these paths can be leveraged by Monte Carlo methods to efficiently explore the posterior.

### 4.3.2 Posterior density visualizations

To further investigate how HMC is able to explore the posterior over the weights, we visualize a cross-section of the posterior density in subspaces of the parameter space containing the samples. Following Garipov et al. [2018], we study two-dimensional subspaces of the parameter space of the form

$$\mathcal{S} = \{w | w = w_1 \cdot a + w_2 \cdot b + w_3 \cdot (1 - a - b)\}. \tag{4.3}$$

$\mathcal{S}$ is the unique two-dimensional affine subspace (plane) of the parameter space that includes parameter vectors $w_1$, $w_2$ and $w_3$.

In Figure 4.3(a) we visualize the posterior log-density, log-likelihood and log-prior density of a ResNet-20-FRN on CIFAR-10. For the visualization, we use the subspace $\mathcal{S}$ defined by the parameter vectors $w_1$, $w_{51}$ and $w_{101}$, the samples produced by HMC at iterations 1, 51 and 101 after burn-in respectively. We observe that HMC is able to navigate complex geometry: the samples fall in three seemingly isolated modes in our two-dimensional cross-section of the posterior. In other words, HMC samples from a single chain are not restricted to any specific convex Gaussian-like mode, and instead explore a region of high posterior density of a complex shape in the parameter space. We note that popular approximate inference procedures, such as variational methods, and Laplace approximations, are typically constrained to unimodal Gaussian approximations to the posterior, which we indeed expect to miss a large space of compelling solutions in the posterior.

In Figure 4.3(b) we provide a visualization for the samples produced by 3 different HMC chains at iteration 51 after burn-in. Comparing the visualizations for samples from the same chain and samples from independent chains in Figure 4.3, we see that the shapes of the posterior surfaces are different, with the latter appearing more regular and symmetric. The qualitative differences between (a) and (b) suggest that while each HMC chain is able to navigate the posterior geometry the chains do not mix perfectly in the weight space, confirming our results in Section 4.3.1.

In Section A.5.4 we provide additional details on our visualization procedure and visualizations using the CNN-LSTM on IMDB.

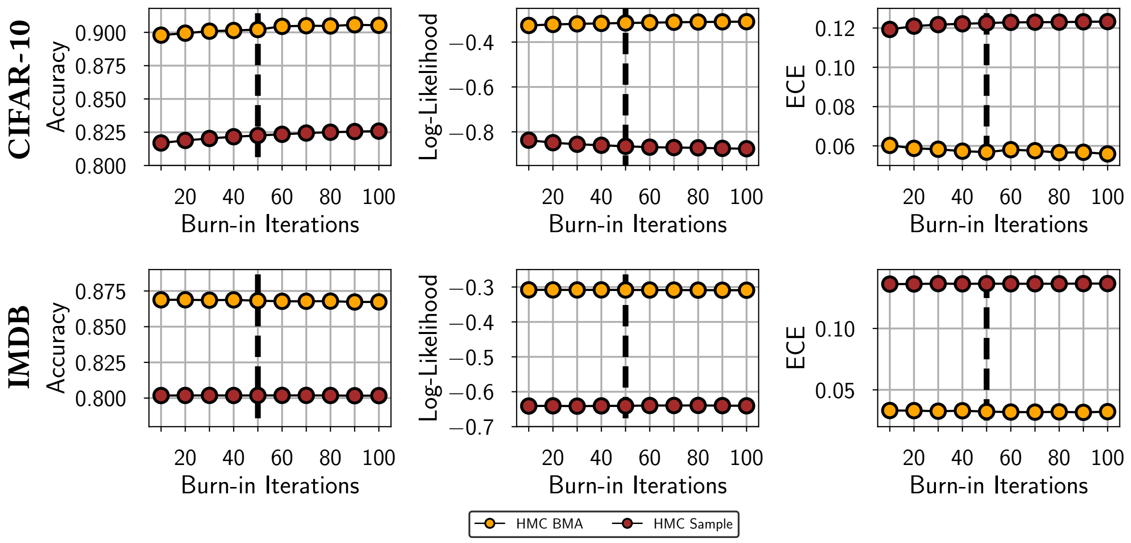


**Figure 4.4: HMC convergence.** The performance of an individual HMC sample and a BMA ensemble of 100 samples from each one of 3 HMC chains after the burn-in as a function of burn-in length. The dashed line indicates the burn-in length of 50 that we used in the main experiments in this work. We use ResNet-20-FRN on CIFAR-10 and CNN-LSTM on IMDB. On IMDB, there is no visible dependence of the results on the burn-in length; on CIFAR-10, there is a weak trend that slows down over time.

### 4.3.3 Convegence of the HMC chains

As another diagnostic, we look at the convergence of the performance of HMC BMA estimates and individual samples as a function of the length of the burn-in period. For a converged chain, the performance of the BMA and individual samples should be stationary not show any visible trends after a sufficiently long burn-in. We use the samples from 3 HMC chains, and evaluate performance of the ensemble of the first 100 HMC samples in each chain after discarding the first $n_{bi}$ samples, where $n_{bi}$ is the length of the burn-in. Additionally, we evaluate the performance of the individual HMC samples after $n_{bi}$ iterations in each of the chains.

We report the results for ResNet-20-FRN on CIFAR-10 and CNN-LSTM on IMDB in Figure 4.4. On IMDB, there is no visible trend in performance, so a burn-in of just 10 HMC iterations should be sufficient. On CIFAR-10, we observe a slowly rising trend that saturates at about 50 iterations, indicating that a longer burn-in period is needed compared to IMDB. We therefore use a burn-in period of 50 HMC iterations on both CIFAR and IMDB for the remainder of the chapter.

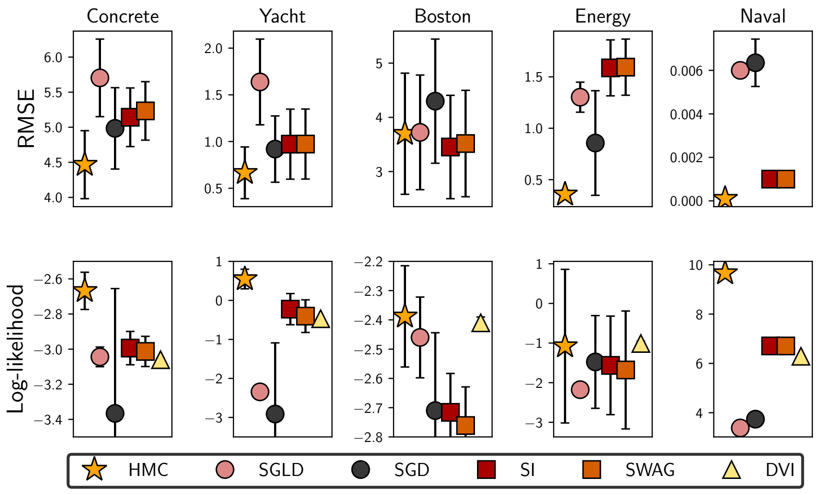


**Figure 4.5: UCI regression datasets.** Performance of Hamiltonian Monte Carlo (HMC), stochastic gradient Langevin dynamics (SGLD), stochastic gradient descent (SGD), subspace inference (SI) [Izmailov et al. 2019], SWAG [Maddox et al. 2019] and deterministic variational inference [DVI; Wu et al. 2018]. We use a fully-connected architecture with a single hidden layer of 50 neurons. The results reported for each method are mean and standard deviation computed over 20 random train-test splits of the dataset. For SI, SWAG and DVI we report the results presented in Izmailov et al. [2019]. **Top:** test root-mean-squared error. **Bottom:** test log-likelihood. HMC performs on par with or better than all other baselines in each experiment, often providing a significant improvement.

## 4.4 Evaluating Bayesian neural networks

Now that we have a procedure for effective HMC sampling, we are primed to explore exciting questions about the fundamental behaviour of Bayesian neural networks, such as the role of tempering, the prior over parameters, generalization performance, and robustness to covariate shift. In this section we evaluate Bayesian neural networks in various problems using our implementation of HMC. Throughout the experiments, we use posterior temperature $T = 1$.

We emphasize that the main goal of our work and this section in particular is to *understand* the behaviour of true Bayesian neural network posteriors using HMC as a precise tool, and *not* to argue for HMC as a practical method for Bayesian deep learning.

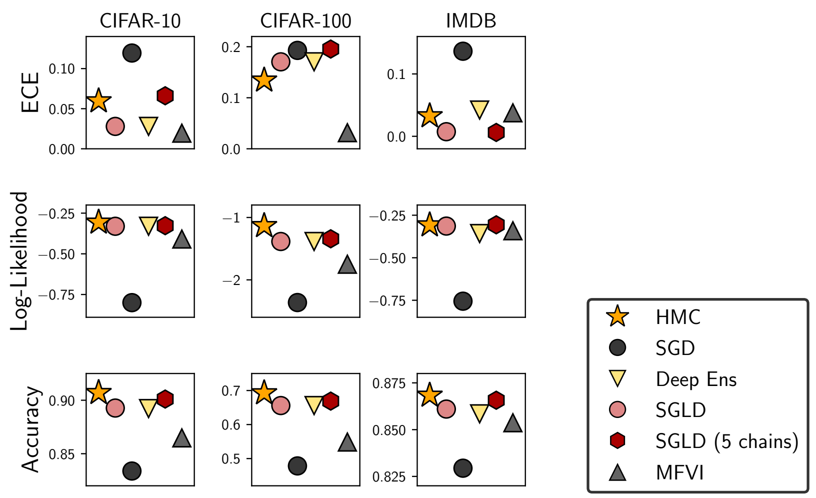


**Figure 4.6: Image and text classification.** Performance of Hamiltonian Monte Carlo (HMC), stochastic gradient Langevin dynamics (SGLD) with 1 and 5 chains, mean field variational inference (MFVI), stochastic gradient descent (SGD), and deep ensembles. We use ResNet-20-FRN on CIFAR datasets, and CNN-LSTM on IMDB. Bayesian neural networks via HMC outperform all baselines on all datasets in terms of accuracy and log-likelihood. On ECE, the methods perform comparably.

**Summary:** Bayesian neural networks achieve strong results outperforming even large deep ensembles in a range of evaluations. Surprisingly, however, BNNs are *less* robust to distribution shift than conventionally-trained models.

### 4.4.1 Regression on UCI datasets

Bayesian deep learning methods are often evaluated on small-scale regression problems using fully connected networks [e.g., Wu et al. 2018; Izmailov et al. 2019; Maddox et al. 2019]. Following these works, we evaluate Bayesian neural networks using HMC on five UCI regression datasets: *Concrete*, *Yacht*, *Boston*, *Energy* and *Naval*. For each of these datasets, we construct 20 random 90-to-10 train-test splits and report the mean and standard deviation of performance over the splits. We use a fully connected neural network with a single hidden layer of size 50 and 2 outputs representing the predictive mean and standard deviation. For HMC we used a single chain with 10 burn-in iterations and 90 iterations of sampling. For more details, please see Section A.5.1.

|  | AUC-ROC |  |  |  |
|---|---|---|---|---|
| OOD Dataset | HMC | DE | ODIN | Mahal. |
| CIFAR-100 | 0.857 | 0.853 | 0.858 | **0.882** |
| SVHN | 0.8814 | 0.8529 | 0.967 | **0.991** |

**Table 4.1: Out-of-distribution detection.** We use a ResNet-20-FRN model trained on CIFAR-10 to detect out-of-distribution data coming from SVHN or CIFAR-100. We report the results for HMC, deep ensembles and specialized ODIN [Liang et al. 2017] and Mahalanobis [Lee et al. 2018] methods. We report the AUC-ROC score (higher is better) evaluating the ability of each method to distinguish between in-distribution and OOD data. The predictive uncertainty from Bayesian neural networks allows us to detect OOD inputs: HMC outpefroms deep ensembles on both datasets. Furthermore, HMC is competitive with ODIN on the harder near-OOD task of detecting CIFAR-100 images, but underperforms on the easier far-OOD task of detecting SVHN images.

We report the results in Figure 4.5. HMC typically outperforms all the baselines, often by a significant margin, both in test RMSE and log-likelihood. On the *Boston* dataset, HMC achieves a slightly higher average RMSE compared to the subspace inference and SWAG [Izmailov et al. 2019; Maddox et al. 2019] but outperforms both these methods significantly in terms of log-likelihood.

### 4.4.2 Image Classification on CIFAR

Next, we evaluate Bayesian neural networks using HMC on image classification problems. We use the ResNet-20-FRN architecture on CIFAR-10 and CIFAR-100. We picked a random subset of 40960 of the 50000 images for each of the datasets to be able to evenly shard the data across the TPU devices; we use the same subset for both HMC and the baselines. We run 3 HMC chains using step size $10^{-5}$ and a prior variance of $1/5$, resulting in 70,248 leapfrog steps per sample. In each chain we discard the first 50 samples as burn-in, and then draw 240 samples (720 in total for 3 chains)[5]. For SGLD, we use a single chain with 1000 burn-in epochs and 9000 epochs of sampling producing 900 samples; we also report the performance of an ensemble of 5 independent SGLD chains. Next, we report the performance of a mean field variational inference (MFVI) solution;

[5]In total, on CIFAR-10 our HMC run requires as many computations as *over 60 million epochs* of standard SGD training

we initialize the mean of MFVI with a solution pre-trained with SGD and use an ensemble of 50 samples from the VI posterior at evaluation. Finally, we report the performance of a single SGD solution and a deep ensemble of 50 models. For more details, see Section A.5.1.

We report the results in Figure 4.6. Bayesian neural networks outperform all baselines in terms of accuracy and log-likelihood on both datasets. In terms of ECE, SGD provides the worst results across the board, and the rest of the methods are competitive; MFVI is particularly well-calibrated on CIFAR-100.

Out-of-distribution detection. Bayesian deep learning methods are often evaluated on out-of-distribution detection. In the Table 4.1 we report the performance of HMC-based Bayesian neural network on out-of-distribution (OOD) detection. To detect OOD data, we use the level of predicted confidence (value of the softmax class probability for the predicted class) from the HMC ensemble, measuring the area under the receiving operator characteristic curve (AUC-ROC). We train the methods on CIFAR-10 and use CIFAR-100 and SVHN as OOD data sources. We find that BNNs perform competitively with the specialized ODIN method in the challenging near-OOD detection setting (i.e. when the OOD data distribution is similar to the training data) of CIFAR-100, while underperforming in the easier far-OOD setting on SVHN relative to the baselines [Liang et al. 2017; Lee et al. 2018].

Robustness to distribution shift. Bayesian methods are often specifically applied to covariate shift problems [Ovadia et al. 2019; Wilson and Izmailov 2020; Dusenberry et al. 2020]. We evaluate the the performance of HMC and Deep Ensemble-based Bayesian neural networks on the CIFAR-10-C dataset [Hendrycks and Dietterich 2019], which applies a set of corruptions to CIFAR-10 with varying intensities. Mimicking the setup in Ovadia et al. [2019], we use the same 16 corruptions, evaluating the performance at all intensities. We report the results in Figure 4.7. Surprisingly, we find that Deep Ensembles and SGLD are consistently more robust to distribution shift than HMC-based BNNs. For high corruption intensities, even a single SGD model outper-

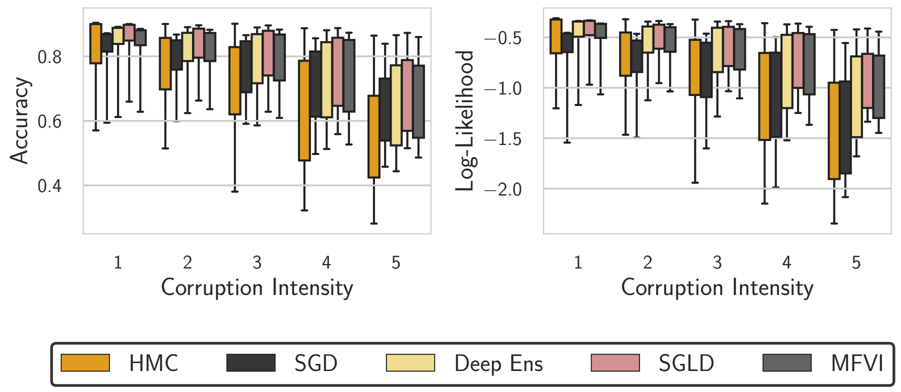


**Figure 4.7: Evaluation on CIFAR-10-C.** Accuracy and log-likelihood of HMC, SGD, deep ensembles, SGLD and MFVI on a distribution shift task, where the CIFAR-10 test set is corrupted in 16 different ways at various intensities on the scale of 1 to 5. We use the ResNet-20-FRN architecture. Boxes capture the quartiles of performance over each corruption, with the whiskers indicating the minimum and maximum. HMC is surprisingly the worst of the considered methods: even a single SGD solution provides better OOD robustness.

forms the HMC ensemble.

In Section A.5.6 we provide further exploration of this effect, where we see HMC samples are significantly less robust to many types of noise compared to conventionally-trained SGD models. Interestingly, the performance of HMC-based BNNs under data corruption can be significantly improved by using posterior tempering.

[$^{P}_{I}$ TODO!]

## 4.4.3 Language Classification on IMDB

We use a CNN-LSTM architecture on the IMDB binary text classification dataset. In Figure 4.6 we report the results for HMC and the same baselines as in Section 4.4.2. We use HMC with a step size of $10^{-5}$ and a prior variance of $1/40$, resulting in 24,836 leapfrog steps per sample. We run 3 chains, burning-in for 50 samples, and drawing 400 samples per chain (1,200 total). For more details on the hyperparameters, please see Section A.5.1. Analogously to the image classification experiments, HMC outperforms the baselines on accuracy and log-likelihood and provides competitive performance on ECE.

## 4.5 Bayesian Neural Networks under Covariate Shift

The predictive distributions of deep neural networks are often deployed in critical applications such as medical diagnosis [Gulshan et al. 2016; Esteva et al. 2017; Filos et al. 2019], and autonomous driving [Bojarski et al. 2016; Al-Shedivat et al. 2017; Michelmore et al. 2020]. These applications typically involve *covariate shift*, where the target data distribution is different from the distribution used for training [Hendrycks and Dietterich 2019; Arjovsky 2021]. Accurately reflecting uncertainty is crucial for robustness to these shifts [Ovadia et al. 2019; Roy et al. 2021]. Since Bayesian methods provide a principled approach to representing model (epistemic) uncertainty, they are commonly benchmarked on out-of-distribution (OOD) generalization tasks [e.g., Kendall and Gal 2017; Ovadia et al. 2019; Chang et al. 2019; Dusenberry et al. 2020; Wilson and Izmailov 2020].

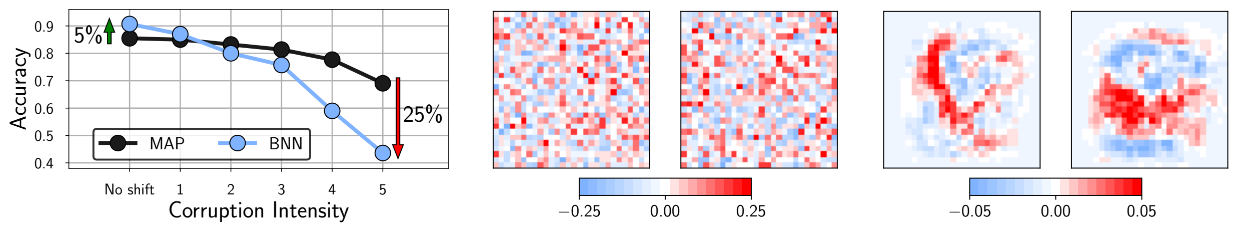
However, in Section 4.4, we showed that Bayesian neural networks (BNNs) with high fidelity inference through Hamiltonian Monte Carlo (HMC) provide shockingly poor OOD generalization performance, despite the popularity and success of approximate Bayesian inference in this setting [Gal and Ghahramani 2016; Lakshminarayanan et al. 2017; Ovadia et al. 2019; Maddox et al. 2019; Wilson and Izmailov 2020; Dusenberry et al. 2020; Benton et al. 2021].

In this section, we seek to understand, further demonstrate, and help remedy this concerning behaviour. We show that Bayesian neural networks perform poorly for different types of covariate shift, namely test data corruption, domain shift, and spurious correlations. In Figure 4.8(a) we see that a ResNet-20 BNN approximated with HMC underperforms a maximum a-posteriori (MAP) solution by 25% on the *pixelate*-corrupted CIFAR-10 test set. This result is particularly surprising given that on the in-distribution test data, the BNN outperforms the MAP solution by over 5%.

Intuitively, we find that Bayesian model averaging (BMA) can be problematic under covariate shift as follows. Due to linear dependencies in the features (inputs) of the training data distribution, model parameters corresponding to these dependencies do not affect the predictions on the training data. As an illustrative special case of this general setting, consider MNIST digits, which always have black corner pixels (*dead pixels*, with intensity zero). The corresponding first layer weights are always multiplied by zero and have no effect on the likelihood. Consequently, these weights are simply sampled from the prior. If at test time the corner pixels are not black, e.g., due to corruption, these pixel values will be multiplied by random weights sampled from the prior, and propagated to the next layer, significantly degrading performance. On the other hand, classical MAP training drives the unrestricted parameters towards zero due to regularization from the prior that penalizes the parameter norm, and will not be similarly affected by noise at test time. Here we see a major difference in robustness between optimizing a posterior for MAP training in comparison to a posterior weighted model average.

As a motivating example, in Figure 4.8(b, c) we visualize the weights in the first layer of a fully-connected network for a sample from the BNN posterior and the MAP solution on the MNIST dataset. The MAP solution weights are highly structured, while the BNN sample appears extremely noisy, similar to a draw from the Gaussian prior. In particular the weights corresponding to dead pixels (i.e. pixel positions that are black for all the MNIST images) near the boundary of the input image are set near zero (shown in white) by the MAP solution, but sampled randomly by the BNN. If at test time the data is corrupted, e.g. by Gaussian noise, and the pixels near the boundary of the image are activated, the MAP solution will ignore these pixels, while the predictions of the BNN will be significantly affected.

Dead pixels are a special case of our more general findings: we show that the dramatic lack of robustness for Bayesian neural networks is fundamentally caused by *any* linear dependencies in the data, combined with models that are non-linear in their parameters. Indeed, we consider a wide range of covariate shifts, including domain shifts. These robustness issues have the potential



(a) ResNet-20, CIFAR-10-C (b) BNN weights (c) MAP weights

**Figure 4.8: Bayesian neural networks under covariate shift**. **(a)**: Performance of a ResNet-20 on the *pixelate* corruption in CIFAR-10-C. For the highest degree of corruption, a Bayesian model average underperforms a MAP solution by 25% (44% against 69%) accuracy. See Izmailov et al. [2021b] for details. **(b)**: Visualization of the weights in the first layer of a Bayesian fully-connected network on MNIST sampled via HMC. **(c)**: The corresponding MAP weights. We visualize the weights connecting the input pixels to a neuron in the hidden layer as a $28 \times 28$ image, where each weight is shown in the location of the input pixel it interacts with.

to impact virtually *every* real-world application of Bayesian neural networks, since train and test rarely come from exactly the same distribution.

Based on our understanding, we introduce a novel prior that assigns a low variance to the weights in the first layer corresponding to directions orthogonal to the data manifold, leading to improved generalization under covariate shift. We additionally study the effect of non-zero mean corruptions and accordingly propose a second prior that constrains the sum of the weights, resulting in further improvements in OOD generalization for Bayesian neural networks.

Our code for this section is available here.

Covariate shift. In this section, we focus on the covariate shift setting. We assume the training dataset $\mathcal{D}_{\text{train}}$ consists of i.i.d. samples from the distribution $p_{\text{train}}(x, y) = p_{\text{train}}(x) \cdot p(y|x)$. However, the test data may come from a different distribution $p_{\text{test}}(x, y) = p_{\text{test}}(x) \cdot p(y|x)$. For concreteness, we assume the conditional distribution $p(y|x)$ remains unchanged, but the marginal distribution of the input features $p_{\text{test}}(x)$ differs from $p_{\text{train}}(x)$; we note that our results do not depend on this particular definition of covariate shift. Arjovsky [2021] provides a detailed discussion of covariate shift.

### 4.5.1 Bayesian neural networks are not robust to covariate shift

In this section, we evaluate Bayesian neural networks under different types of covariate shift. Specifically, we focus on two types of covariate shift: test data corruption and domain shift. In Section A.6.4, we additionally evaluate BNNs in the presence of spurious correlations in the data.

Methods. We evaluate BNNs against two deterministic baselines: a MAP solution approximated with stochastic gradient descent (SGD) with momentum [Robbins and Monro 1951; Polyak 1964] and a deep ensemble of 10 independently trained MAP solutions [Lakshminarayanan et al. 2017]. For BNNs, we provide the results using a Gaussian prior and a more heavy-tailed Laplace prior following Fortuin et al. [2021]. Izmailov et al. [2021b] conjectured that *cold posteriors* [Wenzel et al. 2020] can improve the robustness of BNNs under covariate shift; to test this hypothesis, we provide results for BNNs with a Gaussian prior and cold posteriors at temperature $10^{-2}$. For all BNN models, we run a single chain of HMC for 100 iterations discarding the first 10 iterations as burn-in, following Izmailov et al. [2021b]. We provide additional experimental details in Section A.6.1.

Datasets and data augmentation. We run all methods on the MNIST [LeCun 1998] and CIFAR-10 [Krizhevsky et al. 2014] datasets. Following Izmailov et al. [2021b] we do not use data augmentation with any of the methods, as it is not trivially compatible with the Bayesian neural network framework [e.g., Izmailov et al. 2021b; Wenzel et al. 2020].

Neural network architectures. On both the CIFAR-10 and MNIST datasets we use a small convolutional network (CNN) inspired by LeNet-5 [LeCun et al. 1998], with 2 convolutional layers followed by 3 fully-connected layers. On MNIST we additionally consider a fully-connected neural network (MLP) with 2 hidden layers of 256 neurons each. We note that high-fidelity posterior sampling with HMC is extremely computationally intensive. Even on the small architectures

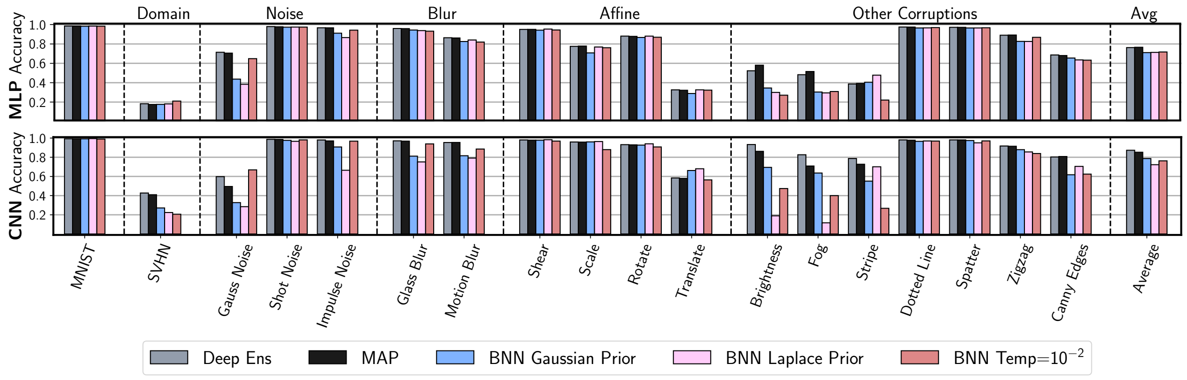


**Figure 4.9: Robustness on MNIST.** Accuracy for deep ensembles, MAP and Bayesian neural networks trained on MNIST under covariate shift. **Top**: Fully-connected network; **bottom**: Convolutional neural network. While on the original MNIST test set BNNs provide competitive performance, they underperform deep ensembles on most of the corruptions. With the CNN architecture, all BNN variants lose to MAP when evaluated on SVHN by almost 20%.

that we consider, the experiments take multiple hours on 8 NVIDIA Tesla V-100 GPUs or 8-core TPU-V3 devices [Jouppi et al. 2020]. See Izmailov et al. [2021b] for details on the computational requirements of full-batch HMC for BNNs.

#### 4.5.1.1 Test data corruption

We start by considering the scenario where the test data is corrupted by some type of noise. In this case, there is no semantic distribution shift: the test data is collected in the same way as the train data, but then corrupted by a generic transformation.

In Figure 4.9, we report the performance of the methods trained on MNIST and evaluated on the MNIST-corrupted (MNIST-C) test sets [Mu and Gilmer 2019] under various corruptions[6]. We report the results for a fully-connected network and a convolutional network.

With the CNN architecture, deep ensembles consistently outperform BNNs. Moreover, even a single MAP solution significantly outperforms BNNs on many of the corruptions. The results are especially striking on *brightness* and *fog* corruptions, where the BNN with Laplace prior shows

[6]In addition to the corruptions from MNIST-C, we consider Gaussian noise with standard deviation 3, as this corruption was considered by Izmailov et al. [2021b].

accuracy at the level of random guessing, while the deep ensemble retains accuracy above 90%. *Gaussian noise* and *impulse noise* corruptions also present significant challenges for BNNs. The BNNs show the most competitive performance in-distribution and on the corruptions representing affine transformations: *shear*, *scale*, *rotate* and *translate*. The results are generally analogous for the MLP: across the board the BNNs underperform deep ensembles, and often even a single MAP solution.

While the cold posteriors provide an improvement on some of the *noise* corruptions, they also hurt performance on *Brightness* and *Stripe*, and do not improve the performance significantly on average across all corruptions compared to a standard BNN with a Gaussian prior. We provide additional results on cold posterior performance in Section A.6.7.

Next, we consider the CIFAR-10-corrupted dataset (CIFAR-10-C) [Hendrycks and Dietterich 2019]. CIFAR-10-C consists of 18 transformations that are available at different levels of intensity (1 − 5). We report the results using corruption intensity 4 (results for other intensities are in Section A.6.3) for each of the transformations in Figure 4.10. Similarly to MNIST-C, BNNs outperform deep ensembles and MAP on in-distribution data, but underperform each over multiple corruptions. On CIFAR-10-C, BNNs are especially vulnerable to different types of *noise* (*Gaussian noise*, *shot noise*, *impulse noise*, *speckle noise*). For each of the *noise* corruptions, BNNs underperform even the classical MAP solution. The cold posteriors improve the performance on the *noise* corruptions, but only provide a marginal improvement across the board.

We note that Izmailov et al. [2021b] evaluated a ResNet-20 model on the same set of CIFAR-10-C corruptions. While they use a much larger architecture, the qualitative results for both architectures are similar: BNNs are the most vulnerable to *noise* and *blur* corruptions. We thus expect that our analysis is not specific to smaller architectures and will equally apply to deeper models.

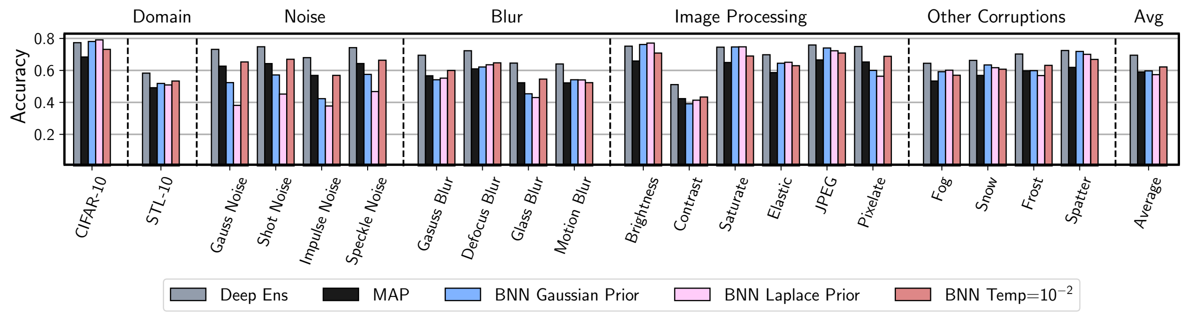


**Figure 4.10: Robustness on CIFAR-10.** Accuracy for deep ensembles, MAP and Bayesian neural networks using a CNN architecture trained on CIFAR-10 under covariate shift. For the corruptions from CIFAR-10-C, we report results for corruption intensity 4. While the BNNs with both Laplace and Gaussian priors outperform deep ensembles on the in-distribution accuracy, they underperform even a single MAP solution on most corruptions.

#### 4.5.1.2 Domain shift

Next, we consider a different type of covariate shift where the test data and train data come from different, but semantically related distributions.

First, we apply our CNN and MLP MNIST models to the SVHN test set [Netzer et al. 2011]. The MNIST-to-SVHN domain shift task is a common benchmark for unsupervised domain adaptation: both datasets contain images of digits, although MNIST contains hand-written digits while SVHN represents house numbers. In order to apply our MNIST models to SVHN, we crop the SVHN images and convert them to grayscale. We report the results in Figure 4.9. While for MLPs all methods perform similarly, with the CNN architecture BNNs underperform deep ensembles and MAP by nearly 20%.

Next, we apply our CIFAR-10 CNN model to the STL-10 dataset [Coates et al. 2011]. Both datasets contain natural images with 9 shared classes between the two datasets[7]. We report the accuracy of the CIFAR-10 models on these 9 shared classes in STL-10 in Figure 4.10. While BNNs outperform the MAP solution, they still significantly underperform deep ensembles.

The results presented in this section highlight the generality and practical importance of the

[7]CIFAR-10 has a class *frog* and STL-10 has *monkey*. The other nine classes coincide.

lack of robustness in BNNs: despite showing strong performance in-distribution, BNNs underperform even a single MAP solution (classical training) over an extensive variety of covariate shifts.

## 4.6 Understanding Bayesian neural networks under covariate shift

Now that we have established that Bayesian neural networks are highly susceptible to many types of covariate shift, we seek to understand why this is the case. In this section, we identify the linear dependencies in the input features as one of the key issues undermining the robustness of BNNs. We emphasize that *linear* dependencies in particular are *not* simply chosen for the simplicity of analysis, and their key role follows from the structure of the fully-connected and convolutional layers.

### 4.6.1 Motivating example: dead pixels and fully-connected layers

To provide an intuition for the results presented in this section, we start with a simple but practically relevant motivating example. Suppose we use a fully-connected Bayesian neural network on a dataset $D = \{(x_i, y_i)\}_{i=1}^n$, with $m$ input features, where $x_i \in \mathbb{R}^m$. We use the upper indices to denote the features $x^1, \ldots, x^m$ and the lower indices to denote the datapoints $x_i$. Then, for each neuron $j$ in the first hidden layer of the network, the activation can be written as $z_j = \phi(\sum_{i=1}^n x^i w_{ij}^1 + b_j^1)$, where $w_{ij}^1$ is the weight of the first layer of the network corresponding to the input feature $i$ and hidden neuron $j$, and $b_j^1$ is the corresponding bias. We show the following Lemma:

**Lemma 4.1.** *Using the notation introduced above, suppose that the input feature $x_k^i$ is equal to zero for all the examples $x_k$ in the training dataset $D$. Suppose the prior distribution over the parameters*

*$p(W)$ factorizes as $p(W) = p(w^1_{ij}) \cdot p(W \setminus w^1_{ij})$ for some neuron $j$ in the first layer, where $W \setminus w^1_{ij}$ represents all the parameters $W$ of the network except $w^1_{ij}$. Then, the posterior distribution $p(W|D)$ will also factorize and the marginal posterior over the parameter $w^1_{ij}$ will coincide with the prior:*

$$p(W|D) = p(W \setminus w^1_{ij}|D) \cdot p(w^1_{ij}). \tag{4.4}$$

*Consequently, the MAP solution will set the weight $w^1_{ij}$ to the value with maximum prior density.*

Intuitively, Lemma 4.1 says that if the prior for some parameter in the network is independent of the other parameters, and the value of the parameter does not affect the predictions of the model on any of the training data, then the posterior of this parameter will coincide with its prior. In particular, if one of the input features is always zero, then the corresponding weights will always be multiplied by zero, and will not affect the predictions of the network. To prove Lemma 4.1, we simply note that the posterior is proportional to the product of prior and likelihood, and both terms factorize with respect to $w^1_{ij}$. We present a formal proof in Section A.6.8.

So, for any sample from the posterior, the weight $w^1_{ij}$ will be a random draw from the prior distribution. Now suppose at test time we evaluate the model on data where the feature $x^i$ is no longer zero. Then for these new inputs, the model will be effectively multiplying the input feature $x^i$ by a random weight $w^1_{ij}$, leading to instability in predictions. In Section A.6.8, we formally state and prove the following proposition:

**Proposition 4.2.** *Suppose the assumptions of Lemma 4.1 hold. Assume also that the prior distribution $p(w^1_{ij})$ has maximum density at $0$ and that the network uses ReLU activations. Then for any test input $\bar{x}$, the expected prediction under Bayesian model averaging (Equation 4.1) will depend on the value of the feature $\bar{x}^i$, while the MAP solution will ignore this feature.*

For example, in the MNIST dataset there is a large number of *dead pixels*: pixels near the boundaries of the image that have intensity zero for all the inputs. In practice, we often use independent zero-mean priors (e.g. Gaussian) for each parameter of the network. So, according to Lemma 4.1, the posterior over all the weights in the first layer of the network corresponding to the dead pixels will coincide with the prior. If at test time the data is corrupted by e.g., Gaussian noise, the dead pixels will receive non-zero intensities, leading to a significant degradation in the performance of the Bayesian model average compared to the MAP solution.

While the situation where input features are constant and equal to zero may be rare and easily addressed, the results presented in this section can be generalized to *any* linear dependence in the data. We will now present our results in the most general form.

### 4.6.2 General linear dependencies and fully-connected layers

We now present our general results for fully-connected Bayesian neural networks when the features are linearly dependent. Intuitively, if there exists a direction in the input space such that all of the training data points have a constant projection on this direction (i.e. the data lies in a hyper-plane), then posterior coincides with the prior in this direction. Hence, the BMA predictions are highly susceptible to perturbations that move the test inputs in a direction orthogonal to the hyper-plane. The MAP solution on the other hand is completely robust to such perturbations. In Section A.6.8 we prove the following proposition.

**Proposition 4.3.** *Suppose that the prior over the weights $w^1_{ij}$ and biases $b^1_j$ in the first layer is an i.i.d. Gaussian distribution $\mathcal{N}(0, \alpha^2)$, independent of the other parameters in the model. Suppose all the inputs $x_1 \dots x_n$ in the training dataset $D$ lie in an affine subspace of the input space: $\sum_{j=1}^m x_i^j c_j = c_0$ for all $i = 1, \dots, n$ and some constants $c$ such that $\sum_{i=0}^m c_i^2 = 1$. Then,*

*(1) For any neuron $j$ in the first hidden layer, the posterior distribution of random variable $w^c_j = \sum_{i=1}^m c_i w^1_{ij} - c_0 b^1_j$ (the projection of parameter vector $(w^1_{1j}, \dots, w^1_{mj}, b^1_j)$ on direction $(c_1, \dots, c_m, -c_0)$) will coincide with the prior $\mathcal{N}(0, \alpha^2)$.*

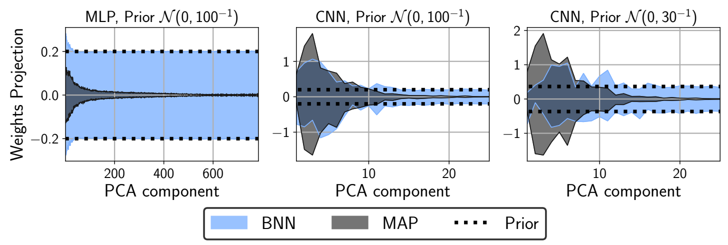
*(2) The MAP solution will set $w^c_j$ to zero.*

(3) *(Informal) Assuming the network uses ReLU activations, at test time, the BMA prediction will be susceptible to the inputs $\bar{x}$ that lie outside of the subspace, i.e. the predictive mean will depend on $\sum_{j=1}^{m} \bar{x}^j c_j - c_0$. The MAP prediction will not depend on this difference.*

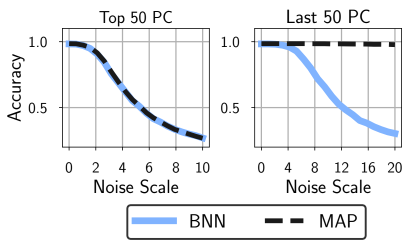
Empirical support. To test Proposition 4.3, we examine the performance of a fully-connected BNN on MNIST. The MNIST training dataset is not full rank, meaning that it has linearly dependent features. For a fully-connected BNN, Proposition 4.3 predicts that the posterior distribution of the first layer weights projected onto directions corresponding to these linearly dependent features will coincide with the prior. In Figure 4.11(a) we test this hypothesis by projecting first layer weights onto the principal components of the data. As expected, the distribution of the projections on low-variance PCA components (directions that are constant or nearly constant in the data) almost exactly coincides with the prior. The MAP solution, on the other hand, sets the weights along these PCA components close to zero, confirming conclusion (2) of the proposition. Finally, in Figure 4.11(b) we visualize the performance of the BMA and MAP solution as we apply noise along high-variance and low-variance directions in the data. As predicted by conclusion (3) of Proposition 4.3, the MAP solution is very robust to noise along the low-variance directions, while BMA is not.

### 4.6.3 Linear dependencies and convolutional layers

Finally, we can extend Proposition 4.3 to convolutional layers. Unlike fully-connected layers, convolutional layers include weight sharing such that no individual weight corresponds to a specific pixel in the input images. For example, dead pixels will not necessarily present an issue for convolutional layers, unlike what is described in Section 4.6.1. However, convolutional layers are still susceptible to a special type of linear dependence. Intuitively, we can think of the outputs of the first convolutional layer on all the input images as the outputs of a fully-connected layer applied to all the $K \times K$ patches of the input image. Therefore the reasoning in Proposition 4.3



(a) Weight projections



(b) Robustness along PCs

**Figure 4.11: Bayesian inference samples weights along low-variance principal components from the prior, while MAP sets these weights to zero. (a)**: The distribution (mean ± 2 std) of projections of the weights of the first layer on the directions corresponding to the PCA components of the data for BNN samples and MAP solution using MLP and CNN architectures with different prior scales. In each case, MAP sets the weights along low-variance components to zero, while BNN samples them from the prior. **(b)**: Accuracy of BNN and MAP solutions on the MNIST test set with Gaussian noise applied along the 50 highest and 50 lowest variance PCA components of the train data (left and right respectively). MAP is very robust to noise along low-variance PCA directions, while BMA is not; the two methods are similarly robust along the highest-variance PCA components.

applies to the convolutional layers, with the difference that the linear dependencies in the $K \times K$ patches cause the instability rather than dependencies in the full feature space. In Section A.6.8 we prove the following proposition.

**Proposition 4.4.** *Suppose that the prior over the parameters of the convolutional filters and biases in the first layer is an i.i.d. Gaussian distribution $\mathcal{N}(0, \alpha^2)$, independent of the other parameters in the model. Suppose that the convolutional filters in the first layer are of size $K \times K \times C$, where $C$ is the number of input channels. Then, consider the set $\hat{D}$ of size $N$ of all the patches of size $K \times K \times C$ extracted from the training images in $D$ after applying the same padding as in the first convolutional layer. Suppose all the patches $z_1 \dots z_N$ in the dataset $\hat{D}$ lie in an affine subspace of the space $\mathbb{R}^{K \times K \times C}$: $\sum_{c=1}^{C} \sum_{a=1}^{K} \sum_{b=1}^{K} z_i^{a,b} \gamma_{c,a,b} = \gamma_0$ for all $i = 1, \dots, N$ and some constants $c_i$ such that $\sum_{c=1}^{C} \sum_{a=1}^{K} \sum_{b=1}^{K} \gamma_{c,a,b}^2 + \gamma_0^2 = 1$. Then, we can prove results analogous to (1)-(3) in Proposition 4.3 (see the Section A.6.8 for the details).*

Empirical support. In Figure 4.11(a), we visualize the projections of the weights in the first layer of the CNN architecture on the PCA components of the $K \times K$ patches extracted from

MNIST. Analogously to fully-connected networks, the projections of the MAP weights are close to zero for low-variance components, while the projections of the BNN samples follow the prior.

### 4.6.4 What corruptions will hurt performance?

Based on Propositions 4.3, 4.4, we expect that the corruptions that are the most likely to break linear dependence structure in the data will hurt the performance the most. In Section A.6.9 we argue that *noise* corruptions are likely to break linear dependence, while the *affine* corruptions are more likely to preserve it, agreeing with our observations in Section 4.5.1.

### 4.6.5 Why do some approximate Bayesian inference methods work well under covariate shift?

Unlike BNNs with HMC inference, some approximate inference methods such as SWAG [Maddox et al. 2019], MC dropout [Gal and Ghahramani 2016], deep ensembles [Lakshminarayanan et al. 2017] and mean field VI [Blundell et al. 2015] provide strong performance under covariate shift [Ovadia et al. 2019; Izmailov et al. 2021b]. For deep ensembles, we can easily understand why: a deep ensemble represents an average of approximate MAP solutions, and we have seen in conclusion (3) of Proposition Theorem 4.3 that MAP is robust to covariate shift in the scenario introduced in Section 4.6.2. Similarly, other methods are closely connected to MAP via characterizing the posterior using MAP optimization iterates (SWAG), or modifying the training procedure for the MAP solution (MC Dropout). We provide further details, including theoretical and empirical analysis of variational inference under covariate shift, in Section A.6.10.

## 4.7 Towards more robust Bayesian model averaging under covariate shift

In this section, we propose a simple new prior inspired by our theoretical analysis. In Section 4.6 we showed that linear dependencies in the input features cause the posterior to coincide with the prior along the corresponding directions in the parameter space. In order to address this issue, we explicitly design the prior for the first layer of the network so that the variance is low along these directions.

### 4.7.1 Data empirical covariance prior

Let us consider the empirical covariance matrix of the inputs $x_i$. Assuming the input features are all preprocessed to be zero-mean $\sum_{i=1}^{n} x_i = 0$, we have $\Sigma = \frac{1}{n-1}\sum_{i=1}^{n} x_i x_i^T$. For fully-connected networks, we propose to use the *EmpCov* prior $p(w^1) = \mathcal{N}(0, \alpha\Sigma + \epsilon I)$ on the weights $w^1$ of the first layer of the network, where $\epsilon$ is a small positive constant ensuring that the covariance matrix is positive definite. The parameter $\alpha > 0$ determines the scale of the prior.

Suppose there is a linear dependence in the input features of the data: $x_i^T p = c$ for some direction $p$ and constant $c$. Then $p$ will be an eigenvector of the empirical covariance matrix with the corresponding eigenvalue equal to $0 : \Sigma p = \frac{1}{n-1}\sum_{i=1}^{n} x_i x_i^T p = \frac{c}{n-1}\sum_{i=1}^{n} x_i = 0$. Hence the prior over $w^1$ will have a variance of $\epsilon$ along the direction $p$.

More generally, the *EmpCov* prior is aligned with the principal components of the data, which are the eigenvectors of the matrix $\Sigma$. The prior variance along each principal component $p_i$ is equal to $\alpha\sigma_i^2 + \epsilon$ where $\sigma_i^2$ is its corresponding explained variance. In Section A.6.11, we discuss a more general family of priors aligned with the principal components of the data.

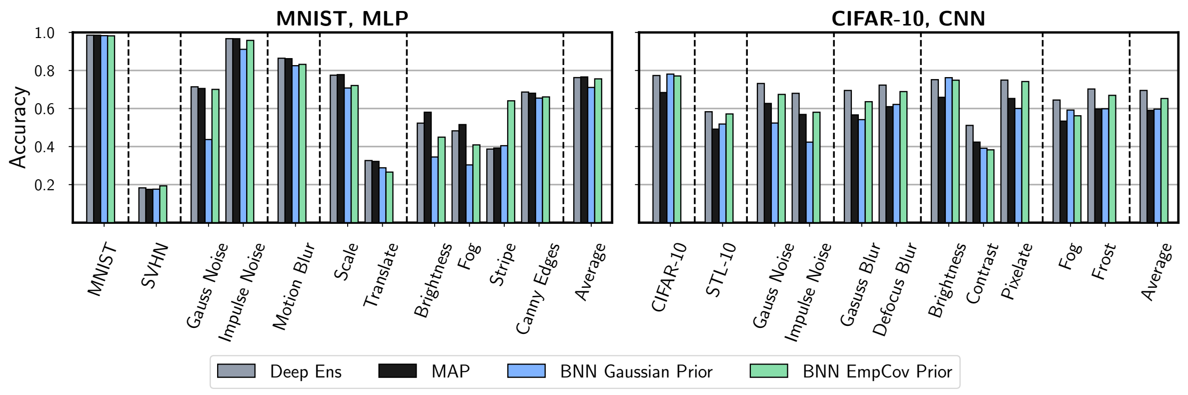


**Figure 4.12: EmpCov prior improves robustness.** Test accuracy under covariate shift for deep ensembles, MAP optimization with SGD, and BNN with Gaussian and *EmpCov* priors. **Left**: MLP architecture trained on MNIST. **Right**: CNN architecture trained on CIFAR-10. The *EmpCov* prior provides consistent improvement over the standard Gaussian prior. The improvement is particularly noticeable on the *noise* corruptions and domain shift experiments (SVHN, STL-10).

Generalization to convolutions. We can generalize the *EmpCov* prior to convolutions by replacing the empirical covariance of the data with the empirical covariance of the patches that interact with the convolutional filter, denoted by $\hat{D}$ in Proposition 4.4.

Is EmpCov a valid prior? *EmpCov* constructs a valid prior by evaluating the empirical covariance matrix of the inputs. This prior does not depend on the train data labels $y_i$, unlike the approach known as *Empirical Bayes* [see e.g. Bishop and Nasrabadi 2006, section 3.5], which is commonly used to specify hyperparameters in Gaussian process and neural network priors [Rasmussen and Williams 2006; MacKay 1995].

### 4.7.2 Experiments

In Figure 4.12 we report the performance of the BNNs using the *EmpCov* prior. In each case, we apply the *EmpCov* prior to the first layer, and a Gaussian prior to all other layers. For more details, please see Section A.6.1. On both MLP on MNIST and CNN on CIFAR-10, the *EmpCov* prior significantly improves performance across the board. In both cases, the BNN with *EmpCov* prior shows competitive performance with deep ensembles, especially in the domain shift experiments.

*EmpCov* is also particularly useful on the *noise* corruptions.

In Section A.6.12, we provide a detailed analysis of the performance of the CNN architecture on MNIST. Surprisingly, we found that using the *EmpCov* prior by itself does not provide a large improvement in this case. In Section A.6.12, we identify an issue specific to this particular setting, and propose another targeted prior that substantially improves performance.

## 4.8 Do we need cold posteriors?

Multiple works have considered tempering the posterior in Bayesian neural networks [e.g. Wenzel et al. 2020; Wilson and Izmailov 2020; Zhang et al. 2020c; Ashukha et al. 2020; Aitchison 2020]. See Section 3.10 for a detailed discussion of posterior tempering. Specifically, we can consider a distribution

$$p_T(w|\mathcal{D}) \propto \left(p(\mathcal{D}|w) \cdot p(w)\right)^{1/T}, \tag{4.5}$$

where $w$ are the parameters of the network, $\mathcal{D}$ is the training dataset, $p(\mathcal{D}|w)$ is the likelihood of $\mathcal{D}$ for the network with parameters $w$ and $T$ is the *temperature*. Note that at temperature $T = 1$, $p_T$ corresponds to the standard Bayesian posterior over the parameters of the network. Temperatures $T < 1$ correspond to *cold posteriors*, distributions that are sharper than the Bayesian posterior. Similarly, temperatures $T > 1$ correspond to *warm posteriors* which are softer than the Bayesian posterior. See Appendix Figure A.25(d) for a visualization of the log-likelihood density surface at different temperatures.

Wenzel et al. [2020] argue that Bayesian neural networks require a cold posterior, and the performance at temperature $T = 1$ is inferior to even a single model trained with SGD. The authors refer to this phenomenon as *the cold posteriors effect*. However, our results are different:

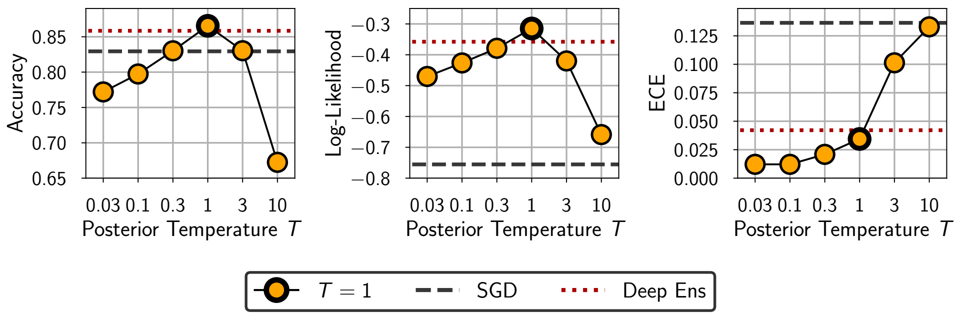


**Figure 4.13: Effect of posterior temperature.** The effect of posterior temperature $T$ on the log-likelihood, accuracy and expected calibration error using the CNN-LSTM model on the IMDB dataset. For both the log-likelihood and accuracy $T = 1$ provides optimal performance, while for the ECE the colder posteriors provide a slight improvement. For all three metrics, the posterior at $T = 1$ outperforms the SGD baseline as well as a deep ensemble of 10 independently trained models.

**Summary**: We show that that cold posteriors are not needed to obtain near-optimal performance with Bayesian neural networks and may even hurt performance. We show that the cold posterior effect is largely an artifact of data augmentation.

### 4.8.1 Testing the cold posteriors effect

Wenzel et al. [2020] demonstrate the cold posteriors with two main experiments: ResNet-20 on CIFAR-10 and CNN-LSTM on IMDB. In these experiments the authors show poor performance at temperature $T = 1$, with strong benefits from decreasing the temperature. However, for the CIFAR-10 experiment, it is apparent [Wenzel et al. 2020, Appendix K, Figure 28] that the results at $T = 1$ are near-optimal for the ResNet on CIFAR-10 if data augmentation is turned off and batch normalization is replaced with filter response normalization, which is in fact necessary for a clear Bayesian interpretation of the inference procedure.

Furthermore, in Section 4.4, we show that Bayesian neural networks can achieve performance superior to SGD and even deep ensembles at temperature $T = 1$, in particular using the same ResNet-20-FRN model on CIFAR-10 and CNN-LSTM model on IMDB used by Wenzel et al. [2020].

To further understand the effect of posterior temperature $T$, we compare the performance of the CNN-LSTM model at different $T$ using our Hamiltonian Monte Carlo sampler. In all runs we used a fixed prior variance $\alpha^2 = \frac{1}{40}$. We report the results in Figure 4.13. We find that the performance of the BNN at $T = 1$ is better than the SGD baseline as well as a deep ensemble of 50 independent models. Moreover, the performance at $T = 1$ is better compared to all other temperatures we tested in terms of both test accuracy and log-likelihood.

We also note that while posterior tempering does not seem necessary for good predictive performance with BNNs, it may be helpful under distribution shift. In Section A.5.6 we show that decreasing the temperature can significantly improve the robustness of BNN predictions to noise in the test inputs. Wilson and Izmailov [2020] additionally argue that tempering may be a reasonable procedure in general, and is not necessarily at odds with Bayesian principles.

Role of data augmentation. Our results are in contrast with Wenzel et al. [2020], who argue that cold posteriors are needed for good performance with BNNs. In Section A.5.7 we provide an additional study of what may have caused the poor performance of BNNs in Wenzel et al. [2020], using the code for inference provided by Wenzel et al. [2020]. We identify data augmentation as the key factor responsible for the cold posterior effect, and also show that batch normalizing does not significantly influence this effect: when the data augmentation is turned off, we do not observe the cold posteriors effect. Data augmentation cannot be naively incorporated in the Bayesian neural network model (see the discussion in appendix K of Wenzel et al. [2020]), and arguably it may be reasonable to decrease the temperature when using data augmentation: intuitively, data augmentation increases the amount of data observed by the model, and should lead to higher posterior contraction. We leave incorporating data augmentation in our HMC evaluation framework as an exciting direction of future work.

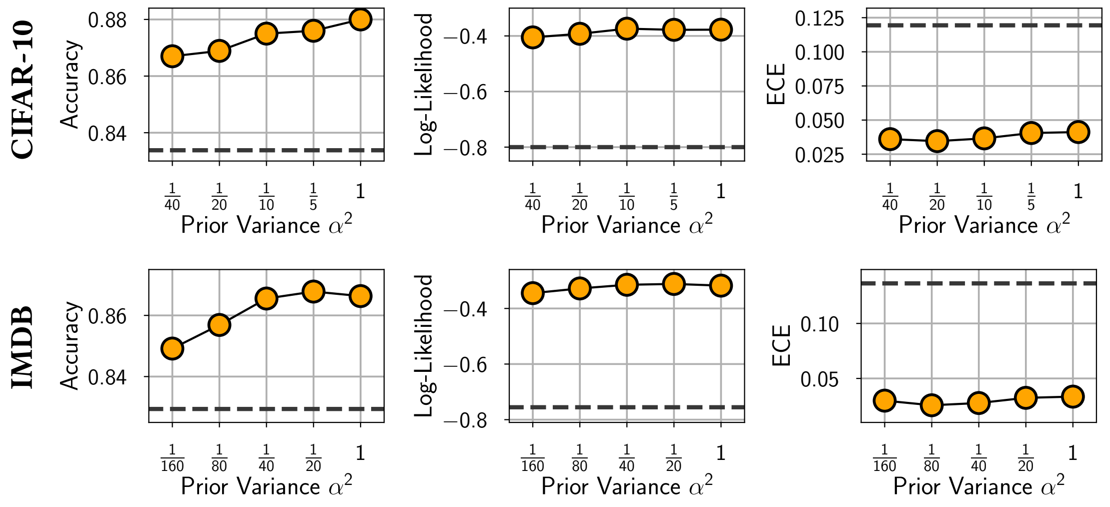


**Figure 4.14: Effect of prior variance.** The effect of prior variance on BNN performance. In each panel, the dashed line shows the performance of the SGD model from Section 4.4. While low prior variance may lead to over-regularization and hurt performance, all the considered prior scales lead to better results than the performance of an SGD-trained neural network of the same architecture.

## 4.9 What is the effect of priors in Bayesian neural networks?

Bayesian deep learning is often criticized for the lack of intuitive priors over the parameters. For example, Wenzel et al. [2020] hypothesize that the popular Gaussian priors of the form $\mathcal{N}(0, \alpha^2 I)$ are inadequate and lead to poor performance. Tran et al. [2020] propose a new prior for Bayesian neural networks inspired by Gaussian processes [Rasmussen and Williams 2006] based on this hypothesis. In concurrent work, Fortuin et al. [2021] also explore several alternatives to standard Gaussian priors inspired by the cold posteriors effect. Wilson and Izmailov [2020] on the other hand, argue that vague Gaussian priors in the parameter space induce useful function-space priors.

In Section 4.4 we have shown that Bayesian neural networks can achieve strong performance with vague Gaussian priors. In this section, we explore the sensitivity of BNNs to the choice of the prior scale as well as several alternative prior families, as a step towards a better understanding of the role of the prior in BNNs.

**Summary:** High-variance Gaussian priors over parameters of BNNs lead to strong performance. The results are robust with respect to the prior scale. Mixture of Gaussian and logistic priors over parameters are not too different in performance to Gaussian priors. These results highlight the relative importance of architecture over parameter priors in specifying a useful prior over functions.

### 4.9.1 Effect of Gaussian prior scale

We use priors of the form $\mathcal{N}(0, \alpha^2 I)$ and vary the prior variance $\alpha^2$. For all cases, we use a single HMC chain producing 40 samples. These are much shorter chains than the ones we used in Section 4.4, so the results are not as good; the purpose of this section is to explore the *relative* performance of BNNs under different priors.

We report the results for the CIFAR-10 and IMDB datasets in Figure 4.14. When the prior variance is too small, the regularization is too strong, hindering the performance. Setting the prior variance too large does not seem to hurt the performance as much. On both problems, the performance is fairly robust: a wide window of prior variances lead to strong performance. In particular, for all considered prior scales, the results are better than those of SGD training.

Why are BNNs so robust to the prior scale? One possible explanation for the relatively flat curves in Figure 4.14 is that large prior variances imply a strong prior belief that the "true" classifier (i.e., the model that would be learned given infinite data) should make high-confidence predictions. Since the model is powerful enough to achieve any desired training accuracy, the likelihood does not overrule this prior belief, and so the posterior assigns most of its mass to very confident classifiers. Past a certain point, increasing the prior variance on the weights may have no effect on the classifiers' already saturated probabilities. Consequently, nearly every member

| Prior | Gaussian | MoG | Logistic |
|---|---|---|---|
| Accuracy | 0.866 | 0.863 | **0.869** |
| ECE | 0.029 | 0.025 | **0.024** |
| Log Likelihood | -0.311 | -0.317 | **-0.304** |

**Table 4.2: Non-Gaussian priors.** BMA accuracy, ECE, and log-likelihood under different prior families using CNN-LSTM on IMDB. We produce 80 samples from a single HMC chain for each of the priors. The heavier-tailed logistic prior provides slightly better performance compared to the Gaussian and mixture of Gaussians (MoG) priors.

of the BMA may be highly overconfident. But the *ensemble* does not have to be overconfident—a mixture of overconfident experts can still make well-calibrated predictions. Appendix Figure A.29 provides some qualitative evidence for this explanation; for some CIFAR-10 test set images, the HMC chain oscillates between assigning the true label probabilities near 1 and probabilities near 0.

### 4.9.2 Non-Gaussian priors

In Table 4.2, we report BMA accuracy, ECE, and log-likelihood for two non-Gaussian priors on the IMDB dataset: *logistic* and *mixture of Gaussians* (MoG). For the MoG prior we use a mixture of two Gaussians centered at 0, one with variance $\frac{1}{40}$ and the other with variance $\frac{1}{160}$. We pick prior scale of the logistic prior to have a variance of $\frac{1}{40}$. We additionally provide the results for a Gaussian prior with variance $\frac{1}{40}$. We approximate the BMA using 80 samples from a single HMC chain for each of the priors. We find that the heavier-tailed logistic prior performs slightly better than the Gaussian and MoG.

### 4.9.3 Importance of Architecture in Prior Specification

We often think of the prior narrowly in terms of a distribution over parameters $p(w)$. But the prior that matters is the prior over functions $p(f(x))$ that is induced when a prior over parameters $p(w)$ is combined with the functional form of a neural network $f(x, w)$. All of the results in this section point to the relative importance of the architecture in defining the prior over functions, compared to the prior over parameters. A vague prior over parameters is not necessarily vague in function-space. Moreover, while the details of the prior distribution over parameters $p(w)$ have only a minor effect on performance, the choice of architecture certainly has a major effect on performance.

## 4.10 Do scalable BDL methods and HMC make similar predictions?

While HMC shows strong performance in our evaluation in Section 4.4, in most realistic BNN settings it is an impractical method. However, HMC can be used as a *reference* to evaluate and calibrate more scalable and practical alternatives. In this section, we evaluate the *fidelity* of SGMCMC, variational methods, and deep ensembles in representing the predictive distribution (Bayesian model average) given by our HMC reference.

**Summary**: While SGMCMC and Deep Ensembles can provide good generalization accuracy and calibration, their predictive distributions differ from HMC. Deep ensembles are similarly close to the HMC predictive distribution as SGLD, and closer than standard variational inference.

### 4.10.1 Comparing the predictive distributions

We consider two primary metrics: *agreement* and *total variation*. We define the agreement between the predictive distributions $\hat{p}$ of HMC and $p$ of another method as the fraction of the test

| Metric | HMC (reference) | SGD | Deep Ens | MFVI | SGMCMC |  |  |  |
|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  | SGLD | SGHMC | SGHMC CLR | SGHMC CLR-Prec |
| **CIFAR-10** |  |  |  |  |  |  |  |  |
| Accuracy | 89.64 ±0.25 | 83.44 ±1.14 | 88.49 ±0.10 | 86.45 ±0.27 | 89.32 ±0.23 | 89.38 ±0.32 | **89.63 ±0.37** | 87.46 ±0.21 |
| Agreement | 94.01 ±0.25 | 85.48 ±1.00 | 91.52 ±0.06 | 88.75 ±0.24 | 91.54 ±0.15 | 91.98 ±0.35 | **92.67 ±0.52** | 90.96 ±0.24 |
| Total Var | 0.074 ±0.003 | 0.190 ±0.005 | 0.115 ±0.000 | 0.136 ±0.000 | 0.110 ±0.001 | 0.109 ±0.001 | **0.099 ±0.006** | 0.111 ±0.002 |
| **CIFAR-10-C** |  |  |  |  |  |  |  |  |
| Accuracy | 70.91 ±0.93 | 71.04 ±1.80 | 76.99 ±0.39 | 75.40 ±0.34 | **78.80 ±0.17** | 78.20 ±0.25 | 76.43 ±0.39 | 73.42 ±0.39 |
| Agreement | 86.00 ±0.44 | 72.01 ±0.82 | 79.29 ±0.18 | 75.47 ±0.27 | 77.99 ±0.22 | 78.98 ±0.22 | **80.93 ±0.73** | 79.65 ±0.35 |
| Total Var | 0.133 ±0.004 | 0.334 ±0.007 | 0.220 ±0.003 | 0.245 ±0.002 | 0.214 ±0.002 | 0.203 ±0.002 | **0.194 ±0.010** | 0.205 ±0.005 |

**Table 4.3: Evaluation of cheaper alternatives to HMC.** Agreement and total variation between predictive distributions of HMC and approximate inference methods: deep ensembles, mean field variational inference (MFVI), and stochastic gradient Monte Carlo (SGMCMC) variations. For all methods we use ResNet-20-FRN trained on CIFAR-10 and evaluate predictions on the CIFAR-10 and CIFAR-10-C test sets. For CIFAR-10-C we report the average results across all corruptions and corruption intensities. We additionally report the results for HMC for reference: we compute the agreement and total variation between one of the chains and the ensemble of the other two chains. For each method we report the mean and standard deviation of the results over three independent runs. MFVI provides the worst approximation of the predictive distribution. Deep ensembles despite often being considered non-Bayesian, significantly outperform MFVI. SG-MCMC methods provide the best results with SGHMC-CLR showing the best overall performance.

data points for which the top-1 predictions of $\hat{p}$ and $p$ are the same:

$$\frac{1}{n}\sum_{i=1}^{n} I[\arg\max_j \hat{p}(y=j|x_i) = \arg\max_j p(y=j|x_i)],$$

where $I[\cdot]$ is the indicator function and $n$ is the number of test data points $x_i$. We define the total variation metric between $\hat{p}$ and $p$ as the total variation distance between the predictive distributions averaged over the test data points:

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\sum_{j}\left|\hat{p}(y=j|x_i)-p(y=j|x_i)\right|.$$

The agreement (higher is better) captures how well a method is able to capture the top-1 predictions of HMC, while the total variation (lower is better) compares the predictive probabilities for each of the classes.

In Table 4.3 we report the agreement and total variation metrics as well as the predictive accuracy on the CIFAR-10 and CIFAR-10-C test sets for a deep ensemble of 50 models and several SGLD variations: standard SGLD [Welling and Teh 2011], SGLD with momentum (SGHMC) [Chen et al. 2014], SGLD with momentum and a cyclical learning rate schedule (SGHMC-CLR) [Zhang et al. 2020c] and SGLD with momentum, cyclical learning rate schedule and a preconditioner (SGHMC-CLR-Prec) [Wenzel et al. 2020]. All methods were trained on CIFAR-10. For more details, please see Section A.5.1.

Overall, the absolute value of agreement achieved by all methods is fairly low on CIFAR-10 and especially on CIFAR-10-C. More advanced SGHMC-CLR and SGHMC-CLR-Prec methods provide a better fit of the HMC predictive distribution while not necessarily improving the accuracy. Notably, these methods are also *less* robust to the data corruptions in CIFAR-10-C, again suggesting that higher fidelity representations of the predictive distribution can lead to decreased robustness to covariate shift, as we found in section 4.4.2.

Deep ensembles, while not typically considered to be a Bayesian method, provide a reasonable approximation to the HMC predictive distribution. In particular, deep ensembles outperform both SGLD and SGHMC in terms of total variation on CIFAR-10 and in terms of agreement on CIFAR-10-C. These results support the argument that deep ensembles, while not typically characterized as a Bayesian method, provide a *higher fidelity* approximation to a Bayesian model average than methods that are conventionally accepted as Bayesian inference procedures in modern deep

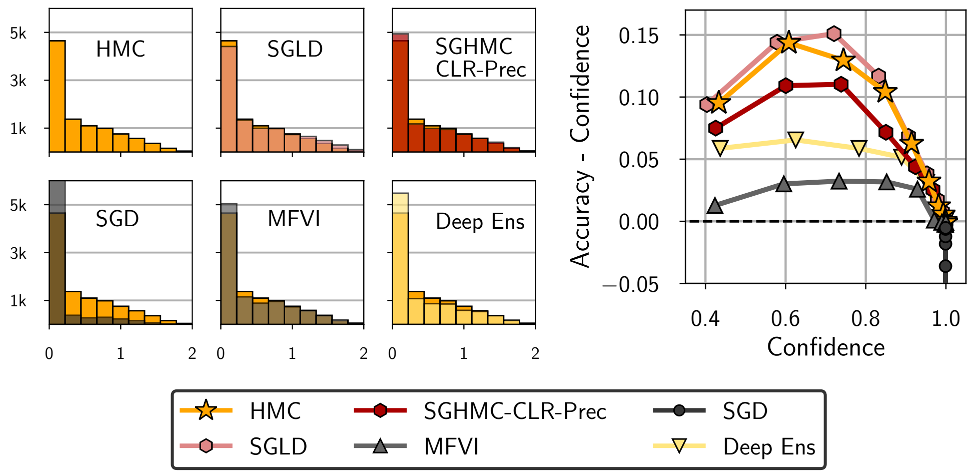


**Figure 4.15:** Distribution of predictive entropies (**left**) and calibration curve (**right**) of posterior predictive distributions for HMC, SGD, deep ensembles, MFVI, SGLD and SGHMC-CLR-Prec for ResNet20-FRN on CIFAR-10. On the left, for all methods, except HMC we plot a pair of histograms: for HMC and for the corresponding method. SGD, Deep ensembles and MFVI provide more confident predictions than HMC. SGMCMC methods appear to fit the predictive distribution of HMC better: SGLD is slightly underconfident relative to HMC while SGHMC-CLR-Prec is slightly over-confident.

learning [see Section 3.2.3; Wilson and Izmailov 2020].

In Section A.5.6 we explore the performance of HMC, SGD, deep ensembles, SGLD and SGHMC-CLR-Prec under different corruptions individually. Interestingly, the behavior of SGLD and SGHMC-CLR-Prec appears more similar to that of deep ensembles than that of HMC. So, while both SGMCMC and deep ensembles are very compelling practically, they provide relatively distinct predictive distributions from HMC. Mean-field variational inference methods are particularly far from the HMC predictive distribution. Thus, we should be very careful when making judgements about *true* Bayesian neural networks based on the SGMCMC or MFVI performance.

## 4.10.2 Predictive entropy and calibration curves

To provide an additional comparison of the predictive distributions between HMC and other methods, in Figure 4.15 we visualize the distribution of predictive entropies and the calibration curves for HMC, SGD, deep ensembles, MFVI, SGLD and SGHMC-CLR-Prec on CIFAR-10 using

ResNet-20-FRN.

All methods except fot SGD make conservative predictions: their confidences tend to underestimate their accuracies (Figure 4.15, right); SGD on the other hand is very over-confident [in agreement with the results in Guo et al. 2017]. Deep ensembles and MFVI provide the most calibrated predictions, while SGLD and SGHMC-CLR-Prec match the HMC entropy distribution and calibration curve closer.

## 4.11 Related work

The bulk of work on Bayesian deep learning has focused on scalable approximate inference methods. These methods include stochastic variational inference [Hoffman et al. 2013; Graves 2011; Blundell et al. 2015; Kingma et al. 2015; Molchanov et al. 2017; Louizos and Welling 2017; Khan et al. 2018; Zhang et al. 2018; Wu et al. 2018; Osawa et al. 2019; Dusenberry et al. 2020], dropout [Srivastava et al. 2014; Gal and Ghahramani 2016; Kendall and Gal 2017; Gal et al. 2017], the Laplace approximation [MacKay 1992b; Kirkpatrick et al. 2017; Ritter et al. 2018b; Li 2000; Daxberger et al. 2020], expectation propagation [Hernández-Lobato and Adams 2015], and leveraging the stochastic gradient descent (SGD) trajectory, either for a deterministic approximation, or sampling as in SGLD [Mandt et al. 2017b; Maddox et al. 2019; Izmailov et al. 2018; Wilson and Izmailov 2020]. Foong et al. [2019] and Farquhar et al. [2020] additionally consider the role of expressive posterior approximations in variational inference. See Section 3.11 for a more detailed discussion of the prior work on approximate inference in Bayesian deep learning.

While these (and many other) methods often provide improved predictions or uncertainty estimates, to the best of our knowledge *none* of these methods have been directly evaluated on their ability to match the true posterior distribution using practical architectures and datasets. Moreover, many of these methods are often designed with train-time constraints in mind, to take roughly the same amount of compute as regular SGD training. To evaluate approximate inference procedures, and explore fundamental questions in Bayesian deep learning, we attempt to construct a posterior approximation of the highest possible quality, ignoring the practicality of the method.

The Monte Carlo literature for Bayesian neural networks has mainly focused on stochastic gradient-based methods [Welling and Teh 2011; Ahn et al. 2014; Chen et al. 2014; Ma et al. 2015; Ahn et al. 2012; Ding et al. 2014; Zhang et al. 2020c; Garriga-Alonso and Fortuin 2021] for computational efficiency reasons. These methods are fundamentally biased: (1) they omit the Metropolis-Hastings correction step, and (2) the noise from subsampling the data perturbs their stationary distribution. In particular, Betancourt [2015] argues that HMC is incompatible with data subsampling. Notably, Zhang et al. [2020b] recently proposed a second-order stochastic gradient MCMC method that is asymptotically exact.

Since the classic work of Neal [1996], there have been a few recent attempts at using full-batch HMC in BNNs [e.g.; Cobb and Jalaian 2020; Wenzel et al. 2020]. These studies tend to use relatively short trajectory lengths (generally not considering a number of leapfrog steps greater than 100), and tend to focus on relatively small datasets and networks. We on the other hand experiment with practical architectures and datasets and use up to $10^5$ leapfrog steps per iteration to ensure good mixing.

Our work is aimed at *understanding* the properties of true Bayesian neural networks. In a similar direction, Wenzel et al. [2020] have recently explored the effect of the posterior temperature in Bayesian neural networks. We discuss their results in detail in Section 4.8, and provide our own exploration of the posterior temperature with a different result: we find that BNNs achieve strong performance at temperature 1 and do not require posterior tempering.

### 4.11.1 Covariate shift

Methods to improve robustness to shift between train and test often explicitly make use of the test distribution in some fashion. For example, it is common to apply semi-supervised methods to the labelled training data augmented by the unlabelled test inputs [e.g., Daume III and Marcu 2006; Athiwaratkun et al. 2019], or to learn a shared feature transformation for both train and test [e.g., Daumé III 2009]. In a Bayesian setting, Storkey and Sugiyama [2007] and Storkey [2009] propose such approaches for linear regression and Gaussian processes under covariate shift, assuming the data comes from multiple sources. Moreover, Shimodaira [2000] propose to re-weight the train data points according to their density in the test data distribution. Sugiyama et al. [2006, 2007] adapt this importance-weighting approach to the cross-validation setting.

We focus on the setting of robustness to covariate shift without any access to the test distribution [e.g., Daume III and Marcu 2006]. Bayesian methods are frequently applied in this setting, often motivated by the ability for a Bayesian model average to provide a principled representation of epistemic uncertainty: there are typically many consistent explanations for out of distribution points, leading to high uncertainty for these points. Indeed, approximate inference approaches for Bayesian neural networks are showing good and increasingly better performance under covariate shift [e.g., Gal and Ghahramani 2016; Lakshminarayanan et al. 2017; Ovadia et al. 2019; Maddox et al. 2019; Wilson and Izmailov 2020; Dusenberry et al. 2020; Benton et al. 2021].

Many works attempt to understand robustness to covariate shift. For example, for classical training Neyshabur et al. [2020] show that models relying on features that encode semantic structure in the data are more robust to covariate shift. Nagarajan et al. [2020] also provide insights into how classically trained max-margin classifiers can fail under covariate shift due to a reliance on spurious correlations between class labels and input features. BNN robustness to adversarial attacks is a related area of study, but generally involves much smaller perturbations to the test covariates than covariate shift. Carbone et al. [2020] prove that BNNs are robust to gradient-based adversarial attacks in the large data, overparameterized limit, while Wicker et al. [2021] present a framework for training BNNs with guaranteed robustness to adversarial examples.

In this work, we propose novel priors that improve robustness of BNNs under covariate shift. In Fortuin et al. [2021], the authors explore a wide range of priors and, in particular, show that the distribution of the weights of SGD-trained networks is heavy-tailed such as Laplace and Student-$t$, and does not appear Gaussian. Other heavy-tailed sparsity inducing priors have also been proposed in the literature [Carvalho et al. 2009; Molchanov et al. 2017; Kessler et al. 2019; Cui et al. 2020; Fortuin 2021]. Inspired by this work, we evaluate Laplace and Student-$t$ priors for BNNs, but find that they do not address the poor performance of BNNs under covariate shift.

Domingos [2000], Minka [2000b], Masegosa [2019] and Morningstar et al. [2020] explore failure modes of Bayesian model averaging when the Bayesian model does not contain a reasonable solution in its hypothesis space, causing issues when the posterior contracts. This situation is orthogonal to the setting in our paper, where we know the Bayesian model does contain a reasonable solution in its hypothesis space, since the MAP estimate is robust to covariate shift. In our setting, robustness issues are caused by a *lack* of posterior contraction.

In general, understanding and addressing covariate shift is a large area of study. For a comprehensive overview, see Arjovsky [2021]. To our knowledge, no prior work has attempted to understand, further demonstrate, or remedy the poor robustness of Bayesian neural networks with high fidelity approximate inference recently discovered in Izmailov et al. [2021b].

## 4.12 Discussion

Despite the rapidly increasing popularity of approximate Bayesian inference in modern deep learning, little is known about the behaviour of truly Bayesian neural networks. To the best of our knowledge, our work provides the first realistic evaluation of Bayesian neural networks with precise and exhaustive posterior sampling. We establish several properties of Bayesian neural networks, including good generalization performance, lack of a cold posterior effect, and a surprising lack of robustness to covariate shift. We hope that our observations and the tools that we develop will facilitate fundamental progress in understanding the behaviour of Bayesian neural networks.

Is HMC Converging? In general, it is not possible to ensure that an MCMC method has converged to sampling from the true posterior distribution: theoretically, there may always remain regions of the posterior that cannot be discovered by the method but that contain most of the posterior mass. To maximize the performance of HMC, we choose the hyper-parameters that are the most likely to provide convergence: long trajectory lengths, and multiple long chains. In Section 4.3, we study the convergence of HMC using the available convergence diagnostics. We find that while HMC does not mix perfectly in weight space, in the space of predictions we cannot find evidence of non-mixing.

Should we use Bayesian neural networks? Our results show that Bayesian neural networks can improve the performance over models trained with SGD in a variety of settings, both in terms of uncertainty calibration and predictive accuracy. On most of the problems we considered in this work, the best results were achieved by Bayesian neural networks. We believe that our results provide motivation to use Bayesian neural networks with accurate posterior approximation in practical applications.

Should we use HMC in practice? For most realistic scenarios in Bayesian Deep Learning, HMC is an impractical method. On the image classification benchmarks, HMC takes orders of magnitude more compute than any of the baselines that we considered. We hope that our work will inspire the community to produce new accurate and scalable approximate inference methods for Bayesian deep learning.

### 4.12.1 Challenging conventional wisdom

A conventional wisdom has emerged that deep ensembles are a non-Bayesian alternative to variational methods, that standard priors for neural networks are poor, and that cold posteriors are a problematic result for Bayesian deep learning. Our results highlight that one should take care in uncritically repeating such claims. In fact, deep ensembles appear to provide a higher fidelity representation of the Bayesian predictive distribution than widely accepted approaches to approximate Bayesian inference. If anything, the takeaway from the relatively good performance of deep ensembles is that we would benefit from approximate inference being *closer* to the Bayesian ideal! Moreover, the details over the priors in weight space can have a relatively minor effect on performance, and there is no strong evidence that standard Gaussian priors are particularly bad. In fact, there are many reasons to believe these priors have useful properties, see e.g. Section 3.7. Similarly, on close inspection, we found no evidence for a general cold posterior effect, which we identify as largely an artifact of data augmentation. Although we see here that tempering does not in fact seem to be required, as argued in Section 3.10 tempering is also not necessarily unreasonable or even divergent from Bayesian principles.

Even the results we found that are less favourable to Bayesian deep learning are contrary to the current orthodoxy. Indeed, higher fidelity Bayesian inference surprisingly appears to suffer more greatly from covariate shift, despite the popularity of approximate Bayesian inference procedures in this setting.

# 5 | Deep Neural Networks under Distribution Shifts

In the previous chapters of this dissertation, we discussed general properties of neural network loss surfaces, probabilistic perspective on generalization, uncertainty estimation and Bayesian neural networks. Now, we will focus on the problem of robustness to distribution shifts. In this chapter, we present a mechanistic understanding of feature learning in neural networks in the presence of shortcut (spurious) features.

We find that the last layer largely determines how much the model will rely on the shortcut features. Moreover, we can retrain the last layer, with a frozen feature extractor, and significantly reduce the reliance on spurious features. It is interesting to consider this result in the context of our analysis in Section 4.5, where we identified the importance of the first layer for robustness to input corruptions. Together, these results highlight the importance of low-level mechanistic understanding of deep learning models for robustness to distribution shift.

This chapter is adapted from the paper "Last Layer Re-Training is Sufficient for Robustness to Spurious Correlations" [Kirichenko et al. 2023] which originally appeared at ICLR 2023, written jointly with Polina Kirichenko and Andrew Gordon Wilson.

## 5.1 Spurious Correlations

Realistic datasets in deep learning are riddled with *spurious correlations* — patterns that are predictive of the target in the train data, but that are irrelevant to the true labeling function. For example, most of the images labeled as butterfly on ImageNet also show flowers [Singla and Feizi 2021], and most of the images labeled as tench show a fisherman holding the tench [Brendel and Bethge 2019]. Deep neural networks rely on these spurious features, and consequently degrade in performance when tested on datapoints where the spurious correlations break, for example, on images with unusual background contexts [Geirhos et al. 2020; Rosenfeld et al. 2018; Beery et al. 2018]. In an especially alarming example, CNNs trained to recognize pneumonia were shown to rely on hospital-specific metal tokens in the chest X-ray scans, instead of features relevant to pneumonia [Zech et al. 2018].

In this chapter, we investigate what features are in fact learned on datasets with spurious correlations. We find that even when neural networks appear to heavily rely on spurious features and perform poorly on minority groups where the spurious correlation is broken, they still learn the core features sufficiently well. These core features, associated with the semantic structure of the problem, are learned even in cases when the spurious features are much simpler than the core features (see Section 5.3.2) and in some cases even when no minority group examples are present in the training data! While both the relevant and spurious features are learned, the spurious features can be highly weighted in the final classification layer of the model, leading to poor predictions on the minority groups.

Inspired by these observations, we propose *Deep Feature Reweighting* (DFR), a simple and effective method for improving worst-group accuracy of neural networks in the presence of spurious features. We illustrate DFR in Figure 5.1. In DFR, we simply retrain the last layer of a classification model trained with standard Empirical Risk Minimization (ERM), using a small set of *reweighting* data where the spurious correlation does not hold. DFR achieves state-of-the-art

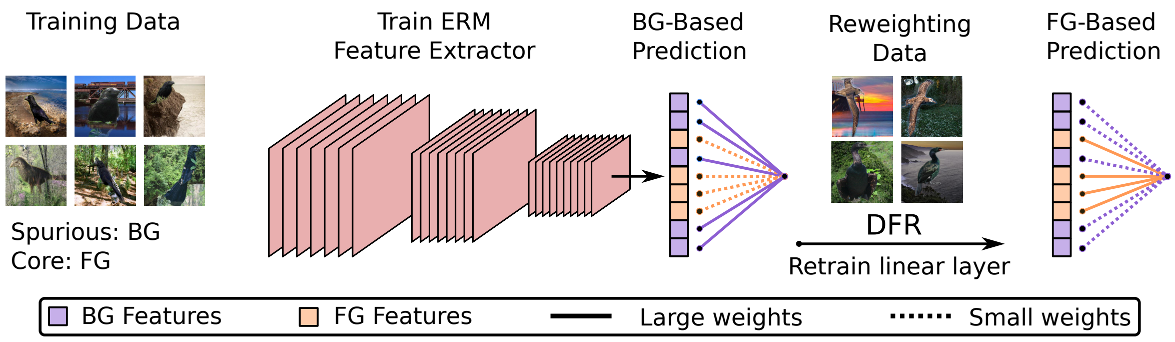


**Figure 5.1: Deep feature reweighting (DFR).** An illustration of the DFR method on the Waterbirds dataset, where the background (BG) is spuriously correlated with the foreground (FG). Standard ERM classifiers learn both features relevant to the background and the foreground, and weight them in a way that the model performs poorly on images with confusing backgrounds. With DFR, we simply reweight these features by retraining the last linear layer on a small dataset where the backgrounds are not spuriously correlated with the foreground. The resulting DFR model primarily relies on the foreground, and performs much better on images with confusing backgrounds.

performance on popular spurious correlation benchmarks by simply reweighting the features of a trained ERM classifier, with no need to re-train the feature extractor. Moreover, we show that DFR can be used to reduce reliance on background and texture information and improve robustness to certain types of covariate shift in large-scale models trained on ImageNet, by simply retraining the last layer of these models. We note that the reason DFR can be so successful is because standard neural networks *are* in fact learning core features, even if they do not primarily rely on these features to make predictions, contrary to recent findings [Hermann and Lampinen 2020; Shah et al. 2020]. Since DFR only requires retraining a last layer, amounting to logistic regression, it is extremely simple, easy to tune and computationally inexpensive relative to the alternatives, yet can provide state-of-the-art performance. Indeed, DFR can reduce texture bias and improve robustness of large ImageNet trained models, in only minutes on a single GPU. Our code is available at `github.com/PolinaKirichenko/deep_feature_reweighting`.

## 5.2 Problem Setting

We consider classification problems, where we assume that the data consists of several *groups* $\mathcal{G}_i$, which are often defined by a combination of a label and spurious attribute. Each group has its own data distribution $p_i(x, y)$, and the training data distribution is a mixture of the group distributions $p(x, y) = \sum_i \alpha_i p_i(x, y)$, where $\alpha_i$ is the proportion of group $\mathcal{G}_i$ in the data. For example, in the Waterbirds dataset [Sagawa et al. 2020], the task is to classify whether an image shows a landbird or a waterbird. The groups correspond to images of waterbirds on water background ($\mathcal{G}_1$), waterbirds on land background ($\mathcal{G}_2$), landbirds on water background ($\mathcal{G}_3$) and landbirds on land background ($\mathcal{G}_4$). See Figure A.44 for a visual description of the Waterbirds data. We will consider the scenario when the groups are not equally represented in the data: for example, on Waterbirds the sizes of the groups are 3498, 184, 56 and 1057, respectively. The larger groups $\mathcal{G}_1, \mathcal{G}_4$ are referred to as *majority groups* and the smaller $\mathcal{G}_2, \mathcal{G}_3$ are referred to as minority groups. As a result of this heavy imbalance, the background becomes a *spurious feature*, i.e. it is a feature that is correlated with the target on the train data, but it is not predictive of the target on the minority groups. Throughout the chapter we will discuss multiple examples of spurious correlations in both natural and synthetic datasets. In this chapter, we study the effect of spurious correlations on the features learned by standard neural networks, and based on our findings propose a simple way of reducing the reliance on spurious features assuming access to a small set of data where the groups are equally represented.

## 5.3 Understanding Representation Learning with Spurious Correlations

In this section we investigate the solutions learned by standard ERM classifiers on datasets with spurious correlations. We show that while these classifiers underperform on the minority groups, h

| **Train Data** | **Test Data (Worst Acc)** | |
|---|---|---|
| **(Spurious Corr.)** | Original | FG-Only |
| Balanced (50%) | 91.9% | 94.7% |
| Original (95%) | 73.8% | 93.7% |
| Original (100%) | 38.4% | 94% |
| FG-Only (-) | 75.2% | 95.5% |

**Table 5.1:** ERM classifiers trained on Waterbirds with Original and FG-Only images achieve similar FG-Only accuracy.

they still learn the core features that can be used to make correct predictions on the minority groups.

### 5.3.1 Feature learning on Waterbirds data

We first consider the Waterbirds dataset [Sagawa et al. 2020] (see Section 5.2) which is generated synthetically by combining images of birds from the CUB dataset [Wah et al. 2011] and backgrounds from the Places dataset [Zhou et al. 2017] (see Sagawa et al. [2020] for a detailed description of the data generation process). For the experiments in this section, we generate several variations of the Waterbirds data following the procedure analogous to Sagawa et al. [2020]. The *Original* dataset is analogous to the standard Waterbirds data, but we vary the degree of spurious correlation between the background and the target: 50% (*Balanced* dataset), 95% (as in Sagawa et al. [2020]) and 100% (no minority group examples in the train dataset). The *FG-Only* dataset contains images of the birds on uniform grey background instead of the Places background, removing the spurious feature. We show examples of datapoints from each variation of the dataset in Appendix Figure A.42. In Appendix A.7.2.1, we provide the full details for the experiment in this section, and additionally consider the reverse Waterbirds problem, where instead of predicting the bird type, the task is to predict the background type, with similar results.

We train a ResNet-50 model with ERM on each of the variations of the data and report the results in Table 5.1. Following prior work [e.g., Sagawa et al. 2020; Liu et al. 2021; Idrissi et al. 2021], we initialize the model with weights pretrained on ImageNet. For the models trained on the Original data, there is a large difference between the mean and worst group accuracy on the Original test data: the model heavily relies on the background information in its predictions, so the performance on minority groups is poor. The model trained without minority groups is especially affected, only achieving 38.4% worst group accuracy. However, surprisingly, we find that all the models trained on the Original data can make much better predictions on the FG-Only test data: if we remove the spurious feature (background) from the inputs at test time, the models make predictions based on the core feature (bird), and achieve worst-group[1] accuracy close to 94%, which is only slightly lower than the accuracy of a model trained directly on the FG-Only data and comparable to the accuracy of a model that was trained on balanced train data[2].

In summary, we conclude that while the models pre-trained on ImageNet and trained on the Original data make use of background information to make predictions, they still learn the features relevant to classifying the birds *almost as well as the models trained on the data without spurious correlations*. We will see that we can retrain the last layer of the Original-trained models and dramatically improve the worst-group performance on the Original data, by emphasizing the features relevant to the bird. While we use pre-trained models in this section, following the common practice on Waterbirds, we show similar results on other datasets without pre-training (see Section 5.3.2, Appendix A.7.3.1).

### 5.3.2 Simplicity bias

Shah et al. [2020] showed that neural networks can suffer from *extreme simplicity bias*, a tendency to completely rely on the simple features, while ignoring similarly predictive (or even

[1]On the FG-Only data the groups only differ by the bird type, as we remove the background. The difference between mean and worst-group accuracy is because the target classes are not balanced in the training data.

[2]In Appendix A.7.2.4, we demonstrate *logit additivity*: the class logits on Waterbirds are well approximated as the sum of the logits for the corresponding background image and the logits for the foreground image.

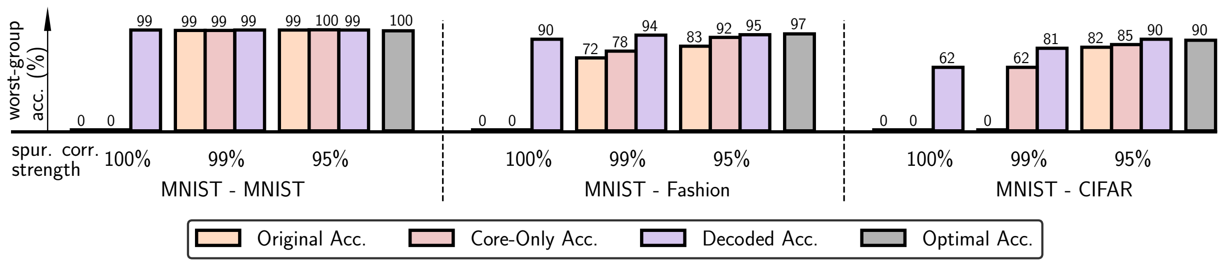


**Figure 5.2: Feature learning and simplicity bias.** ResNet-20 ERM classifiers trained on Dominoes data with varying levels of spurious correlation between core and spurious features. We show worst-group test accuracy for: Original data, data with only core features present (Core-Only), and accuracy of decoding the core feature from the latent representations of the Original data with logistic regression. We additionally report optimal accuracy: accuracy of a model trained and evaluated on the Core-Only data. Even in cases when the model achieves 0% accuracy on the Original data, the core features can still be decoded from latent representations.

more predictive) complex features. In this section, we explore whether the neural networks can still learn the core features in the extreme simplicity bias scenarios, where the spurious features are simple and highly correlated with the target. Following Shah et al. [2020] and Pagliardini et al. [2022], we consider *Dominoes* binary classification datasets, where the top half of the image shows MNIST digits [LeCun et al. 1998] from classes $\{0, 1\}$, and the bottom half shows MNIST images from classes {7, 9} (*MNIST-MNIST*), Fashion-MNIST [Xiao et al. 2017] images from classes {coat, dress} (*MNIST-Fashion*) or CIFAR-10 [Krizhevsky and Hinton 2009] images from classes {car, truck} (*MNIST-CIFAR*). In all Dominoes datasets, the top half of the image (MNIST $0-1$ images) presents a linearly separable feature; the bottom half of the image presents a harder to learn feature. See Appendix A.7.2.2 for more details about the experimental set-up and datasets, and Appendix Figure A.42 for image examples.

We use the simple feature (top half of the images) as a spurious feature and the complex feature (bottom half) as the core feature; we generate datasets with 100%, 99% and 95% correlations between the spurious feature and the target in train data, while the core feature is perfectly aligned with the target. We then define 4 groups $\mathcal{G}_i$ based on the values of the spurious and core features, where the minority groups correspond to images with top and bottom halves that do not match. The groups on validation and test are balanced.

We train a ResNet-20 model on each variation of the dataset. In Figure 5.2 we report the worst group performance for each of the datasets and each spurious correlation strength. In addition to the worst-group accuracy on the Original test data, we report the *Core-Only* worst-group accuracy, where we evaluate the model on datapoints with the spurious top half of the image replaced with a black image. Similarly to Shah et al. [2020] and Pagliardini et al. [2022], we observe that when the spurious features are perfectly correlated with the target on Dominoes datasets, the model relies just on the simple spurious feature to make predictions and achieves 0% worst-group accuracy. However, with 99% and 95% spurious correlation levels on train, we observe that models learned the core features well, as indicated both by their performance on the Original test data and especially increased performance on Core-Only test data where spurious features are absent. For reference, on each dataset we also report the *Optimal accuracy*, which is the accuracy of a model trained and evaluated on the Core-Only data. The optimal accuracy provides an upper bound on the accuracy that we can expect on each of the datasets.

Decoding feature representations. The performance on the Original and even Core-Only data might not be fully representative of whether or not the network learned a high-quality representation of the core features. Indeed, even if we remove the MNIST digit from the top half of the image, the network can still primarily rely on the (empty) top half in its predictions: an empty image may be more likely to come from class 1, which typically has fewer white pixels than class 0. To see how much information about the core feature is contained in the latent representation, we evaluate the *decoded* accuracy: for each problem we train a logistic regression classifier on the features extracted by the final convolutional layer of the network. We use a group-balanced validation set to train the logistic regression model, and then report the worst-group accuracy on a test set. In Figure 5.2, we observe that for MNIST-MNIST and MNIST-FashionMNIST even when the spurious correlation is 100%, reweighting the features leads to high worst group test accuracy.

Moreover, on all Dominoes datasets for 99% and 95% spurious correlations level the core features can be decoded with high accuracy and almost match the optimal accuracy. This decoding serves as a basis of our DFR method, which we describe in detail in Section 5.4. In Appendix A.7.2.2 we report an additional baseline and verify that the model indeed learns a non-trivial representation of the core feature.

Relation to prior work. With the same Dominoes datasets that we consider in this section, Shah et al. [2020] showed that neural networks tend to rely entirely on the simple features. However, they only considered the 100% spurious correlation strength and accuracy on the Original test data. Our results do not contradict their findings but provide new insights: even in these most challenging cases, the networks still represent information about the complex core feature. Moreover, this information can be decoded to achieve high accuracy on the mixed group examples. Hermann and Lampinen [2020] considered a different set of synthetic datasets, showing that in some cases neural networks fail to represent information about some of the predictive features. In particular, they also considered decoding the information about these features from the latent representations and different spurious correlation strengths. Our results add to their observations and show that while it is possible to construct examples where predictive features are suppressed, in many challenging practical scenarios, neural networks learn a high quality representation of the core features relevant to the problem even if they rely on the spurious features.

ColorMNIST. In Appendix A.7.2.3 we additionally show results on ColorMNIST dataset with varied spurious correlations strength in train data: decoding core features from the trained representations on this problem also achieves results close to optimal accuracy and demonstrates that the core features are learned even in the cases when the model initially achieved 0% worst-group accuracy.

In summary, we find that, surprisingly, if (1) the strength of the spurious correlation is lower than 100% or (2) the difference in complexity between the core and spurious features is not as stark as on MNIST-CIFAR, *the core feature can be decoded from the learned embeddings with high accuracy.*

## 5.4 Deep Feature Reweighting

In Section 5.3 we have seen that neural networks trained with standard ERM learn multiple features relevant to the predictive task, such as features of both the background and the object represented in the image. Inspired by these observations, we propose *Deep Feature Reweighting (DFR)*, a simple and practical method for improving robustness to spurious correlations and distribution shift.

Let us assume that we have access to a dataset $\mathcal{D} = \{x_i, y_i\}$ which can exhibit spurious correlations. Furthermore, we assume that we have access to a (typically much smaller) dataset $\hat{\mathcal{D}}$, where the groups are represented equally. $\hat{\mathcal{D}}$ can be a subset of the train dataset $\mathcal{D}$, or a separate set of datapoints. We will refer to $\hat{\mathcal{D}}$ as *reweighting dataset*. We start by training a neural network on all of the available data $\mathcal{D}$ with standard ERM without any group or class reweighting. For this stage we do not need any information beyond the training data and labels. Here, we assume that the network consists of a feature extractor (such as a sequence of convolutional or transformer layers), followed by a fully-connected classification layer mapping the features to class logits. In the second stage of the procedure, we simply discard the classification head and train a new classification head from scratch on the available balanced data $\hat{\mathcal{D}}$. We use the new classification head to make predictions on the test data. We illustrate DFR in Figure 5.1.

Notation. We will use notation $\text{DFR}_{\mathcal{D}}^{\hat{\mathcal{D}}}$, where $\mathcal{D}$ is the dataset used to train the base feature extractor model and $\hat{\mathcal{D}}$ – to train the last linear layer.

## 5.5 Feature Reweighting Improves Robustness

In this section, we evaluate DFR on benchmark problems with spurious correlations.

Data. See Section 5.2 for a description of the *Waterbirds* data. On *CelebA* hair color prediction[Liu et al. 2015], the groups are non-blond females ($\mathcal{G}_1$), blond females ($\mathcal{G}_2$), non-blond males ($\mathcal{G}_3$) and blond males ($\mathcal{G}_4$) with proportions 44%, 14%, 41%, and 1% of the data, respectively; the group $\mathcal{G}_4$ is the minority group, and the gender serves as a spurious feature. On *MultiNLI*, the goal is to classify the relation between a pair of sentences (premise and hypothesis) as contradiction, entailment or neutral; the presence of negation words ("no", "never", ...) is correlated with the contradiction class, and serves as a spurious feature. Finally, we consider the *CivilComments* [Borkan et al. 2019] dataset implemented in the WILDS benchmark [Koh et al. 2021; Sagawa et al. 2021], where the goal is to classify comments as toxic or neutral. Each comment is labeled with 8 attributes $s_i$ (male, female, LGBT, black, white, Christian, Muslim, other religion) based on whether or not the corresponding characteristic is mentioned in the comment. For evaluation, we follow the standard WILDS protocol and report the worst accuracy across the 16 overlapping groups corresponding to pairs $(y, s_i)$ for each of the attributes $s_i$. See Figures A.44, A.45 for a visual description of all four datasets.

Baselines. We consider 6 baseline methods that work under different assumptions on the information available at the training time. *Empirical Risk Minimization* (*ERM*) represents conventional training without any procedures for improving worst-group accuracies. *Just Train Twice* (*JTT*) [Liu et al. 2021] is a method that detects the minority group examples on train data, only using group labels on the validation set to tune hyper-parameters. *Correct-n-Contrast* (*CnC*) [Zhang et al. 2022] detects the minority group examples similarly to JTT, and uses a contrastive objective to learn representations robust to spurious correlations. *Group DRO* [Sagawa et al. 2020] is a state-of-the-art method, which uses group information on train and adaptively upweights worst-group examples during training. *SUBG* is ERM applied to a random subset of the data where the groups are equally represented, which was recently shown to be a surprisingly strong baseline [Idrissi et al. 2021]. Finally, *Spread Spurious Attribute* (*SSA*) [Nam et al. 2022] attempts to fully exploit the group-labeled validation data with a semi-supervised approach that propagates the group labels to the the training data. We discuss the assumptions on the data for each of these baselines in Appendix A.7.3.2.

DFR. We evaluate $DFR^{Val}_{Tr}$ , where we use a group-balanced[3] subset of the validation data available for each of the problems as the reweighting dataset $\hat{\mathcal{D}}$. We use the standard training dataset (with group imbalance) $\mathcal{D}$ to train the feature extractor. In order to make use of more of the available data, we train logistic regression 10 times using different random balanced subsets of the data, and average the weights of the learned models. We report full details and several ablation studies in Appendix A.7.3.

Hyper-parameters. Following prior work [e.g. Liu et al. 2021], we use a ResNet-50 model [He et al. 2016] pretrained on ImageNet for Waterbirds and CelebA and a BERT model [Devlin et al. 2018] pre-trained on Book Corpus and English Wikipedia data for MultiNLI and CivilComments. For $\mathrm{DFR}^{\mathrm{Val}}_{\mathrm{Tr}}$ , the size of the reweighting set $\hat{\mathcal{D}}$ is small relative to the number of features produced by the feature extractor. For this reason, we use $\ell_1$-regularization to allow the model to learn sparse solutions and drop irrelevant features. For $\mathrm{DFR}^{\mathrm{Val}}_{\mathrm{Tr}}$ *we only tune a single hyper-parameter* — the strength of the regularization term. We tune all the hyper-parameters on the validation data provided with each of the datasets. We note that the prior work methods including Group DRO, JTT, CnC, SUBG, SSA and others extensively tune hyper-parameters on the validation data. For $\mathrm{DFR}^{\mathrm{Val}}_{\mathrm{Tr}}$ we split the validation in half, and use one half to tune the regularization strength param-

[3]We keep all of the data from the smallest group, and subsample the data from the other groups to the same size. On CivilComments the groups overlap, so for each datapoint we use the smallest group that contains it when constructing the group-balanced reweighting dataset.

| Method | Group Info | Waterbirds | | CelebA | | MultiNLI | | CivilComments | |
|---|---|---|---|---|---|---|---|---|---|
| | Train / Val | Worst(%) | Mean(%) | Worst(%) | Mean(%) | Worst(%) | Mean(%) | Worst(%) | Mean(%) |
| JTT | ✗ / ✓ | 86.7 | 93.3 | 81.1 | 88.0 | 72.6 | 78.6 | 69.3 | 91.1 |
| CnC | ✗ / ✓ | $88.5_{\pm 0.3}$ | $90.9_{\pm 0.1}$ | $88.8_{\pm 0.9}$ | $89.9_{\pm 0.5}$ | - | - | $68.9_{\pm 2.1}$ | $81.7_{\pm 0.5}$ |
| SUBG | ✓/ ✓ | $89.1_{\pm 1.1}$ | - | $85.6_{\pm 2.3}$ | - | $68.9_{\pm 0.8}$ | - | - | - |
| SSA | ✗ / ✓✓ | $89.0_{\pm 0.6}$ | $92.2_{\pm 0.9}$ | **$89.8$**$_{\pm 1.3}$ | $92.8_{\pm 0.1}$ | $76.6_{\pm 0.7}$ | $79.9_{\pm 0.87}$ | **$69.9$**$_{\pm 2}$ | $88.2_{\pm 2.}$ |
| Group DRO | ✓/ ✓ | 91.4 | 93.5 | 88.9 | 92.9 | **77.7** | 81.4 | **69.9** | 88.9 |
| Base (ERM) | ✗ / ✗ | $74.9_{\pm 2.4}$ | $98.1_{\pm 0.1}$ | $46.9_{\pm 2.8}$ | $95.3_{\pm 0}$ | $65.9_{\pm 0.3}$ | $82.8_{\pm 0.1}$ | $55.6_{\pm 0.6}$ | $92.1_{\pm 0.1}$ |
| $\text{DFR}^{\text{Val}}_{\text{Tr}}$ | ✗ / ✓✓ | **$92.9$**$_{\pm 0.2}$ | $94.2_{\pm 0.4}$ | $88.3_{\pm 1.1}$ | $91.3_{\pm 0.3}$ | $74.7_{\pm 0.7}$ | $82.1_{\pm 0.2}$ | **$70.1$**$_{\pm 0.8}$ | $87.2_{\pm 0.3}$ |

**Table 5.2: Spurious correlation benchmark results.** Worst-group and mean test accuracy of DFR variations and baselines on benchmark datasets. For mean accuracy, we follow Sagawa et al. [2020] and weight the group accuracies according to their prevalence in the training data. The Group Info column shows whether group labels are available to the methods on train and validation datasets. $\text{DFR}^{\text{Val}}_{\text{Tr}}$ uses the validation data to train the model parameters (last layer) in addition to hyper-parameter tuning, indicated with ✓✓; SSA also uses the validation set to train the model. For DFR we report the mean±std over 5 independent runs. DFR is competitive with state-of-the-art.

eter; then, we retrain the logistic regression with the optimal regularization on the full validation set.

Results. We report the results for $\text{DFR}^{\text{Val}}_{\text{Tr}}$ and the baselines in Table 5.2. $\text{DFR}^{\text{Val}}_{\text{Tr}}$ is competitive with the state-of-the-art Group DRO across the board, achieving the best results among all methods on Waterbirds and CivilComments. Moreover, $\text{DFR}^{\text{Val}}_{\text{Tr}}$ achieves similar performance to SSA, a method designed to make optimal use of the group information on the validation data ($\text{DFR}^{\text{Val}}_{\text{Tr}}$ is slightly better on Waterbirds and CivilComments while SSA is better on CelebA and MultiNLI). Both methods use the same exact setting and group information to train and tune the model. We explore other variations of DFR in Appendix A.7.3.3.

To sum up, $\text{DFR}^{\text{Val}}_{\text{Tr}}$ matches the performance of the best available methods, while only using the group labels on a small validation set. This state-of-the-art performance is achieved by simply reweighting the features learned by standard ERM, with no need for advanced regularization techniques.

## 5.6 Natural Spurious Correlations on ImageNet

In the previous section we focused on benchmark problems constructed to highlight the effect of spurious features. Computer vision classifiers are known to learn undesirable patterns and rely on spurious features in real-world problems [see e.g. Rosenfeld et al. 2018; Singla and Feizi 2021; Geirhos et al. 2020]. In this section we explore two prominent shortcomings of ImageNet classifiers: background reliance [Xiao et al. 2020] and texture bias [Geirhos et al. 2018].

### 5.6.1 Background reliance

Prior work has demonstrated that computer vision models such as ImageNet classifiers can rely on image background to make their predictions [Zhang et al. 2007; Ribeiro et al. 2016; Shetty et al. 2019; Xiao et al. 2020; Singla and Feizi 2021]. Here, we show that it is possible to reduce the background reliance of ImageNet-trained models by simply retraining their last layer with DFR.

Xiao et al. [2020] proposed several datasets in the *Backgrounds Challenge* to study the effect of the backgrounds on predictions of ImageNet models. The datasets in the Backgrounds Challenge are based on the ImageNet-9 dataset, a subset of ImageNet structured into 9 coarse-grain classes (see Xiao et al. [2020] for details). ImageNet-9 contains 45$k$ training images and 4050 validation images. We consider three datasets from the Backgrounds Challenge: (1) *Original* contains the original images; (2) *Mixed-Rand* contains images with random combinations of backgrounds and foregrounds (objects); (3) *FG-Only* contains images showing just the object with a black background. We additionally consider *Paintings-BG* using paintings from Kaggle's `painter-by-numbers` dataset ( `https://www.kaggle.com/c/painter-by-numbers/`) as background for the ImageNet-9 validation data. Finally, we consider the ImageNet-R dataset [Hendrycks et al. 2021] restricted to the ImageNet-9 classes. See Appendix A.7.5 for details and Figure A.49 for example images.

We use an ImageNet-trained ResNet-50 as a feature extractor and train DFR with different

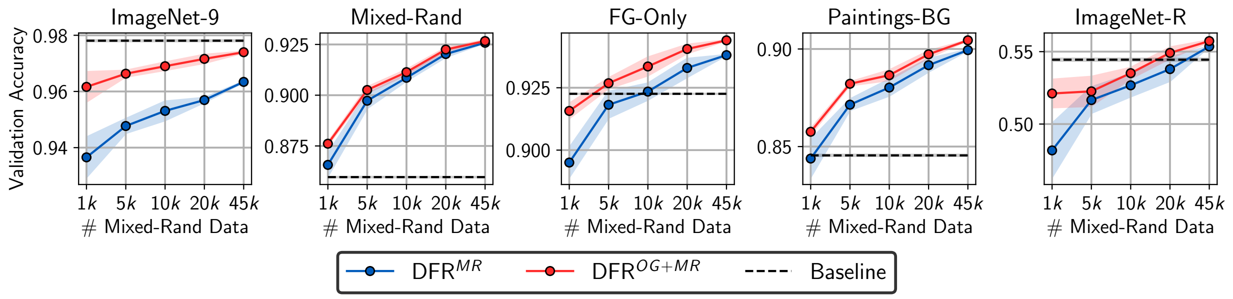


**Figure 5.3: Background reliance.** Accuracy of DFR$^{\text{MR}}$and DFR$^{\text{OG+MR}}$ on different ImageNet-9 validation splits with an ImageNet-trained ResNet-50 feature extractor. DFR reduces background reliance with a minimal drop in performance on the Original data.

reweighting datasets. As a *Baseline*, we train DFR on the Original 45$k$ training datapoints (we cannot use the ImageNet-trained ResNet-50 last layer, as ImageNet-9 has a different set of classes). We train *DFR*$^{MR}$ and *DFR*$^{OG+MR}$ on Mixed-Rand training data and a combination of Mixed-Rand and Original training data respectively. In Figure 5.3, we report the predictive accuracy of these methods on different validation datasets as a function of the number of Mixed-Rand data observed. We select the observed subset of the data randomly; for DFR$^{\text{OG+MR}}$, in each case we use the same amount of Mixed-Rand and Original training datapoints. We repeat this experiment with a VIT-B-16 model [Dosovitskiy et al. 2020] pretrained on ImageNet-21$k$ and report the results in the Appendix Figure A.47.

First, we notice that the baseline model provides significantly better performance on FG-Only (92%) than on the Mixed-Rand (86%) validation set, suggesting that the feature extractor learned the features needed to classify the foreground, as well as the background features. With access to Mixed-Rand data, we can reweight the foreground and background features with DFR and significantly improve the performance on mixed-rand, FG-Only and Paintings-BG datasets. At the same time, DFR$^{\text{OG+MR}}$ is able to mostly maintain the performance on the Original ImageNet-9 data; the small drop in performance is because the background is relevant to predicting the class on this validation set. Finally, on ImageNet-R, DFR provides a small improvement when we use all of 45$k$ datapoints; the covariate shift in ImageNet-R is not primarily background-based, so reducing background reliance does not provide a big improvement. We provide additional results in See Appendix A.7.5.

### 5.6.2 Texture-vs-shape bias

Geirhos et al. [2018] showed that on images with conflicting texture and shape, ImageNet-trained CNNs tend to make predictions based on the texture, while humans usually predict based on the shape of the object. The authors designed the *GST* dataset with conflicting cues, and proposed the term *texture bias* to refer to the fraction of datapoints on which a model (or a human) makes predictions based on texture; conversely, *shape bias* is the fraction of the datapoints on which prediction is made based on the shape of the object. Geirhos et al. [2018] showed that it is possible to increase the shape bias of CNNs by training on Stylized ImageNet (*SIN*), a dataset obtained from ImageNet by removing the texture information via style transfer (see Appendix Figure A.49 for example images). Using SIN in combination with ImageNet (*SIN+IN*), they also obtained improved robustness to corruptions. Finally, they proposed the *Shape-RN-50* model, a ResNet-50 (RN-50) trained on SIN+IN and finetuned on IN, which outperforms the ImageNet-trained ResNet-50 on in-distribution data and out-of-distribution robustness.

Here, we explore whether it is possible to change the shape bias of ImageNet-trained models by simply retraining the last layer with DFR. Intuitively, we expect that the standard ImageNet-trained model already learns *both* the shape and texture features. Indeed, Hermann et al. [2020] showed that shape information is partially decodable from the features of ImageNet models on the GST dataset. Here, instead of targeting GST, we apply DFR to the large-scale SIN dataset, and explore both the shape bias and the predictive performance of the resulting models. In Table 5.3, we report the shape bias, as well as predictive accuracy of ResNet-50 models trained on ImageNet (*IN*), SIN, IN+SIN and the Shape-RN-50 model, and the DFR models trained on SIN and IN+SIN. The DFR models use an IN-trained ResNet-50 model as a feature extractor. See Appendix A.7.6 for details.

| Method | Training Data | Shape bias (%) | Top-1 Acc (%) | | |
|---|---|---|---|---|---|
| | | | ImageNet | ImageNet-R | ImageNet-C |
| RN-50 | IN | 21.4 | 76.0 | 23.8 | 39.8 |
| | SIN | **81.4** | 60.3 | 26.9 | 38.1 |
| | IN+SIN | 34.7 | 74.6 | **27.6** | **45.7** |
| Shape-RN-50 | IN+SIN | 20.5 | **76.8** | 25.6 | 42.3 |
| DFR | SIN | 34.0 | 65.1 | 24.6 | 36.7 |
| | IN+SIN | 30.6 | 74.5 | **27.2** | 40.7 |

**Table 5.3: Shape bias.** Shape bias and accuracy on ImageNet validation set variations for ResNet-50 trained on different datasets and DFR with an ImageNet-trained ResNet-50 as a feature extractor. For each metric, we show the best result in bold. By retraining just the last layer with DFR, we can significantly increase the shape bias compared to the base model (21.4% $\rightarrow$ 34% for DFR(SIN)) and improve performance on ImageNet-R/C.

First, we observe that while the SIN-trained RN-50 achieves a shape bias of 81.4%, as reported by Geirhos et al. [2018], the models trained on combinations of IN and SIN are still biased towards texture. Curiously, the Shape-RN-50 model proposed by Geirhos et al. [2018] has almost identical shape bias to a standard IN-trained RN-50! At the same time, Shape-RN-50 outperforms IN-trained RN-50 on all the datasets that we consider, and Geirhos et al. [2018] showed that Shape-RN-50 significantly outperforms IN-trained RN-50 in transfer learning to a segmentation problem, suggesting that it learned better shape-based features. The fact that the shape bias of this model is lower than that of an IN-trained RN-50, suggests that the shape bias is largely affected by the last linear layer of the model: even if the model extracted high-quality features capturing the shape information, the last layer can still assign a higher weight to the texture information.

Next, DFR can significantly increase the shape bias of an IN-trained model. $\text{DFR}_{\text{IN}}^{\text{SIN}}$ achieves a comparable shape bias to that of a model trained from scratch on a combination of IN and SIN datasets. Finally, $\text{DFR}_{\text{IN}}^{\text{IN+SIN}}$ improves the performance on both ImageNet-R and ImageNet-C [Hendrycks and Dietterich 2019] datasets compared to the base RN-50 model. In the Appendix Table A.32 we show similar results for a VIT-B-16 model pretrained on ImageNet-21$k$ and finetuned on ImageNet; there, $\text{DFR}_{\text{IN}}^{\text{IN+SIN}}$ can also improve the shape bias and performance on ImageNet-C, but does not help on ImageNet-R. To sum up, by reweighting the features learned by an ImageNet-trained model, we can significantly increase its shape bias and improve robustness to certain corruptions. However, to achieve the highest possible shape bias, it is still preferable to re-train the model from scratch, as RN-50(SIN) achieves a much higher shape bias compared to all other methods.

## 5.7 Related Work

Feature learning in the presence of spurious correlations. The poor performance of neural networks on datasets with spurious correlations inspired research in understanding when and how the spurious features are learned. Geirhos et al. [2020] provide a detailed survey of the results in this area. Several works explore the behavior of maximum-margin classifiers, SGD training dynamics and inductive biases of neural network models in the presence of spurious features [Nagarajan et al. 2020; Pezeshki et al. 2021; Rahaman et al. 2019]. Shah et al. [2020] show empirically that in certain scenarios neural networks can suffer from *extreme simplicity bias* and rely on simple spurious features, while ignoring the core features; in Section 5.3.2 we revisit these problems and provide further discussion. Hermann and Lampinen [2020] and Jacobsen et al. [2018] also show synthetic and natural examples, where neural networks ignore relevant features, and Scimeca et al. [2021] explore which types of shortcuts are more likely to be learned. Kolesnikov and Lampert [2016] on the other hand show that on realistic datasets core and spurious features can often be distinguished from the latent representations learned by a neural network in the context of object localization.

Group robustness. The methods achieving the best worst-group performance typically build on the distributionally robust optimization (DRO) framework, where the worst-case loss is minimized instead of the average loss [Ben-Tal et al. 2013; Hu et al. 2018; Sagawa et al. 2020; Oren et al. 2019; Zhang et al. 2020a]. Notably, Group DRO [Sagawa et al. 2020], which optimizes a soft version of the worst-group loss holds state-of-the-art results on multiple benchmarks with spurious correlations. Several methods have been proposed for the scenario where group labels are not known, and need to be inferred from the data; these methods typically train a pair of networks, where the first model is used to identify the challenging minority examples and define a weighted loss for the second model [Liu et al. 2021; Nam et al. 2020; Yaghoobzadeh et al. 2019; Utama et al. 2020; Creager et al. 2021; Dagaev et al. 2021; Zhang et al. 2022]. Other works proposed semi-supervised methods for the scenario where the group labels are provided for a small fraction of the train datapoints [Sohoni et al. 2021; Nam et al. 2022]. Idrissi et al. [2021] recently showed that with careful tuning simple approaches such as data subsampling and reweighting can provide competitive performance. We note that all of the methods described above use a validation set with a high representation of minority groups to tune the hyper-parameters and optimize worst-group performance.

In a closely related work, Menon et al. [2020] considered classifier retraining and threshold correction on a subset of the training data for correcting the subgroup bias. In our work, we focus on classifier retraining and show that re-using train data for last layer retraining as done in Menon et al. [2020] is suboptimal, while retraining the last layer on held-out data achieves significantly better performance across various spurious correlations benchmarks (see Section 5.5). Moreover, we make multiple contributions beyond the scope of Menon et al. [2020], for example: we analyze feature learning in the extreme simplicity bias scenarios (Section 5.3), show strong performance on natural language processing datasets with spurious correlations (Section 5.5) and demonstrate how last layer retraining can be used to correct background and texture bias in large-scale models on ImageNet (Section 5.6).

In an independent and concurrent work, Rosenfeld et al. [2022] show that ERM learns high quality representations for domain generalization, and by training the last layer on the target domain it is possible to achieve strong results. Kang et al. [2019] propose to use classifier re-training, among other methods, in the context of long-tail classification. The observations in our work are complimentary, as we focus on spurious correlation robustness instead of domain generalization and long-tail classification. Moreover, there are important algorithmic differences between DFR, and the methods of Kang et al. [2019] and Rosenfeld et al. [2022]. In particular, Kang et al. [2019] retrain the last layer on the training data with class-balanced data sampling, while we use held-out data and group-balanced subsampling; they also do not use regularization. Rosenfeld et al. [2022] only present last layer retraining results as motivation, and do not use it in their proposed DARE method. We present a detailed discussion of these differences in Appendix A.7.1, and empirical comparisons in Appendix A.7.7.

Other group robustness methods. Another group of papers proposes regularization techniques to learn diverse solutions on the train data, focusing on different groups of features [Teney et al. 2021a; Lee et al. 2022; Pagliardini et al. 2022; Pezeshki et al. 2021]. Xu et al. [2022] show how to train *orthogonal classifiers*, i.e. classifiers invariant to given spurious features in the data. Other papers proposed methods based on meta-learning the weights for a weighted loss [Ren et al. 2018] and group-agnostic adaptive regularization [Cao et al. 2019, 2020]. Sohoni et al. [2020] use clustering of internal features to provide approximate labels for group distributionally robust optimization. A number of prior works address de-biasing classifiers by leveraging prior knowledge on the bias type [Li and Vasconcelos 2019; Kim et al. 2019; Tartaglione et al. 2021; Teney et al. 2021b; Zhu et al. 2021; Wang et al. 2019; Cadene et al. 2019; Li et al. 2018a].

Spurious correlations in natural image datasets. Multiple works demonstrated that natural image datasets contain spurious correlations that hurt neural network models [Kolesnikov and Lampert 2016; Xiao et al. 2020; Shetty et al. 2019; Alcorn et al. 2019; Singla and Feizi 2021; Singla et al. 2021; Moayeri et al. 2022]. Notably, Geirhos et al. [2018] demonstrated that ImageNet-trained CNNs are biased towards texture rather than shape of the objects. Follow-up work explored this texture bias and showed that despite being texture-biased, CNNs still often represent information about the shape in their feature representations [Hermann et al. 2020; Islam et al. 2021]. In Section 5.6 we show that it is possible to reduce the reliance of ImageNet-trained models on background context and texture information by retraining just the last layer of the model.

Transfer learning. Transfer learning [Pan and Yang 2009; Sharif Razavian et al. 2014] is an extremely popular framework in modern machine learning. Multiple works demonstrate its effectiveness [e.g. Girshick 2015; Huh et al. 2016; He et al. 2017; Sun et al. 2017; Mahajan et al. 2018; Kolesnikov et al. 2020; Maddox et al. 2021], and study when and why transfer learning can be effective [Zhai et al. 2019; Neyshabur et al. 2020; Abnar et al. 2021; Kornblith et al. 2019; Kumar et al. 2022]. Bommasani et al. [2021] provide a comprehensive discussion of modern transfer learning with large-scale models. While algorithmically our proposed DFR method is related to transfer learning, it has a different motivation and works on different problems. We discuss the relation between DFR and transfer learning in detail in Section 5.4.

Related methods have been developed in several areas of machine learning, such as ML Fairness [Dwork et al. 2012; Hardt et al. 2016; Kleinberg et al. 2016; Pleiss et al. 2017; Agarwal et al. 2018; Khani et al. 2019], NLP [McCoy et al. 2019; Lovering et al. 2020; Kaushik et al. 2020, 2021; Eisenstein 2022; Veitch et al. 2021] domain adaptation [Ganin and Lempitsky 2015; Ganin et al. 2016] and domain generalization [Blanchard et al. 2011; Muandet et al. 2013; Li et al. 2018b; Gulrajani and Lopez-Paz 2020; Ruan et al. 2021] including works on Invariant Risk Minimization and causality [Peters et al. 2016; Arjovsky et al. 2019; Krueger et al. 2021; Aubin et al. 2021].

## 5.8 Discussion

We have shown that neural networks simultaneously learn multiple different features, including relevant semantic structure, even in the presence spurious correlations. By retraining the last layer of the network with DFR, we can significantly reduce the impact of spurious features and improve worst-group-performance of the models. In particular, DFR achieves state-of-the-art performance on spurious correlation benchmarks, and can reduce the reliance of ImageNet trained models on background and texture information. DFR is extremely simple, cheap and effective: it only has one hyper-parameter, and we can run it on ImageNet-scale data in a matter of minutes. We hope that our results will be useful to practitioners and will inspire further research in feature learning in the presence of spurious correlations.

Spurious correlations and representation learning. Prior work has often associated poor robustness to spurious correlations with the quality of *representations* learned by the model [Arjovsky et al. 2019; Bahng et al. 2020; Ruan et al. 2021] and suggested that the entire model needs to be carefully trained to avoid relying on spurious features [e.g. Sagawa et al. 2020; Idrissi et al. 2021; Liu et al. 2021; Zhang et al. 2022; Sohoni et al. 2021; Pezeshki et al. 2021; Lee et al. 2022]. Our work presents a different view: *representations learned with standard ERM even without seeing any minority group examples are sufficient to achieve state-of-the-art performance on popular spurious correlation benchmarks*. The issue of spurious correlations is not in the features extracted by the models (though the representations learned by ERM can be improved [e.g., Hermann and Lampinen 2020; Pezeshki et al. 2021]), but in the weights assigned to these features. Thus we can simply re-weight these features for substantially improved robustness.

Practical advantages of DFR. DFR is extremely simple, cheap and effective. In particular, DFR has only one tunable hyper-parameter — the strength of regularization of the logistic regression. Furthermore, DFR is highly robust to the choice of base model, as we demonstrate in Appendix Table A.29, and does not require early stopping or other highly problem-specific tuning such as in Idrissi et al. [2021] and other prior works. Moreover, as DFR only requires re-training the last linear layer of the model, it is also extremely fast and easy to run. For example, we can train DFR on the 1.2$M$-datapoint Stylized ImageNet dataset in Section 5.6 on a single GPU in a matter of minutes, after extracting the embeddings on all of the datapoints, which only needs to be done once. On the other hand, existing methods such as Group DRO [Sagawa et al. 2020] require training the model from scratch multiple times to select the best hyper-parameters, which may be impractical for large-scale problems.

On the need for reweighting data in DFR. Virtually all methods in the spurious correlation literature, even the ones that do not explicitly use group information to train the model, use group-labeled data to tune the hyper-parameters (we discuss the assumptions of different methods in prior work in Appendix A.7.3.2). If a practitioner has access to group-labeled data, we believe that they should leverage this data to find a better model instead of just tuning the hyper-parameters. Finally, we note that DFR can be easily combined with methods that automatically estimate group labels [e.g. Liu et al. 2021; Creager et al. 2021; Sohoni et al. 2021]: we can retrain the last layer using these estimated labels instead of ground truth.

Future work. There are many exciting directions for future research. DFR largely reduces the issue of spurious correlations to a linear problem: how do we train an optimal linear classifier on given features to avoid spurious correlations? In particular, we can try to avoid the need for a balanced reweighting dataset by carefully studying this linear problem and only using the features that are robustly predictive across all of the training data. We can also consider other types of supervision, such as saliency maps [Singla and Feizi 2021] or segmentation masks to tell the last layer of the model what to pay attention to in the data. Finally, we can leverage better representation learning methods [Pezeshki et al. 2021], including self-supervised learning methods [e.g. Chen et al. 2020; He et al. 2021], to further improve the performance of DFR.

# 6 | Conclusion

Despite the exciting progress in methodology, our understanding of deep learning models remains very limited. In this dissertation, we explored a range of results that aim to impprove this understanding. In these works, we dissect models and training procedures and attempt to look inside the black box and understand the mechanisms and principles.

In Chapter 2, we explored the loss surfaces of neural networks. We demonstrated mode connectivity, the phenomenon that the optima of neural network loss surfaces are connected by simple paths of low loss. This observation inspired us to propose SWA, an impactful method for training deep neural networks.

In Chapter 3, we presented a principled treatment of generalization in deep learning from a Bayesian probabilistic perspective. We used our insights about the loss surfaces from Chapter 2 to develop scalable methods for approximate inference in deep learning with state-of-the-art performance in uncertainty quantification.

In Chapter 4, for the first time, we applied highly precise Hamiltonian Monte Carlo approximate inference to practically-sized Bayesian neural networks. With this tool, we were able to answer multiple foundational questions around Bayesian deep learning. In particular, we have seen that Bayesian neural networks with standard priors provide strong performance in-distribution but generalize poorly when the inputs are corrupted at test time. We presented a precise mechanistic explanation of this result and propose a fix.

Finally, in Chapter 5, we explored feature learning in neural networks in the presence of shortcut features. We showed the last layer often plays a distinct mechanistic role in the model and can control how much the network relies on the shortcut features. Motivated by this observation, we proposed a practical method for reducing reliance on shortcut features in pretrained models.

I believe that now is the time when foundational research on understanding deep learning models is incredibly important. With our models becoming larger and more powerful, and applied more and more widely in the real world, we need to understand these models, the decisions that they make, and when we can trust them. The improved understanding can also motivate practical improvements in the methodology, as we have seen many times throughout this dissertation.

# A | Appendix

## A.1 Appendix for Mode Connectivity

We organize the supplementary material as follows. Section A.1.1 discusses the computational complexity of the proposed curve finding method. Section A.1.2 describes how to apply batch normalization at test time to points on curves connecting pairs of local optima. Section A.1.3 provides formulas for a polygonal chain and Bezier curve with $n$ bends. Section A.1.4 provides details and results of experiments on curve finding and contains a table summarizing all path finding experiments. Section A.1.5 provides additional visualizations of the train loss and test accuracy surfaces. Section A.1.6 contains details on curve ensembling experiments. Section **??** describes experiments on relation between mode connectivity and the number of parameters in the networks. Section A.1.8 discusses a trivial construction of curves connecting two modes, where points on the curve represent reparameterization of the endpoints, unlike the curves in the main text. Section A.1.9 provides details of experiments on FGE. Finally, Section A.1.10 describes pathways traversed by FGE.

### A.1.1 Computational complexity of curve finding

The forward pass of the proposed method consists of two steps: computing the point $\phi_\theta(t)$ and then passing a mini-batch of data through the DNN corresponding to this point. Similarly, the backward pass consists of first computing the gradient of the loss with respect to $\phi_\theta(t)$, and then multiplying the result by the Jacobian $\frac{\partial \phi_\theta}{\partial \theta}$. The second step of the forward pass and the first step of the backward pass are exactly the same as the forward and backward pass in the training of a single DNN model. The additional computational complexity of the procedure compared to single model training comes from the first step of the forward pass and the second step of the backward pass and in general depends on the parametrization $\phi_\theta(\cdot)$ of the curve.

In our experiments we use curve parametrizations of a specific form. The general formula for a curve with one bend is given by

$$\phi_\theta(t) = \hat{w}_1 \cdot c_1(t) + \theta \cdot c(t) + \hat{w}_2 \cdot c_2(t).$$

Here the parameters of the curve are given by $\theta \in \mathbb{R}^{|net|}$ and coefficients $c_1, c_2, c : [0, 1] \to \mathbb{R}$.

For this family of curves the computational complexity of the first step of the method is $O(|net|)$, as we only need to compute a weighted sum of $\hat{w}_1$, $\hat{w}_2$ and $\theta \in \mathbb{R}^{|net|}$. The Jacobian matrix

$$\frac{\partial \phi_\theta(t)}{\partial \theta} = c(t) \cdot I,$$

thus the additional computational complexity of the backward pass is also $O(|net|)$, as we only need to multiply the gradient with respect to $\phi_\theta(t)$ by a scalar. Thus, the total additional computational complexity is $O(|net|)$. In practice we observe that the gap in time-complexity between one epoch of training a single model and one epoch of the proposed method with the same network architecture is usually below 50%.

### A.1.2 Batch Normalization

Batch normalization (Ioffe and Szegedy [2015]) is essential to modern deep learning architectures. Batch normalization re-parametrizes the output of each layer as

$$\hat{x} = \gamma \frac{x - \mu(x)}{\sigma(x) + \epsilon} + \beta,$$

where $\mu(x)$ and $\sigma(x)$ are the mean and standard deviation of the output $x$, $\epsilon > 0$ is a constant for numerical stability and $\gamma$ and $\beta$ are free parameters. During training, $\mu(x)$ and $\sigma(x)$ are computed separately for each mini-batch and at test time statistics aggregated during training are used.

When connecting two DNNs that use batch normalization, along a curve $\phi(t)$, we compute $\mu(x)$ and $\sigma(x)$ for any given $t$ over mini-batches during training, as usual. In order to apply batch-normalization to a network on the curve at the test stage we compute these statistics with one additional pass over the data, as running averages for these networks are not collected during training.

### A.1.3 Formulas for curves with $n$ bends

For $n$ bends $\theta = \{w_1, w_2, \dots, w_n\}$, the parametrization of a polygonal chain connecting points $w_0, w_{n+1}$ is given by

$$\phi_\theta(t) = (n+1) \cdot \left( \left( t - \frac{i}{n+1} \right) \cdot w_{i+1} + \left( \frac{i+1}{n+1} - t \right) \cdot w_i \right),$$

for $\frac{i}{n+1} \leqslant t \leqslant \frac{i+1}{n+1}$ and $0 \leq i \leq n$.

For $n$ bends $\theta = \{w_1, w_2, \dots, w_n\}$, the parametrization of a Bezier curve connecting points $w_0$ and $w_{n+1}$ is given by

$$\phi_\theta(t) = \sum_{i=0}^{n+1} w_i C_{n+1}^i t^i (1-t)^{n+1-i}$$

**Table A.1:** The properties of loss and error values along the found curves for different architectures and tasks

| Model | | Length | Train Loss | | | | Train Error (%) | | | Test Error (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DNN | Curve | Ratio | Min | Int | Mean | Max | Min | Int | Max | Min | Int | Max |
| | | | | | | MNIST | | | | | | |
| FC | Single | – | 0.018 | – | – | 0.018 | 0.01 | – | 0.01 | 1.46 | – | 1.5 |
| FC | Segment | 1 | 0.018 | 0.252 | 0.252 | 0.657 | 0.01 | 0.53 | 2.13 | 1.45 | 1.96 | 3.18 |
| FC | Bezier | 1.58 | 0.016 | 0.02 | 0.02 | 0.024 | 0.01 | 0.02 | 0.04 | 1.46 | 1.52 | 1.56 |
| FC | Polychain | 1.73 | 0.013 | 0.022 | 0.022 | 0.029 | 0 | 0.03 | 0.07 | 1.46 | 1.51 | 1.58 |
| | | | | | | CIFAR-10 | | | | | | |
| 3conv3fc | Single | – | 0.05 | – | – | 0.05 | 0.06 | – | 0.06 | 12.3 | – | 12.36 |
| 3conv3fc | Segment | 1 | 0.05 | 1.124 | 1.124 | 2.416 | 0.06 | 35.69 | 88.24 | 12.28 | 43.3 | 88.27 |
| 3conv3fc | Bezier | 1.30 | 0.034 | 0.038 | 0.037 | 0.05 | 0.05 | 0.1 | 0.2 | 12.06 | 12.7 | 13.66 |
| 3conv3fc | Polychain | 1.67 | 0.04 | 0.044 | 0.044 | 0.05 | 0.06 | 0.15 | 0.31 | 12.17 | 12.68 | 13.31 |
| VGG-16 | Single | – | 0.04 | – | – | 0.04 | 0 | – | 0.01 | 6.87 | – | 7.01 |
| VGG-16 | Segment | 1 | 0.039 | 1.759 | 1.759 | 2.569 | 0 | 61.43 | 90 | 6.87 | 63.75 | 90 |
| VGG-16 | Bezier | 1.55 | 0.028 | 0.03 | 0.03 | 0.04 | 0 | 0.01 | 0.02 | 6.59 | 6.77 | 7.01 |
| VGG-16 | Polychain | 1.83 | 0.025 | 0.031 | 0.031 | 0.045 | 0 | 0.01 | 0.04 | 6.54 | 6.89 | 7.28 |
| ResNet-158 | Single | – | 0.015 | – | – | 0.015 | 0.02 | – | 0.02 | 5.56 | – | 5.74 |
| ResNet-158 | Segment | 1 | 0.013 | 0.551 | 0.551 | 2.613 | 0 | 16.37 | 81.41 | 5.57 | 20.79 | 80.00 |
| ResNet-158 | Bezier | 2.13 | 0.013 | 0.017 | 0.018 | 0.022 | 0 | 0.02 | 0.07 | 5.48 | 5.82 | 6.24 |
| ResNet-158 | Polychain | 3.48 | 0.013 | 0.017 | 0.017 | 0.047 | 0 | 0.05 | 0.139 | 5.48 | 5.88 | 7.35 |
| WRN-10-28 | Single | – | 0.033 | – | – | 0.035 | 0 | – | 0 | 4.49 | – | 4.56 |
| WRN-10-28 | Segment | 1 | 0.033 | 0.412 | 0.412 | 2.203 | 0 | 5.44 | 65.62 | 4.49 | 10.55 | 66.6 |
| WRN-10-28 | Bezier | 1.83 | 0.03 | 0.033 | 0.038 | 0.038 | 0 | 0.01 | 0.04 | 4.4 | 4.62 | 4.83 |
| WRN-10-28 | Polychain | 1.95 | 0.026 | 0.029 | 0.029 | 0.037 | 0 | 0 | 0 | 4.38 | 6.93 | 10.38 |
| | | | | | | CIFAR-100 | | | | | | |
| VGG-16 | Single | – | 0.14 | – | – | 0.141 | 0.05 | – | 0.06 | 29.44 | – | 29.94 |
| VGG-16 | Segment | 1 | 0.137 | 3.606 | 3.606 | 4.941 | 0.04 | 73.25 | 99 | 29.44 | 80.59 | 99.01 |
| VGG-16 | Bezier | 1.52 | 0.095 | 0.107 | 0.105 | 0.141 | 0.03 | 0.08 | 0.18 | 29.28 | 30.49 | 31.23 |
| VGG-16 | Polychain | 1.64 | 0.118 | 0.139 | 0.139 | 0.2 | 0.04 | 0.19 | 0.39 | 29.33 | 30.13 | 30.92 |
| ResNet-164 | Single | – | 0.079 | – | – | 0.08 | 0.06 | – | 0.09 | 24.41 | – | 24.4 |
| ResNet-164 | Segment | 1 | 0.076 | 1.844 | 1.844 | 5.53 | 0.06 | 38.03 | 98.65 | 24.4 | 53.69 | 98.83 |
| ResNet-164 | Bezier | 1.87 | 0.074 | 0.083 | 0.084 | 0.098 | 0.05 | 0.28 | 0.96 | 24.15 | 24.99 | 26.1 |
| ResNet-164 | Polychain | 2.56 | 0.067 | 0.078 | 0.078 | 0.109 | 0.06 | 0.28 | 0.85 | 23.98 | 24.92 | 26.12 |

### A.1.4 Curve Finding Experiments

All experiments on curve finding were conducted with TensorFlow (Abadi et al. [2016]) and as baseline models we used the following implementations:

**Table A.2:** The value of perplexity along the found curves for PTB dataset

| Model | | Train | | Validation | | Test | |
|---|---|---|---|---|---|---|---|
| DNN | Curve | Min | Max | Min | Max | Min | Max |
| PTB | | | | | | | |
| RNN | Single | 37.5 | 39.2 | 82.7 | 83.1 | 78.7 | 78.9 |
| RNN | Segment | 37.5 | 596.3 | 82.7 | 682.1 | 78.7 | 615.7 |
| RNN | Bezier | 29.8 | 39.2 | 82.7 | 88.7 | 78.7 | 84.0 |

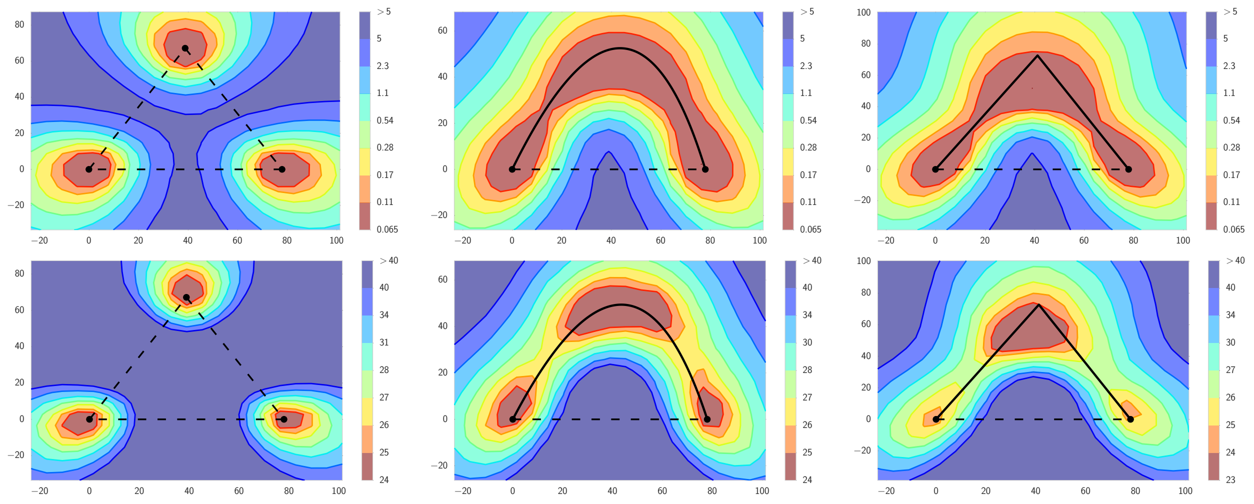


**Figure A.1:** The $\ell_2$-regularized cross-entropy train loss (**Top**) and test error (**Bottom**) surfaces of a deep residual network (ResNet-164) on CIFAR-100. **Left:** Three optima for independently trained networks. **Middle** and **Right**: A quadratic Bezier curve, and a polygonal chain with one bend, connecting the lower two optima on the left panel along a path of near-constant loss. Notice that in each panel, a direct linear path between each mode would incur high loss.

- ResNet-bottleneck-164 and Wide ResNet-28-10 (`https://github.com/tensorflow/models/tree/master/research/resnet`);
- ResNet-158 (`https://github.com/tensorflow/models/tree/master/official/resnet`);
- A reimplementation of VGG-16 without batch-normalization from (`https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py`);

Table A.1 summarizes the results of the curve finding experiments with all datasets and architectures. For each of the models we report the properties of loss and the error on the train and test

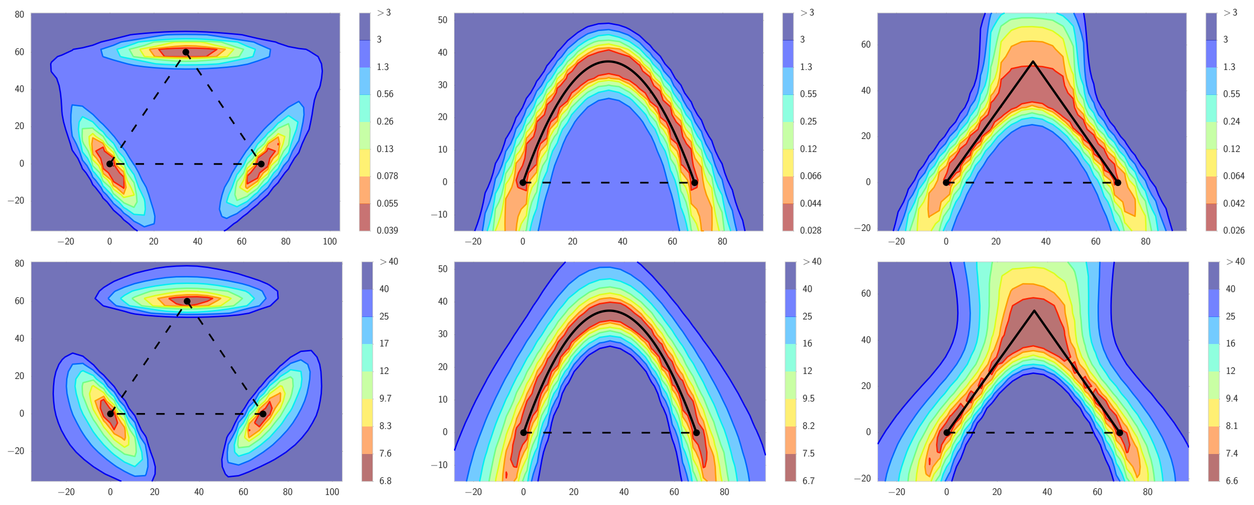


**Figure A.2:** Same as Fig. A.1 for VGG-16 on CIFAR-10.

datasets. For each of these metrics we report 3 values: "Max" is the maximum values of the metric along the curve, "Int" is a numerical approximation of the integral $\int$ <metric>$(\phi_\theta)d\phi_\theta/\int d\phi_\theta$, where <metric> represents the train loss or the error on the train or test dataset and "Min" is the minimum value of the error on the curve. "Int" represents a mean over a uniform distribution on the curve, and for the train loss it **coincides with the loss** (1) in the paper. We use an equally-spaced grid with 121 points on $[0, 1]$ to estimate the values of "Min", "Max", "Int". For "Int" we use the trapezoidal rule to estimate the integral. For each dataset and architecture we report the performance of single models used as the endpoints of the curve as "Single", the performance of a line segment connecting the two single networks as "Segment", the performance of a quadratic Bezier curve as "Bezier" and the performance of a polygonal chain with one bend as "Polychain". Finally, for each curve we report the ratio of its length to the length of a line segment connecting the two modes.

We also examined the quantity "Mean" defined as $\int < \text{metric} > (\phi_\theta(t))dt$, which **coincides with the loss** (2) from the paper, but in all our experiments it is nearly equal to "Int".

Besides convolutional and fully-connected architectures we also apply our approach to RNN architecture on next word prediction task, PTB dataset (Marcus et al. [1993]). As a base model we used the implementation available at https://www.tensorflow.org/tutorials/recurrent. As the main loss we consider perplexity. The results are presented in Table A.2.

### A.1.5 Train loss and test accuracy surfaces

In this section we provide additional visualizations. Fig. A.1 and Fig. A.2 show visualizations of the train loss and test accuracy for ResNet-164 on CIFAR-100 and VGG-16 on CIFAR-10.

### A.1.6 Curve Ensembling

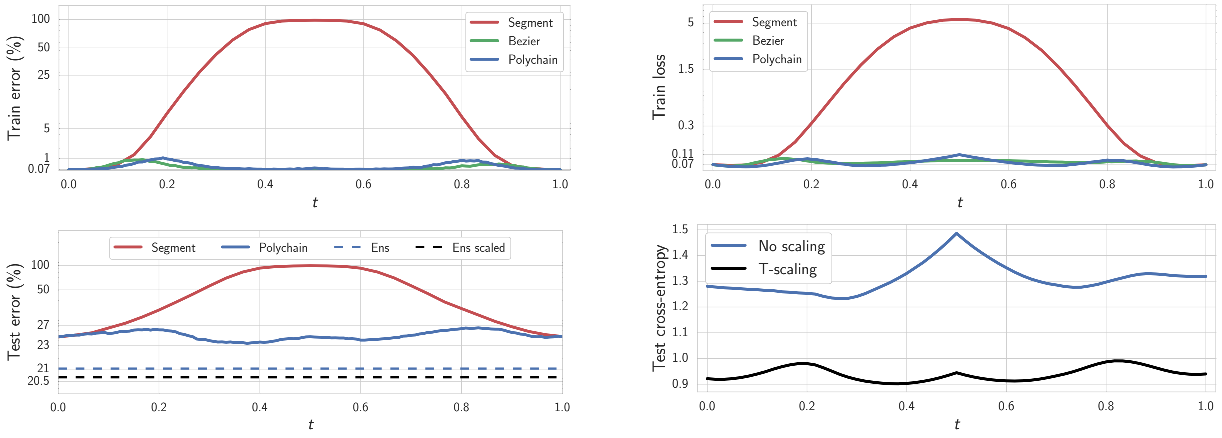


**Figure A.3:** Error as a function of the point on the curves $\phi_\theta(t)$ found by the proposed method, using a ResNet-164 on CIFAR-100. **Top left:** train error. **Bottom left:** test error; dashed lines correspond to quality of ensemble constructed from curve points before and after logits rescaling. **Top right:** train loss ($\ell_2$ regularized cross-entropy). **Bottom right:** cross-entropy before and after logits rescaling for the polygonal chain.

Here we explore ensembles constructed from points sampled from these high accuracy curves. In particular, we train a polygonal chain with one bend connecting two independently trained ResNet-164 networks on CIFAR-100 and construct an ensemble of networks corresponding to 50 points placed on an equally-spaced grid on the curve. The resulting ensemble had 21.03% error-rate on the test dataset. The error-rate of the ensemble constructed from the endpoints of the curve was 22.0%. An ensemble of three independently trained networks has an error rate of 21.01%. Thus, the ensemble of the networks on the curve outperformed an ensemble of its endpoints implying that the curves found by the proposed method are actually passing through diverse networks that produce predictions different from those produced by the endpoints of the curve. Moreover, the ensemble based on the polygonal chain has the same number of parameters as three independent networks, and comparable performance.

Furthermore, we can improve the ensemble on the chain without adding additional parameters or computational expense, by accounting for the pattern of increased training and test loss towards the centres of the linear paths shown in Figure A.3. While the training and test accuracy are relatively constant, the pattern of loss, shared across train and test sets, indicates overconfidence away from the three points defining the curve: in this region, networks tend to output probabilities closer to 1, sometimes with the wrong answers. This overconfidence decreases the performance of ensembles constructed from the networks sampled on the curves. In order to correct for this overconfidence and improve the ensembling performance we use temperature scaling [Guo et al. 2017], which is inversely proportional to the loss. Figure A.3, bottom right, illustrates the test loss of ResNet-164 on CIFAR-100 before and after temperature scaling. After rescaling the predictions of the networks, the test loss along the curve decreases and flattens. Further, the test error-rate of the ensemble constructed from the points on the curve went down from 21.03% to 20.7% after applying the temperature scaling, outperforming 3 independently trained networks.

However, directly ensembling on the curves requires manual intervention for temperature scaling, and an additional pass over the training data for each of the networks (50 in this case) at test time to perform batch normalization as described in section A.1.2. Moreover, we also need to train at least two networks for the endpoints of the curve.

### A.1.7 The Effects of Increasing Parametrization

One possible factor that influences the connectedness of a local minima set is the overparameterization of neural networks. In this section, we investigate the relation between the observed

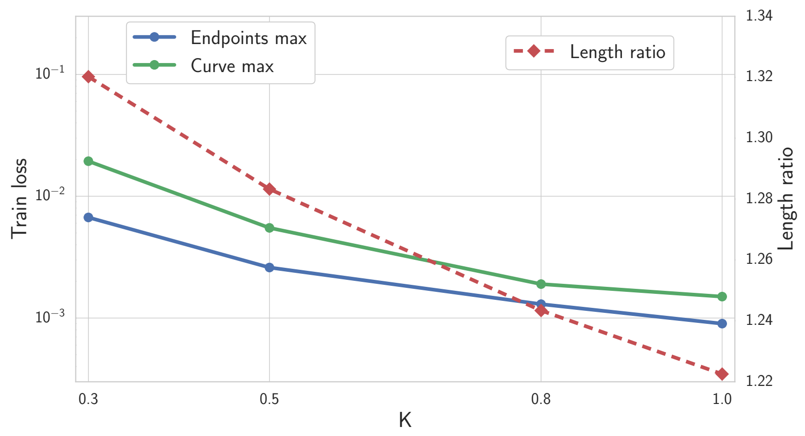


**Figure A.4:** The worst train loss along the curve, maximum of the losses of the endpoints, and the ratio of the length of the curve and the line segment connecting the two modes, as a function of the scaling factor $K$ of the sizes of fully-connected layers.

connectedness of the local optima and the number of parameters (weights) in the neural network. We start with a network that has three convolutional layers followed by three fully-connected layers, where each layer has $1000K$ neurons. We vary $K \in \{0.3, 0.5, 0.8, 1\}$, and for each value of $K$ we train two networks that we connect with a Bezier curve using the proposed procedure.

For each value of $K$, Figure A.4 shows the worst training loss along the curve, maximum of losses of the endpoints, and the ratio of the length of the curve and the line segment connecting the two modes. Increasing the number of parameters we are able to reduce the difference between the worst value of the loss along the curve and the loss of single models used as the endpoints. The ratio of the length of the found curve and the length of the line segment connecting the two modes also decreases monotonically with $K$. This result is intuitive, since a greater parametrization allows for more flexibility in how we can navigate the loss surfaces.

### A.1.8 Trivial connecting curves

For convolutional networks with ReLU activations and without batch normalization we can construct a path connecting two points in weight space such that the accuracy of each point on the curve (excluding the origin of the weight space) is at least as good as the minimum of the accura-

**Algorithm 3** Fast Geometric Ensembling

```
Require:
   weights $\hat{w}$, LR bounds $\alpha_1, \alpha_2$,
   cycle length $c$ (even), number of iterations $n$
Ensure: ensemble
   $w \leftarrow \hat{w}$ {Initialize weight with $\hat{w}$}
   ensemble $\leftarrow$ [ ]
   for $i \leftarrow 1, 2, \ldots, n$ do
      $\alpha \leftarrow \alpha(i)$ {Calculate LR for the iteration}
      $w \leftarrow w - \alpha \nabla \mathcal{L}_i(w)$ {Stochastic gradient update}
      if $\mathrm{mod}(i, c) = c/2$ then
         ensemble $\leftarrow$ ensemble + $[w]$ {Collect weights}
      end if
   end for
```

cies of the endpoints. Unlike the paths found by our procedure, these paths are trivial and merely exploit redundancies in the parametrization. Also, the training loss goes up substantially along these curves. Below we give a construction of such paths.

Let $\hat{w}_1$ and $\hat{w}_2$ be two sets of weights. This path of interest consists of two parts. The first part connects the point $\hat{w}_1$ with 0 and the second one connects the point $\hat{w}_2$ with 0. We describe only the first part $\phi(t)$ of the path, such that $\phi(0) = 0, \phi(1) = \hat{w}_1$, as the second part is completely analogous. Let the weights of the network $\hat{w}_1$ be $\{W_i, b_i\}_{1 \le i \le n}$ where $W_i, b_i$ are the weights and biases of the $i$-th layer, and $n$ is the total number of layers. Throughout the derivation we consider the inputs of the network fixed. The output of the $i$-th layer $o_i = W_i \mathrm{ReLU}(o_{i-1}) + b_i$, $1 \le i \le n$, where $i = 0$ corresponds to the first layer and $i = n$ corresponds to logits (the outputs of the last layer). We construct $\phi(t) = \{W_i(t), b_i(t)\}_{1 \le i \le n}$ in the following way. We set $W_i(t) = W_i t$ and $b_i(t) = b_i t^i$. It is easy to see that logits of the network with weights $\phi(t)$ are equal to $o_n(t) = t^n o_n$ for all $t > 0$. Note that the predicted labels corresponding to the logits $o_n(t)$ and $o_n$ are the same, so the accuracy of all networks corresponding to $t > 0$ is the same.

### A.1.9 Fast geometric ensembling experiments

In this section we compare the proposed Fast Geometric Ensembling (**FGE**) technique against ensembles of independently trained networks (**Ind**), and SnapShot Ensembles (**SSE**) [Huang et al. 2017a], a recent state-of-the-art fast ensembling approach. Alg. 3 provides an outline of the FGE algorithm.

For the ensembling experiments we use a 164-layer Preactivation-ResNet in addition to the VGG-16 and Wide ResNet-28-10 models. As baseline models we used the following implementations:

- VGG-16 (`https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py`);
- Preactivation-ResNet-164 (`https://github.com/bearpaw/pytorch-classification/blob/master/models/cifar/preresnet.py`);
- ResNet-50 ImageNet (`https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py`);
- Wide ResNet-28-10 (`https://github.com/meliketoy/wide-resnet.pytorch/blob/master/networks/wide_resnet.py`);

We compare the accuracy of each method as a function of computational budget. For each network architecture and dataset we denote the number of epochs required to train a single model as $B$. For a $kB$ budget, we run each of Ind, FGE and SSE $k$ times from random initializations and ensemble the models gathered from the $k$ runs. In our experiments we set $B = 200$ for VGG-16 and Wide ResNet-28-10 (WRN-28-10) models, and $B = 150$ for ResNet-164, since 150 epochs is typically sufficient to train this model. We note the runtime per epoch for FGE, SSE, and Ind is the same, and so the total computation associated with $kB$ budgets is the same for all ensembling approaches.

**Table A.3:** Error rates (%) on CIFAR-100 and CIFAR-10 datasets for different ensembling techniques and training budgets. The best results for each dataset, architecture, and budget are **bolded**.

| | | CIFAR-100 | | | CIFAR-10 | | |
|---|---|---|---|---|---|---|---|
| DNN (Budget) | method | $1B$ | $2B$ | $3B$ | $1B$ | $2B$ | $3B$ |
| VGG-16 (200) | Ind | 27.4 ± 0.1 | 25.28 | 24.45 | 6.75 ± 0.16 | 5.89 | 5.9 |
| | SSE | 26.4 ± 0.1 | 25.16 | 24.69 | 6.57 ± 0.12 | 6.19 | 5.95 |
| | FGE | **25.7 ± 0.1** | **24.11** | **23.54** | **6.48 ± 0.09** | **5.82** | **5.66** |
| ResNet-164 (150) | Ind | 21.5 ± 0.4 | 19.04 | 18.59 | 4.72 ± 0.1 | **4.1** | **3.77** |
| | SSE | 20.9 ± 0.2 | 19.28 | 18.91 | 4.66 ± 0.02 | 4.37 | 4.3 |
| | FGE | **20.2 ± 0.1** | **18.67** | **18.21** | **4.54 ± 0.05** | 4.21 | 3.98 |
| WRN-28-10 (200) | Ind | 19.2 ± 0.2 | 17.48 | 17.01 | 3.82 ± 0.1 | 3.4 | **3.31** |
| | SSE | 17.9 ± 0.2 | 17.3 | 16.97 | 3.73 ± 0.04 | 3.54 | 3.55 |
| | FGE | **17.7 ± 0.2** | **16.95** | **16.88** | **3.65 ± 0.1** | **3.38** | 3.52 |

For the FGE (Fast Geometric Ensembling) strategy on ResNet we run the FGE routine summarized in Alg. 1 after epoch 125 of the usual (same as Ind) training for 22 epochs. The total training time is thus $125 + 22 = 147$ epochs. For VGG and Wide ResNet models we run the pre-training procedure for 156 epochs to initialize FGE. Then we run FGE for 22 epochs starting from checkpoints corresponding to epochs 120 and 156 and ensemble all the gathered models. The total training time is thus $156 + 22 + 22 = 200$ epochs. For VGG we use cycle length $c = 2$ epochs, which means that the total number of models in the final ensemble is 22. For ResNet and Wide ResNet we use $c = 4$ epochs, and the total number of models in the final ensemble is 12 for Wide ResNets and 6 for ResNets.

For Ind, we use an initial learning rate of 0.1 for ResNet and Wide ResNet, and 0.05 for VGG. For FGE, with VGG we use cycle length $c = 2$ epochs, and a total of 22 models in the final ensemble. With ResNet and Wide ResNet we use $c = 4$ epochs, and the total number of models in the final ensemble is 12 for Wide ResNets and 6 for ResNets. For VGG we set the learning rates to $\alpha_1 = 10^{-2}$, $\alpha_2 = 5 \cdot 10^{-4}$; for ResNet and Wide ResNet models we set $\alpha_1 = 5 \cdot 10^{-2}$, $\alpha_2 = 5 \cdot 10^{-4}$. For SSE, we followed Huang et al. [2017a] and varied the initial learning rate $\alpha_0$ and number of snapshots per run $M$. We report the best results we achieved, which corresponded to

$\alpha_0 = 0.1, M = 4$ for ResNet, $\alpha_0 = 0.1, M = 5$ for Wide ResNet, and $\alpha_0 = 0.05, M = 5$ for VGG. The total number of models in the FGE ensemble is constrained by network choice and computational budget. Further experimental details are in the Appendix A.1.9.

Table A.3 summarizes the results of the experiments. In all conducted experiments FGE outperforms SSE, particularly as we increase the computational budget. The performance improvement against Ind is most noticeable for CIFAR-100. With a large number of classes, any two models are less likely to make the same predictions. Moreover, there will be greater uncertainty over which representation one should use on CIFAR-100, since the number of classes is increased tenfold from CIFAR-10, but the number of training examples is held constant. Thus smart ensembling strategies will be especially important on this dataset. Indeed in all experiments on CIFAR-100, FGE outperformed all other methods. On CIFAR-10, FGE consistently improved upon SSE for all budgets and architectures. FGE also improved against Ind for all training budgets with VGG, but is more similar in performance to Ind on CIFAR-10 when using ResNets.

Prediction diversity. Diversity of predictions of the individual networks is crucial for the ensembling performance [e.g., Lee et al. 2016b]. We note that the diversity of the networks averaged by FGE is lower than that of completely independently trained networks. Specifically, two independently trained ResNet-164 on CIFAR-100 make different predictions on 19.97% of test objects, while two networks from the same FGE run make different predictions on 14.57% of test objects. Further, performance of individual networks averaged by FGE is slightly lower than that of fully trained networks (e.g. 78.0% against 78.5% on CIFAR100 for ResNet-164). However, for a given computational budget FGE can propose many more high-performing networks than independent training, leading to better ensembling performance (see Table A.3).

ImageNet results. ImageNet ILSVRC-2012 [Russakovsky et al. 2012] is a large-scale dataset containing 1.2 million training images and 50000 validation images divided into 1000 classes. CIFAR-100 is the primary focus of our ensemble experiments. However, we also include ImageNet

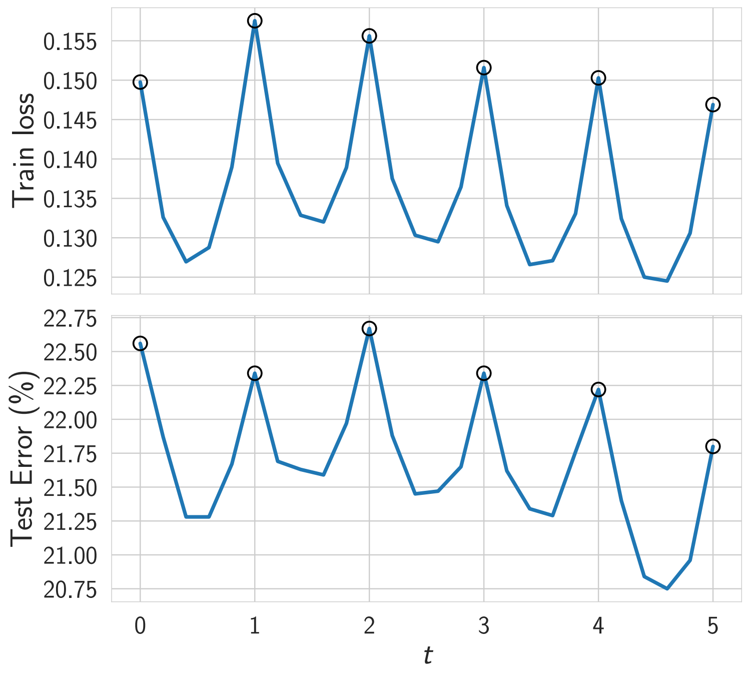


**Figure A.5:** Train loss and test error along the polygonal chain connecting the sequence of points ensembled in FGE. The plot is generated using PreResNet-164 on CIFAR 100. Circles indicate the bends on the polygonal chain, i.e. the networks ensembled in FGE.

results for the proposed FGE procedure, using a ResNet-50 architecture. We used a pretrained model with top-1 test error of 23.87 to initialize the FGE procedure. We then ran FGE for 5 epochs with a cycle length of 2 epochs and with learning rates $\alpha_1 = 10^{-3}$, $\alpha_2 = 10^{-5}$. The top-1 test error-rate of the final ensemble was 23.31. Thus, in just 5 epochs we could improve the accuracy of the model by 0.56 using FGE. The final ensemble contains 4 models (including the pretrained one). Despite the harder setting of only 5 epochs to construct an ensemble, FGE performs comparably to the best result reported by Huang et al. [2017a] on ImageNet, 23.33 error, which was also achieved using a ResNet-50.

### A.1.10 Polygonal chain connecting FGE proposals

In order to better understand the trajectories followed by FGE we construct a polygonal chain connecting the points that FGE ensembles. Suppose we run FGE for $n$ learning rate cycles obtaining $n$ points $w_1, w_2, \ldots, w_n$ in the weight space that correspond to the lowest values of the learning rate. We then consider the polygonal chain consisting of the line segments connecting $w_i$ to $w_{i+1}$ for $i = 1, \ldots, n-1$. We plot test accuracy and train error along this polygonal chain in Figure A.5. We observe that along this curve both train loss and test error remain low, agreeing with our intuition that FGE follows the paths of low loss and error. Surprisingly, we find that the points on the line segments connecting the weights $w_i, w_{i+1}$ have lower train loss and test error than $w_i$ and $w_{i+1}$. See Izmailov et al. [2018] for a detailed discussion of this phenomenon.

## A.2 Appendix for Stochastic Weight Averaging

This supplementary material is structured as follows. In Section A.2.1 we describe the details of the experiments with SWA. In Section A.2.2 how SWA can be used to train a neural network from scratch with a constant learning rate schedule.

### A.2.1 Experimental Details

For the experiments on CIFAR datasets (section 2.6.1) we used the following implementations (embedded links):

- Shake-Shake-2x64d
- PyramidNet-272
- VGG-16
- Preactivation-ResNet-164
- Wide ResNet-28-10

Models for ImageNet are from here. Pretrained networks can be found here.

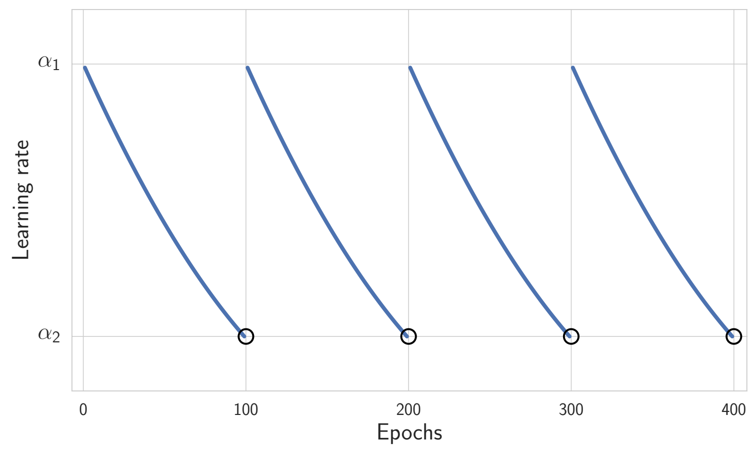


**Figure A.6:** Cyclical learning rate used for Shake-Shake as a function of iteration.

SWA LEARNING RATES. For PyramidNet SWA uses a cyclic learning rate with $\alpha_1 = 0.05$ and $\alpha_2 = 0.001$ and cycle length 3. For VGG and Wide ResNet we used constant learning $\alpha_1 = 0.01$. For ResNet we used constant learning rates $\alpha_1 = 0.01$ on CIFAR-10 and 0.05 on CIFAR-100.

For Shake-Shake Net we used a custom cyclic learning rate based on the cosine annealing used when training Shake-Shake with SGD. Each of the cycles replicate the learning rates corresponding to epochs $1600 - 1700$ of the standard training and the cycle length $c = 100$ epochs. The learning rate schedule is depicted in Figure A.6 and follows the formula

$$\alpha(i) = 0.1 \cdot \left(1 + \cos\left(\pi \cdot \frac{1600 + \text{epoch}(i) \bmod 100)}{1800}\right)\right),$$

where epoch(i) is the number of data passes completed before iteration $i$.

For all experiments with ImageNet we used cyclic learning rate schedule with the same hyperparameters $\alpha_1 = 0.001$, $\alpha_2 = 10^{-5}$ and $c = 1$.

SGD LEARNING RATES. For conventional SGD training we used SGD with momentum 0.9 and with an annealed learning rate schedule. For VGG, Wide ResNet and Preactivation ResNet we fixed the learning rate to $\alpha_1$ for the first half of epochs ($0B$–$0.5B$), then linearly decreased the learning rate to $0.01\alpha_1$ for the next 40% of epochs ($0.5B$–$0.9B$), and then kept it constant for the

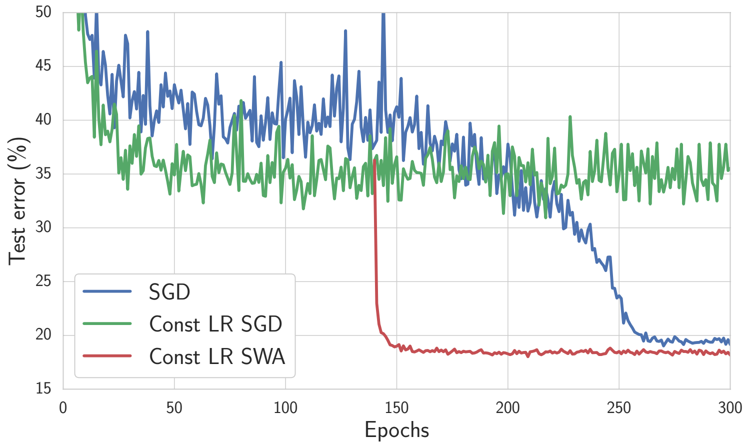


**Figure A.7:** Test error as a function of training epoch for constant (green) and decaying (blue) learning rate schedules for a Wide ResNet-28-10 on CIFAR-100. In red we average the points along the trajectory of SGD with constant learning rate starting at epoch 140.

last 10% of epochs ($0.9B - 1B$). For VGG we set $\alpha_1 = 0.05$, and for Preactivation ResNet and Wide ResNet we set $\alpha_1 = 0.1$. For Shake-Shake Net and PyramidNets we used the cosine and piecewise-constant learning rate schedules described in Gastaldi [2017] and Han et al. [2016] respectively.

### A.2.2 Training with a Constant Learning Rate

In this section we show that it is possible to train DNNs from scratch with a fixed learning rate using SWA. We run SGD with a fixed learning rate of 0.05 on a Wide ResNet-28-10 [Zagoruyko and Komodakis 2016] for 300 epochs from a random initialization on CIFAR-100. We then averaged the weights at the end of each epoch from epoch 140 and until the end of training. The final test accuracy of this SWA model was 81.7.

Figure A.7 illustrates the test error as a function of the number of training epochs for SWA and conventional training. The accuracy of the individual models with weights averaged by SWA stays at the level of $\approx$ 65% which is 16% less than the accuracy of the SWA model. These results correspond to our intuition presented in section 3.4 that SGD with a constant learning rate oscillates around the optimum, but SWA converges.

While being able to train a DNN with a fixed learning rate is a surprising property of SWA, for practical purposes we recommend initializing SWA from a model pretrained with conventional training (possibly for a reduced number of epochs), as it leads to faster and more stable convergence than running SWA from scratch.

## A.3 Appendix for SWA-Gaussian

[$^{P}_{I}$ ToDo]

### A.3.1 Asymptotic Normality of SGD

Under conditions of decaying learning rates, smoothness of gradients, and the existence of a full rank stationary distribution, martingale based analyses of stochastic gradient descent [e.g., Asmussen and Glynn 2007, Chapter 8] show that SGD has a Gaussian limiting distribution. That is, in the limit as the time step goes to infinity, $t^{1/2}(\theta_t - \theta^*) \to \mathcal{N}(0, \mathcal{H}(\theta)^{-1}\mathbb{E}(\nabla \log p(\theta) \nabla \log p(\theta)^T)\mathcal{H}(\theta)^{-1}))$, where $\mathcal{H}(\theta)^{-1}$ is the inverse of the Hessian matrix of the log-likelihood and $\mathbb{E}(\nabla \log p(\theta) \nabla \log p(\theta)^T)$ is the covariance of the gradients and $\theta^*$ is a stationary point or minima. Note that these analyses date back to Ruppert [1988] and Polyak and Juditsky [1992] for Polyak-Ruppert averaging, and are still popular in the analysis of stochastic gradient descent.

Mandt et al. [2017a], Chen et al. [2016], and Babichev and Bach [2018] all use the same style of analyses, but for different purposes. We will test the specific assumptions of Mandt et al. [2017a] in the next section. Finally, note that the technical conditions are essentially the same conditions as for the Bernstein von Mises Theorem [e.g., Van der Vaart 2000, Chapter 10] which implies that the asymptotic posterior will also be Gaussian.

It may be counter-intuitive that, as we show in Section 3.4, SWAG captures the geometry of the objective correctly. One might even expect SWAG estimates of variance to be inverted, as gradient descent would oscillate more in the sharp directions of the objective. To gain more intuition about SGD dynamics we visualize SGD trajectory on a quadratic problem. More precisely, we define a

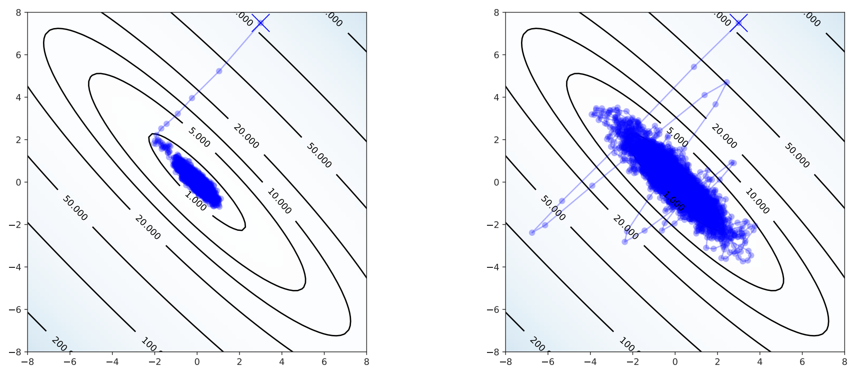


**Figure A.8:** Trajectory of SGD with isotropic Gaussian gradient noise on a quadratic loss function. **Left**: SGD without momentum; **Right**: SGD with momentum.

2-dimensional quadratic function $f(x, y) = (x+y)^2 + 0.05 \cdot (x-y)^2$ shown in Figure A.8. We then run SGD to minimize this function.

It turns out that the gradient noise plays a crucial role in forming the SGD stationary distribution. If there is no noise in the gradients, we are in the full gradient descent regime, and optimization either converges to the optimizer, or diverges to infinity depending on the learning rate. However, when we add isotropic Gaussian noise to the gradients, SGD converges to the correct Gaussian distribution, as we visualize in the left panel of Figure A.8. Furthermore, adding momentum affects the scale of the distribution, but not its shape, as we show in the right panel of Figure A.8. These conclusions hold as long as the learning rate in SGD is not too large.

The results we show in Figure A.8 are directly predicted by theory in Mandt et al. [2017a]. In general, if the gradient noise is not isotropic, the stationary distribution of SGD would be different from the exact posterior distribution. Mandt et al. [2017a] provide a thorough empirical study of the SGD trajectory for convex problems, such as linear and logistic regression, and show that SGD can often provide a competitive baseline on these problems.

### A.3.1.1 Other Considerations for Gaussian Approximation

Given the covariance matrix $A = \mathcal{H}(\theta)^{-1}\mathbb{E}(\nabla \log p(\theta) \nabla \log p(\theta)^T)\mathcal{H}(\theta)^{-1}$, Chen et al. [2016] show that a batch means estimator of the iterates (similar to what SWAG uses) themselves will converge to $A$ in the limit of infinite time. We tried batch means based estimators but saw no improvement; however, it could be interesting to explore further in future work.

Intriguingly, the covariance $A$ is the same form as sandwich estimators [see e.g. Müller 2013, for a Bayesian analysis in the model mis-specification setting], and so $A = \mathcal{H}(\theta)^{-1}$ under model well-specification [Müller 2013; Chen et al. 2016]. We then tie the covariance matrix of the iterates back to the well known Laplace approximation, which uses $\mathcal{H}(\theta)^{-1}$ as its covariance as described by MacKay [2003, Chapter 28], thereby justifying SWAG theoretically as a sample based Laplace approximation.

Finally, in Chapter 4 of Berger [2013] constructs an example (Example 10) of fitting a Gaussian approximation from a MCMC chain, arguing that it empirically performs well in Bayesian decision theoretic contexts. Berger [2013] give the explanation for this as the Bernstein von Mises Theorem providing that in the limit the posterior will itself converge to a Gaussian. However, we would expect that even in the infinite data limit the posterior of DNNs would converge to something very non-Gaussian, with connected modes surrounded by gorges of zero posterior density [Garipov et al. 2018]. One could use this justification for fitting a Gaussian from the iterates of SGLD or SGHMC instead.

### A.3.2 Do the assumptions of Mandt et al. [2017a] hold for DNNs?

In this section, we investigate the results of Mandt et al. [2017a] in the context of deep learning. Mandt et al. [2017a] uses the following assumptions:

1. Gradient noise at each point $\theta$ is $\mathcal{N}(0, C)$.
2. $C$ is independent of $\theta$ and full rank.
3. The learning rates, $\eta$, are small enough that we can approximate the dynamics of SGD by a continuous-time dynamic described by the corresponding stochastic differential equation.

4. In the stationary distribution, the loss is approximately quadratic near the optima, i.e. approximately $(\theta - \theta^*)^\top \mathcal{H}(\theta)(\theta - \theta^*)$, where $\mathcal{H}(\theta^*)$ is the Hessian at the optimum; further, the Hessian is assumed to be positive definite.

Assumption 1 is motivated by the central limit theorem, and Assumption 3 is necessary for the analysis in Mandt et al. [2017a]. Assumptions 2 and 4 may or may not hold for deep neural networks (as well as other models). Under these assumptions, Theorem 1 of Mandt et al. [2017a] derives the optimal constant learning rate that minimizes the KL-divergence between the SGD stationary distribution and the posterior[1]:

$$\eta^* = 2\frac{B}{N}\frac{d}{tr(C)}, \tag{A.1}$$

where $N$ is the size of the dataset, $d$ is the dimension of the model, $B$ is the minibatch size and $C$ is the gradient noise covariance.

We computed Equation A.1 over the course of training for two neural networks in Figure A.A.9(a), finding that the predicted optimal learning rate was an order of magnitude larger than what would be used in practice to explore the loss surface in a reasonable time (about 4 compared to 0.1).

We now focus on seeing how Assumptions 2 and 4 fail for DNNs; this will give further insight into what portions of the theory do hold, and may give insights into a corrected version of the optimal learning rate.

#### A.3.2.1 Assumption 2: Gradient Covariance Noise.

In Figure A.A.9(b), the trace of the gradient noise covariance and thus the optimal learning rates *are* nearly constant; however, the total variance is much too small to induce effective learning rates, probably due to over-parameterization effects inducing non full rank gradient covariances

[1]An optimal diagonal preconditioner is also derived; our empirical work applies to that setting as well. A similar analysis with momentum holds as well, adding in only the momentum coefficient.

as was found in Chaudhari and Soatto [2018]. We note that this experiment is not sufficient to be fully confident that $C$ is independent of the parameterization near the local optima, but rather that $tr(C)$ is close to constant; further experiments in this vein are necessary to test if the diagonals of $C$ are constant. The result that $tr(C)$ is close to constant suggests that a constant learning rate could be used for sampling in a stationary phase of training. The dimensionality parameter in Equation A.1 could be modified to use the number of effective parameters or the rank of the gradient noise to reduce the optimal learning rate to a feasible number.

To estimate $tr(C)$ from the gradient noise we need to divide the estimated variance by the batch size (as $V(\hat{g}(\theta)) = BC(\theta)$), for a correct version of Equation A.1. From Assumption 1 and Equation 6 of Mandt et al. [2017a], we see that

$$\hat{g}(\theta) \approx g(\theta) + \frac{1}{\sqrt{B}} \nabla g(\theta), \nabla g(\theta) \sim N(0, C(\theta)),$$

where $B$ is the batch size. Thus, collecting the variance of $\hat{g}(\theta)$ (the variance of the stochastic gradients) will give estimates that are upscaled by a factor of $B$, leading to a cancellation of the batch size terms:

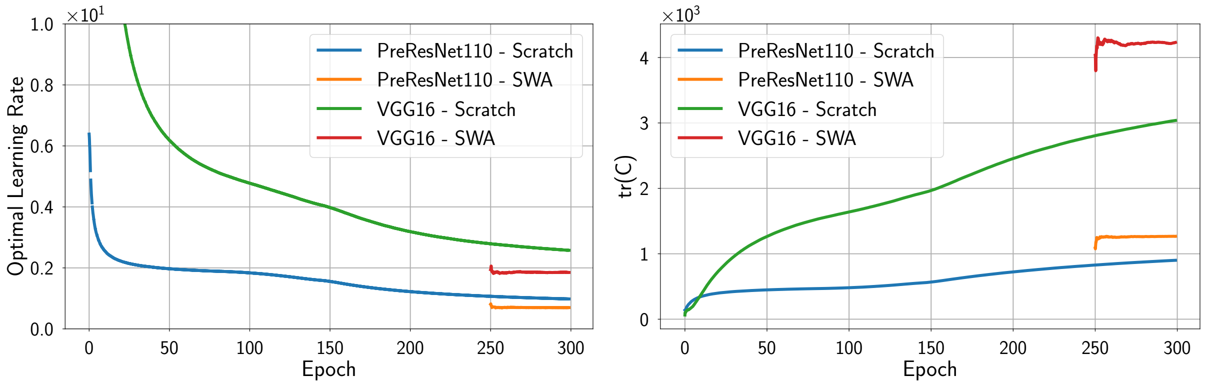
$$\eta \approx \frac{2}{N} \frac{d}{tr(V(\hat{g}(\theta)))}.$$

To include momentum, we can repeat the analysis in Sections 4.1 and 4.3 of Mandt et al. [2017a] finding that this also involves scaling the optimal learning rate but by a factor of $\mu$, the momentum term.[2] This gives the final optimal learning rate equation as

$$\eta \approx \frac{2\mu}{N} \frac{d}{tr(V(\hat{g}(\theta)))}. \tag{A.2}$$

In Figure A.9(b), we computed $tr(C)$ for VGG-16 and PreResNet-164 on CIFAR-100 beginning from the start of training (referred to as from scratch), as well as the start of the SWAG procedure

[2]Our experiments used $\mu = 0.1$ corresponding to $\rho = 0.9$ in PyTorch's SGD implementation.



**(a)** Optimal learning rate. **(b)** $tr(C)$

**Figure A.9:** Gradient variance norm and computed optimal learning rates for VGG-16 and PreResNet-164. The computed optimal learning rates are always too large by a factor of 10, while the gradient variance stabilizes over the course of training.

(referred to in the legend as SWA). We see that $tr(C)$ is never quite constant when trained from scratch, while for a period of constant learning rate near the end of training, referred to as the stationary phase, $tr(C)$ is essentially constant throughout. This discrepancy is likely due to large gradients at the very beginning of training, indicating that the stationary distribution has not been reached yet.

Next, in Figure A.9(a), we used the computed $tr(C)$ estimate for all four models and Equation A.2 to compute the optimal learning rate under the assumptions of Mandt et al. [2017a], finding that these learning rates are not constant for the estimates beginning at the start of training and that they are too large (1-3 at the minimum compared to a standard learning rate of 0.1 or 0.01).

#### A.3.2.2 Assumption 4: Hessian Eigenvalues at the Optima

To test assumption 4, we used a GPU-enabled Lanczos method from GPyTorch [Gardner et al. 2018] and used restarting to compute the minimum eigenvalue of the train loss of a pre-trained PreResNet-164 on CIFAR-100. We found that even at the end of training, the minimum eigenvalue was $-272$ (the maximum eigenvalue was 3580 for comparison), indicating that the Hessian is not positive definite. This result harmonizes with other work analyzing the spectra of the Hessian for

DNN training [Li et al. 2017; Sagun et al. 2018]. Further, Garipov et al. [2018] and Draxler et al. [2018a] argue that the loss surfaces of DNNs have directions along which the loss is completely flat, suggesting that the loss is nowhere near a positive-definite quadratic form.

### A.3.3 Further Geometric Experiments

In Figure A.10 we present figures/swag analogous to those in Section 3.4 for PreResNet-110 and VGG-16 on CIFAR-10 and CIFAR-100. For all dataset-architecture pairs we see that SWAG is able to capture the geometry of the posterior in the subspace spanned by SGD trajectory.

### A.3.4 Hyper-Parameters and Limitations

In this section, we discuss the hyper-parameters in SWAG, as well as some current theoretical limitations.

### A.3.5 Rank of Covariance Matrix

We now evaluate the effect of the covariance matrix rank on the SWAG approximation. To do so, we trained a PreResNet56 on CIFAR-100 with SWAG beginning from epoch 161, and evaluated 30 sample Bayesian model averages obtained at different epochs; the accuracy plot from this experiment is shown in Figure A.11 (a). The rank of each model after epoch 161 is simply $\min(epoch - 161, 140)$, and 30 samples from even a low rank approximation reach the same predictive accuracy as the SWA model. Interestingly, both SWAG and SWA outperform ensembles of a SGD run and ensembles of the SGD models in the SWA run.

#### A.3.5.1 Number of Samples in the Forwards Pass

In most situations where SWAG will be used, no closed form expression for the integral $\int f(y)q(\theta|y)d\theta$, will exist. Thus, Monte Carlo approximations will be used; Monte Carlo integration converges

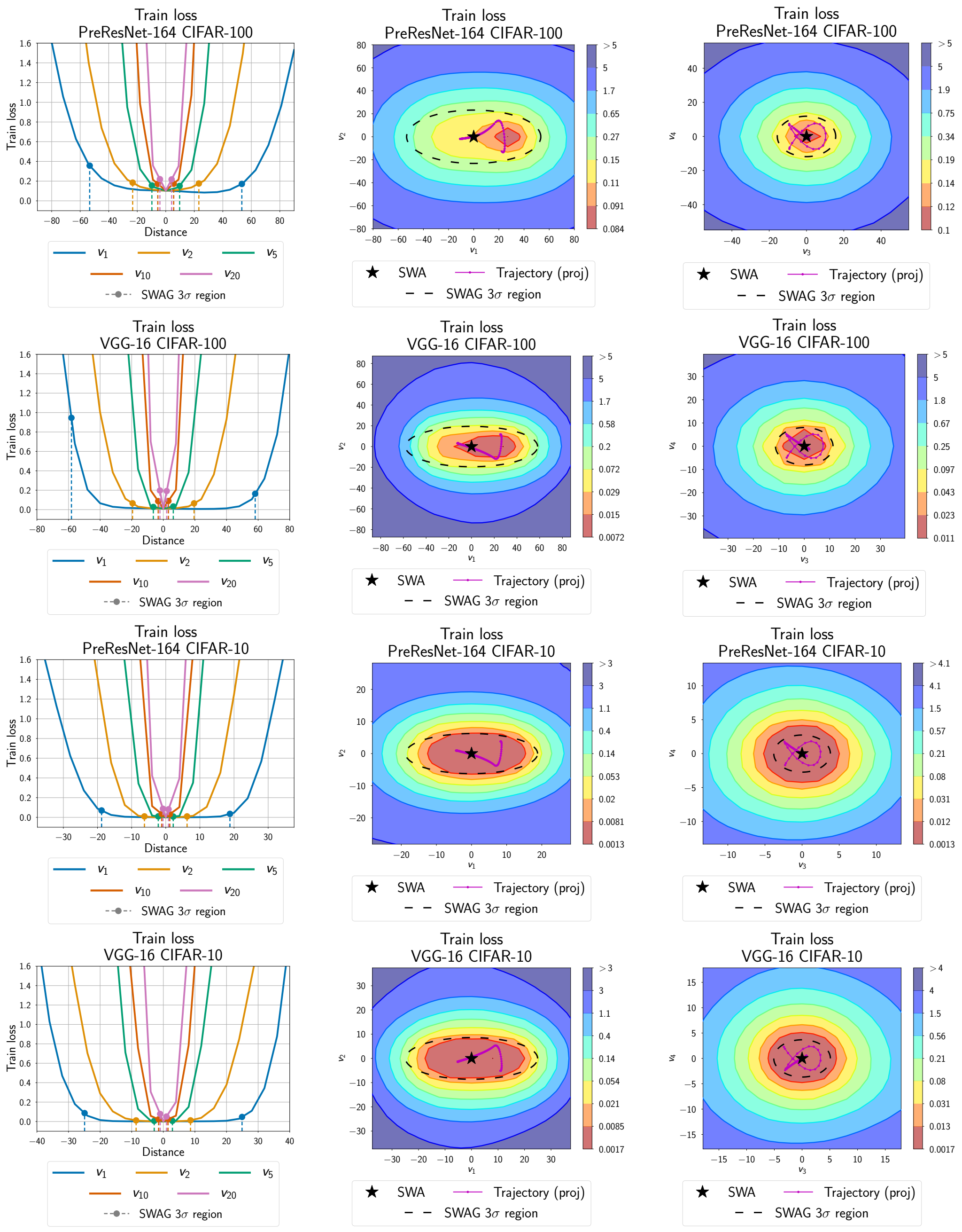


**Figure A.10: Left:** Posterior-density cross-sections along the rays corresponding to different eigenvectors of the SWAG covariance matrix. **Middle:** Posterior-density surface in the plane spanned by eigenvectors of SWAG covariance matrix corresponding to the first and second largest eigenvalues and (**Right:**) the third and fourth largest eigenvalues. Each row in the figure corresponds to an architecture-dataset pair indicated in the title of each panel.

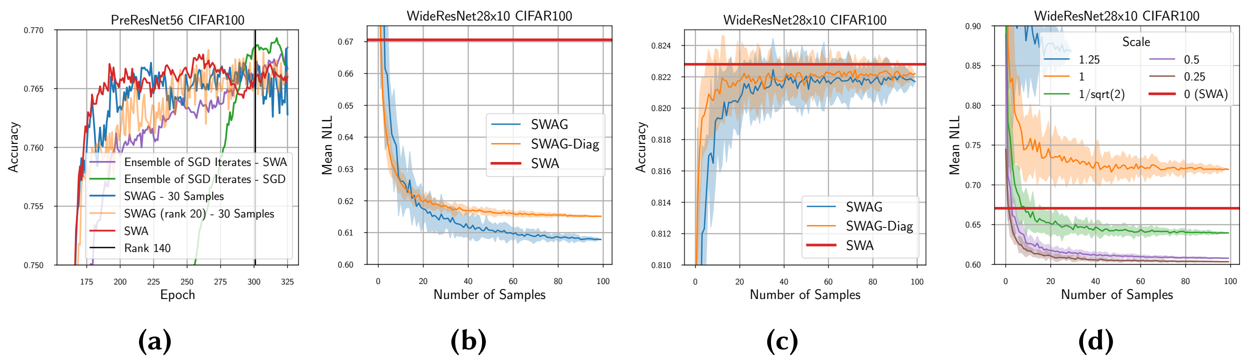


**Figure A.11: (a)** 30 samples of SWAG with a rank 20 covariance matches the SWA result over the course of training for PreResNet56 on CIFAR-100. SWAG with a rank of 140, SWAG with a rank of 20, and SWA all outperform ensembles of SGD iterates from the SWA procedure and from a standard SGD training path. **(b)** NLL and **(c)** accuracy by number of samples for WideResNet on CIFAR-100 for SWAG, SWAG-Diag, and SWA. 30 samples is adequate for stable accuracies and NLLs. **(d)** NLL by number of samples for different scales for WideResNet on CIFAR-100 for SWAG, SWAG-Diag, and SWA. Scales beneath 1 perform better, with 0.5 and 0.25 best.

at a rate of $1/\sqrt{K}$, where $K$ is the number of samples used, but practically good results may be found with very few samples (e.g. Chapter 29 of MacKay [2003]).

To test how many samples are needed for good predictive accuracy in a Bayesian model averaging task, we used a rank 20 approximation for SWAG and then tested the NLL on the test set as a function of the number of samples for WideResNet28x10 [Zagoruyko and Komodakis 2016] on CIFAR-100.

The results from this experiment are shown in Figure A.11 (b, c), where it is possible to see that about 3 samples will match the SWA result for NLL, with about 30 samples necessary for stable accuracy (about the same as SWA for this network). In most of our experiments, we used 30 samples for consistency. In practice, we suggest tuning this number by looking at a validation set as well as the computational resources available and comparing to the free SWA predictions that come with SWAG.

#### A.3.5.2 Dependence on Learning Rate

First, we note that the covariance, $\Sigma$, estimated using SWAG, is a function of the learning rate (and momentum) for SGD. While the theoretical work of Mandt et al. [2017a] suggests that it is possible to optimally set the learning rate, our experiments in Appendix A.3.2 show that currently the assumptions of the theory do not match the empirical reality in deep learning. In practice the learning rate can be chosen to maximize negative log-likelihood on a validation set. In the linear setting as in Mandt et al. [2017a], the learning rate controls the scale of the asymptotic covariance matrix. If the optimal learning rate (Equation A.1) is used in this setting, the covariance matches the true posterior. To attempt to disassociate the learning rate from the covariance in practice, we rescale the covariance matrix when sampling by a constant factor for a WideResNet on CIFAR-100 shown in Figure A.11 (d).

Over several replications, we found that a scale of 0.5 worked best, which is expected because the low rank plus diagonal covariance incorporates the variance twice (once for the diagonal and once from the low rank component).

#### A.3.5.3 Necessity of Batch Norm Updates

One possible slowdown of SWAG at inference time is in the usage of updated batch norm parameters. Following Izmailov et al. [2018], we found that in order for the averaging and sampling to work well, it was necessary to update the batch norm parameters of networks after sampling a new model. This is shown in Figure A.12 for a WideResNet on CIFAR-100 for two independently trained models.

#### A.3.5.4 Usage in Practice

From our experimental findings, we see that given an equal amount of training time, SWAG typically outperforms other methods for uncertainty calibration. SWAG additionally does not

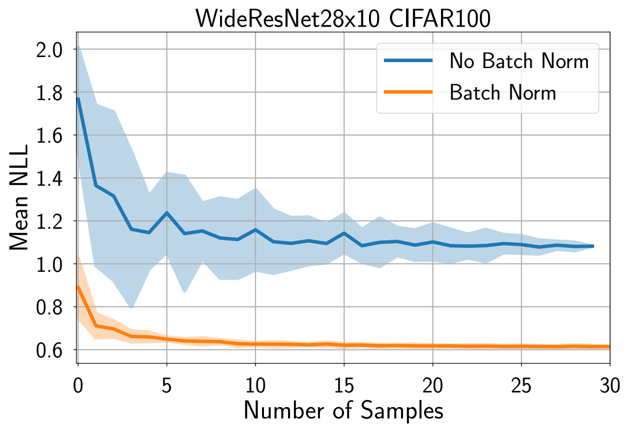


**Figure A.12:** NLL by number of samples for SWAG with and without batch norm updates after sampling. Updating the batch norm parameters after sampling results in a significant improvement in NLL.

require a validation set like temperature scaling and Platt scaling (e.g. Guo et al. [2017]; Kuleshov et al. [2018]). SWAG also appears to have a distinct advantage over temperature scaling, and other popular alternatives, when the target data are from a different distribution than the training data, as shown by our transfer learning experiments.

Deep ensembles [Lakshminarayanan et al. 2017] require several times longer training for equal calibration, but often perform somewhat better due to incorporating several independent training runs. Thus SWAG will be particularly valuable when training time is limited, but inference time may not be. One possible application is thus in medical applications when image sizes (for semantic segmentation) are large, but predictions can be parallelized and may not have to be instantaneous.

## A.3.6 Further Classification Uncertainty Results

### A.3.6.1 Reliability Diagrams

We provide the additional reliability diagrams for all methods and datasets in Figure A.13. SWAG consistently improves calibration over SWA, and performs on par or better than temperature scaling. In transfer learning temperature scaling fails to achieve good calibration, while SWAG

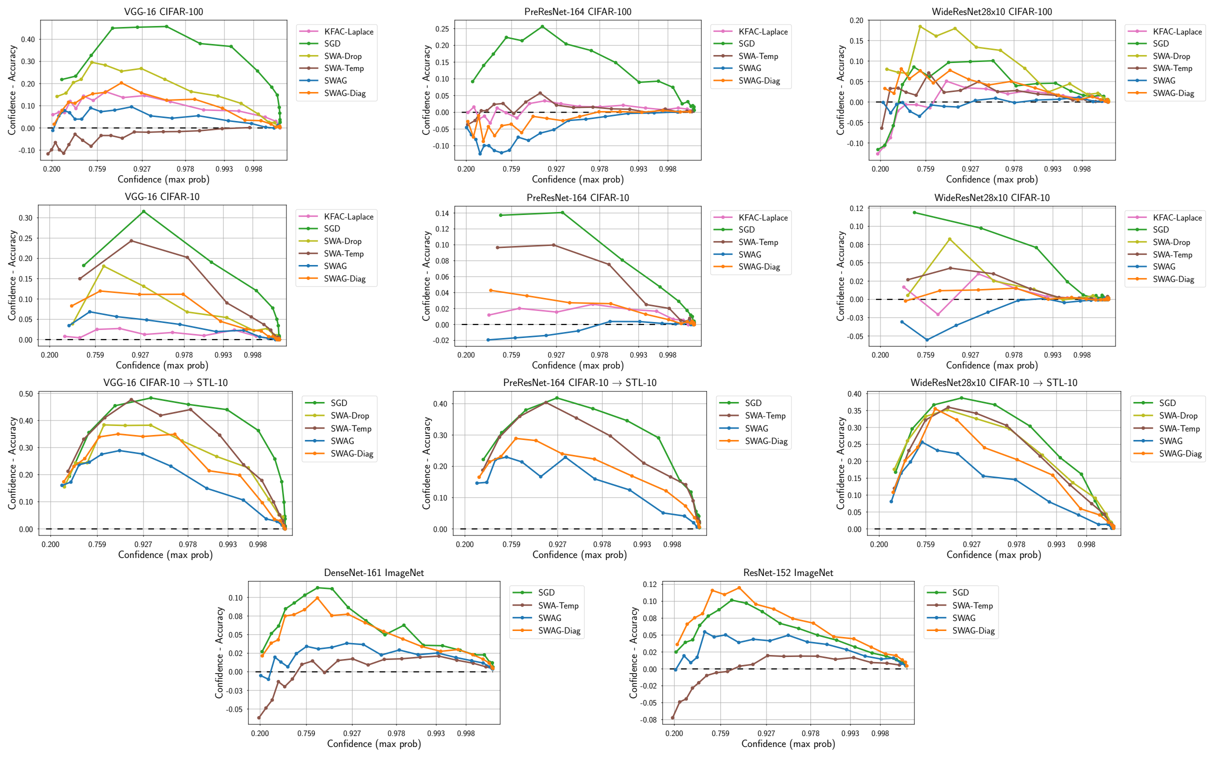


**Figure A.13:** Reliability diagrams (see Section 3.5.1) for all models and datasets. The dataset and architecture are listed in the title of each panel.

still provides a significant improvement over SWA.

#### A.3.6.2 Out-of-Domain Image Detection

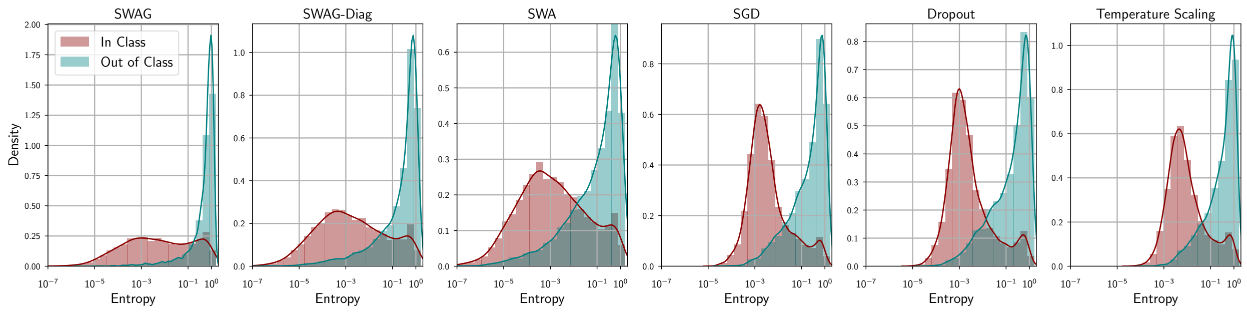


**Figure A.14:** In and out of sample entropy distributions for WideResNet28x10 on CIFAR5 + 5.

**Table A.4:** Symmetrized, discretized KL divergence between the distributions of predictive entropies for data from the first and last five classes of CIFAR-10 for models trained only on the first five classes. The entropy distributions for SWAG are more different than the baseline models.

| Method | JS-Distance |
|---|---|
| SWAG | **3.31** |
| SWAG-Diag | 2.27 |
| MC Dropout | 3.04 |
| SWA | 1.68 |
| SGD (Baseline) | 3.14 |
| SGD + Temp. Scaling | 2.98 |

Next, we evaluate the SWAG variants along with the baselines on out-of-domain data detection. To do so we train a WideResNet as described in Section A.3.9 on the data from five classes of the CIFAR-10 dataset, and then analyze their predictions on the full test set. We expect the outputted class probabilities on objects that belong to classes that were not present in the training data to have high-entropy reflecting the model's high uncertainty in its predictions, and considerably lower entropy on the images that are similar to those on which the network was trained.

To make this comparison quantitative, we computed the symmetrized KL divergence between the binned in and out of sample distributions in Table A.4, finding that SWAG and Dropout perform best on this measure. We plot the histograms of predictive entropies on the in-domain (classes that were trained on) and out-of-domain (classes that were not trained on) in Figure A.A.14 for a qualitative comparison.

Table A.4 shows the computed symmetrized, discretized KL distance between in and out of sample distributions for the CIFAR5 out of sample image detection class. We used the same bins as in Figure A.14 to discretize the entropy distributions, then smoothed these bins by a factor of 1e-7 before calculating $KL(\text{IN}||\text{OUT}) + KL(\text{OUT}||\text{IN})$ using the `scipy.stats.entropy` function. We can see even qualitatively that the distributions are more distinct for SWAG and SWAG-Diagonal than for the other methods, particularly temperature scaling.

#### A.3.6.3 Tables of ECE, NLL, and Accuracy.

We provide test accuracies (Tables A.11,A.12,A.13) and negative log-likelihoods (NLL) (Tables A.8,A.9,A.10) all methods and datasets. We observe that SWAG is competitive with SWA, SWA with temperature scaling and SWA-Dropout in terms of test accuracy, and typically outperforms all the baselines in terms of NLL. SWAG-Diagonal is generally inferior to SWAG for log-likelihood, but outperforms SWA.

In Tables A.5,A.6,A.7 we additionally report expected calibration error [ECE, Naeini et al. 2015], a metric of calibration of the predictive uncertainties. To compute ECE for a given model we split the test points into 20 bins based on the confidence of the model, and we compute the absolute value of the difference of the average confidence and accuracy within each bin, and average the obtained values over all bins. Please refer to [Naeini et al. 2015; Guo et al. 2017] for more details. We observe that SWAG is competitive with temperature scaling for ECE. Again, SWAG-Diagonal achieves better calibration than SWA, but using the low-rank plus diagonal covariance approximation in SWAG leads to substantially improved performance.

**Table A.5:** ECE for various versions of SWAG, temperature scaling, and MC Dropout on CIFAR-10 and CIFAR-100.

| | CIFAR-10 | CIFAR-10 | CIFAR-10 | CIFAR-100 | CIFAR-100 | CIFAR-100 |
|---|---|---|---|---|---|---|
| Model | VGG-16 | PreResNet-164 | WideResNet28x10 | VGG-16 | PreResNet-164 | WideResNet28x10 |
| SGD | 0.0483 ± 0.0022 | 0.0255 ± 0.0009 | 0.0166 ± 0.0007 | 0.1870 ± 0.0014 | 0.1012 ± 0.0009 | 0.0479 ± 0.0010 |
| SWA | 0.0408 ± 0.0019 | 0.0203 ± 0.0010 | 0.0087 ± 0.0002 | 0.1514 ± 0.0032 | 0.0700 ± 0.0056 | 0.0684 ± 0.0022 |
| SWAG-Diag | 0.0267 ± 0.0025 | 0.0082 ± 0.0008 | **0.0047** ± 0.0013 | 0.0819 ± 0.0021 | 0.0239 ± 0.0047 | 0.0322 ± 0.0018 |
| SWAG | 0.0158 ± 0.0030 | **0.0053** ± 0.0004 | 0.0088 ± 0.0006 | 0.0395 ± 0.0061 | 0.0587 ± 0.0048 | **0.0113** ± 0.0020 |
| KFAC-Laplace | 0.0094 ± 0.0005 | 0.0092 ± 0.0018 | 0.0060 ± 0.0003 | 0.0778 ± 0.0054 | **0.0158** ± 0.0014 | 0.0379 ± 0.0047 |
| SWA-Dropout | 0.0284 ± 0.0036 | 0.0162 ± 0.0000 | 0.0094 ± 0.0014 | 0.1108 ± 0.0181 | * | 0.0574 ± 0.0028 |
| SWA-Temp | 0.0366 ± 0.0063 | 0.0172 ± 0.0010 | 0.0080 ± 0.0007 | **0.0291** ± 0.0097 | 0.0175 ± 0.0037 | 0.0220 ± 0.0007 |
| SGLD | **0.0082** ± 0.0012 | 0.0251 ± 0.0012 | 0.0192 ± 0.0007 | 0.0424 ± 0.0029 | 0.0363 ± 0.0008 | 0.0296 ± 0.0008 |

**Table A.6:** ECE on ImageNet.

| Model | DenseNet-161 | ResNet-152 |
|---|---|---|
| SGD | 0.0545 ± 0.0000 | 0.0478 ± 0.0000 |
| SWA | 0.0509 ± 0.0000 | 0.0605 ± 0.0000 |
| SWAG-Diag | 0.0459 ± 0.0000 | 0.0566 ± 0.0000 |
| SWAG | 0.0204 ± 0.0000 | 0.0279 ± 0.0000 |
| SWA-Temp | **0.0190** ± 0.0000 | **0.0183** ± 0.0000 |

**Table A.7:** ECE on CIFAR10 to STL 10.

| Model | VGG-16 | PreResNet-164 | WideResNet28x10 |
|---|---|---|---|
| SGD | 0.2149 ± 0.0027 | 0.1758 ± 0.0000 | 0.1561 ± 0.0000 |
| SWA | 0.2082 ± 0.0056 | 0.1739 ± 0.0000 | 0.1413 ± 0.0000 |
| SWAG-Diag | 0.1719 ± 0.0075 | 0.1312 ± 0.0000 | 0.1241 ± 0.0000 |
| SWAG | **0.1463** ± 0.0075 | **0.1110** ± 0.0000 | **0.1017** ± 0.0000 |
| SWA-Dropout | 0.1803 ± 0.0024 | | 0.1421 ± 0.0000 |
| SWA-Temp | 0.2089 ± 0.0055 | 0.1646 ± 0.0000 | 0.1371 ± 0.0000 |

### A.3.7 Language Modeling

We evaluate SWAG using standard Penn Treebank and WikiText-2 benchmark language modeling datasets. Following [Merity et al. 2017] we use a 3-layer LSTM model with 1150 units in the hidden layer and an embedding of size 400; we apply dropout, weight-tying, activation regularization (AR) and temporal activation regularization (TAR) techniques. We follow [Merity et al. 2017] for

**Table A.8:** NLL on CIFAR10 and CIFAR100.

| Dataset | CIFAR-10 | | | CIFAR-100 | | |
|---|---|---|---|---|---|---|
| Model | VGG-16 | PreResNet-164 | WideResNet28x10 | VGG-16 | PreResNet-164 | WideResNet28x10 |
| SGD | 0.3285 ± 0.0139 | 0.1814 ± 0.0025 | 0.1294 ± 0.0022 | 1.7308 ± 0.0137 | 0.9465 ± 0.0191 | 0.7958 ± 0.0089 |
| SWA | 0.2621 ± 0.0104 | 0.1450 ± 0.0042 | 0.1075 ± 0.0004 | 1.2780 ± 0.0051 | 0.7370 ± 0.0265 | 0.6684 ± 0.0034 |
| SWAG-Diag | 0.2200 ± 0.0078 | 0.1251 ± 0.0029 | 0.1077 ± 0.0009 | 1.0163 ± 0.0032 | 0.6837 ± 0.0186 | 0.6150 ± 0.0029 |
| SWAG | 0.2016 ± 0.0031 | **0.1232** ± 0.0022 | 0.1122 ± 0.0009 | 0.9480 ± 0.0038 | **0.6595** ± 0.0019 | **0.6078** ± 0.0006 |
| KFAC-Laplace | 0.2252 ± 0.0032 | 0.1471 ± 0.0012 | 0.1210 ± 0.0020 | 1.1915 ± 0.0199 | 0.7881 ± 0.0025 | 0.7692 ± 0.0092 |
| SWA-Dropout | 0.2328 ± 0.0049 | 0.1270 ± 0.0000 | 0.1094 ± 0.0021 | 1.1872 ± 0.0524 | | 0.6500 ± 0.0049 |
| SWA-Temp | 0.2481 ± 0.0245 | 0.1347 ± 0.0038 | **0.1064** ± 0.0004 | 1.0386 ± 0.0126 | **0.6770** ± 0.0191 | 0.6134 ± 0.0023 |
| SGLD | **0.2001** ± 0.0059 | 0.1418 ± 0.0005 | 0.1289 ± 0.0009 | 0.9699 ± 0.0057 | 0.6981 ± 0.0052 | 0.678 ± 0.0022 |
| SGD-Ens | 0.1881 ± 0.002 | 0.1312 ± 0.0023 | 0.1855 ± 0.0014 | **0.8979** ± 0.0065 | 0.7839 ± 0.0046 | 0.7655 ± 0.0026 |

**Table A.9:** NLL on ImageNet.

| Model | DenseNet-161 | ResNet-152 |
|---|---|---|
| SGD | 0.9094 ± 0.0000 | 0.8716 ± 0.0000 |
| SWA | 0.8655 ± 0.0000 | 0.8682 ± 0.0000 |
| SWAG-Diag | 0.8559 ± 0.0000 | 0.8584 ± 0.0000 |
| SWAG | **0.8303** ± 0.0000 | **0.8205** ± 0.0000 |
| SWA-Temp | 0.8359 ± 0.0000 | 0.8226 ± 0.0000 |

**Table A.10:** NLL when transferring from CIFAR10 to STL10.

| Model | VGG-16 | PreResNet-164 | WideResNet28x10 |
|---|---|---|---|
| SGD | 1.6528 ± 0.0390 | 1.4790 ± 0.0000 | 1.1308 ± 0.0000 |
| SWA | 1.3993 ± 0.0502 | 1.3552 ± 0.0000 | 1.0047 ± 0.0000 |
| SWAG-Diag | 1.2258 ± 0.0446 | 1.0700 ± 0.0000 | 0.9340 ± 0.0000 |
| SWAG | **1.1402** ± 0.0342 | **0.9706** ± 0.0000 | **0.8710** ± 0.0000 |
| SWA-Dropout | 1.3133 ± 0.0000 | | 0.9914 ± 0.0000 |
| SWA-Temp | 1.4082 ± 0.0506 | 1.2228 ± 0.0000 | 0.9706 ± 0.0000 |

specific hyper-parameter settings such as dropout rates for different types of layers. We train all models for language modeling tasks and evaluate validation and test perplexity. For SWA and SWAG we pre-train the models using standard SGD for 500 epochs, and then run the model for 100 more epochs to estimate the mean $\theta_{\text{SWA}}$ and covariance $\Sigma$ in SWAG. For this experiment we introduce a small change to SWA and SWAG: to estimate the mean $\theta_{\text{SWA}}$ we average weights after each mini-batch of data rather than once per epoch, as we found more frequent averaging to

**Table A.11:** Accuracy on CIFAR-10 and CIFAR-100.

| Dataset | CIFAR-10 | | | CIFAR-100 | | |
|---|---|---|---|---|---|---|
| Model | VGG-16 | PreResNet-164 | WideResNet28x10 | VGG-16 | PreResNet-164 | WideResNet28x10 |
| SGD | 93.17 ± 0.14 | 95.49 ± 0.06 | 96.41 ± 0.10 | 73.15 ± 0.11 | 78.50 ± 0.32 | 80.76 ± 0.29 |
| SWA | 93.61 ± 0.11 | 96.09 ± 0.08 | **96.46** ± 0.04 | 74.30 ± 0.22 | **80.19** ± 0.52 | 82.40 ± 0.16 |
| SWAG-Diag | **93.66** ± 0.15 | 96.03 ± 0.10 | 96.41 ± 0.05 | 74.68 ± 0.22 | 80.18 ± 0.50 | **82.40** ± 0.09 |
| SWAG | 93.60 ± 0.10 | 96.03 ± 0.02 | 96.32 ± 0.08 | **74.77** ± 0.09 | 79.90 ± 0.50 | 82.23 ± 0.19 |
| KFAC-Laplace | 92.65 ± 0.20 | 95.49 ± 0.06 | 96.17 ± 0.00 | 72.38 ± 0.23 | 78.51 ± 0.05 | 80.94 ± 0.41 |
| SWA-Dropout | 93.23 ± 0.36 | **96.18** ± 0.00 | 96.39 ± 0.09 | 72.50 ± 0.54 | | 82.30 ± 0.19 |
| SWA-Temp | 93.61 ± 0.11 | 96.09 ± 0.08 | 96.46 ± 0.04 | 74.30 ± 0.22 | 80.19 ± 0.52 | 82.40 ± 0.16 |
| SGLD | 93.55 ± 0.15 | 95.55 ± 0.04 | 95.89 ± 0.02 | 74.02 ± 0.30 | 80.09 ± 0.05 | 80.94 ± 0.17 |

**Table A.12:** Accuracy on ImageNet.

| Model | DenseNet-161 | ResNet-152 |
|---|---|---|
| SGD | 77.79 ± 0.00 | 78.39 ± 0.00 |
| SWA | **78.60** ± 0.00 | 78.92 ± 0.00 |
| SWAG-Diag | 78.59 ± 0.00 | 78.96 ± 0.00 |
| SWAG | 78.59 ± 0.00 | **79.08** ± 0.00 |
| SWA-Temp | 78.60 ± 0.00 | 78.92 ± 0.00 |

**Table A.13:** Accuracy when transferring from CIFAR-10 to STL-10.

| Model | VGG-16 | PreResNet-164 | WideResNet28x10 |
|---|---|---|---|
| SGD | **72.42** ± 0.07 | 75.56 ± 0.00 | 76.75 ± 0.00 |
| SWA | 71.92 ± 0.01 | **76.02** ± 0.00 | **77.50** ± 0.00 |
| SWAG-Diag | 72.09 ± 0.04 | 75.95 ± 0.00 | 77.26 ± 0.00 |
| SWAG | 72.19 ± 0.06 | 75.88 ± 0.00 | 77.09 ± 0.00 |
| SWA-Dropout | 71.45 ± 0.11 | | 76.91 ± 0.00 |
| SWA-Temp | 71.92 ± 0.01 | 76.02 ± 0.00 | 77.50 ± 0.00 |

greatly improve performance. After SWAG distribution is constructed we sample and ensemble 30 models from this distribution. We use rank-10 for the low-rank part of the covariance matrix of SWAG distribution.

### A.3.8 Regression

For the small UCI regression datasets, we use the architecture from Wu et al. [2018] with one hidden layer with 50 units, training for 50 epochs (starting SWAG at epoch 25) and using 20 repetitions of 90/10 train test splits. We fixed a single seed for tuning before using 20 different seeds for the results in the paper.

We use SGD[3], manually tune learning rate and weight decay, and use batch size of $N/10$ where $N$ is the dataset size. All models predict heteroscedastic uncertainty (i.e. output a variance). In Table A.14, we compare subspace inference methods to deterministic VI (DVI, Wu et al. [2018]) and deep Gaussian processes with expectation propagation (DGP1-50 Bui et al. [2016]). SWAG outperforms DVI and the other methods on three of the six datasets and is competitive on the other three despite its vastly reduced computational time (the same as SGD whereas DVI is known to be 300x slower). Additionally, we note the strong performance of well-tuned SGD as a baseline against the other approximate inference methods, as it consistently performs nearly as well as both SWAG and DVI.

Finally, in Table A.14, we compare the calibration (coverage of the 95% credible sets of SWAG and 95% confidence regions of SGD) of both SWAG and SGD. Note that neither is ever too over-confident (far beneath 95% coverage) and that SWAG is considerably better calibrated on four of the six datasets.

### A.3.9 Classification Experimental Details and Parameters

In this section we describe all of the architectures and hyper-parameters we use in Sections 3.5.1, A.3.6.2.

On ImageNet we use architecture implementations and pre-trained weights from `https://github.com/pytorch/vision/tree/master/torchvision`. For the experiments on CIFAR datasets

[3]Except for concrete where we use Adam due to convergence issues.

**Table A.14:** Unnormalized test log-likelihoods on small UCI datasets for proposed methods, as well as direct comparisons to the numbers reported in deterministic variational inference (DVI, Wu et al. [2018]) and Deep Gaussian Processes with expectation propagation (DGP1-50, Bui et al. [2016]), and variational inference (VI) with the re-parameterization trick [Kingma et al. 2015]. * denotes reproduction from [Wu et al. 2018]. Note that SWAG wins on two of the six datasets, and that SGD serves as a strong baseline throughout.

| dataset | N | D | SGD | SWAG | DVI* | DGP1-50* | VI* | SGLD* | PBP* |
|---|---|---|---|---|---|---|---|---|---|
| boston | 506 | 13 | -2.536 ± 0.240 | -2.469 ± 0.183 | -2.41 ± 0.02 | **-2.33** ± 0.06 | -2.43 ±0.03 | -2.40 ± 0.05 | -2.57 ± 0.09 |
| concrete | 1030 | 8 | **-3.02** ± 0.126 | -3.05 ± 0.1 | -3.06 ± 0.01 | -3.13 ± 0.03 | -3.04 ±0.02 | -3.08 ± 0.03 | -3.16 ± 0.02 |
| energy | 768 | 8 | -1.736 ± 1.613 | -1.679 ± 1.488 | **-1.01** ± 0.06 | -1.32 ± 0.03 | -2.38 ±0.02 | -2.39 ± 0.01 | -2.04 ± 0.02 |
| naval | 11934 | 16 | 6.567 ± 0.185 | **6.708** ± **0.105** | 6.29 ± 0.04 | 3.60 ± 0.33 | 5.87 ±0.29 | 3.33 ± 0.01 | 3.73 ± 0.01 |
| yacht | 308 | 6 | -0.418 ± 0.426 | **-0.404** ± **0.418** | -0.47 ± 0.03 | -1.39 ± 0.14 | -1.68 ±0.04 | -2.90 ± 0.01 | -1.63 ± 0.02 |
| power | 9568 | 4 | -2.772 ± 0.04 | -2.775 ± 0.038 | -2.80 ± 0.00 | -2.81 ± 0.01 | **-2.66** ± 0.01 | -2.67 ± 0.00 | -2.84 ± 0.01 |

**Table A.15:** Calibration on small-scale UCI datasets. Bolded numbers are those closest to 0.95 %the predicted coverage).

| | N | D | SGD | SWAG |
|---|---|---|---|---|
| boston | 506 | 13 | 0.913 ± 0.039 | **0.936** ± 0.036 |
| concrete | 1030 | 8 | 0.909 ± 0.032 | **0.930** ± 0.023 |
| energy | 768 | 8 | 0.947 ± 0.026 | **0.951** ± 0.027 |
| naval | 11934 | 16 | **0.948** ± 0.051 | 0.967 ± 0.008 |
| yacht | 308 | 6 | 0.895 ± 0.069 | **0.898** ± 0.067 |
| power | 9568 | 4 | **0.956** ± 0.006 | 0.957 ± 0.005 |

we adapted the following implementations:

- VGG-16: `https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py`
- Preactivation-ResNet-164: `https://github.com/bearpaw/pytorch-classification/blob/master/models/cifar/preresnet.py`
- WideResNet28x10: `https://github.com/meliketoy/wide-resnet.pytorch/blob/master/networks/wide_resnet.py`

For all datasets and architectures we use the same piecewise constant learning rate schedule and weight decay as in Izmailov et al. [2018], except we train Pre-ResNet for 300 epochs and start averaging after epoch 160 in SWAG and SWA. For all of the methods we are using our own implementations in PyTorch. We describe the hyper-parameters for all experiments for each model:

SWA We use the same hyper-parameters as Izmailov et al. [2018] on CIFAR datasets. On ImageNet we used a constant learning rate of $10^{-3}$ instead of the cyclical schedule, and averaged 4 models per epoch. We adapt the code from `https://github.com/timgaripov/swa` for our implementation of SWA.

SWAG In all experiments we use rank $K = 20$ and use 30 weight samples for Bayesian model averaging. We re-use all the other hyper-parameters from SWA.

KFAC-Laplace For our implementation we adapt the code for KFAC Fisher approximation from `https://github.com/Thrandis/EKFAC-pytorch` and implement our own code for sampling. Following [Ritter et al. 2018b] we tune the scale of the approximation on validation set for every model and dataset.

MC-Dropout In order to implement MC-dropout we add dropout layers before each weight layer and sample 30 different dropout masks for Bayesian model averaging at inference time. To choose the dropout rate, we ran the models with dropout rates in the set $\{0.1, 0.05, 0.01\}$ and chose the one that performed best on validation data. For both VGG-16 and WideResNet28x10 we found that dropout rate of 0.05 worked best and used it in all experiments. On PreResNet-164 we couldn't achieve reasonable performance with any of the three dropout rates, which has been reported from the work of He et al. [2016]. We report the results for MC-Dropout in combination with both SWA (SWA-Drop) and SGD (SGD-Drop) training.

Temperature Scaling For SWA and SGD solutions we picked the optimal temperature by minimizing negative log-likelihood on validation data, adapting the code from `https://github.com/gpleiss/temperature_scaling`.

SGLD We initialize SGLD from checkpoints pre-trained with SGD. We run SGLD for 100 epochs on WideResNet and for 150 epochs on PreResNet-156. We use the learning rate schedule of [Welling and Teh 2011]:

$$\eta_t = \frac{\eta_0}{(\eta_1 + t)^{0.55}}.$$

We tune constants $a, b$ on validation. For WideResNet we use $a = 38.0348$, $b = 13928.7$ and for PreResNet we use $a = 40.304$, $b = 15476.4$; these values are selected so that the initial learning rate is 0.2 and final learning rate is 0.1. We also had to rescale the noise in the gradients by a factor of $5 \cdot 10^{-4}$ compared to [Welling and Teh 2011]. Without this rescaling we found that even with learning rates on the scale of $10^{-7}$ SGD diverged. We note that noise rescaling is commonly used with stochastic gradient MCMC methods (see e.g. the implementation of [Zhang et al. 2020c]).

On CIFAR datasets for tuning hyper-parameters we used the last 5000 training data points as a validation set. On ImageNet we used 5000 of test data points for validation. On the transfer task for CIFAR10 to STL10, we report accuracy on all 10 STL10 classes even though frogs are not a part of the STL10 test set (and monkeys are not a part of the CIFAR10 training set).

## A.4 Appendix for "Bayesian Deep Learning and a Probabilistic Perspective of Generalization"

This appendix is organized as follows. In Section A.4.1, we visualize predictive functions corresponding to weight samples within high posterior density valleys on a regression problem. In Section A.4.2, we provide background material on Gaussian processes. In Section A.4.3, we present further results comparing MultiSWAG and MultiSWA to Deep Ensembles under data distribution shift on CIFAR-10. In Section A.4.4, we provide the details of all experiments presented in the paper. In Section A.4.5, we present analytic results on the dependence of the prior distribution in function space on the variance of the prior over parameters. In Section A.4.6, we study

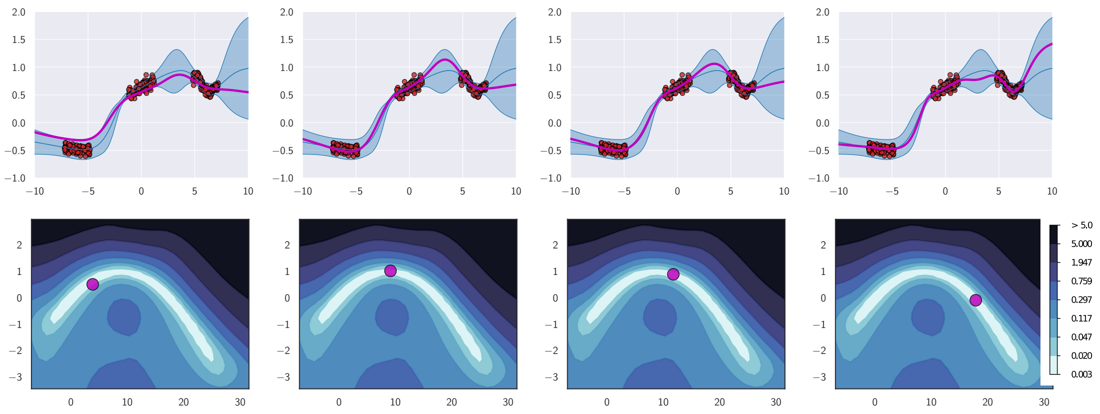


**Figure A.15: Diversity of high performing functions. Bottom**: a contour plot of the posterior log-density in the subspace containing a pair of independently trained modes (as with deep ensembles), and a path of high posterior density connecting these modes. In each panel, the purple point represents a sample from the posterior in the parameter subspace. **Top**: the predictive distribution constructed from samples in the subspace. The shaded blue area shows the $3\sigma$-region of the predictive distribution at each of the input locations, and the blue line shows the mean of the predictive distribution. In each panel, the purple line shows the predictive function corresponding to the sample shown in the corresponding bottom row panel. For the details of the experimental setup see Section 5.1 of Izmailov et al. [2019].

the prior correlations between BNN logits on perturbed images.

### A.4.1 Loss Valleys

We demonstrate that different points along the valleys of high posterior density (low loss) connecting pairs of independently trained optima [Garipov et al. 2018; Draxler et al. 2018b; Fort and Jastrzebski 2019] correspond to different predictive functions. We use the regression example from Izmailov et al. [2019] and show the results in Figure A.15.

### A.4.2 Gaussian processes

With a Bayesian neural network, a distribution over parameters $p(w)$ induces a distribution over functions $p(f(x;w))$ when combined with the functional form of the network. Gaussian processes (GPs) are often used to instead *directly* specify a distribution over functions.

A Gaussian process is a distribution over functions, $f(x) \sim \mathcal{GP}(m, k)$, such that any collection of function values, queried at any finite set of inputs $x_1, \ldots, x_n$, has a joint Gaussian distribution:

$$f(x_1), \ldots, f(x_n) \sim \mathcal{N}(\mu, K)\,. \tag{A.3}$$

The mean vector, $\mu_i = \mathbb{E}[f(x_i)] = m(x_i)$, and covariance matrix, $K_{ij} = \text{cov}(f(x_i), f(x_j)) = k(x_i, x_j)$, are determined by the *mean function* $m$ and *covariance function* (or *kernel*) $k$ of the Gaussian process.

The popular RBF kernel has the form

$$k(x_i, x_j) = \exp\left(-\frac{1}{2\ell^2}\|x_i - x_j\|^2\right)\,. \tag{A.4}$$

The *length-scale* hyperparameter $\ell$ controls the extent of correlations between function values. If $\ell$ is large, sample functions from a GP prior are simple and slowly varying with inputs $x$.

Gaussian processes with RBF kernels (as well as many other standard kernels) assign positive density to any set of observations. Moreover, these models are *universal approximators* [Rasmussen and Williams 2006]: as the number of observations increase, they are able to approximate any function to arbitrary precision.

Work on Gaussian processes in machine learning was triggered by the observation that Bayesian neural networks become Gaussian processes with particular kernel functions as the number of hidden units approaches infinity [Neal 1996]. This result resembles recent work on the neural tangent kernel [e.g., Jacot et al. 2018].

### A.4.3 Deep Ensembles and MultiSWAG Under Distribution Shift

In Figures A.18, A.19, A.20, A.21 we show the negative log-likelihood for Deep Ensembles, MultiSWA and MultiSWAG using PreResNet-20 on CIFAR-10 with various corruptions as a function of the number of independently trained models (SGD solutions, SWA solutions or SWAG models, respectively). For MultiSWAG, we generate 20 samples from each independent SWAG model. Typically MultiSWA and MultiSWAG significantly outperform Deep Ensembles when a small number of independent models is used, or when the level of corruption is high.

In Figure A.22, following Ovadia et al. [2019], we show the distribution of negative log likelihood, accuracy and expected calibration error as we vary the type of corruption. We use a fixed training time budget: 10 independently trained models for every method. For MultiSWAG we ensemble 20 samples from each of the 10 SWAG approximations. MultiSWAG particularly achieves better NLL than the other two methods, and MultiSWA outperforms Deep Ensembles; the difference is especially pronounced for higher levels of corruption. In terms of ECE, MultiSWAG again outperforms the other two methods for higher corruption intensities.

We note that Ovadia et al. [2019] found Deep Ensembles to be a very strong baseline for prediction quality and calibration under distribution shift. For this reason, we focus on Deep Ensembles in our comparisons.

### A.4.4 Details of Experiments

In this section we provide additional details of the experiments presented in the paper.

#### A.4.4.1 Approximating the True Predictive Distribution

For the results presented in Figure 3.7 we used a network with 3 hidden layers of size 10 each. The network takes two inputs: $x$ and $x^2$. We pass both $x$ and $x^2$ as input to ensure that the network can represent a broader class of functions. The network outputs a single number $y = f(x)$.

To generate data for the plots, we used a randomly-initialized neural network of the same architecture described above. We sampled the weights from an isotropic Gaussian with variance $0.1^2$ and added isotropic Gaussian noise with variance $0.1^2$ to the outputs:

$$y = f(x; w) + \epsilon(x),$$

with $w \sim \mathcal{N}(0, 0.1^2 \cdot I)$, $\epsilon(x) \sim \mathcal{N}(0, 0.1^2 \cdot I)$. The training set consists of 120 points shown in Figure 3.7.

For estimating the ground truth we ran 200 chains of Hamiltonian Monte Carlo (HMC) using the `hamiltorch` package [Cobb et al. 2019]. We initialized each chain with a network pre-trained with SGD for 3000 steps, then ran Hamiltonian Monte Carlo (HMC) for 2000 steps, producing 200 samples.

For Deep Ensembles, we independently trained 50 networks with SGD for 20000 steps each. We used minus posterior log-density as the training loss. For SVI, we used a fully-factorized Gaussian approximation initialized at an SGD solution trained for 20000 steps. For all inference methods we set prior variance to $10^2$ and noise variance to $0.02^2$.

Discrepancy with true BMA. For the results presented in panel (d) of Figure 3.7 we computed Wasserstein distance between the predictive distribution approximated with HMC and the predictive distribution for Deep Ensembles and SVI. We used the one-dimensional Wasserstein distance function[4] from the *scipy* package [Virtanen et al. 2020a]. We computed the Wasserstein distance between marginal distributions at each input location, and averaged the results over the input locations. In the top sub-panels of panels (b), (c) of Figure 3.7 we additionally visualize the marginal Wasserstein distance between the HMC predictive distribution and Deep Ensembles and SVI predictive distrbutions respectively for each input location.

#### A.4.4.2 Deep Ensembles and MultiSWAG

We evaluate Deep Ensembles, MultiSWA and MultiSWAG under distribution shift in Section 3.6. Following Ovadia et al. [2019], we use a PreResNet-20 network and the CIFAR-10 dataset with different types of corruptions introduced in Hendrycks and Dietterich [2019]. For training individual SGD, SWA and SWAG models we use the hyper-parameters used for PreResNet-164 in

[4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wasserstein_distance.html

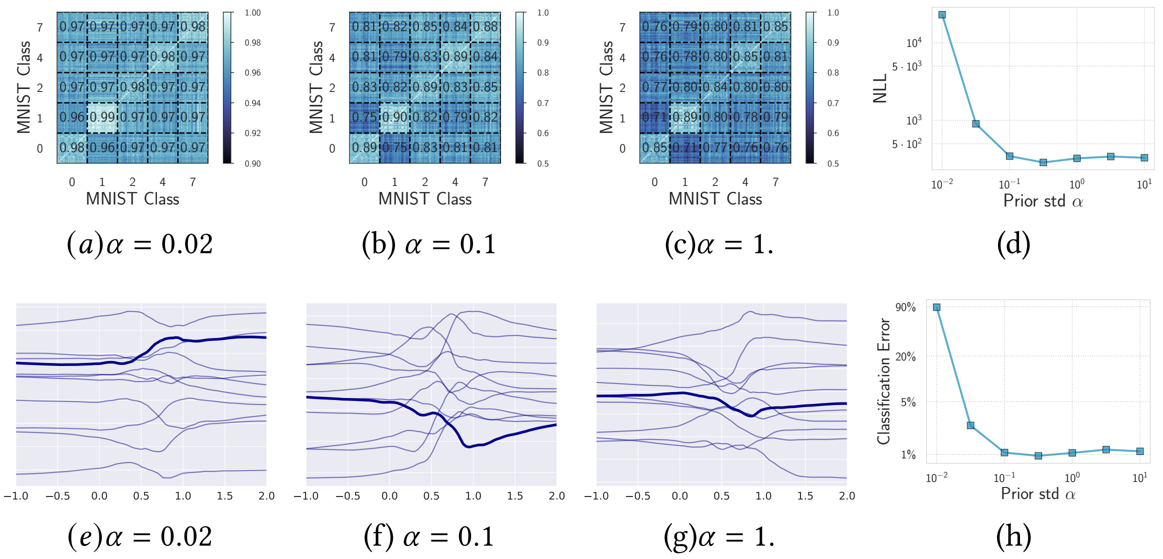


**Figure A.16: (a)–(c)**: Average pairwise prior correlations for pairs of objects in classes $\{0, 1, 2, 4, 7\}$ of MNIST induced by LeNet-5 for $p(f(x; w))$ when $p(w) = \mathcal{N}(0, \alpha^2 I)$. Images in the same class have higher prior correlations than images from different classes, suggesting that $p(f(x; w))$ has desirable inductive biases. The correlations slightly decrease with increases in $\alpha$. Panels **(e)–(g)** show sample functions from LeNet-5 along the direction connecting a pair of MNIST images of 0 and 1 digits. The complexity of the samples increases with $\alpha$. **(d)**: NLL and **(h)** classification error of an ensemble of 20 SWAG samples on MNIST as a function of $\alpha$ using a LeNet-5. The NLL is high for overly small $\alpha$ and near-optimal for larger values with an optimum near $\alpha = 0.3$.

Maddox et al. [2019]. For each SWAG model we sample 20 networks and ensemble them. So, Deep Ensembles, MultiSWA and MultiSWAG are all evaluated under the same training budget; Deep Ensembles and MultiSWA also use the same test-time budget.

For producing the corrupted data we used the code[5] released by Hendrycks and Dietterich [2019]. We had issues producing the data for the *frost* corruption type, so we omit it in our evaluation, and include *Gaussian blur* which was not included in the evaluation of Hendrycks and Dietterich [2019].

### A.4.4.3 Neural Network Priors

In the main text we considered different properties of the prior distribution over functions induced by a spherical Gaussian distribution over the weights, with different variance scales.

[5]https://github.com/hendrycks/robustness/blob/master/ImageNet-C/create_c/make_cifar_c.py

Prior correlation diagrams. In panels (a)–(c) of Figure A.16 we show pairwise correlations of the logits for different pairs of datapoints. To make these plots we produce $S = 100$ samples of the weights $w_i$ of a LeNet-5 from the prior distribution $\mathcal{N}(0, \alpha^2 I)$ and compute the logits corresponding to class 0 for each data point and each weight sample. We then compute the correlations for each pair $x, x'$ of data points as follows:

$$\mathrm{corr}_{\mathrm{logit}}(x, x') = \frac{\sum_{i=1}^{S}(f(x, w_i) - \bar{f}(x))(f(x', w_i) - \bar{f}(x'))}{\sqrt{\sum_{i=1}^{S}(f(x, w_i) - \bar{f}(x))^2 \cdot \sum_{i=1}^{S}(f(x', w_i) - \bar{f}(x'))^2}},$$

where $f(x, w)$ is the logit corresponding to class 0 of the network with weights $w$ on the input $x$, and $\bar{f}(x)$ is the mean value of the logit $\bar{f}(x) = \frac{1}{S}\sum_i f(x, w_i)$. For evaluation, we use 200 random datapoints per class for classes $0, 1, 2, 4, 7$ (a total of 1000 datapoints). We use this set of classes to ensure that the structure is clearly visible in the figure. We combine the correlations into a diagram, additionally showing the average correlation for each pair of classes. We repeat the experiment for different values of $\alpha \in \{0.02, 0.1, 1\}$. For a discussion of the results see Section 3.7.2.

Sample functions. In panels (e)–(g) of Figure A.16 we visualize the functions sampled from the LeNet-5 network along the direction connecting a pair of MNIST images. In particular, we take a pair of images $x_0$ and $x_1$ of digits 0 and 1, respectively, and construct the path $x(t) = t \cdot x_0 + (1 - t) \cdot x_1$. We then study the samples of the logits $z(t) = f(x(t) \cdot \|x_0\| / \|x(t)\|, w)$ along the path; here we adjusted the norm of the images along the path to be constant as the values of the logits are sensitive to the norm of the inputs. The complexity of the samples increases as we increase the variance of the prior distribution over the weights. This increased complexity of sample functions explains why we might expect the prior correlations for pairs of images to be lower when we increase the variance of the prior distribution over the weights.

Performance dependence on prior variance. In panels (d), (h) of Figure A.16 we show the test negative log-likelihood and accuracy of SWAG applied to LeNet-5 on MNIST. We train the model for 50 epochs, constructing the rank-20 SWAG approximation from the last 25 epochs. We use an initial learning rate of 0.05 and SWAG learning rate of 0.01 with the learning rate schedule of Maddox et al. [2019]. We use posterior log-density as the objective, and vary the prior variance $\alpha^2$. In panels (f), (g) of Figure 3.12 we perform an analogous experiment using a PreResNet-20 and a VGG-16 on CIFAR-10, using the hyper-parameters reported in Maddox et al. [2019] (for PreResNet-20 we use the hyper-parameters used with PreResNet-164 in Maddox et al. [2019]). Both on MNIST and CIFAR-10 we observe that the performance is poor for overly small values of $\alpha$, close to optimal for intermediate values, and still reasonable for larger values of $\alpha$. For further discussion of the results see Section 3.7.3.

Predictions from prior samples. Following Wenzel et al. [2020] we study the predictive distributions of prior samples using PreResNet-20 on CIFAR-10. In Figure 3.12 we show the sample predictive functions averaged over datapoints for different scales $\alpha$ of the prior distribution. We also show the predictive distribution for each $\alpha$, which is the average of the sample predictive distributions over 200 samples of weights. In Figure 3.13 we show how the predictive distribution changes as we vary the number of observed data for prior scale $\alpha = \sqrt{10}$. We see that the marginal predictive distribution for all considered values of $\alpha$ is reasonable — roughly uniform across classes, when averaged across the dataset. For the latter experiment we used stochastic gradient Langevin dynamics (SGLD) [Welling and Teh 2011] with a cosine lerning rate schedule. For each sample we restart SGLD, and we only use the sample obtained at the last iteration. We discuss the results in Section 3.10.4.

Prior correlations with corrupted images. In Section A.4.6 and Figure A.23 we study the decay of the prior correlations between logits on an original image and a perturbed image as we increase the intensity of perturbations. For the BNN we use PreResNet-20 architecture with the standard Gaussian prior $\mathcal{N}(0, I)$. For the linear model, the correlations are not affected by the prior variance $\alpha^2$:

$$cov(w^T x, w^T y) = \mathbb{E}(w^T x \cdot w^T y) =$$
$$\mathbb{E} x^T w w^T y = x^T \mathbb{E} w w^T y = \alpha^2 x^T y,$$

and hence

$$corr(w^T x, w^T y) =$$
$$\frac{cov(w^T x, w^T y)}{\sqrt{cov(w^T y, w^T y) \cdot cov(w^T x, w^T x)}} = x^T y.$$

We use the $\mathcal{N}(0, I)$ prior for the weights of the linear model. Finally, we also evaluate the correlations associated with an RBF kernel (see Equation (A.4)). To set the lengthscale $\ell$ of the kernel we evaluate the pairwise correlations for the PreResnet-20 and RBF kernel on the 100 uncorrupted CIFAR-10 images that were used for the experiment, and ensure that the average correlations match. The resulting value of $\ell$ is 10000, and the average correlation for the RBF kernel and PreResNet was $\approx 0.9$; for the linear model the average correlation was $\approx 0.82$. For the perturbations we used the same set of corruptions introduced in Hendrycks and Dietterich [2019] as in the experiments in Section 3.6 with the addition of a random translation: for a random translation of intensity $i$ we pad the image with $2 \cdot i$ zeros on each side and crop the image randomly to $32 \times 32$.

#### A.4.4.4 Rethinking Generalization

In Section 3.8, we experiment with Bayesian neural networks and Gaussian processes on CIFAR-10 with noisy labels, inspired by the results in Zhang et al. [2017a] that suggest we need to re-think generalization to understand deep learning.

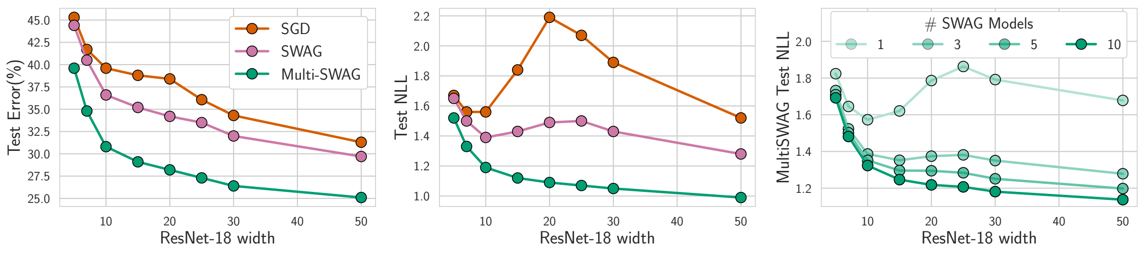
Following Zhang et al. [2017a], we train PreResNet-20 on CIFAR-10 with different fractions of random labels. To ensure that the networks fits the train data, we turn off weight decay and data augmentation, and use a lower initial learning rate of 0.01. Otherwise, we follow the hyper-parameters that were used with PreResNet-164 in Maddox et al. [2019]. We use diagonal Laplace approximation to compute an estimate of marginal likelihood for each level of label corruption. Following Ritter et al. [2018b] we use the diagonal of the Fisher information matrix rather than the Hessian.

We perform a similar experiment with a Gaussian process with RBF kernel on the binary classification problem for two classes of CIFAR-10. We use variational inference to fit the model, and we use the variational evidence lower bound to approximate the marginal likelihood. We use variational inference to overcome the non-Gaussian likelihood and not for scalability reasons; i.e., we are not using inducing inputs. We use the `GPyTorch` package [Gardner et al. 2018] to train the models. We use an RBF kernel with default initialization from `GPyTorch` and divide the inputs by 5000 to get an appropriate input scale. We train the model on a binary classification problem between classes 0 and 1.

For the 10-class GP classification experiment we train 10 one-vs-all models that classify between a given class and the rest of the data. To reduce computation, in training we subsample the data not belonging to the given class to $10k$ datapoints, so each model is trained on a total of $15k$ datapoints. We then combine the 10 models into a single multi-class model: an observation is attributed to the class that corresponds to the one-vs-all model with the highest confidence. We use the same hyper-parameters as in the binary classification experiments.

#### A.4.4.5 Double Descent

In Section 3.9 we evaluate SGD, SWAG and MultiSWAG for models of varying width. Following Nakkiran et al. [2019] we use ResNet-18 on CIFAR-100; we consider original labels, 10% and 20% label corruption. For networks of every width we reuse the hyper-paramerers used for PreResNet-164 in Maddox et al. [2019]. For original labels and 10% label corruption we use 5 independently



(a) 10% Corrupted (Err) (b) 10% Corrupted (NLL) (c) 20% Corrupted (# Models)

**Figure A.17: Double Descent. (a)**: Test error and **(b)**: NLL loss for ResNet-18 with varying width on CIFAR-100 for SGD, SWAG and MultiSWAG when 10% of the labels are randomly reshuffled. MultiSWAG alleviates double descent both on the original labels and under label noise, both in accuracy and NLL. **(e)**: Test NLLs for MultiSWAG with varying number of independent models under 20% label corruption; NLL monotonically decreases with increased number of independent models, alleviating double descent.

trained SWAG models with MultiSWAG, and for 20% label corruption we use 10 models; for 20% label corruption we also show performance varying the number of independent models in Figures **??** and **??**. Both for SWAG and MultiSWAG we use an ensemble of 20 sampled models from each of the SWAG solutions; for example, for MultiSWAG with 10 independent SWAG solutions, we use an ensemble of 200 networks.

### A.4.5 Analysis of Prior Variance Effect

In this section we provide simple analytic results for the effect of prior variance in ReLU networks. A related derivation is presented in the Appendix Section A.8 of Garipov et al. [2018] about connecting paths from symmetries in parametrization.

We will consider a multilayer network $f(x, w)$ of the form

$$f(x, \{W_i, b_i\}_{i=1}^n) = \tag{A.5}$$

$$W_n(\dots\phi(W_2\phi(W_1x + b_1) + b_2)) + b_n, \tag{A.6}$$

where $\phi$ is the ReLU (or in fact any positively-homogeneous activation function), $W_i$ are weight matrices and $b_i$ are bias vectors. In particular, $f$ can be a regular CNN with ReLU activations up to the logits (with softmax activation removed).

Now, suppose we have a prior distribution of the form

$$W_i \sim \mathcal{N}(0, \alpha_i^2 I), \quad b_i \sim \mathcal{N}(0, \beta_i^2 I), \tag{A.7}$$

where the identity matrices $I$ are implicitly assumed to be of appropriate shapes, so each weight matrix and bias vector has a spherical Gaussian distribution. We can reparameterize this distribution as

$$W_i = \alpha_i \mathbb{E}_i, \quad \mathbb{E}_i \sim \mathcal{N}(0, I), \tag{A.8}$$

$$b_i = \beta_i \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, I). \tag{A.9}$$

We can then express the predictions of the network on the input $x$ for weights sampled from the prior as the random variable

$$\begin{aligned} & f(x, \{\alpha_i, \beta_i\}_{i=1}^n) = \\ & \alpha_n \cdot \mathbb{E}_n(\ldots \phi(\alpha_1 \mathbb{E}_1 x + \beta_1 \cdot \epsilon_1)) + \beta_n \cdot \epsilon_n. \end{aligned} \tag{A.10}$$

Through Equation (A.10), we can observe some simple properties of the dependence between the prior scales $\alpha_i$, $\beta_i$ and the induced function-space prior.

**Proposition A.1.** *Suppose the network has no bias vectors, i.e.* $\beta_1 = \ldots = \beta_n = 0$. *Then the scales* $\alpha_i$ *of the prior distribution over the weights only affect the output scale of the network.*

*Proof.* In the case when there are no bias vectors Equation (A.10) simplifies to

$$\begin{aligned}
&f(x,\{\alpha_i, \beta_i = 0\}_{i=1}^n) = \\
&\quad \alpha_n \cdot \mathbb{E}_n(\ldots \phi(\alpha_1 \mathbb{E}_1 x + \beta_1 \cdot \epsilon_1)) + \beta_n \cdot \epsilon_n = \\
&\quad \alpha_n \cdot \ldots \cdot \alpha_1 \cdot \mathbb{E}_n(\ldots \phi(\mathbb{E}_1 x)) = \\
&\quad \alpha_n \cdot \ldots \cdot \alpha_1 \cdot f(x, \{\alpha_i = 1, \beta_i = 0\}_{i=1}^n).
\end{aligned}$$

In the derivation above we used positive homogeneity of ReLU: $\phi(\alpha z) = \alpha\phi(z)$ for any positive $\alpha$. □

In other words, to sample from the distribution over functions corresponding to a prior with variances $\{\alpha_i, \beta_i = 0\}_{i=1}^n$, we can sample from the spherical Gaussian prior (without bias terms) $\{\alpha_i = 1, \beta_i = 0\}_{i=1}^n$ and then rescale the outputs of the network by the product of variances $\alpha_n \cdot \ldots \cdot \alpha_2 \cdot \alpha_1$.

We note that the result above is different from the results for *sigmoid* networks considered in MacKay [1995], where varying the prior on the weights leads to changing the length-scale of the sample functions. For ReLU networks without biases, increasing prior variance only increases the output scale of the network and not the complexity of the samples. If we apply the softmax activation on the outputs of the last layer of such network, we will observe increasingly confident predictions as we increase the prior variance. We observe this effect in Figure 3.12 and discuss it in Section 3.10.4.

In case bias vectors are present, we can obtain a similar result using a specific scaling of the prior variances with layer, as in the following proposition.

**Proposition A.2.** *Suppose the prior scales depend on the layer of the network as follows for some*

*$\gamma > 0$:*

$$\alpha_i = \gamma, \quad \beta_i = \gamma^i,$$

*for all layers $i = 1 \ldots n$. Then $\gamma$ only affects the scale of the predictive distribution at any input $x$:*

$$f(x, \{\alpha_i = \gamma, \beta_i = \gamma^i\}_{i=1}^n) = \gamma^n \cdot f(x, \{\alpha_i = 1, \beta_i = 1\}_{i=1}^n).$$

*Proof.* The proof is analogous to the proof of Proposition A.1. We can use the positive homogenety of ReLU activations to factor the prior scales outside of the network:

$$\begin{aligned}
&f(x, \{\alpha_i = \gamma, \beta_i = \gamma^i\}_{i=1}^n) = \\
&\quad \gamma \cdot \mathbb{E}_n(\ldots \phi(\gamma \cdot \mathbb{E}_1 x + \gamma \cdot \epsilon_1)) + \gamma^n \cdot \epsilon_n = \\
&\quad \gamma^n \cdot \big(\mathbb{E}_n(\ldots \phi(\mathbb{E}_1 x + \epsilon_1))) + \epsilon_n\big) = \\
&\quad \gamma^n \cdot f(x, \{\alpha_i = 1, \beta_i = 1\}_{i=1}^n).
\end{aligned}$$

□

The analysis above can be applied to other simple scaling rules of the prior, e.g.

$$\begin{aligned}
&f(x, \{\alpha_i = \gamma\hat{\alpha}_i, \beta_i = \gamma^i \hat{\beta}_i\}_{i=1}^n) = \\
&\quad \gamma^n \cdot f(x, \{\alpha_i = \hat{\alpha}_i, \beta_i = \hat{\beta}_i\}_{i=1}^n),
\end{aligned} \tag{A.11}$$

can be shown completely analogously to Proposition A.2.

More general types of scaling of the prior affect both the output scale of the network and also the relative effect of prior and variance terms. For example, by Equation (A.11) we have

$$\begin{aligned} & f(x,\{\alpha_i = \gamma, \beta_i = \gamma\}_{i=1}^n) = \\ & f(x,\{\alpha_i = \gamma \cdot 1, \beta_i = \gamma^i \cdot \gamma^{1-i}\}_{i=1}^n) = \\ & \quad \gamma^n \cdot f(x, \{\alpha_i = 1, \beta_i = \gamma^{1-i}\}_{i=1}^n). \end{aligned}$$

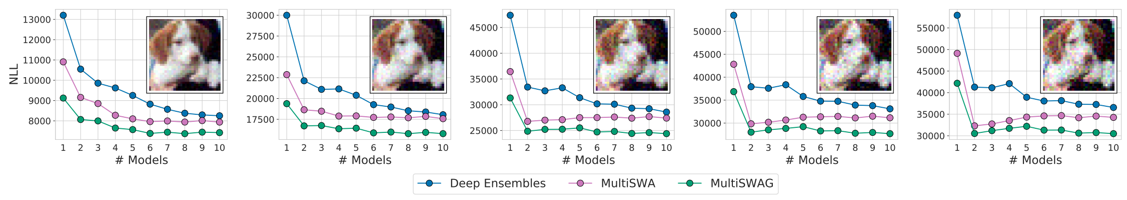
We note that the analysis does not cover residual connections and batch normalization, so it applies to LeNet-5 but cannot be directly applied to PreResNet-20 networks used in many of our experiments.

### A.4.6 Prior Correlation Structure under Perturbations

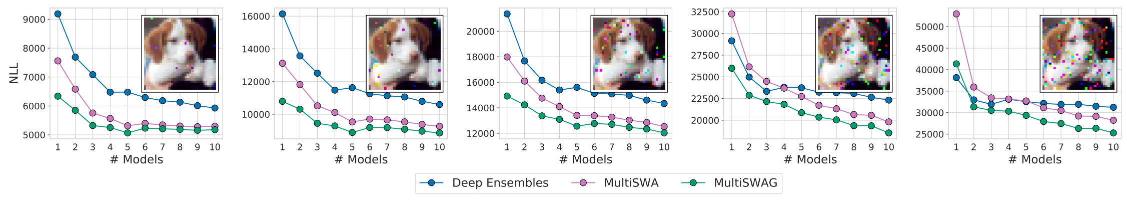
In this section we explore the prior correlations between the logits on different pairs of datapoints induced by a spherical Gaussian prior on the weights of a PreResNet-20. We sample a 100 random images from CIFAR-10 (10 from each class) and apply 17 different perturbations introduced by Hendrycks and Dietterich [2019] at 5 different levels of intensity. We then compute correlations between the logits $f(x, w)$ for the original image $x$ and $f(\tilde{x}, w)$ for the corrupted image $\tilde{x}$, as we sample the weights of the network from the prior $w \sim \mathcal{N}(0, I)$.

In Figure A.23 we show how the correlations decay with perturbation intensity. For reference we also show how the correlations decay for a linear model and for an RBF kernel. For the RBF kernel we set the lengthscale so that the average correlations on the uncorrupted datapoints match those of a PreResNet-20. Further experimental details can be found in Appendix A.4.4.3.

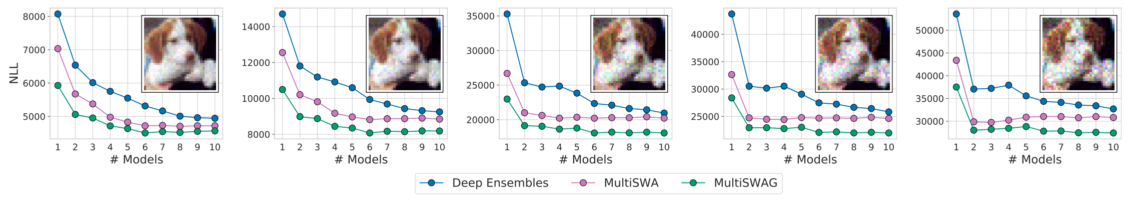
For all types of corruptions except *saturate*, *snow*, *fog* and *brightness* the PreResNet logits decay slower compared to the RBF kernel and linear model. It appears that the prior samples are sensitive to corruptions that alter the brightness or more generally the colours in the image. For many types of corruptions (such as e.g. *Gaussian Noise*) the prior correlations for PreResNet are close to 1 for all levels of corruption.



(a) Gaussian Noise



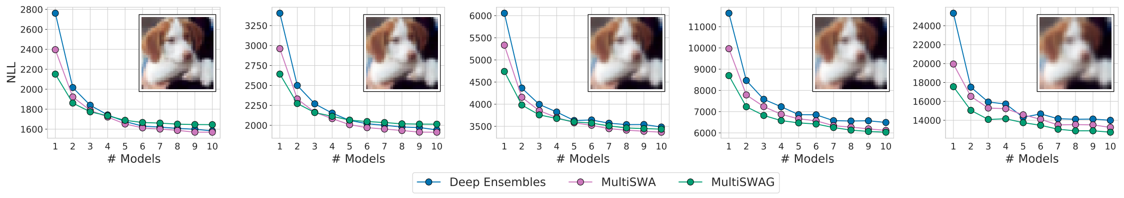
(b) Impulse Noise



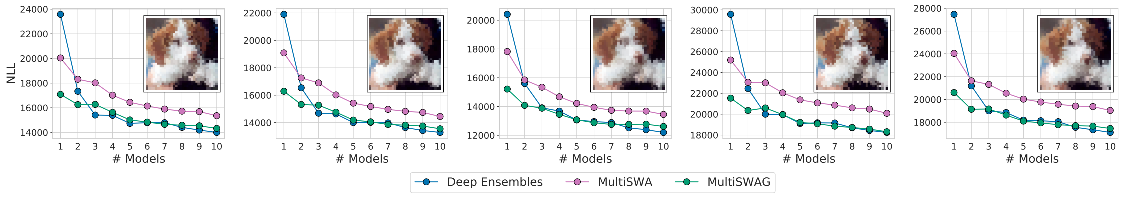
(c) Shot Noise

**Figure A.18: Noise Corruptions.** Negative log likelihood on CIFAR-10 with a PreResNet-20 for Deep Ensembles, MultiSWAG and MultiSWA as a function of the number of independently trained models for different types of corruption and corruption intensity (increasing from left to right).

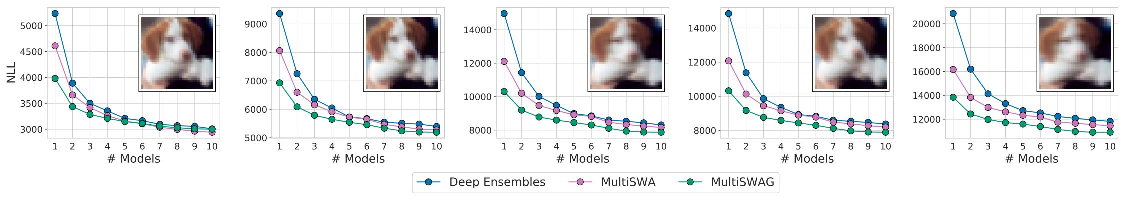
Overall, these results indicate that the prior over *functions* induced by a vague prior over parameters $w$ in combination with a PreResNet has useful equivariance properties: before seeing data, the model treats images of the same class as highly correlated, even after an image has undergone significant perturbations representative of perturbations we often see in the real world. These types of symmetries are a large part of what makes neural networks a powerful model class for high dimensional natural signals.



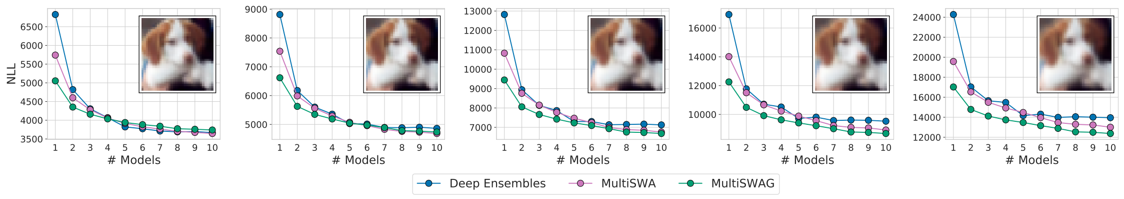
(a) Defocus Blur



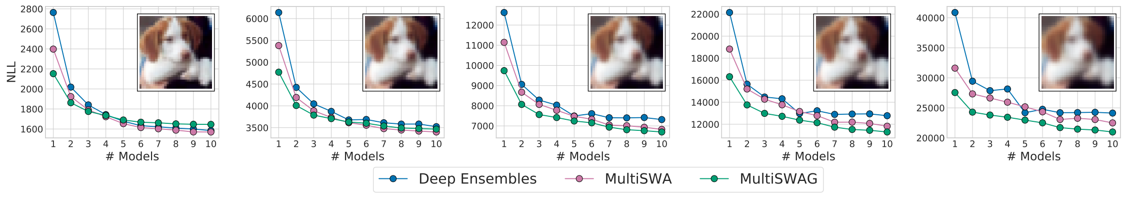
(b) Glass Blur



(c) Motion Blur

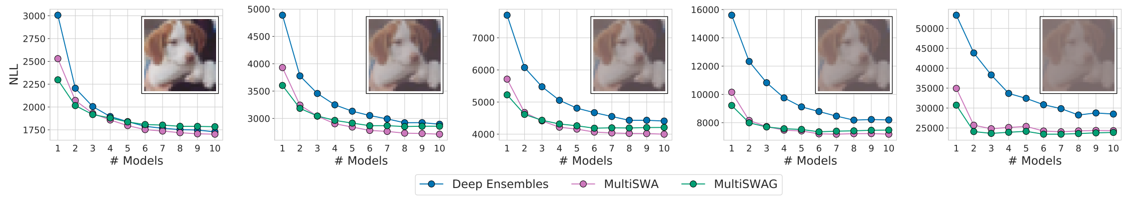


(d) Zoom Blur

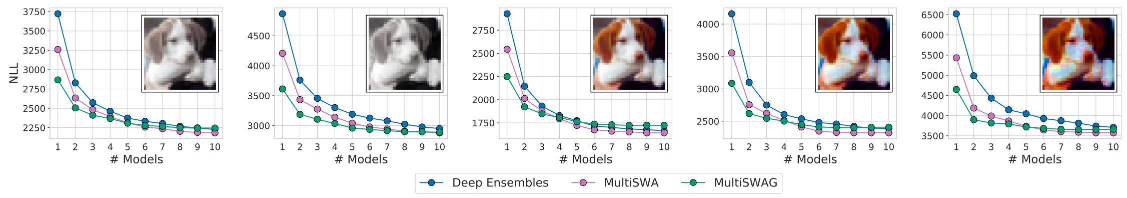


(e) Gaussian Blur

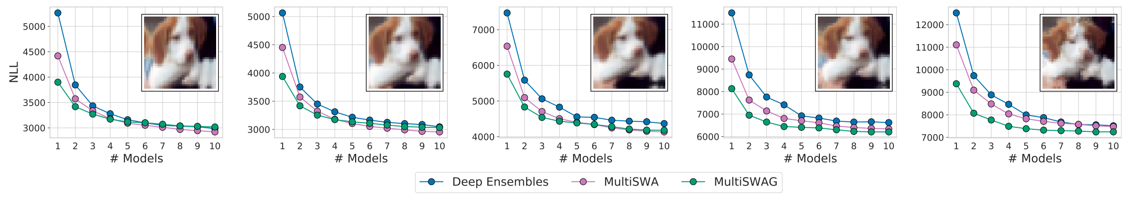
**Figure A.19: Blur Corruptions.** Negative log likelihood on CIFAR-10 with a PreResNet-20 for Deep Ensembles, MultiSWAG and MultiSWA as a function of the number of independently trained models for different types of corruption and corruption intensity (increasing from left to right).



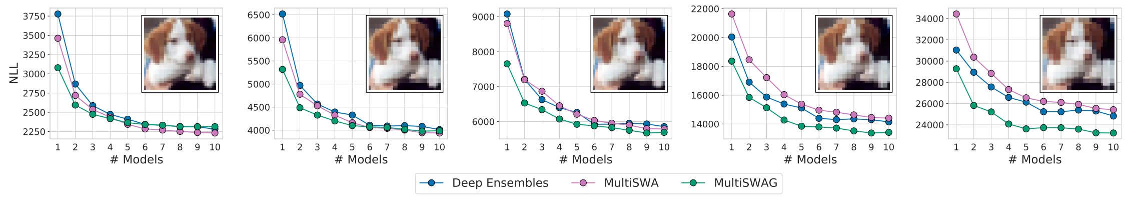
(a) Contrast



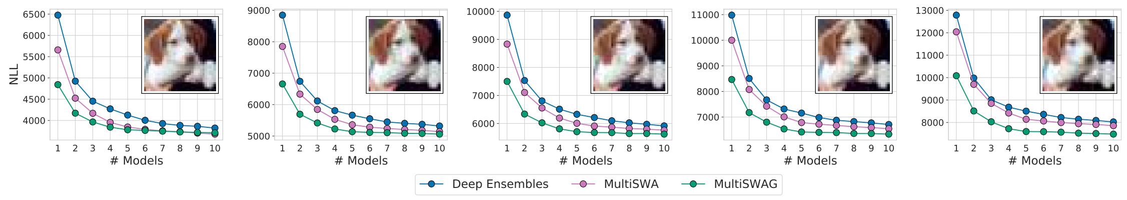
(b) Saturate



(c) Elastic Transform

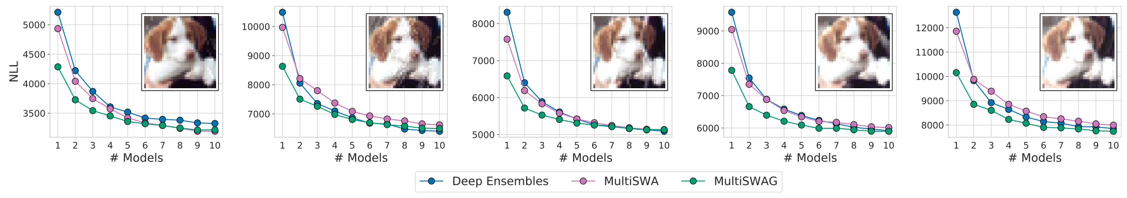


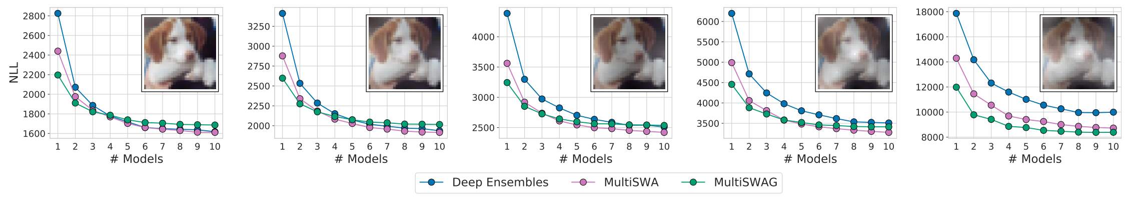
(d) Pixelate



(e) JPEG Compression

**Figure A.20: Digital Corruptions.** Negative log likelihood on CIFAR-10 with a PreResNet-20 for Deep Ensembles, MultiSWAG and MultiSWA as a function of the number of independently trained models for different types of corruption and corruption intensity (increasing from left to right).


(a) Snow


(b) Fog

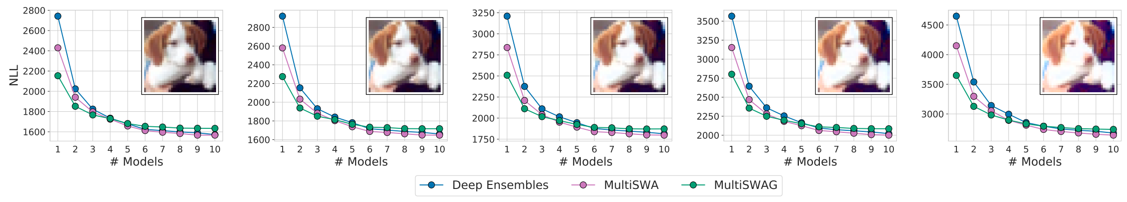
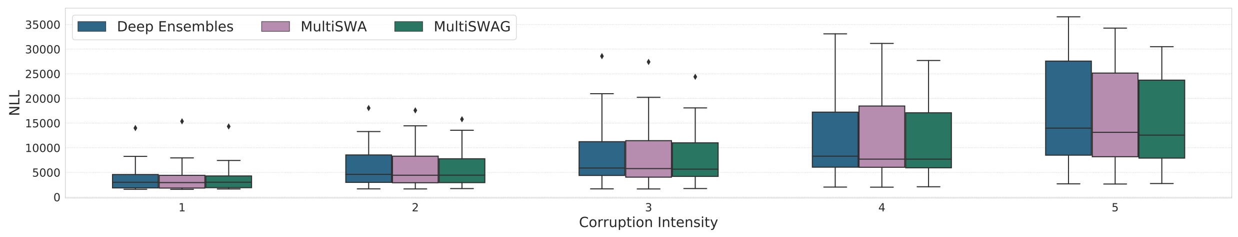
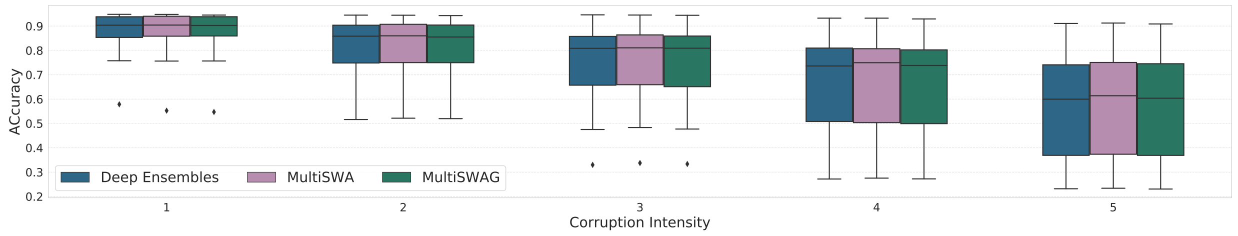

(c) Brightness

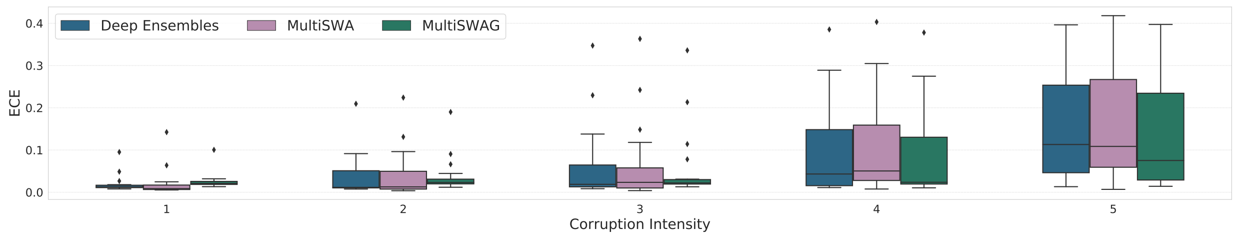
**Figure A.21: Weather Corruptions.** Negative log likelihood on CIFAR-10 with a PreResNet-20 for Deep Ensembles, MultiSWAG and MultiSWA as a function of the number of independently trained models for different types of corruption and corruption intensity (increasing from left to right).



(a) Negative log likelihood



(b) Accuracy



(c) Expected calibration error

**Figure A.22:** Negative log likelihood, accuracy and expected calibration error distribution on CIFAR-10 with a PreResNet-20 for Deep Ensembles, MultiSWAG and MultiSWA as a function of the corruption intensity. Following Ovadia et al. [2019] we summarize the results for different types of corruption with a boxplot. For each method, we use 10 independently trained models, and for MultiSWAG we sample 20 networks from each model. As in Figures 5, 11-14, there are substantial differences between these three methods, which are hard to see due to the vertical scale on this plot. MultiSWAG particularly outperforms Deep Ensembles and MultiSWA in terms of NLL and ECE for higher corruption intensities.

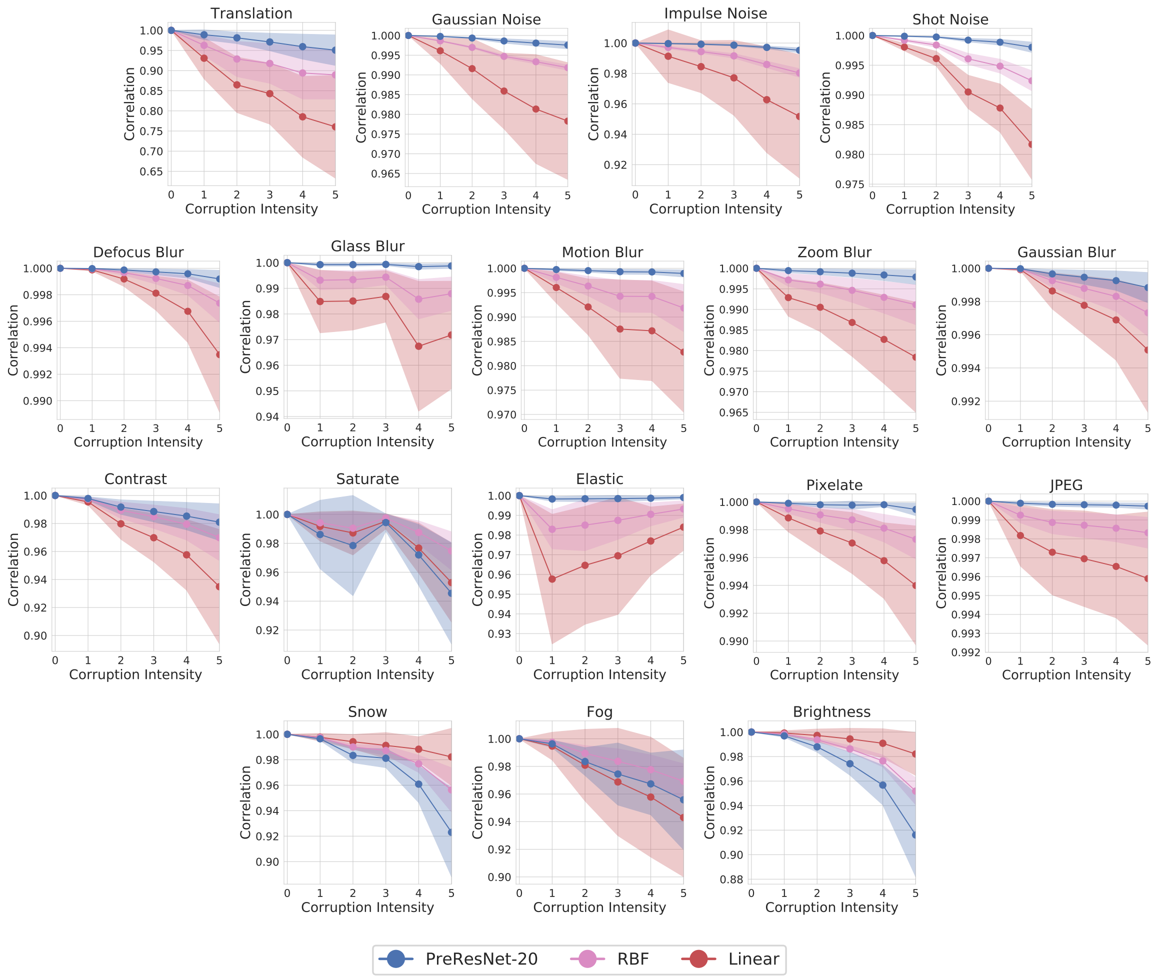


**Figure A.23: Prior correlations under corruption.** Prior correlations between predictions (logits) for PreResNet-20, Linear Model and RBF kernel on original and corrupted images as a function of corruption intensity for different types of corruptions. The lengthscale of the RBF kernell is calibrated to produce similar correlations to PreResNet on uncorrupted datapoints. We report the mean correlation values over 100 different images and show the $1\sigma$ error bars with shaded regions. For all corruptions except Snow, Saturate, Fog and Brightness the correlations decay slower for PreResNet compared to baselines.

| Method | Hyper-parameter | Was Tuned | Experiments<br>CIFAR-10<br>ResNet-20-FRN | <br>CIFAR-100<br>ResNet-20-FRN | <br>IMDB<br>CNN LSTM |
|---|---|---|---|---|---|
| HMC | Prior Variance | ✓ | $\frac{1}{5}$ | $\frac{1}{5}$ | $\frac{1}{40}$ |
| | Step Size | ✓ | $10^{-5}$ | $10^{-5}$ | $10^{-5}$ |
| | Num. Burnin Iterations | ✗ | 50 | 50 | 50 |
| | Num. Samples per Chain | ✗ | 240 | 40 | 400 |
| | Num. of Chains | ✗ | 3 | 3 | 3 |
| | Total Samples | ✗ | 720 | 120 | 1200 |
| | Total Epochs | | $5 \cdot 10^7$ | $8.5 \cdot 10^6$ | $3 \cdot 10^7$ |
| SGD | Weight Decay | ✓ | 10 | 10 | 3 |
| | Initial Step Size | ✓ | $3 \cdot 10^{-7}$ | $1 \cdot 10^{-6}$ | $3 \cdot 10^{-7}$ |
| | Step Size Schedule | ✗ | cosine | cosine | cosine |
| | Batch Size | ✓ | 80 | 80 | 80 |
| | Num. Epochs | ✗ | 500 | 500 | 500 |
| | Momentum | ✗ | 0.9 | 0.9 | 0.9 |
| | Total Epochs | | $5 \cdot 10^2$ | $5 \cdot 10^2$ | $5 \cdot 10^2$ |
| Deep Ensembles | Num. Models | ✗ | 50 | 50 | 50 |
| | Total Epochs | | $2.5 \cdot 10^4$ | $2.5 \cdot 10^4$ | $2.5 \cdot 10^4$ |
| SGLD | Prior Variance | ✓ | $\frac{1}{5}$ | $\frac{1}{5}$ | $\frac{1}{5}$ |
| | Step Size | ✓ | $10^{-6}$ | $3 \cdot 10^{-6}$ | $1 \cdot 10^{-5}$ |
| | Step Size Schedule | ✓ | constant | constant | constant |
| | Batch Size | ✓ | 80 | 80 | 80 |
| | Num. Epochs | ✗ | 10000 | 10000 | 10000 |
| | Num. Burnin Epochs | ✗ | 1000 | 1000 | 1000 |
| | Num. Samples per Chain | ✗ | 900 | 900 | 900 |
| | Num. of Chains | ✗ | 5 | 5 | 5 |
| | Total Samples | ✗ | 4500 | 4500 | 4500 |
| | Total Epochs | | $5 \cdot 10^4$ | $5 \cdot 10^4$ | $5 \cdot 10^4$ |
| MFVI | Prior Variance | ✗ | $\frac{1}{5}$ | $\frac{1}{5}$ | $\frac{1}{5}$ |
| | Num. Epochs | ✗ | 300 | 300 | 300 |
| | Optimizer | ✓ | Adam | Adam | Adam |
| | Initial Step Size | ✓ | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| | Step Size Schedule | ✗ | cosine | cosine | cosine |
| | Batch Size | ✓ | 80 | 80 | 80 |
| | VI mean init | ✗ | SGD solution | SGD solution | SGD solution |
| | VI variance init | ✓ | $10^{-2}$ | $10^{-2}$ | $10^{-2}$ |
| | Number of samples | ✗ | 50 | 50 | 50 |
| | Total Epochs | | $8 \cdot 10^2$ | $8 \cdot 10^2$ | $8 \cdot 10^2$ |

**Table A.16: Hyper-parameters for CIFAR and IMDB.** We report the hyper-parameters for each method our main evaluations on CIFAR and IMDB datasets in Section 4.4. For each method we report the total number of training epochs equivalent to the amount of compute spent. We run HMC on a cluster of 512 TPUs, and the baselines on a cluster of 8 TPUs. For each of the hyper-parameters we report whether it was tuned via cross-validation, or whether a value was selected without tuning.

| Method | Hyper-parameter | Was Tuned | Experiments | | | | |
|---|---|---|---|---|---|---|---|
| | | | Concrete | Yacht | Energy | Boston | Naval |
| HMC | Prior Variance | ✓ | $\frac{1}{10}$ | $\frac{1}{10}$ | $\frac{1}{10}$ | $\frac{1}{10}$ | $\frac{1}{40}$ |
| | Step Size | ✓ | $10^{-5}$ | $10^{-5}$ | $10^{-5}$ | $10^{-5}$ | $5 \cdot 10^{-7}$ |
| | Num. Burnin Iterations | ✗ | 10 | 10 | 10 | 10 | 10 |
| | Num. Iterations | ✗ | 90 | 90 | 90 | 90 | 90 |
| | Num. of Chains | ✗ | 1 | 1 | 1 | 1 | 1 |
| SGD | Weight Decay | ✓ | 10 | $10^{-1}$ | 10 | $10^{-1}$ | 1 |
| | Initial Step Size | ✓ | $3 \cdot 10^{-5}$ | $3 \cdot 10^{-6}$ | $3 \cdot 10^{-6}$ | $3 \cdot 10^{-6}$ | $10^{-6}$ |
| | Step Size Schedule | ✗ | cosine | cosine | cosine | cosine | cosine |
| | Batch Size | ✗ | 927 | 277 | 691 | 455 | 10740 |
| | Num. Epochs | ✓ | 1000 | 5000 | 5000 | 500 | 1000 |
| | Momentum | ✗ | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 |
| SGLD | Prior Variance | ✓ | $\frac{1}{10}$ | $\frac{1}{10}$ | $\frac{1}{10}$ | $\frac{1}{10}$ | 1 |
| | Step Size | ✓ | $3 \cdot 10^{-5}$ | $10^{-4}$ | $3 \cdot 10^{-5}$ | $3 \cdot 10^{-5}$ | $10^{-6}$ |
| | Step Size Schedule | ✗ | constant | constant | constant | constant | constant |
| | Batch Size | ✗ | 927 | 277 | 691 | 455 | 10740 |
| | Num. Epochs | ✗ | 10000 | 10000 | 10000 | 10000 | 10000 |
| | Num. Burnin Epochs | ✗ | 1000 | 1000 | 1000 | 1000 | 1000 |
| | Num. Samples per Chain | ✗ | 900 | 900 | 900 | 900 | 900 |
| | Num. of Chains | ✗ | 1 | 1 | 1 | 1 | 1 |

**Table A.17: Hyper-parameters for UCI.** We report the hyper-parameters for each method our main evaluations on UCI datasets in Section 4.4. For HMC, the number of iterations is the number of HMC iterations after the burn-in phase; the number of accepted samples is lower. For each of the hyper-parameters we report whether it was tuned via cross-validation, or whether a value was selected without tuning.

| Hyper-parameter | Was Tuned | SGLD | SGHMC | SGHMC CLR | SGHMC CLR-Prec |
|---|---|---|---|---|---|
| Initial Step size | ✓ | $10^{-6}$ | $3 \cdot 10^{-7}$ | $3 \cdot 10^{-7}$ | $3 \cdot 10^{-5}$ |
| Step Size Schedule | ✗ | constant | constant | cyclical | cyclical |
| Momentum | ✓ | 0. | 0.9 | 0.95 | 0.95 |
| Preconditioner | ✗ | None | None | None | RMSprop |
| Num. Samples per chain | ✗ | 900 | 900 | 180 | 180 |
| Num. of Chains | ✗ | 3 | 3 | 3 | 3 |

**Table A.18: SGMCMC hyper-parameters on CIFAR-10.** We report the hyper-parameter values used by each of the SGMCMC methods in Section 4.10. The remaining hyper-parameters are the same as the SGLD hyper-parameters reported in Table A.16. For each of the hyper-parameters we report whether it was tuned via cross-validation, or whether a value was selected without tuning.

## A.5 Appendix for "What Are Bayesian Neural Network Posteriors Really Like?"

This appendix is organized as follows. We present the Hamiltonian Monte Carlo algorithm that we implement in the paper in Figure 4, Figure 5. In Section A.5.1 we provide the details on hyper-parameters used in our experiments. In Section A.5.2 we provide a description of the $\hat{R}$ statistic used in Section 4.3.1. In Section A.5.3 we study the marginals of the posterior over parameters estimated by HMC. In Section A.5.4 we provide additional posterior density surface visualizations. In Section A.5.5 we compare the BMA predictions using two independent HMC chains on a synthetic regression problem. In Section A.5.6 we show that BNNs are not robust to distribution shift and discuss the reasons for this behavior. In Section A.5.7 we provide a further discussion of the effect of posterior temperature. Finally, in Section A.5.8 we visualize the predictions of HMC samples from a single chain on several CIFAR-10 test inputs.

### A.5.1 Hyper-Parameters and Details

**CIFAR and IMDB.** In Table A.16 we report the hyperparameters used by each of the methods in our main evaluation on CIFAR and IMDB datasets in Section 4.4. HMC was run on a cluster of 512 TPUs and the other baselines were run on a cluster of 8 TPUs. On CIFAR datasets the methods used a subset of 40960 datapoints. All methods were ran at posterior temperature 1. We tuned the hyper-parameters for all methods via cross-validation maximizing the accuracy on a validation set. For the step-sizes we considered an exponential grid with a step of $\sqrt{10}$ with 5-7 different values, where the boundaries were selected for each method so it would not diverge. We considered weight decays 1, 5, 10, 20, 40, 100 and the corresponding prior variances. For batch sizes we considered values 80, 200, 400 and 1000; for all methods lower batch sizes resulted in the best performance. For HMC we set the trajectory length according to the strategy described in

Section 4.2.1. For SGLD, we experimented with using a cosine learning rate schedule decaying to a non-zero final step size, but it did not improve upon a constant schedule. For MFVI we experimented with the SGD and Adam optimizers; we initialize the mean of the MFVI distribution with a pre-trained SGD solution, and the per-parameter variance with a value $\sigma_{\text{Init}}^{\text{VI}}$; we tested values $10^{-2}, 10^{-1}, 10^{0}$ for $\sigma_{\text{Init}}^{\text{VI}}$. For all HMC hyper-parameters, we provide ablations illustrating their effect in Section 4.2. Producing a single sample with HMC on CIFAR datasets takes roughly one hour on our hardware, and on IMDB it takes 105 seconds; we can run up to three chains in parallel.

**Temperature scaling on IMDB.** For the experiments in Section 4.8 we run a single HMC chain producing 40 samples after 10 burn-in epochs for each temperature. We used step-sizes $5 \cdot 10^{-5}$, $3 \cdot 10^{-5}$, $10^{-5}$, $3 \cdot 10^{-6}$, $10^{-6}$ and $3 \cdot 10^{-7}$ for temperatures 10, 3, 1, 0.3, 0.1 and 0.03 respectively, ensuring that the accept rates were close to 100%. We used a prior variance of 1/50 in all experiments; the lower prior variance compared to Table A.16 was chosen to reduce the number of leapfrog iterations, as we chose the trajectory length according to the strategy described in Section 4.2.1. We ran the experiments on 8 NVIDIA Tesla V-100 GPUs, as we found that sampling at low temperatures requires `float64` precision which is not supported on TPUs.

**UCI Datasets.** In Table A.17 we report the hyperparameters used by each of the methods in our main evaluation on UCI datasets in Section 4.4. For each datasets we construct 20 random splits with 90% of the data in the train and 10% of the data in the test split. In the evaluation, we report the mean and standard deviation of the results across the splits. We use another random split for cross-validation to tune the hyper-parameters. For all datasets we use a fully-connected network with a single hidden layer with 50 neurons and 2 outputs representing the predictive mean and variance for the given input. We use a Gaussian likelihood to train each of the methods. For the SGD and SGLD baselines, we did not use mini-batches: the gradients were computed over the entire dataset. We run each experiment on a single NVIDIA Tesla V-100 GPU.

**SGMCMC Methods.** In Table A.18 we report the hyper-parameters of the SGMCMC methods on the CIFAR-10 dataset used in the evaluation in Section 4.10. We considered momenta in the set of $\{0.9, 0.95, 0.99\}$ and step sizes in $\{10^{-4}, 3 \cdot 10^{-5}, 10^{-5}, 3 \cdot 10^{-6}, 10^{-6}, 3 \cdot 10^{-7}, 10^{-7}\}$. We selected the hyper-parameters with the best accuracy on the validation set. SGLD does not allow a momentum.

### A.5.2 Description of $\hat{R}$ Statistics

$\hat{R}$ [Gelman et al. 1992] is a popular MCMC convergence diagnostic. It is defined in terms of some scalar function $\psi(\theta)$ of the Markov chain iterates $\{\theta_{mn} | m \in \{1, \ldots, M\}, n \in \{1, \ldots, N\}\}$, where $\theta_{mn}$ denotes the state of the $m$th of $M$ chains at iteration $n$ of $N$. Letting $\psi_{mn} \triangleq \psi(\theta_{mn})$, $\hat{R}$ is defined as follows:

$$\bar{\psi}_{m\cdot} \triangleq \frac{1}{N} \sum_n \psi_{mn}; \quad \bar{\psi}_{\cdot\cdot} \triangleq \frac{1}{MN} \sum_{m,n} \psi_{mn}; \tag{A.12}$$

$$\frac{B}{N} \triangleq \frac{1}{M-1} \sum_m (\bar{\psi}_{m\cdot} - \bar{\psi}_{\cdot\cdot})^2; \tag{A.13}$$

$$W \triangleq \frac{1}{M(N-1)} \sum_{m,n} (\psi_{mn} - \bar{\psi}_{m\cdot})^2; \tag{A.14}$$

$$\hat{\sigma}_+^2 \triangleq \frac{N-1}{N} W + \frac{B}{N}; \tag{A.15}$$

$$\hat{R} \triangleq \frac{M+1}{M} \frac{\hat{\sigma}_+^2}{W} - \frac{N-1}{MN}. \tag{A.16}$$

If the chains were initialized from their stationary distribution, then $\hat{\sigma}_+^2$ would be an unbiased estimate of the stationary distribution's variance. $W$ is an estimate of the average within-chain variance; if the chains are stuck in isolated regions, then $W$ should be smaller than $\hat{\sigma}_+^2$, and $\hat{R}$ will be clearly larger than 1. The $\frac{M+1}{M}$ and $\frac{N-1}{MN}$ terms are there to account for sampling variability—they vanish as $N$ gets large if $W$ approaches $\hat{\sigma}_+^2$.

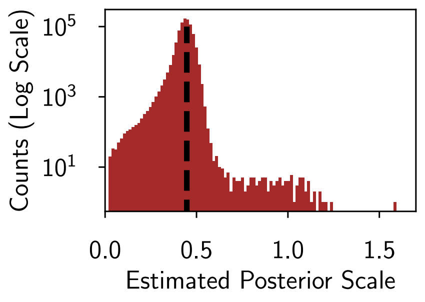
Since $\hat{R}$ is defined in terms of a function of interest $\psi$, we can compute it for many such functions. In Section 4.3.1 we evaluated it for each weight and each predicted softmax probability

**Algorithm 4** Hamiltonian Monte Carlo

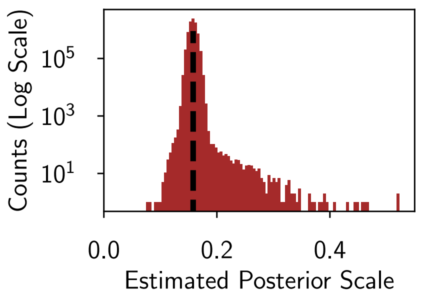
```
**Input:** Trajectory length $\tau$, number of burn-in interations $N_{\text{burnin}}$, initial parameters $w_{\text{init}}$, step size $\Delta$, number of samples $K$, unnormalized posterior log-density function $f(w) = \log p(D|w) + \log p(w)$.
**Output:** Set $S$ of samples $w$ of the parameters.
$w \leftarrow w_{\text{init}}$;   $N_{\text{leapfrog}} \leftarrow \frac{\tau}{\Delta}$;
# Burn-in stage
**for** $i \leftarrow 1 \dots N_{\text{burnin}}$ **do**
    $m \sim \mathcal{N}(0, I)$;
    $(w, m) \leftarrow \text{Leapfrog}(w, m, \Delta, N_{\text{leapfrog}}, f)$;
**end for**
# Sampling
$S \leftarrow \varnothing$;
**for** $i \leftarrow 1 \dots K$ **do**
    $m \sim \mathcal{N}(0, I)$;
    $(w', m') \leftarrow \text{Leapfrog}(w, m, \Delta, N_{\text{leapfrog}}, f)$;

    # Metropolis-Hastings correction
    $p_{\text{accept}} \leftarrow \min\left\{1, \frac{f(w')}{f(w)} \cdot \exp\left(\frac{1}{2}\|m\|^2 - \|m'\|^2\right)\right\}$;
    $u \sim \text{Uniform}[0, 1]$;
    **if** $u \leqslant p_{\text{accept}}$ **then**
        $w \leftarrow w'$;
    **end if**
    $S \leftarrow S \cup \{w\}$;
**end for**
```

**Algorithm 5** Leapfrog integration

```
**Input:** Parameters $w_0$, initital momentum $m_0$, step size $\Delta$, number of leapfrog steps $N_{\text{leapfrog}}$, posterior log-density function $f(w) = \log p(w|D)$.
**Output:** New parameters $w$; new momentum $m$.
$w \leftarrow w_0$;   $m \leftarrow m_0$;
**for** $i \leftarrow 1 \dots N_{\text{leapfrog}}$ **do**
    $m \leftarrow m + \frac{\Delta}{2} \cdot \nabla f(w)$;
    $w \leftarrow w + \Delta \cdot m$;
    $m \leftarrow m + \frac{\Delta}{2} \cdot \nabla f(w)$;
**end for**
$\text{Leapfrog}(w_0, m_0, \Delta, N_{\text{leapfrog}}, f) \leftarrow (w, m)$
```



(a) ResNet-20-FRN



(b) CNN-LSTM

**Figure A.24: Marginal distributions of the weights.** Log-scale histograms of estimated marginal posterior standard deviations for ResNet-20-FRN on CIFAR-10 and CNN-LSTM on IMDB. The histograms show how many parameters have empirical standard deviations that fall within a given bin. For most of the parameters (notice that the plot is logarithmic) the posterior scale is very similar to that of the prior distribution.

in the test set.

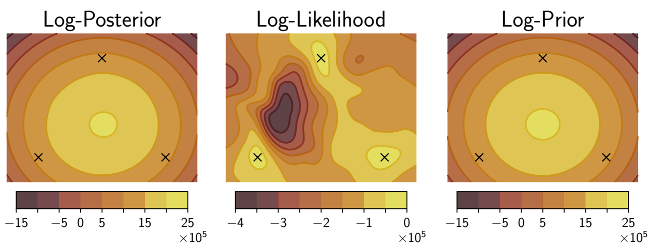
### A.5.3 Marginal distributions of the weights

In Section 4.2.1, we argued for using a trajectory length $\tau = \frac{\pi \sigma_{\text{prior}}}{2}$ based on the intuition that the posterior scale is determined primarily by the prior scale. In Figure A.24 we examine this intuition. For each parameter, we estimate the marginal standard deviation of that parameter under the distribution sampled by HMC. Most of these marginal scales are close to the prior scale, and only a few are significantly larger (note logarithmic scale on y-axis), confirming that the posterior's scale is determined by the prior.

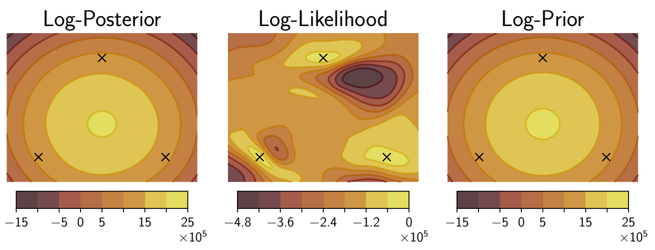
### A.5.4 Additional Posterior Visualizations

In Section 4.3.2 we study two-dimensional cross-sections of posterior log-density, log-likelihood and log-prior surfaces. We provide additional visualizations on the IMDB dataset in Figure A.25.

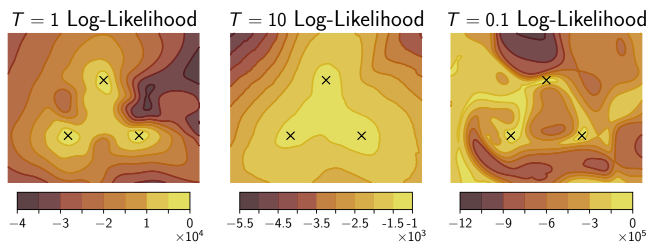
On IMDB, the posterior log-density is dominated by the prior, and the corresponding panels are virtually indistinguishable in Figure A.25. For the CNN-LSTM on IMDB the number of parameters is much larger than the number of data points, and hence the scale of the prior density



(a) IMDB, same chain



(b) IMDB, independent chain



(c) IMDB, Log-Likelihood at different $T$

**Figure A.25: Additional posterior density visualizations.** Visualizations of posterior log-density, log-likelihood and log-prior in two-dimensional subspaces of the parameter space spanned by three HMC samples on IMDB using CNN-LSTM. **(a):** samples from the same chain and **(b)**: independent chains; **(c):** Log-likelihood surfaces for samples from the same chain at posterior temperatures $T = 1$, 10 and 0.1.

values is much larger than the scale of the likelihood. Note that the likelihood still affects the posterior typical set, and the HMC samples land in the modes of the likelihood in the visualization. In contrast, on ResNet-20, the number of parameters is smaller and the number of data points is larger, so the posterior is dominated by the likelihood in Figure 4.3. The log-likelihood panels for both datasets show that HMC is able to navigate complex geometry: the samples fall in three isolated modes in our two-dimensional cross-sections. On IMDB, the visualizations for samples from a single chain and for samples from three independent chains are qualitatively quite similar, hinting at better parameter-space mixing compared to CIFAR-10 (see Section 4.3.1).

In Figure A.25 (c), we visualize the likelihood cross-sections using our runs with varying posterior temperature on IMDB. The visualizations show that, as expected, low temperature leads to a sharp likelihood, while the high-temperature likelihood appear soft. In particular, the scale of the lowest likelihood values at $T = 10$ is only $10^3$ while the scale at $T = 0.1$ is $10^6$.

**How are the visualizations created?** To create the visualizations we pick the points in the parameter space corresponding to three HMC samples: $w_1, w_2, w_3$. We construct a basis in the 2-dimensional affine subspace passing through these three points: $u = w_2 - w_1$ and $v = w_3 - w_1$. We then orthogonalize the basis: $\hat{u} = u/\|u\|$, $\hat{v} = (v - \hat{u}^T v)/\|v - \hat{u}^T v\|$. We construct a 2-dimensional uniform grid in the basis $\hat{u}, \hat{v}$. Each point in the grid corresponds to a vector of parameters of the network. We evaluate the log-likelihood, log-prior and posterior log-density for each of the points in the grid, converting them to the corresponding network parameters. Finally, we produce contour plots using the collected values. The procedure is analogous to that used by Garipov et al. [2018][6].

### A.5.5 HMC Predictive Distributions in Synthetic Regression

We consider a one-dimensional synthetic regression problem. We follow the general setup of Izmailov et al. [2019] and Wilson and Izmailov [2020]. We generate the training inputs as a uniform grid with 40 points in each of the following intervals (120 datapoints in total): $[-10, -6]$, $[6, 10]$ and $[14, 18]$. We construct the ground truth target values using a neural network with 3 hidden layers, each of dimension 100, one output and two inputs: following Izmailov et al. [2019], for each datapoint $x$ we pass $x$ and $x^2$ as inputs to the network to enlarge the class of functions that the network can represent. We draw the parameters of the network from a Gaussian distribution with mean 0 and standard deviation 0.1. We show the sample function used to generate the target values as a black line in each of the panels in Figure A.26. We then add Gaussian noise with

[6]See also the blogpost `https://izmailovpavel.github.io/curves_blogpost/`, Section "How to Visualize Loss Surfaces?".

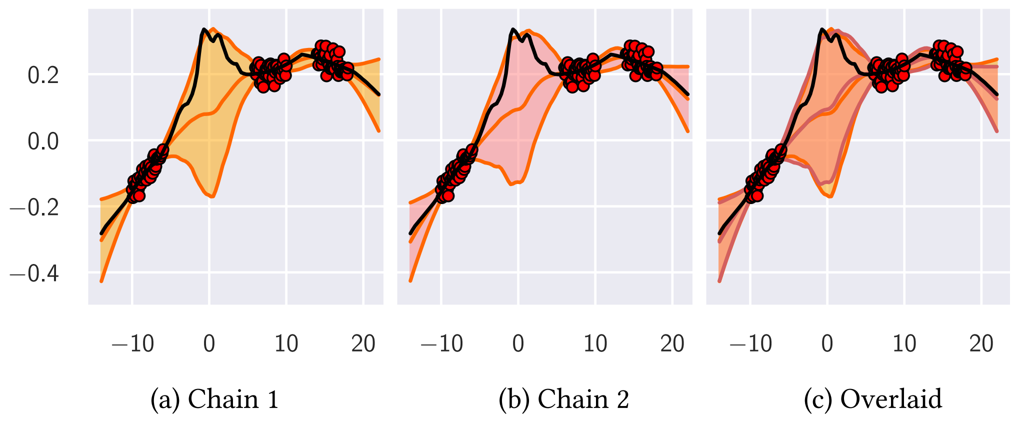


**Figure A.26: HMC chains on synthetic regression.** We visualize the predictive distributions for two independent HMC chains on a synthetic regression problem with a fully-connected network. The data is shown with red circles, and the true data generating function is shown with a black line. The shaded region shows 3 standard deviations of the predictive distribution, and the predictive mean is shown with a line of the same color. In panels **(a)**, **(b)** we show the predictive distributions for each of the two chains individually, and in panel **(c)** we overlay them on top of each other. The chains provide almost identical predictions, suggesting that HMC mixes well in the prediction space.

mean 0 and standard deviation 0.02 to each of the target values. The final dataset used in the experiment is shown with red circles in Figure A.26.

For inference, we use the same model architecture that was used to generate the data. We sample the initialization parameters of the network from a Gaussian distribution with mean 0 and standard deviation 0.005. We use a Gaussian distribution with mean zero and standard deviation 0.1 as the prior over the parameters, same as the distribution used to sample the parameters of the ground truth solution. We use a Gaussian likelihood with standard deviation 0.02, same as the noise distribution in the data. We run two HMC chains from different random initializations. Each chain uses a step-size of $10^{-5}$ and the trajectory length is set according to the strategy described in Section 4.2.1, resulting in 15708 leapfrog steps per HMC iteration. We run each chain for 100 HMC iterations and collect the predictions corresponding to all the accepted samples, resulting in 89 and 82 samples for the first and second chain respectively. We discard the first samples and only use the last 70 samples from each chain. For each input point we compute the mean and standard deviation of the predictions.

We report the results in Figure A.26. In panels (a), (b) we show the predictive distributions for each of the chains, and in panel (c) we show them overlaid on top of each other. Both chains provide high uncertainty away from the data, and low uncertainty near the data as desired [Yao et al. 2019]. Moreover, the true data-generating function lies in the $3\sigma$-region of the predictive distribution for each chain. Finally, the predictive distributions for the two chains are almost identical. This result suggests that on the synthetic problem under consideration HMC is able to mix in the space of predictions, and provides similar results independent of initialization and random seed. We come to the same conclusion for more realistic problems in Section 4.3.

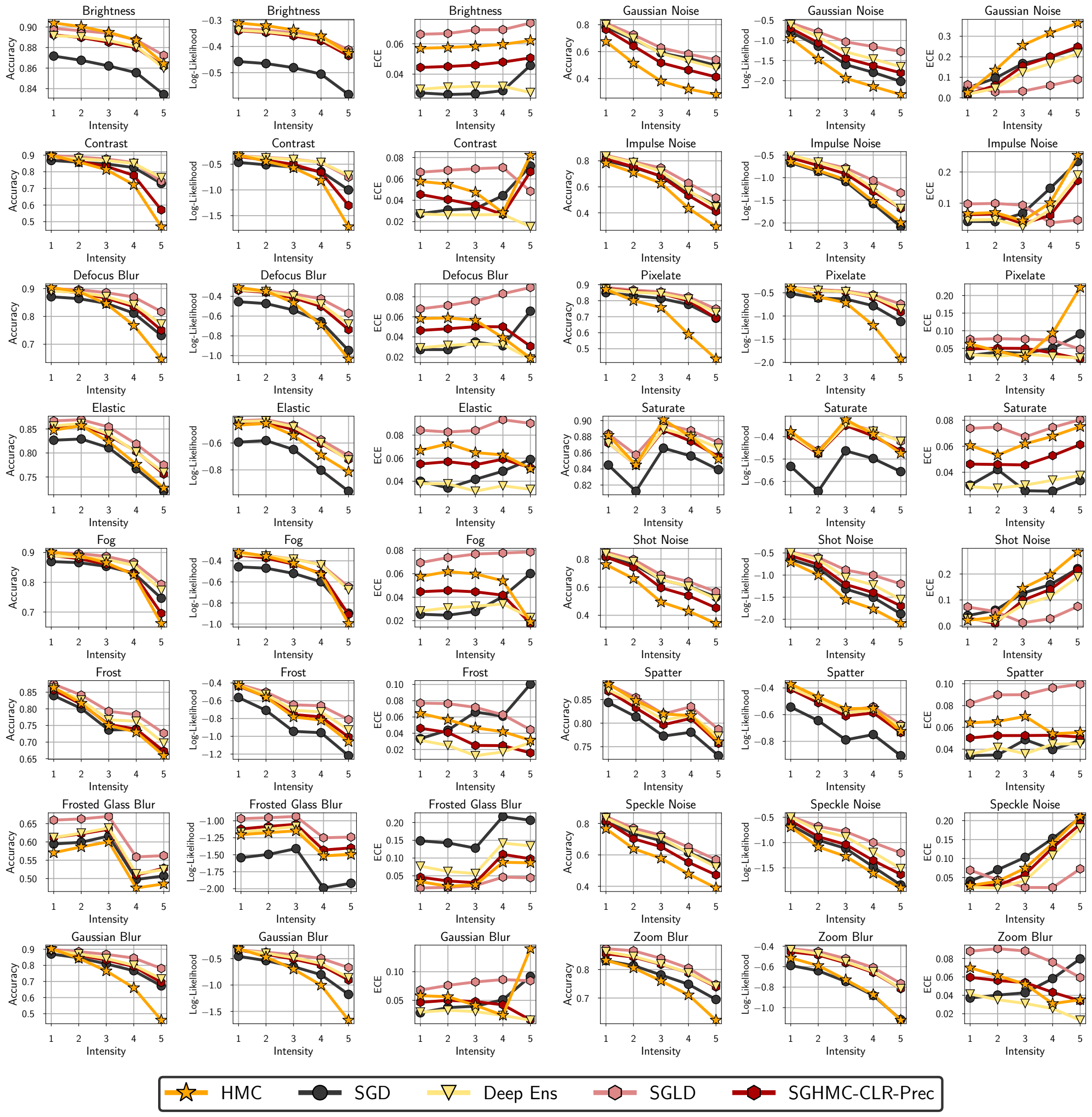


**Figure A.27: Performance under corruption.** We show accuracy, log-likelihood and ECE of HMC, SGD, Deep Ensembles, SGLD and SGHMC-CLR-Prec for all 16 CIFAR-10-C corruptions as a function of corruption intensity. HMC shows poor accuracy on most of the corruptions with a few exceptions. SGLD provides the best robustness on average.

### A.5.6 BNNs are not Robust to Domain Shift

In Section 4.4.2, Figure 4.7 we have seen that surprisingly BNNs via HMC underperform significantly on corrupted data from CIFAR-10-C compared to SGLD, deep ensembles and even MFVI and SGD. We provide detailed results in Figure A.27. HMC shows surprisingly poor robustness in terms of accuracy and log-likelihood across the corruptions. The ECE results are mixed. In most cases, the HMC ensemble of 720 models loses to a single SGD solution!

The poor performance of HMC on OOD data is surprising. Bayesian methods average the predictions over multiple models for the data, and faithfully represent uncertainty. Hence, Bayesian deep learning methods are expected to be robust to noise in the data, and are often explicitly evaluated on CIFAR-10-C [e.g. Wilson and Izmailov 2020; Dusenberry et al. 2020]. Our results suggest that the improvements achieved by Bayesian methods on corrupted data may be a sign of *poor* posterior approximation.

To further understand the robustness results, we reproduce the same effect on a small fully-connected network with two hidden layers of width 256 on MNIST. We run HMC at temperatures $T = 1$ and $T = 10^{-3}$ and SGD and report the results for both the BMA ensembles and individual samples in Figure A.28. For all methods, we train the models on the original MNIST training set, and evaluate on the test set with random Gaussian noise $\mathcal{N}(0, \sigma^2 I)$ of varying scale $\sigma$. We report the test accuracy as a function of $\sigma$. We find that while the performance on the original test set is very close for all methods, the accuracy of HMC at $T = 1$ drops much quicker compared to that of SGD as we increase the noise scale.

Notably, the individual sample performance of $T = 1$ HMC is especially poor compared to SGD. For example, at noise scale $\sigma = 3$ the SGD accuracy is near 60% while the HMC sample only achieves around 20% accuracy!

HMC can be though of as sampling points at a certain sub-optimal level of the training loss, significantly lower than that of SGD solutions. As a result, HMC samples are individually inferior

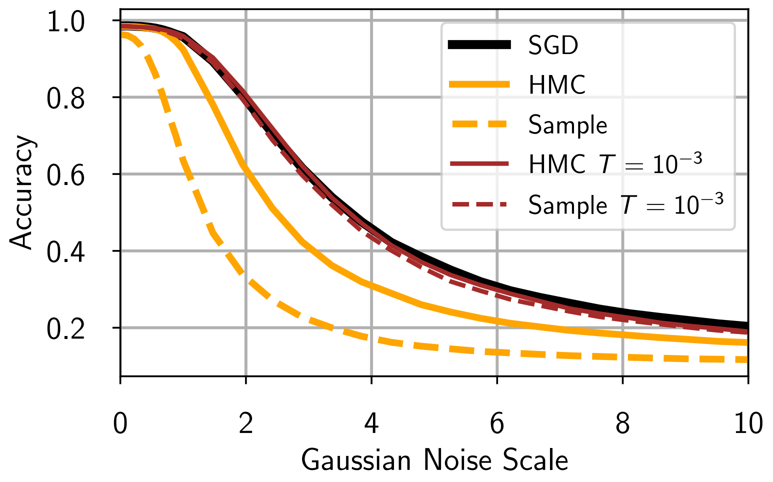


**Figure A.28: Robustness on MNIST.** Performance of SGD, BMA ensembles and individual samples constructed by HMC at temperatures $T = 1$ and $T = 10^{-3}$ on the MNIST test set corrupted by Gaussian noise. We use a fully-connected network. Temperature 1 HMC shows very poor robustness, while lowering the temperature allows us to close the gap to SGD.

to SGD solutions. On the original test data ensembling the HMC samples leads to strong performance significantly outperforming SGD (see Section 4.4). However, as we apply noise to the test data, ensembling can no longer close the gap to the SGD solutions. To provide evidence for this explanation, we run evaluate HMC at a very low temperature $T = 10^{-3}$, as low temperature posteriors concentrate on high-performance solutions similar to the ones found by SGD. We find that at this temperature, HMC performs comparably with SGD, closing the gap in robustness We have also experimented with varying the prior scale but were unable to close the gap in robustness at temperature $T = 1$.

We hypothesize that using a lower temperature with HMC would also significantly improve robustness on CIFAR-10-C. Verifying this hypothesis, and generally understanding the robustness of BNNs further is an exciting direction of future work.

|  | Acc, $T = 1$ | Acc, $T = 0.1$ | CE, $T = 1$ | CE, $T = 0.1$ |
|---|---|---|---|---|
| BN + Aug | 87.46 | 91.12 | 0.376 | 0.2818 |
| FRN + Aug | 85.47 | 89.63 | 0.4337 | 0.317 |
| BN + No Aug | 86.93 | 85.20 | 0.4006 | 0.4793 |
| FRN + No Aug | 84.27 | 80.84 | 0.4708 | 0.5739 |

**Table A.19: Role of data augmentation in the cold posterior effect.** Results of a single chain ensemble constructed with the SGHMC-CLR-Prec sampler of Wenzel et al. [2020] at temperatures $T = 1$ and $T = 0.1$ for different combinations of batch normalization (BN) or filter response normalization (FRN) and data augmentation (Aug). We use the ResNet-20 architecture on CIFAR-10. Regardless of the normalization technique, the cold posteriors effect is present when data augmentation is used, and not present otherwise.

## A.5.7 Further discussion of cold posteriors

In Section 4.8 we have seen that the cold posteriors are not needed to achieve strong performance with BNNs. We have even shown that cold (as well as warm) posteriors may hurt the performance. On the other hand, in Section A.5.6 we have shown that lowering the temperature can improve robustness under the distribution shift, at least for a small MLP on MNIST. Here, we discuss the potential reasons for why the cold posteriors effect was observed in Wenzel et al. [2020].

### A.5.7.1 What causes the difference with Wenzel et al. [2020]?

There are several key differences between the experiments in our study and Wenzel et al. [2020].

First of all, the predictive distributions of SGLD (a version of which was used in Wenzel et al. [2020]) are highly dependent on the hyper-parameters such as the batch size and learning rate, and are inherently biased: SGLD with a non-vanishing step size samples from a perturbed version of the posterior, both because it omits a Metropolis-Hastings accept-reject step and because its updates include minibatch noise. Both of these perturbations should tend to make the entropy of SGLD's stationary distribution increase with its step size; we might expect this to translate to approximations to the BMA that are overdispersed.

Furthermore, Wenzel et al. [2020] show in Figure 6 that with a high batch size they achieve

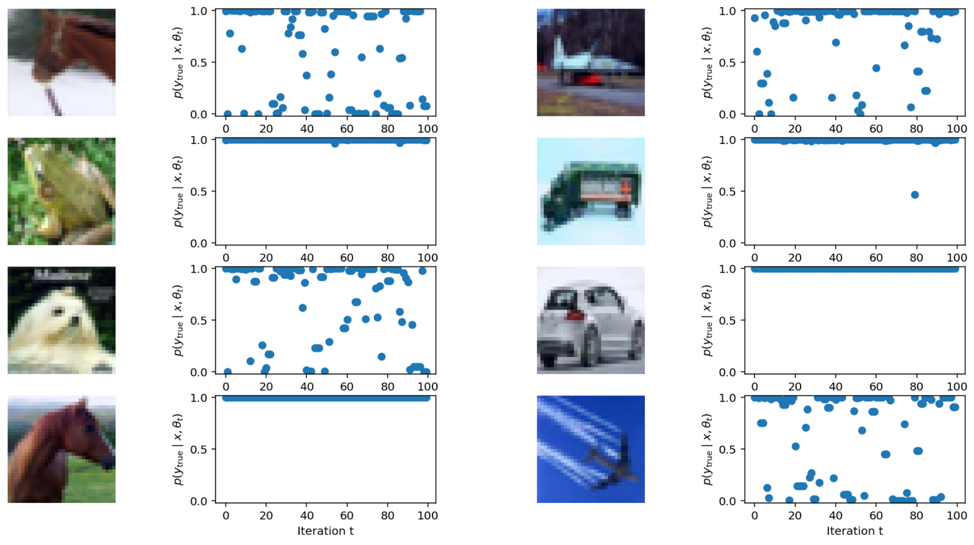


**Figure A.29: HMC samples are (over)confident classifiers.** Plots show the probability assigned by a series of HMC samples to the true label of a held-out CIFAR-10 image. In many cases these probabilities are overconfident (i.e., assign the right answer probability near 0), but there are always *some* samples that assign the true label high probability, so the Bayesian model average is both accurate and well calibrated. These samples were generated with a spherical Gaussian prior with variance $\frac{1}{5}$.

good performance at $T = 1$ for the CNN-LSTM. Using the code provided by the autors[7] with default hyper-parameters we achieved strong performance at $T = 1$ for the CNN-LSTM (accuracy of 0.855 and cross-entropy of 0.35, compared to 0.81 and 0.45 reported in Figure 1 of Wenzel et al. [2020]); we were, however, able to reproduce the cold posteriors effect on CIFAR-10 using the same code.

On CIFAR-10, the main difference between our setup and the configuration in Wenzel et al. [2020] is the use of batch normalization and data augmentation. In the appendix K and Figure 28 of Wenzel et al. [2020], the authors show that if both the data augmentation and batch normalization are turned off, we no longer observe the cold posteriors effect. In Table A.19 we confirm using the code provided by the authors that in fact it is sufficient to turn off just the data augmentation to remove the cold posteriors effect. It is thus likely that the results in Wenzel et al. [2020] are at least partly affected by the use of data augmentation.

[7]https://github.com/google-research/google-research/tree/master/cold_posterior_bnn

### A.5.8 Visualizing prediction variations in HMC samples

In Figure A.29 we visualize the predicted class probability of the true class for 100 HMC samples on eight different input images. While on some images the predicted class probability is always close to 1, on other inputs it is close to 1 for some of the samples (confidently correct), close to 0 for some of the samples (confidently wrong) and in between 0 and 1 for the remaining samples (unconfident). So, some of the samples are individually over-confident, but the ensemble is well-calibrated as other samples assign high probabilities to the correct class.

## A.6 Appendix for "Dangers of Bayesian Model Averaging under Covariate Shift"

This appendix is organized as follows.

- In Section A.6.1, we provide details on hyper-parameters, datasets and architectures used in our experiments.
- In Section A.6.2, we discuss whether the poor generalization of BNNs under covariate shift is surprising.
- In Section A.6.3, we examine BNN performance under covariate shift for a variety of different standard priors with different hyper-parameter settings.
- In Section A.6.4, we study the effect of spurious correlations on BMA performance using the shift-MNIST dataset.
- In Section A.6.5, we show that the same issues that hurt BNN generalization under covariate shift can cause poor performance in low-data regime.

- In Section A.6.6, we explore the convergence of the BNN performance as a function of the number of HMC samples we produce.
- In Section A.6.7, we study how temperature scaling impacts BMA performance under covariate shift.
- In Section A.6.8, we provide proofs of our propositions from Section 4.6.
- In Section A.6.9, we visualize how different corruptions introduce noise along different principal components of the data, and relate this to BMA performance on these corruptions.
- In Section A.6.10, we explain why approximate inference methods SWAG and MC Dropout do not suffer the same performance degradation under covariate shift as HMC.
- In Section A.6.11, we analyze a more general family of priors that includes the *EmpCov* prior from Section 4.7.
- In Section A.6.12, we introduce the sum filter prior for improving BNN robustness to non-zero-mean noise.
- In Section A.6.13, we provide an example of a model architecture where the BMA will be impacted by nonlinear dependencies in the training data.
- In Section A.6.14, we examine how BNNs can be impacted by linear dependencies beyond the first layer using the example of dead neurons.
- In Section A.6.15 we prove that linear dependencies do not hurt BMAs of linear models under covariate shift.
- In Section A.6.16, we examine covariate shift from an optimization perspective.
- Lastly, in subsubsection A.6.16.4, we provide details on licensing.

### A.6.1 Hyper-parameters and details of experiments

#### A.6.1.1 Prior definitions

Here we define the prior families used in the main text and the appendix, and the corresponding hyper-parameters.

**Gaussian priors.** We consider iid Gaussian priors of the form $\mathcal{N}(0, \alpha^2 I)$, where $\alpha^2$ is the prior variance. Gaussian priors are the default choice in Bayesian neural networks [e.g. Fortuin et al. 2021; Izmailov et al. 2021b; Wilson and Izmailov 2020].

**Laplace priors.** We consider priors of the form $\text{Laplace}(\alpha) : \frac{1}{2\alpha}\exp(-\|x\|_1/\alpha)$, where $\|\cdot\|_1$ is the $\ell_1$-norm.

**Student-t priors.** In Section A.6.3, we consider iid Student-t priors of the form $\text{Student-}t(\nu, \alpha^2) : \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\nu\pi}}(1 + \frac{w^2}{\nu\alpha^2})^{-\frac{\nu+1}{2}}$, where $\nu$ represents the degrees of freedom and $\alpha^2$ is the prior variance.

**Exp-norm priors.** In Section A.6.3, we consider the prior family of the form $\text{ExpNorm}(p, \alpha^2) : \exp(-\|w\|^p/2\alpha^2)$. Notice that for $p = 2$, we get the Gaussian prior family. By varying $p$ we can construct more heavy-tailed ($p < 2$) or less heavy-tailed ($p > 2$) priors.

#### A.6.1.2 Hyper-parameters and details

**HMC hyper-parameters.** The hyper-parameters for HMC are the step size, trajectory length and any hyper-parameters of the prior. Following Izmailov et al. [2021b], we set the trajectory length $\tau = \frac{\pi\sigma_{prior}}{2}$ where $\sigma_{prior}$ is the standard deviation of the prior. We choose the step size to ensure that the accept rates are high; for most of our MLP runs we do $10^4$ leapfrog steps per sample, while for CNN we do $5 \cdot 10^3$ leapfrog steps per sample. For each experiment, we run a single HMC chain for 100 iterations discarding the first 10 iterations as burn-in; in Section A.6.6 we show that 100 samples are typically sufficient for convergence of the predictive performance.

**Data Splits.** For all CIFAR-10 and MNIST experiments, we use the standard data splits:

50000 training samples for CIFAR-10, 60000 training samples for MNIST, and 10000 test samples for both. For all data corruption experiments, we evaluate on the corrupted 10000 test samples. For domain shift experiments, we evaluate on 26032 SVHN test samples for MNIST to SVHN and 7200 STL-10 test samples for CIFAR-10 to STL-10. In all cases, we normalize the inputs using train data statistics, and do not use any data augmentation.

**Neural network architectures.** Due to computational constraints, we use smaller neural network architectures for our experiments. All architectures use ReLU activations. For MLP experiments, we use a fully-connected network with 2 hidden layers of 256 neurons each. For CNN experiments, we use a network with 2 convolutional layers followed by 3 fully-connected layers. Both convolutional layers have $5 \times 5$ filters, a stride of 1, and use $2 \times 2$ average pooling with stride 2. The first layer has 6 filters and uses padding, while the second layer has 16 filters and does not use padding. The fully connected layers have 400, 120, and 84 hidden units.

**MAP and deep ensemble hyper-parameters.** We use the SGD optimizer with momentum 0.9, cosine learning rate schedule and weight decay 100 to approximate the MAP solution. In Section A.6.16 we study the effect of using other optimizers and weight decay values. On MNIST, we run SGD for 100 epochs, and on CIFAR we run for 300 epochs. For the deep ensemble baselines, we train 10 MAP models independently and ensemble their predictions.

**Prior hyper-parameters.** To select prior hyper-parameters we perform a grid search, and report results for the optimal hyperparameters in order to compare the best versions of different models and priors. We report the prior hyper-parameters used in our main evaluation in Table A.20. In Section A.6.3 we provide detailed results for various priors with different hyper-parameter choices.

**Tempering hyper-parameters.** For the tempering experiments, we use a Gaussian prior with variance $\alpha^2 = \frac{1}{100}$ on MNIST and $\alpha^2 = \frac{1}{3}$ on CIFAR-10. We set the posterior temperature to $10^{-2}$. We provide additional results for other prior variance and temperature combinations in Section A.6.7.

**Table A.20:** Prior hyper-parameters

| Hyper-parameter | MNIST MLP | MNIST CNN | CIFAR-10 CNN |
|---|---|---|---|
| BNN, Gaussian prior; $\alpha^2$ | $\frac{1}{100}$ | $\frac{1}{100}$ | $\frac{1}{100}$ |
| BNN, Laplace prior; $\alpha$ | $\sqrt{\frac{1}{6}}$ | $\sqrt{\frac{1}{6}}$ | $\sqrt{\frac{1}{200}}$ |

**Compute.** We ran all the MNIST experiments on TPU-V3-8 devices, and all CIFAR experiments on 8 NVIDIA Tesla-V100 devices. A single HMC chain with 100 iterations on these devices takes roughly 1.5 hours for MNIST MLP, 2 hours for CIFAR CNN and 3 hours for MNIST CNN. As a rough upper-bound, we ran on the order of 100 different HMC chains, each taking 2 hours on average, resulting in 200 hours on our devices, or roughly 1600 GPU-hours (where we equate 1 hour on TPU-V3-8 to 8 GPU-hours).

### A.6.2 Should we find the lack of BMA robustness surprising?

Bayesian neural networks are sometimes presented as a way of improving *just* the uncertainties, often at the cost of degradation in accuracy. Consequently, one might assume that the poor performance of BNNs under covariate shift is not surprising, and we should use BNN uncertainty estimates solely to *detect* OOD, without attempting to make predictions, even for images that are still clearly recognizable.

In recent years, however, Bayesian deep learning methods [e.g., Maddox et al. 2019; Dusenberry et al. 2020; Daxberger et al. 2020], as well as high-fidelity approximate inference with HMC [Izmailov et al. 2021b], achieve improved uncertainty *and* accuracy compared to standard MAP training with SGD. In this light, we believe there are many reasons to find the significant performance degradation under shift surprising:

- The BNNs are often providing significantly better accuracy on in-distribution points. For example, HMC BNNs achieve a 5% improvement over MAP on CIFAR-10, but 25% worse accuracy on the pixelate corruption, when the images are still clearly recognizable (see Figure 4.8). To go from clearly better to profoundly worse would not typically be expected of any method on these shifts.

- In fact, recent work [e.g. Miller et al. 2021] shows that there is typically a strong correlation between in-distribution and OOD generalization accuracy on related tasks, which is the opposite of what we observe in this work.
- Many approximate Bayesian inference procedures do improve accuracy over MAP on shift problems [Ovadia et al. 2019; Wilson and Izmailov 2020; Dusenberry et al. 2020], and newer inference procedures appear to be further improving on these results. For example, MultiSWAG [Wilson and Izmailov 2020] is significantly more accurate than MAP under shift. The fact that these methods are more Bayesian than MAP, and improve upon MAP in these settings, makes it particularly surprising that a high-fidelity BMA would be so much worse than MAP. This is a nuanced point — how is it that methods getting closer in some ways to the Bayesian ideal are improving on shift, when a still higher-fidelity representation of the Bayesian ideal is poor on shift? — we discuss this point in Section A.6.10.
- Recent results highlight that there need not be a tension between OOD detection and OOD generalization accuracy: indeed deep ensembles provide much better performance than MAP on both [Ovadia et al. 2019].
- Bayesian methods are closely associated with trying to provide a good representation of uncertainty, and a good representation of uncertainty should not say "I have little idea" when a point is only slightly out of distribution, but still clearly recognizable, e.g., through noise corruption or mild domain shift.
- In Figure A.30 we report the log-likelihood and ECE metrics which evaluate the quality of uncertainty estimates for deep ensembles, MAP and BNNs. The log-likelihood and ECE of

standard BNNs are better than the corresponding values for the MAP solution on average, but they are much worse than the corresponding numbers for deep ensembles for high degrees of corruption. Furthermore, for some corruptions (*impulse noise*, *pixelate*) BNNs lose to MAP on both log-likelihood and ECE at corruption intensity 5. Also for larger ResNet-20 architecture on CIFAR-10-C, Izmailov et al. [2021b] reported that the log-likelihoods of BNNs are on average slightly worse than for MAP solution at corruption intensity 5.

## A.6.3 Additional results on BNN robustness

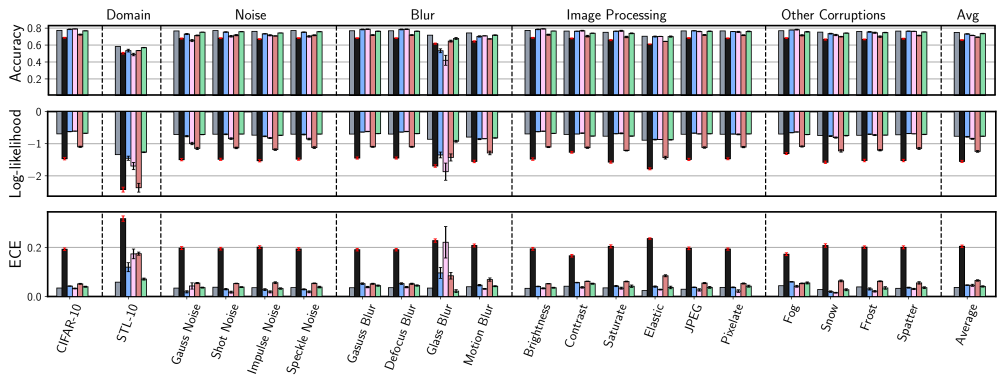
### A.6.3.1 Error-bars and additional metrics

We report the accuracy, log-likelihood and expected calibration error (ECE) for deep ensembles, MAP solutions and BMA variations in Figure A.30. We report the results for different corruption intensities (1, 3, 5) and provide error-bars computed over 3 independent runs. Across the board, *EmpCov* priors provide the best performance among BNN variations on all three metrics.

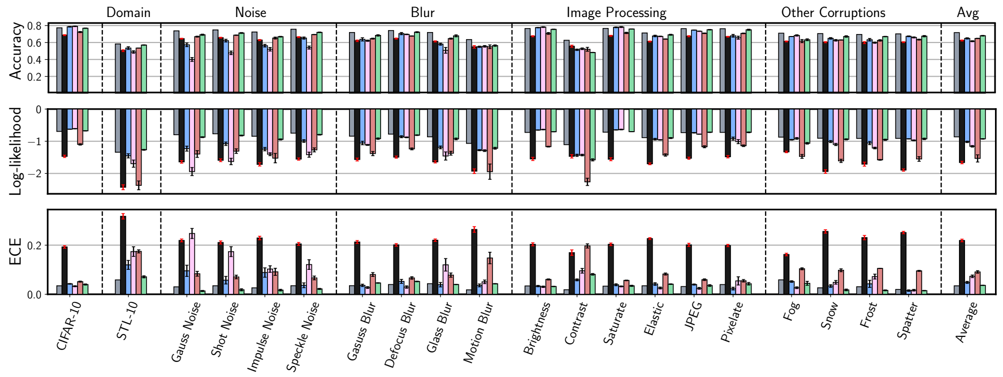
### A.6.3.2 Detailed results for different priors

In this section, we evaluate BNNs with several prior families and provide results for different choices of hyper-parameters. The priors are defined in subsubsection A.6.1.1.

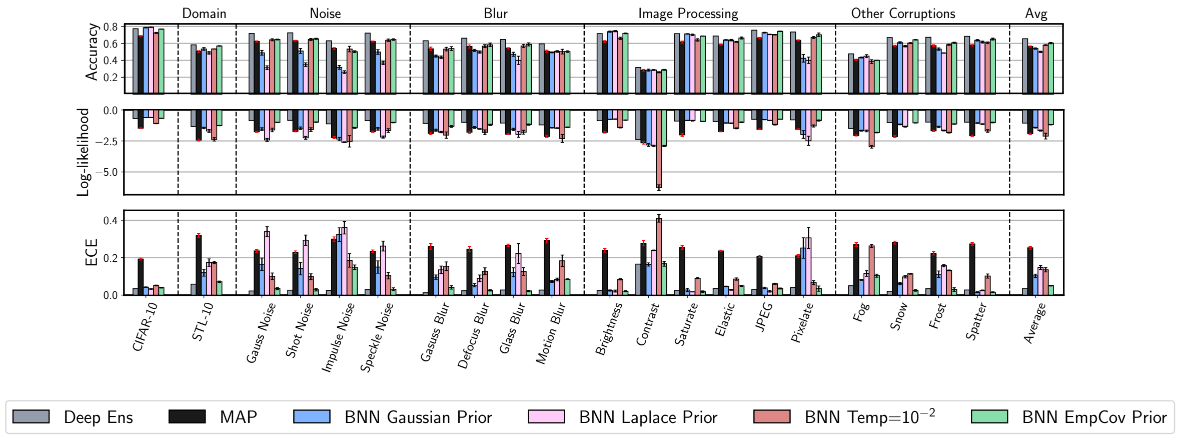
We report the results using CNNs and MLPs on MNIST in Figure A.31. None of the considered priors completely close the gap to MAP under all corruptions. Gaussian priors show the worst results, losing to MAP on all MNIST-C corruptions and Gaussian noise, at all prior standard deviations. Laplace priors show similar results to Gaussian priors under Gaussian noise, but beat MAP on the *stripe* corruption in MNIST-C. Student-*t* priors show better results, matching or outperforming MAP on all affine transformations, but still underpeform significantly under *Gaussian noise*, *Brightness* and *Fog* corruptions. Finally, exp-norm priors can match MAP on Shot noise and also outperform MAP on *stripe*, but lose on other corruptions. The in-domain



**(a)** Corruption Intensity 1

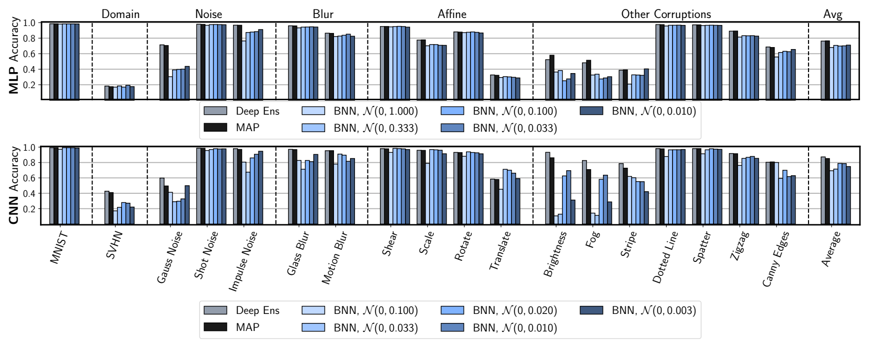


**(b)** Corruption Intensity 3

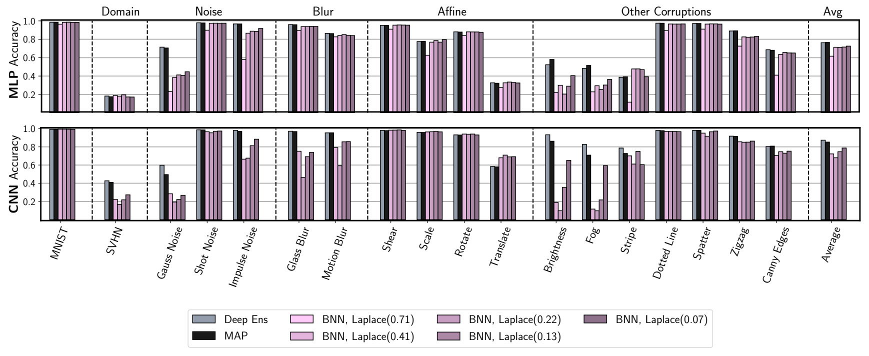


**(c)** Corruption Intensity 5

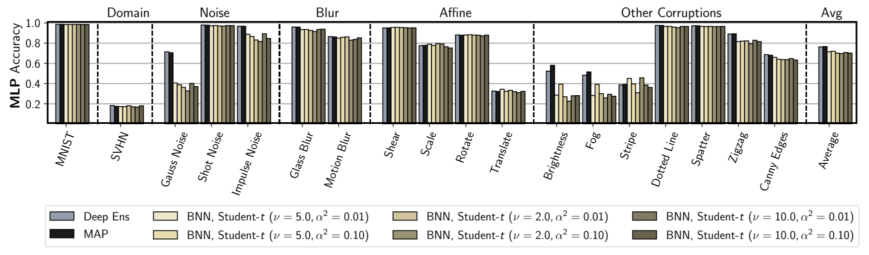
**Figure A.30: Detailed results on CIFAR-10.** Accuracy, log-likelihood and log-likelihood for deep ensembles, MAP solution, and BNN variants under covariate shift on CIFAR-10. We report the performance at corruption intensity levels 1, 3 and 5 (corruption intensity does not affect the CIFAR-10 and STL-10 columns in the plots). For all methods except deep ensembles we report the mean and standard deviation (via error-bars) over 3 random seeds. *EmpCov* priors provide the only BNN variation that consistently performs on par with deep ensembles in terms of log-likelihood and ECE. Tempered posteriors improve the accuracy on some of the corruptions, but significantly hurt in-domain performance.



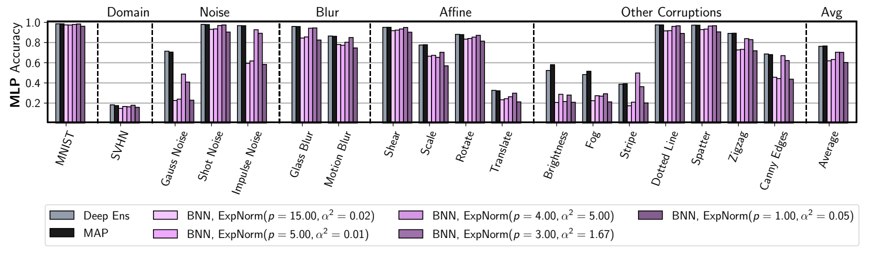
**(a)** Gaussian priors



**(b)** Laplace priors



**(c)** Student-$t$ priors



**(d)** ExpNorm priors

**Figure A.31: Priors on MNIST.** We report the performance of different prior families under covariate shift on MNIST. For Gaussian and Laplace prior families, we report the results using both the MLP and CNN architectures; for Student-$t$ and ExpNorm we only report the results for MLP. None of the priors can match the MAP performance across the board, with particularly poor results under *Gaussian noise*, *Brightness* and *Fog* corruptions.

performance with exp-norm priors is also lower compared to the other priors considered.

To sum up, none of the priors considered is able to resolve the poor robustness of BNNs under covariate shift. In particular, all priors provide poor performance under *Gaussian noise*, *Brightness* and *Fog* corruptions.

### A.6.4 Bayesian neural networks and spurious correlations

For corrupted data, models experience worse performance due to additional noisy features being introduced. However, it's also possible that the reverse can occur, and a seemingly highly predictive feature in the training data will not be present in the test data. This distinct category of covariate shift is often called spurious correlation. To test performance with spurious correlations, we use the Shift-MNIST dataset [Jacobsen et al. 2018], where we introduce spurious features via modifying the training data so that a set of ten pixels in the image perfectly correlates with class labels.

**Table A.21: Spurious correlations.** Accuracy and log-likelihood of MAP, deep ensembles and BNNs with Gaussian and *EmpCov* priors on the Shift-MNIST dataset.

| Model | MLP Accuracy | MLP LL | CNN Accuracy | CNN LL |
|---|---|---|---|---|
| MAP | 88.70% | -0.527 | 48.63% | -2.206 |
| Deep Ensemble | 88.73% | -0.527 | 72.99% | -1.041 |
| BNN Gaussian Prior | 90.83% | -0.598 | 64.27% | -1.326 |
| BNN EmpCov Prior | 86.95% | -1.146 | 64.41% | -1.450 |

Table A.21 shows the results for deep ensembles, MAP and BNNs with Gaussian and *EmpCov* priors on the Shift-MNIST dataset. We see worse accuracy for CNN architectures, demonstrating how more complex architectures can more easily over-fit the spurious correlations. BNNs with Gaussian prior perform better than MAP on both MLP and CNN, but significantly worse than deep ensembles for CNNs. Notice that the *EmpCov* prior does not improve performance here for either architecture, highlighting the difference between spurious correlations and other forms of covariate shift. In particular, the largest principal components of the Shift-MNIST training dataset place large magnitude weights on the spurious features, and so using the *EmpCov* prior results in samples with larger weights for the activated (spurious) pixels. When those same pixels are not activated in the test set, such samples will have a larger shift in their predictions.

An in-depth analysis of BNNs in the presence of spurious correlations remains an exciting direction for further research.

### A.6.5 Bayesian neural networks in low-data regime

The intuition presented in Propositions 4.3, 4.4 suggests that Bayesian neural networks may also underperform in low-data regime. Indeed, if the model only observes a small number of datapoints, some of the directions in the parameter space will not be sufficiently constrained by the data. Empirically, in Table A.22 found that the performance of BNNs is indeed inferior to MAP when the training dataset is very small, but the results become more similar as the size of the dataset increases.

**Table A.22: Spurious correlations.** Accuracy of MAP and HMC BNNs using the MLP architecture on MNIST in low-data regime. When the dataset is very small, MAP significantly outperforms the BNN.

| | 50 datapoints | 100 datapoints | 1000 datapoints |
|---|---|---|---|
| MAP | 66.4% | 74.3% | 90.2% |
| HMC BNN | 53.4% | 65.4% | 90.3% |

We believe that the reason why we do not observe the poor generalization of the Bayesian models in the 1000 datapoints regime is that the low-variance directions are fairly consistent across the dataset. However, in extreme low-data cases, we cannot reliably estimate the low-variance directions leading to poor performance according to Propositions 4.3, 4.4. A detailed exploration of BNN performance in low-data regime is an exciting direction of future work.

### A.6.6 Convergence of HMC accuracy with samples

In our experiments, we use 90 HMC samples from the posterior to evaluate the performance of BNNs. In this section, we verify that the Monte Carlo estimates of accuracy of the Bayesian model average converge very quickly with the number of samples, and 90 samples are sufficient for performing qualitative comparison of the methods. In Table A.23 we show the accuracy for a fully-connected HMC BNN with a Gaussian prior on MNIST under different corruptions as a function of the number of samples:

**Table A.23: Spurious correlations.** Accuracy of MAP and HMC BNNs using the MLP architecture on MNIST in low-data regime. When the dataset is very small, MAP significantly outperforms the BNN.

| corruption | 10 samples | 50 samples | 100 samples | 500 samples | 1200 samples |
|---|---|---|---|---|---|
| MNIST | 98.2% | 98.19% | 98.19% | 98.32% | 98.26% |
| Impulse Noise | 85.34% | 89.86% | 90.68% | 91.3% | 91.33% |
| Motion Blur | 81.56% | 81.82% | 82.14% | 82.47% | 82.61% |
| Scale | 67.32% | 68.69% | 69.45% | 69.91% | 70.18% |
| Brightness | 23.66% | 20.26% | 22.31% | 24.08% | 23.4% |
| Stripe | 28.18% | 30.09% | 34.8% | 39.26% | 37.96% |
| Canny Edges | 58.79% | 62.85% | 63.34% | 64.36% | 64.32% |

In each case, the performance estimated from 100 samples is very similar to the performance for 1200 samples. The slowest convergence is observed on the stripe corruption, but even there the performance at 100 samples is very predictive of the performance at 1200 samples.

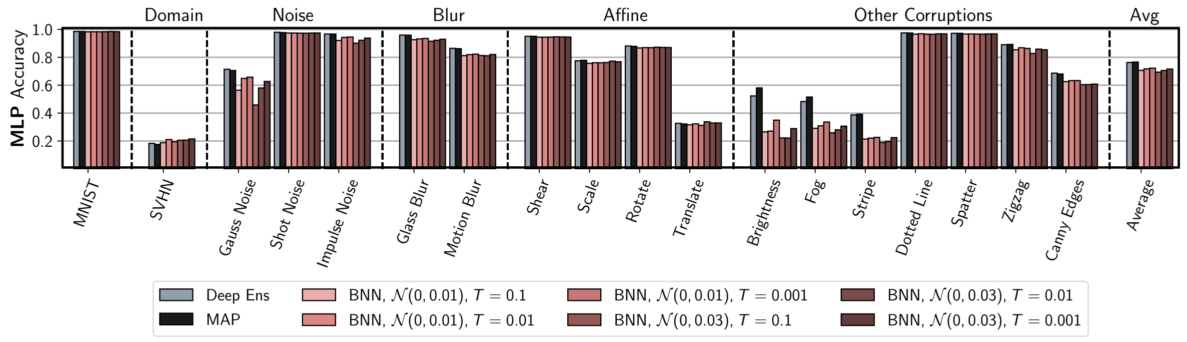


**Figure A.32: Temperature ablation.** We report the performance of BNNs with Gaussian priors and tempered posteriors for different temperatures and prior scales. Low temperatures ($T = 10^{-2}, 10^{-3}$) can provide a significant improvement on the *noise* corruptions, but do not improve the results significantly under other corruptions.

### A.6.7 Tempered posteriors

In this section we explore the effect of posterior tempering on the performance of the MLP on MNIST. In particular, following Wenzel et al. [2020] we consider the cold posteriors:

$$p_T(W|D) \propto (p(D|W)p(W))^{1/T}, \tag{A.17}$$

where $T \leqslant 1$. In Figure A.32 we report the results for BNNs with Gaussian priors with variances 0.01 and 0.03 and posterior temperatures $T \in \{10^{-1}, 10^{-2}, 10^{-3}\}$. As observed by Izmailov et al. [2021b], lower temperatures ($10^{-2}, 10^{-3}$) improve performance under the *Gaussian noise* corruption; however, low temperatures do not help with other corruptions significantly.

### A.6.8 Proofs of the theoretical results

For convenience, in this section we assume that a constant value of 1 is appended to the input features instead of explicitly modeling a bias vector $b$. We assume that the output $f(x, W)$ of the network with parameters $W$ on an input $x$ is given by

$$f(x, W) = \psi(\phi(\ldots\phi(\phi(xW^1)W^2 + b^2))W^l + b^l), \tag{A.18}$$

where $\phi$ are non-linearities of the intermediate layers (e.g. ReLU) and $\psi$ is the final link function (e.g. softmax).

We will also assume that the likelihood is a function $\ell(\cdot,\cdot)$ that only depends on the output of the network and the target label:

$$p(y|x,W) = \ell(y, f(x,W)). \tag{A.19}$$

For example, in classification $\ell(y, f(x,W)) = f(x,W)[y]$, the component of the output of the softmax layer corresponding to the class label $y$. Finally, we assume that the likelihood factorizes over the inputs:

$$p(\mathcal{D}|W) = \prod_{x,y\in\mathcal{D}} p(y|x,W) \tag{A.20}$$

for any collection of datapoints $\mathcal{D}$.

#### A.6.8.1 Proof of Lemma 4.1

We restate the Lemma:

**Lemma A.3.** *Suppose that the input feature $x_k^i$ is equal to zero for all the examples $x_k$ in the training dataset $D$. Suppose the prior distribution over the parameters $p(W)$ factorizes as $p(W) = p(w_{ij}^1)\cdot p(W\setminus w_{ij}^1)$ for some neuron $j$ in the first layer, where $W\setminus w_{ij}^1$ represents all the parameters $W$ of the network except $w_{ij}^1$. Then, the posterior distribution $p(W|D)$ will also factorize and the marginal posterior over the parameter $w_{ij}^1$ will coincide with the prior:*

$$p(W|D) = p(W\setminus w_{ij}^1|D)\cdot p(w_{ij}^1). \tag{A.21}$$

*Consequently, the MAP solution will set the weight $w_{ij}^1$ to the value with maximum prior density.*

**Proof.** Let us denote the input vector $x$ without the input feature $i$ by $x^{-i}$, and the matrix

$W^1$ without the row $i$ by $W^1_{-i}$. We can rewrite Equation A.18 as follows:

$$f(x, W) = \psi(\phi(\dots\phi(\phi(x^{-i}W^1_{-i} + \underbrace{x^i W^1_i}_{=0})W^2 + b^2))W^l + b^l). \tag{A.22}$$

As for all the training inputs $x_k$ the feature $x^i_k$ is equal to 0, the vector $x^i W^1_i$ is equal to zero and can be dropped:

$$f(x_k, W) = \psi(\phi(\dots\phi(\phi(x^{-i}_k W^1_{-i})W^2 + b^2))W^l + b^l)) =: f'(x_k, W_{-i}), \tag{A.23}$$

where $W_{-i}$ denotes the vector of parameters $W$ without $W^1_i$, and we defined a new function $f'$ that does not depend on $W_i$ and is equivalent to $f$ on the training data. Consequently, according to Equation A.19 and Equation A.20, we can write

$$p(D|W) = \prod_{k=1}^{n} \ell(y_k, f(x_k, W)) = \prod_{k=1}^{n} \ell(y_k, f'(x_k, W_{-i})). \tag{A.24}$$

In other words, the likelihood does not depend on $W^1_i$ and in particular $w^1_{ij}$ for any $j$.

Let us write down the posterior over the parameters using the factorization of the prior:

$$p(W|D) = \frac{\overbrace{p(D|W \setminus w^1_{ij})p(W \setminus w^1_{ij})}^{\text{does not depend on } w^1_{ij}}\, p(w^1_{ij})}{Z}, \tag{A.25}$$

where $Z$ is a normalizing constant that does not depend on $W$. Hence, the posterior factorizes as a product of two distributions: $p(D|W \setminus w^1_{ij})p(W \setminus w^1_{ij})/Z$ over $W \setminus w^1_{ij}$ and $p(w^1_{ij})$. The marginal posterior over $w^1_{ij}$ thus coincides with the prior and is independent of the other parameters.

Maximizing the factorized posterior Equation A.25 to find the MAP solution, we set the $w^1_{ij}$ to the maximum of its marginal posterior, as it is independent of the other parameters. ■

### A.6.8.2 Formal statement and proof of Proposition 4.1

First, let us prove the following result for the MAP solution.

**Proposition A.4.** *Consider the following assumptions:*

*(a) The input feature $x_k^i$ is equal to zero for all the examples $x_k$ in the training dataset $D$.*

*(b) The prior over the parameters factorizes as $p(W) = p(W_{-i}) \cdot p(W_i^1)$, where $W_{-i}$ is the vector of all parameters except for $W_i^1$, the row $i$ of the weight matrix $W^1$ of the first layer.*

*(c) The prior distribution $p(W_i^1)$ has maximum density at $0$.*

*Consider an input $x(c) = [x^1, \dots, x^{i-1}, c, x^{i+1}, \dots, x^m]$. Then, the prediction with the MAP model $W_{MAP}$ does not depend on $c$: $f(x(c), W_{MAP}) = f(x(0), W_{MAP})$.*

**Proof.** Analogous to the proof of Lemma A.3, we can show that under the assumptions (a), (b) the posterior over the parameters factorizes as

$$p(W|D) = p(W^{-i}|D)p(W_i^1). \tag{A.26}$$

Then, the MAP solution will set the weights $W_i^1$ to the point of maximum density, which is $0$ under assumption (c). Consequently, based on Equation A.22, we can see that the output of the MAP model will not depend on $x^i = c$. ■

Next, we provide results for the Bayesian model average. We define *positive-homogeneous* activations as functions $\phi$ that satisfy $\phi(c \cdot x) = c \cdot \phi(x)$ for any positive scalar $c$ and any $x$. For example, ReLU and Leaky ReLU activations are positive-homogeneous.

We will call a vector $z$ of class logits (inputs to softmax) *$\epsilon$-separable* if the largest component $z_i$ is larger than all the other components by at least $\epsilon$:

$$z_i - z_j > \epsilon \quad \forall j \neq i. \tag{A.27}$$

We can prove the following general proposition.

**Proposition A.5.** *We will need the following assumptions:*

*(d) The support of the prior over the parameters $W_{-i}$ is bounded: $\|W_{-i}\| < B$.*

*(e) The activations $\phi$ are positive-homogeneous and have a Lipschitz constant bounded by $L_\phi$.*

*Consider an input $x(c) = [x^1, \ldots, x^{i-1}, c, x^{i+1}, \ldots, x^m]$. Then, we can prove the following conclusions*

*(2) Suppose the link function $\psi$ is identity. Suppose also that the expectation $\mathbb{E}[\phi(\ldots\phi(\phi(W_i^1)W^2)\ldots)W^l]$ over $W$ sampled from the posterior is non-zero. Then the predictive mean under BMA (see Equation 4.1) on the input $x(c)$ depends on $c$.*

*(3) Suppose the link function $\psi$ is softmax. Then, for sufficiently large $c > 0$ the predicted class $\hat{y}(c) = \arg\max_y f(x(c), W)[y]$ does not depend on $x(c)$ for any sample $W$ from the posterior such that $z = \phi(\ldots\phi(\phi(W_i^1)W^2)\ldots)W^l$ is $\epsilon$-separable.*

**Proof.** We can rewrite Equation A.18 as follows:

$$\begin{aligned} f(x(c), W) = \psi(\phi(\ldots\phi(\phi(x^{-i}W_{-i}^1 + cW_i^1)W^2 + b^2))W^l + b^l) = \\ \psi(\phi(\ldots\phi(c \cdot \phi([x^{-i}W_{-i}^1]/c + W_i^1)W^2 + b^2))W^l + b^l) = \\ \psi(c \cdot (\phi(\ldots\phi(\phi([x^{-i}W_{-i}^1]/c + W_i^1)W^2 + b^2/c))W^l + b^l/c)). \end{aligned} \tag{A.28}$$

Now, under our assumptions the prior and hence the posterior over the weights $W_{-i}$ is bounded. As in finite-dimensional Euclidean spaces all norms are equivalent, in particular we imply that (1) the spectral norms $\|W^t\|_2 < L_W$ are bounded for all layers $t = 2, \ldots, l$ by a constant $L_W$, and (2) the Frobenious norms $\|\cdot\|$ of the bias parameters $b_t$ and the weights $W_{-i}^1$ are all bounded by a constant $B$. We will also assume that the norm of the vector $x^{-i}$ is bounded by the same constant: $\|x^{-i}\| \leqslant B$.

Consider the difference

$$\left\| \left( \phi \left( \dots \phi \left( \phi \left( \frac{x^{-i} W_{-i}^1}{c} + W_i^1 \right) W^2 + \frac{b^2}{c} \right) \right) W^l + \frac{b^l}{c} \right) - \right.$$
$$\left. \phi \left( \dots \phi \left( \phi \left( \frac{x^{-i} W_{-i}^1}{c} + W_i^1 \right) W^2 + \frac{b^2}{c} \right) \right) W^l \right\| \leqslant \frac{B}{c}. \tag{A.29}$$

Indeed, by the $\|b^l\| \leqslant B$. Next, for an arbitrary $z$ we can bound

$$\left\| \phi \left( z + \frac{b^{l-1}}{c} \right) W^l - \phi(z) W^l \right\| \leqslant L_W \cdot L_\phi \cdot \frac{B}{c}, \tag{A.30}$$

where we used the fact that $\phi$ is Lipschitz with $L_\phi$ and the Lipschitz constant for matrix multiplication by $W^l$ coincides with the spectral norm of $W^l$ which is bounded by $L_W$.

Using the bound in Equation A.30, we have

$$\left\| \left( \phi \left( \phi \left( \dots \phi \left( \phi \left( \frac{x^{-i} W_{-i}^1}{c} + W_i^1 \right) W^2 + \frac{b^2}{c} \right) W^{l-1} + \frac{b^{l-1}}{c} \right) \dots \right) W^l + \frac{b^l}{c} \right) - \right.$$
$$\left. \phi \left( \phi \left( \dots \phi \left( \phi \left( \frac{x^{-i} W_{-i}^1}{c} + W_i^1 \right) W^2 + \frac{b^2}{c} \right) \dots \right) W^{l-1} \right) W^l \right\| \leqslant \frac{B}{c} + L_W \cdot L_\phi \cdot \frac{B}{c}. \tag{A.31}$$

Applying the same argument to all layers of the network (including the first layer where $\frac{x^{-i} W_{-i}^1}{c}$ plays the role analogous to $\frac{b^{l-1}}{c}$ in Equation A.30), we get

$$\left\| \left( \phi \left( \dots \phi \left( \phi \left( \frac{x^{-i} W_{-i}^1}{c} + W_i^1 \right) W^2 + \frac{b^2}{c} \right) \dots \right) W^l + \frac{b^l}{c} \right) \right.$$
$$\left. - \phi \left( \dots \phi \left( \phi \left( W_i^1 \right) W^2 \right) \dots \right) W^l \right\|$$
$$\leqslant \frac{B}{c} (1 + L_W \cdot L_\phi + L_W^2 \cdot L_\phi^2 + \dots + L_W^{l-1} \cdot L_\phi^{l-1}). \tag{A.32}$$

Choosing $c$ to be sufficiently large, we can make the bound in Equation A.32 arbitrarily tight.

**Conclusion (2)** Suppose $\psi$ is the identity. Then, we can write

$$f(x(c), W) = c \cdot \phi\left(\ldots \phi\left(\phi\left(W_i^1\right) W^2\right) \ldots\right) W^l + \Delta, \tag{A.33}$$

where $\Delta$ is bounded: $\|\Delta\| \leqslant B(1 + L_W \cdot L_\phi + L_W^2 \cdot L_\phi^2 + \ldots + L_W^{l-1} \cdot L_\phi^{l-1})$. Consider the predictive mean under BMA,

$$\mathbb{E}_W f(x(c), W) = c \cdot \underbrace{\mathbb{E}_W \phi\left(\ldots \phi\left(\phi\left(W_i^1\right) W^2\right) \ldots\right) W^l}_{\neq 0} + \underbrace{\mathbb{E}_W \Delta}_{\text{Bounded}}, \tag{A.34}$$

where the first term is linear in $c$ and the second term is bounded uniformly for all $c$. Finally, we assumed that the expectation $\mathbb{E}_W \phi\left(\ldots \phi\left(\phi\left(W_i^1\right) W^2\right) \ldots\right) W^l \neq 0$, so for large values of $c$ the first term in Equation A.34 will dominate, so the output depends on $c$.

**Conclusion (3)** Now, consider the softmax link function $\psi$. Note that for the softmax we have $\arg\max_y \psi(c \cdot z)[y] = \arg\max_y z[y]$. In other words, multiplying the logits (inputs to the softmax) by a positive constant $c$ does not change the predicted class. So, we have

$$\hat{y}(c, W) = \arg\max_y f(x(c), W)[y] = \\ \arg\max_y \left(\phi\left(\ldots \phi\left(\phi\left(\frac{x^{-i} W_{-i}^1}{c} + W_i^1\right) W^2 + \frac{b^2}{c}\right)\right) W^l + \frac{b^l}{c}\right)[y]. \tag{A.35}$$

Notice that $z_W = \phi(\ldots \phi(\phi(W_i^1) W^2) \ldots) W^l$ does not depend on the input $x(c)$ in any way. Furthermore, if $z_W$ is $\epsilon$-separable, with class $y_W$ corresponding to the largest component of $z_W$, then by taking

$$c > \frac{B(1 + L_W \cdot L_\phi + L_W^2 \cdot L_\phi^2 + \ldots + L_W^{l-1} \cdot L_\phi^{l-1})}{\epsilon}, \tag{A.36}$$

we can guarantee that the predicted class for $f(x(c), W)$ will be $y_W$ according to Equation A.32.

■

#### A.6.8.3 General linear dependencies, Proposition 4.3

We will prove the following proposition, reducing the case of general linear dependencies to the case when an input feature is constant.

Suppose that the prior over the weights $W^1$ in the first layer is an i.i.d. Gaussian distribution $\mathcal{N}(0, \alpha^2)$, independent of the other parameters in the model. Suppose all the inputs $x_1 \dots x_n$ in the training dataset $D$ lie in a subspace of the input space: $x_i^T c = 0$ for all $i = 1, \dots, n$ and some constant vector $c$ such that $\sum_{i=1}^m c_i^2 = 1$.

Let us introduce a new basis $v_1, \dots, v_m$ in the input space, such that the vector $c$ is the first basis vector. We can do so e.g. by starting with the collection of vectors $\{c, e_2, \dots, e_m\}$, where $e_i$ are the standard basis vectors in the feature space, and using the Gram–Schmidt process to orthogonalize the vectors. We will use $V$ to denote the matrix with vectors $v_1, \dots, v_m$ as colunms. Due to orthogonality, we have $VV^T = I$.

We can rewrite our model from Equation A.18 as

$$f(x, W) = \psi(\phi(\dots\phi(\phi(\underbrace{xV}_{\bar{x}}\ \underbrace{V^T W^1}_{\bar{W}^1})W^2 + b^2))W^l + b^l). \tag{A.37}$$

We can thus re-parameterize the first layer of the model by using transformed inputs $\bar{x} = xV$, and transformed weights $\bar{W}^1 = V^T W^1$. Notice that this re-parameterized model is equivalent to the original model, and doing inference in the re-parameterized model is equivalent to doing inference in the original model.

The induced prior over the weights $\bar{W}^1$ is $\mathcal{N}(0, \alpha^2 I)$, as we simply rotated the basis. Furthermore, the input $\bar{x}_k^1 = x_k^T v_1 = 0$ for all training inputs $k$. Thus, with the re-parameterized model we are in the setting of Lemma 4.1 and Propositions A.4, A.5.

In particular, the posterior over the parameters $\bar{W}_1^1 = v_1^T W^1$ will coincide with the prior $\mathcal{N}(0, \alpha^2 I)$ (Lemma 4.1). The MAP solution will ignore the feature combination $\bar{x}^1 = x^T v_1$, while the BMA predictions will depend on it (Propositions A.4, A.5).

#### A.6.8.4 Convolutional layers, Proposition 4.4

Suppose that the convolutional filters in the first layer are of size $K \times K \times C$, where $C$ is the number of input channels. Let us consider the set $\hat{D}$ of size $N$ of all the patches of size $K \times K \times C$ extracted from the training images in $D$ after applying the same padding as in the first convolutional layer. Let us also denote the set of patches extracted from a fixed input image by $D_x$.

A convolutional layer applied to $x$ can be thought of as a fully-connected layer applied to all patches in $D_x$ individually, and with results concatenated:

$$conv(w, x) = \left\{ \left( i, j, \sum_{a=i}^{i+k} \sum_{b=j}^{j+k} \sum_{c=1}^{C} x_{a,b,c} \cdot W^1_{a,b,c} \right) \right\}, \tag{A.38}$$

where $x_{a,b,c}$ is the intensity of the image at location $(a, b)$ in channel $c$, $W^1_{a,b,c}$ is the corresponding weight in the convolutional filter, and the tuples $(i, j, v)$ for all $i, j$ represent the intensities at location $(i, j)$ in the output image.

In complete analogy with Lemma 4.1 and Propositions A.4, A.5, we can show that if all the patches in the dataset $\hat{D}$ are linearly dependent, then we can re-parameterize the convolutional layer so that one of the convolutional weights will always be multiplied by 0 and will not affect the likelihood of the data. The MAP solution will set this weight to zero, while the BMA will sample this weight from the prior, and it will affect predictions.

### A.6.9 How corruptions break linear dependence in the data

In Figure A.33, we visualize the projections of the original and corrupted MNIST data on the PCA components extracted from the MNIST train set and the set of all $5 \times 5$ patches of the MNIST train set. As we have seen in Section 4.6, the former are important for the MLP robustness, while the latter are important for CNNs.

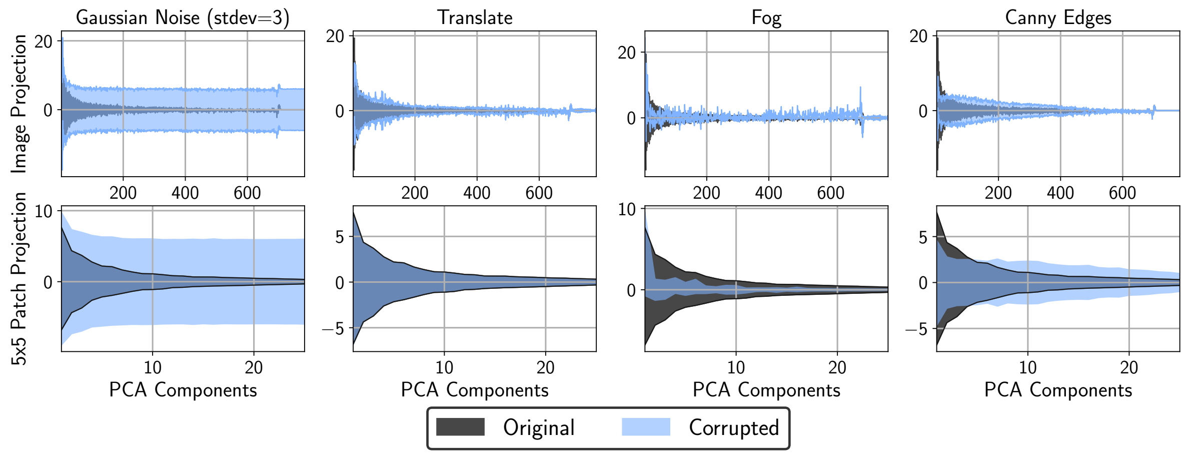


**Figure A.33: Corruptions and linear dependence. Top**: The distribution (mean ± 2 std) of MNIST and MNIST-C images and **bottom**: 5×5 patches extracted from these images projected onto the corresponding principal components of the training data images and patches. *Gaussian noise* corruption breaks linear dependencies in both cases, while *Translate* does not change the projection distribution for the $5 \times 5$ patches.

Certain corruptions increase variance along the lower PC directions more than others. For example, the *Translate* corruption does not alter the principal components of the $5 \times 5$ patches in the images, and so a convolutional BNN with a Gaussian prior is very robust to this corruption. In contrast, Gaussian noise increases variance similarly along all directions, breaking any linear dependencies present in the training data and resulting in much worse BNN performance.

## A.6.10 Analyzing other approximate inference methods

In this section, we provide additional discussion on why popular approximate inference methods SWAG and MC Dropout do not exhibit the same poor performance under covariate shift.

### A.6.10.1 Variational inference

Suppose the prior is $p(w) = \mathcal{N}(0, \alpha^2 I)$ and the variational family contains distributions of the form $q(w) = \mathcal{N}(\mu, \Lambda)$, where the mean $\mu$ and the covariance matrix $\Lambda$ are parameters. Variational inference solves the following optimization problem: maximize $\mathbb{E}_{w\sim\mathcal{N}(\mu,\Lambda)} p(D|w) - KL\left(\mathcal{N}(\mu, \Lambda) || \mathcal{N}(0, \alpha^2 I)\right)$

with respect to $\mu$, $\Lambda$ [Blundell et al. 2015; Kingma and Welling 2013].

First, let us consider the case when the parameter $\Lambda$ is unconstrained and can be any positive-definite matrix. Suppose we are using a fully-connected network, and there exists a linear dependence in the features, as in Proposition 4.3. Then, there exists a direction $d$ in the parameter space of the first layer of the model, such that the projection of the weights on this direction will not affect the likelihood, and the posterior over this projection will coincide with the prior and will be independent from other directions (Proposition 4.3,), which is Gaussian. Consequently, the optimal variational distribution will match the prior in this projection, and will also be independent from the other directions, or, in other words, $d$ will be an eigenvector of the optimal $\Lambda$ with eigenvalue $\alpha^2$. So, variational inference with a general Gaussian variational family will suffer from the same exact issue that we identified for the true posterior. Furthermore, we can generalize this result to convolutional layers completely analogously to Proposition 4.4,.

Now, let us consider the mean-field variational inference (MFVI) which is commonly used in practice in Bayesian deep learning. In MFVI, the covariance matrix $\Lambda$ is constrained to be diagonal. Consequently, for general linear dependencies in the features the variational distribution will not have sufficient capacity to make the posterior over the direction $d$ independent from the other directions. As a result, MFVI will not suffer as much as exact Bayesian inference from the issue presented in Propositions 4.3, 4.4,.

One exception is the dead pixel scenario described in Section 5.1, where one of the features in the input is a constant zero. In this scenario, MFVI will have capacity to make the variational posterior over the corresponding weight match the prior, leading to the same lack of robustness described in Proposition 4.2.

Empirical results. In addition to the theoretical analysis above, we ran mean field variational inference on our fully-connected network on MNIST and evaluated robustness on the MNIST-C corruptions. Below we report the results for MFVI, MAP and HMC BNN with a Gaussian prior:

**Table A.24:** Accuracy of MAP, MFVI and HMC BNNs under different corruptions. MFVI is more robust than HMC and even outperforms the MAP solution for some of the corruptions.

| method | CIFAR-10 | Gaussian Noise | Motion Blur | Scale | Brightness | Stripe | Canny Edges |
|---|---|---|---|---|---|---|---|
| MAP | **98.5%** | **70.6%** | **86.7%** | **77.6%** | 50.6% | 34.1% | 68% |
| MFVI | 97.9% | 62.5% | 82.2% | 70.5% | **68.9%** | **47.7%** | **70%** |
| HMC | 98.2% | 43.2% | 82.1% | 69.5% | 22.3% | 34.8% | 63.3% |

As expected from our theoretical analysis, MFVI is much more robust to noise than HMC BNNs. However, on some of the corruptions (Gaussian Noise, Motion Blur, Scale) MFVI underperforms the MAP solution. At the same time, MFVI even outperforms MAP on Brightness, Stripe and Canny Edges.

#### A.6.10.2 SWAG

SWA-Gaussian (SWAG) [Maddox et al. 2019] approximates the posterior distribution as a multivariate Gaussian with the SWA solution [Izmailov et al. 2018] as its mean. To construct the covariance matrix of this posterior, either the second moment (SWAG-Diagonal) or the sample covariance matrix of the SGD iterates is used. For any linear dependencies in the training data, the corresponding combinations of weights become closer to zero in later SGD iterates due to weight decay. Since SWAG only uses the last $K$ iterates in constructing its posterior, the resulting posterior will likely have very low variance in the directions of any linear dependencies. Furthermore, because SGD is often initialized at low magnitude weights, even the earlier iterates will likely have weights close to zero in these directions.

#### A.6.10.3 MC Dropout

MC Dropout applies dropout at both train and test time, thus allowing computation of model uncertainty from a single network by treating stochastic forward passes through the network as posterior samples. The full model learned at train time is still an approximate MAP solution, and

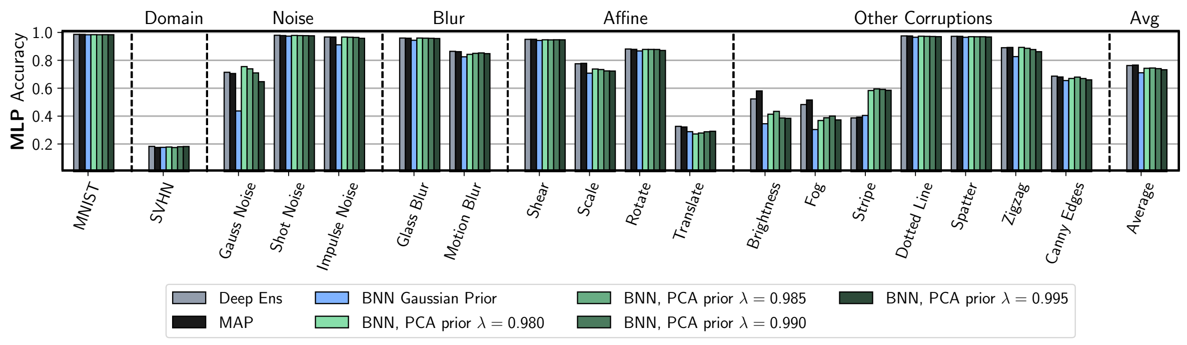


**Figure A.34: General PCA priors.** Performance of the PCA priors introduced in Section A.6.11 for various decay rates $\lambda$. PCA priors generally improve performance significantly under *Gaussian noise* and *Stripe*. Lower decay rates $\lambda$ provide better results under *Gaussian noise*.

thus will be minimally affected by linear dependencies in the data being broken at test time. As for the test-time dropout, we can conclude that if the expected output of the network is not affected by linear dependencies being broken, then any subset of that network (containing a subset of the network's hidden units) would be similarly unaffected. Additionally, if dropping an input breaks a linear dependency from the training data, the network (as an approximate MAP solution) is robust to such a shift.

## A.6.11 General PCA Priors

In Section 4.7 we introduced the *EmpCov* prior, which improves robustness to covariate shift by aligning with the training dataset's principal components. Following the notation used in Section 4.7, we can define a more general family of PCA priors as

$$p(w^1) = \mathcal{N}(0, \alpha V \mathrm{diag}(s) V^T + \epsilon I), \quad s_i = f(i) \tag{A.39}$$

where for an architecture with $n_w$ first layer weights, $s$ is a length $n_w$ vector, $\mathrm{diag}(s)$ is the $n_w \times n_w$ diagonal matrix with $s$ as its diagonal, and $V$ is an $n_w \times n_w$ matrix such that the $i^{th}$ column of $V$ is the $i^{th}$ eigenvector of $\Sigma$.

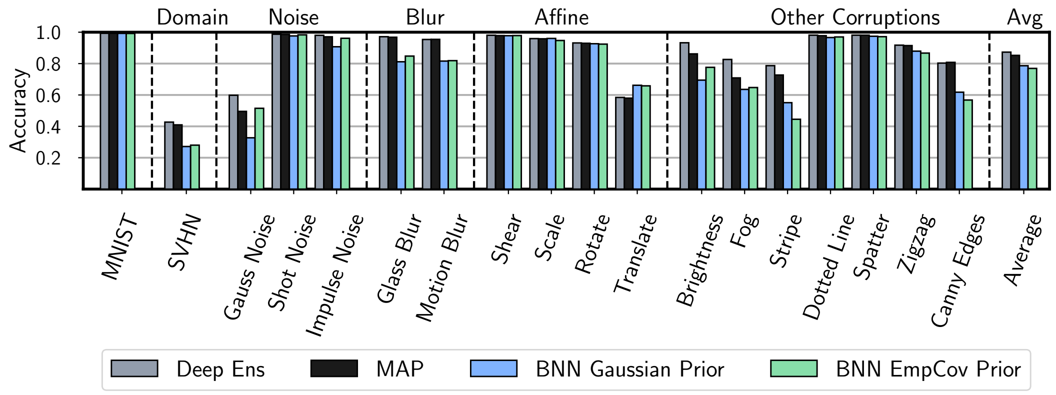


**Figure A.35: MNIST CNN results.** Test accuracy under covariate shift for deep ensembles, MAP optimization with SGD, and BNN with Gaussian and *EmpCov* priors.

The *EmpCov* prior is the PCA prior where $f(i)$ returns the $i^{th}$ eigenvalue (explained variance) of $\Sigma$. However, there might be cases where we do not want to directly use the empirical covariance, and instead use an alternate $f$. For example, in a dataset of digits written on a variety of different wallpapers, the eigenvalues for principal components corresponding to the wallpaper pattern could be much higher than those corresponding to the digit. If the task is to identify the digit, using *EmpCov* might be too restrictive on digit-related features relative to wallpaper-related features.

We examine alternative PCA priors where $f(i) = \lambda^i$ for different decay rates $\lambda$. We evaluate BNNs with these priors on MNIST-C, and find that the choice of decay rate can significantly alter the performance on various corruptions. Using priors with faster decay rates (smaller $\lambda$) can provide noticeable improvement on Gaussian Noise and Zigzag corruptions, while the opposite occurs in corruptions like Translate and Fog. Connecting this result back to Section A.6.9 and Figure A.33, we see that the corruptions where faster decay rates improve performance are often the ones which add more noise along the smallest principal components.

## A.6.12 Effect of non-zero mean corruptions

In Figure A.35, we report the results of deep ensembles, MAP and BNNs with Gaussian and *EmpCov* priors under various corruptions using the CNN architecture on MNIST. *EmpCov* improves performance on all of the noise corruptions. For example, on the *Gaussian noise* corruption, *EmpCov* achieves 58.3% accuracy while the Gaussian prior achieves 32.7% accuracy; similarly on the *impulse noise* the results are 96.5% and 90.73% respectively.

However, *EmpCov* does not improve the results significantly on *brightness* or *fog*, and even hurts the performance slightly on *stripe*. Below, we explain that these corruptions are non-zero mean, and the performance is affected by the sum of the filter weights. We thus propose the SumFilter prior which greatly improves the performance on these corruptions.

### A.6.12.1 Non-zero mean corruptions

As we have seen in various experiments (e.g. Figure 4.9, Figure A.31), convolutional Bayesian neural networks are particularly susceptible to the *brightness* and *fog* corruptions on MNIST-C. Both of these corruptions are not zero-mean: they shift the average value of the input features by 1.44 and 0.89 standard deviations respectively. In order to understand why non-zero mean corruptions can be problematic, let us consider a simplified corruption that applies a constant shift $c$ to all the pixels in the input image. Ignoring the boundary effects, the convolutional layers are linear in their input. Denoting the output of the convolution with a filter $w$ on an input $x$ as $conv(w, x)$, and an image with all pixels equal to 1 as $\mathbb{1}$ we can write

$$conv(w, x + c \cdot \mathbb{1}) = conv(w, x) + c \cdot conv(w, \mathbb{1}) = conv(w, x) + \mathbb{1} \cdot c \cdot \sum_{a,b} w_{a,b}, \tag{A.40}$$

where the last term represents an image of the same size as the output of the $conv(\cdot, \cdot)$ but with all pixels equal to the sum of the weights in the convolutional filter $w$ multiplied by $c$. So, if the input of the convolution is shifted by a constant value $c$, the output will be shifted by a constant value $c \cdot \sum_{a,b} w_{a,b}$.

As the convolutional layer is typically followed by an activation such as ReLU, the shift in the output of the convolution can significantly hinder the performance of the network. For example, suppose $c \cdot \sum_{a,b} w_{a,b}$ is a negative value such that all the output pixels in $conv(w, x) + \mathbb{1} \cdot c \cdot \sum_{a,b} w_{a,b}$ are negative. In this case, the output of the ReLU activation applied after the convolutional filter will be 0 at all output locations, making it impossible to use the learned features to make predictions.

In the next section, we propose a prior that reduces the sum $\sum_{a,b} w_{a,b}$ of the filter weights, and show that it significantly improves robustness to multiple corruptions, including *fog* and *brightness*.

#### A.6.12.2 SumFilter prior

As we've discussed, if the sum of filter weights for CNNs is zero, then corrupting the input by adding a constant has no effect on our predictions. We use this insight to propose a novel prior that constrains the sum of the filter weights. More specifically, we place a Gaussian prior on the parameters and Laplace prior on the sum of the weights:

$$p(w) \sim \mathcal{N}\left(w|0, \alpha^2 I\right) \times \text{Laplace}\Big(\sum_{\text{filter}} w|0, \gamma^2\Big). \tag{A.41}$$

For our experiments, we only place the additional Laplace prior on the sum of weights in first layer filters. An alternative version could place the prior over filter sums in subsequent layers, which may be useful for deeper networks.

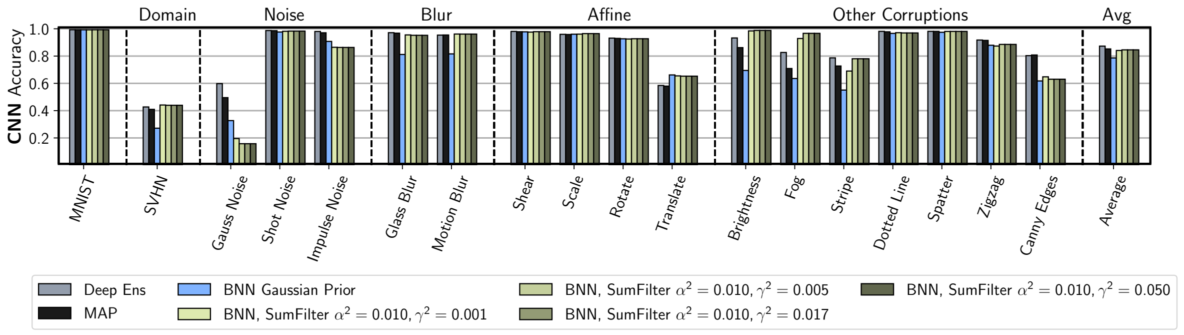


**Figure A.36: SumFilter priors.** Performance of BNNs with the *SumFilter* priors introduced in subsubsection A.6.12.2 for the CNN architecture on MNIST. *SumFilter* priors do not improve the performance under *Gaussian noise* unlike *EmpCov priors*, but provide a significant improvement on the *Brightness* and *Fog* corruptions.

#### A.6.12.3 Experiments

Figure A.36 shows that this prior substantially improves the performance of a convolutional BNN on MNIST-C. The BNN with a filter sum prior yields a better or comparable performance to MAP for all MNIST corruptions, with the exception of *Canny Edges* and *Impulse Noise*. We also implemented this prior for MLPs, but found that it only improved BNN performance on two corruptions, fog and brightness. Overall, this prior addresses a more specific issue than *EmpCov*, and we would not expect it to be applicable to as many forms of covariate shift.

### A.6.13 Example: Bayesian NALU under covariate shift

The Neural Arithmetic Logic Unit (NALU) [Trask et al. 2018] is an architecture which can learn arithmetic functions that extrapolate to values outside those observed during training. A portion of the unit is of the form $\prod_{j=1}^{m} |x_j|^{w_j}$, and in this section we examine a simplified form of this unit in order to demonstrate an instance where nonlinear dependencies hurt BMA under covariate shift.

Let's consider the NALU-inspired architecture with input features $x^1, \dots, x^m$ that takes the form $f(x, w) = \prod_{j=1}^{m} (x^j)^{w_j}$. Suppose the prior over the weights $w = [w_1, \dots, w_m]$ is an i.i.d.

Gaussian distribution $\mathcal{N}(0, \alpha^2)$. Suppose all inputs $x_1, \ldots x_n$ in training dataset $\mathcal{D}$ lie in a subspace of the input space: $\prod_{j=1}^{m} (x_i^j)^{p_j} = 1$ for all $i = 1, \ldots, n$ and some constant vector $p$ such that $\sum_{j=1}^{m} p_j^2 = 1$. Following the same approach as subsubsection A.6.8.3, we can introduce a new basis $v_1, \ldots, v_m$ in the input space such that $v_1 = p$. We can similarly re-parameterize the model using the weights rotated into this new basis, $\bar{w} = w^T v_1, \ldots, w^T v_m$, and it follows that $w_i = \bar{w}_1 \cdot v_1^i + \cdots + \bar{w}_m \cdot v_m^i$ for all $i = 1, \ldots, n$. Using the corresponding transformed inputs $\bar{x}^i = \prod_{j=1}^{m} (x^j)^{v_i^j}$ for all $i = 1, \ldots, n$, we can rewrite our model as follows:

$$f(x, w) = f(\bar{x}, \bar{w}) = (\prod_{j=2}^{m} (\bar{x}^j)^{\bar{w}_j}) \cdot \underbrace{(\bar{x}^1)^{\bar{w}_1}}_{=1}. \tag{A.42}$$

Since $f(\bar{x}, \bar{w})$ does not depend on $\bar{w}_1$ for all $\bar{x} \in \bar{\mathcal{D}}$, we can follow the same reasoning from subsubsection A.6.8.1 to conclude that the marginal posterior over $\bar{w}_1$ coincides with the induced prior. Since $\bar{w}$ is the result of simply rotating $w$ into a new basis, it also follows that the induced prior over $\bar{w}$ is $\mathcal{N}(0, \alpha^2 I)$, and that the posterior can be factorized as $p(\bar{w}|\bar{\mathcal{D}}) = p(\bar{w} \setminus \bar{w}_1|\bar{\mathcal{D}}) \cdot p(\bar{w}_1)$.

Consider a test input $\bar{x}_k(c) = [c, \bar{x}_k^2, \ldots, \bar{x}_k^m]$. The predictive mean under BMA will be:

$$\mathbb{E}_{\bar{w}} f(\bar{x}_k(c), \bar{w}) = \mathbb{E}_{\bar{w} \setminus \bar{w}_1} \prod_{j=2}^{m} (\bar{x}_k^j)^{\bar{w}_j} \cdot \mathbb{E}_{\bar{w}_1} c^{\bar{w}_1}. \tag{A.43}$$

Thus the predictive mean depends upon $c$, and so the BMA will not be robust to the nonlinear dependency being broken at test time. In comparison, the MAP solution would set $\bar{w}_1 = 0$, and its predictions would not be affected by $c$.

While the dependency described in this section may not necessarily be common in real datasets, we highlight this example to demonstrate how a nonlinear dependency can still hurt BMA robustness. This further demonstrates how the BMA issue we've identified does not only involve *linear* dependencies, but rather involves dependencies which have some relationship to the model architecture.

### A.6.14 Dead neurons

Neural network models can often contain *dead neurons*: hidden units which output zero for all inputs in the training set. This behaviour occurs in classical training when a neuron is knocked off the training data manifold, resulting in zero non-regularized gradients for the corresponding weights and thus an inability to train the neuron using the gradient signal from the non-regularized loss. However, we can envision scenarios where a significant portion of the BNN posterior distribution contains models with dead neurons, such as when using very deep, over-parameterized architectures.

Let us consider the posterior distribution over the parameters $W$ conditioned on the parameters $W^1, W^2, b^2, \ldots, W^k, b^k$ of the first $k$ layers, where we use the notation of Section A.6.8. Suppose for the parameters $W^1, W^2, b^2, \ldots, W^k, b^k$ the $k$-th layer contains a dead neuron, i.e. an output that is 0 for all the inputs $x_j$ in the training dataset $D$. Then, consider the sub-network containing layers $k+1, \ldots, l$. For this sub-network, the output of a dead neuron in the $k$-th layer is an input that is 0 for all training inputs. We can then apply the same reasoning as we did in subsubsection A.6.8.1, subsubsection A.6.8.2 to show that there will exist a direction in the parameters $W^k$ of the $k+1$-st layer, such that along this direction the posterior *conditioned* on the parameters $W^1, W^2, b^2, \ldots, W^k, b^k$ coincides with the prior (under the assumption that the prior over the parameters $W^k$ is iid and independent of the other parameters). If a test input is corrupted in a way that activates the dead neuron, the predictive distribution of the BMA *conditioned* on the parameters $W^1, W^2, b^2, \ldots, W^k, b^k$ will change.

### A.6.15 Bayesian linear regression under covariate shift

We examine the case of Bayesian linear regression under covariate shift. Let us define the following Bayesian linear regression model:

$$y = w^\top \phi(x, z) + \epsilon(x) \tag{A.44}$$

$$\epsilon \sim \mathcal{N}(0, \sigma^2) \tag{A.45}$$

where $w \in \mathbb{R}^d$ are linear weights and $z$ are the deterministic parameters of the basis function $\phi$. We consider the dataset $D = \{(x_i, y_i)\}_{i=1}^n$ and define $y := (y_1, \dots, y_N)^\top$, $X := (x_1, \dots, x_n)^\top$, and $\Phi := (\phi(x_1, z), \dots, \phi(x_n, z))^\top$

The likelihood function is given by:

$$p(y|X, w, \sigma^2) = \prod_{i=1}^{n} \mathcal{N}(y_i | w^\top \phi(x_i, z), \sigma^2). \tag{A.46}$$

Let us choose a conjugate prior on the weights:

$$p(w) = \mathcal{N}(w|\mu_0, \Sigma_0), \tag{A.47}$$

The posterior distribution is given by:

$$
\begin{aligned}
p(w|\mathcal{D}) &\propto \mathcal{N}(w|\mu_0, \Sigma_0) \times \prod_{i=1}^{n} \mathcal{N}(y_i|w^\top \phi(x_i, z), \sigma^2) \\
&= \mathcal{N}(w|\mu, \Sigma), \\
\mu &= \Sigma \left( \Sigma_0^{-1}\mu_0 + \frac{1}{\sigma^2}\Phi^\top y \right), \\
\Sigma^{-1} &= \Sigma_0^{-1} + \frac{1}{\sigma^2}\Phi^\top \Phi.
\end{aligned}
$$

The MAP solution is therefore equal to the mean,

$$
w_{\text{MAP}} = \Sigma \left( \Sigma_0^{-1}\mu_0 + \frac{1}{\sigma^2}\Phi^\top y \right) = \left( \Sigma_0^{-1} + \frac{1}{\sigma^2}\Phi^\top \Phi \right)^{-1} \left( \Sigma_0^{-1}\mu_0 + \frac{1}{\sigma^2}\Phi^\top y \right) \tag{A.48}
$$

Thus, we see that the BMA and MAP predictions coincide in Bayesian linear regression, and both will have equivalent performance under covariate shift in terms of accuracy.

**What happens away from the data distribution?** If the data distribution spans the entire input space, than the posterior will contract in every direction in the weight space. However, if the data lies in a linear (or affine, if we are using a Gaussian prior) subspace of the input space, there will be directions in the parameter space for which the posterior would coincide with the prior. Now, if a test input does not lie in the same subspace, the predictions on that input would be affected by the shift vector according to the prior. Specifically, if the input $x$ is shifted from the subspace containing the data by a vector $v$ orthogonal to the subspace, then the predictions between $x$ and its projection to the subspace would differ by $w^T v$, where $w \sim \mathcal{N}(\mu_0, \Sigma_0)$, which is itself $\mathcal{N}(\mu_0^T v, v^T \Sigma_0 v)$. Assuming the prior is zero-mean, the mean of the prediction would not be affected by the shift, but the uncertainty will be highly affected. The MAP solution on the other hand does not model uncertainty.

## A.6.16 An optimization perspective on the covariate shift problem.

In this section, we examine SGD's robustness to covariate shift from an optimization perspective.

### A.6.16.1 Effect of regularization and initialization on SGD's robustness to covariate shift

In Section A.6.14, we discussed how SGD pushes the weights that correspond to dead neurons, a generalization of the dead pixels analysis, towards zero thanks to the regularization term. In this section, we study the effect of regularization and initialization on SGD's robustness under covariate shift.

**Regularization**

To study the effect of regularization on SGD's robustness under covarite shift, we hold all hyper-parameters fixed and we change the value of the regularization parameter. Figure A.37 shows the outcome. Most initialization schemes for neural networks initialize the weights with values close to zero, hence we expect SGD not perform as poorly on out-of-distribution data as HMC on these networks even without regularization. Therefore, we see that SGD without regularization (reg = 0.0) is still competitive with reasonably regularized SGD.

**Initialization**

The default initialization scheme for fully-connected layers in Pytorch for example is the He initialization [He et al. 2015]. We use a uniform initialization $\mathcal{U}(-b, b)$ and study the effect of varying $b$ on the performance of SGD under covarite shift for a fully-connected neural network. Figure A.38 shows our empirical results, where smaller weights result in better generalization on most of the corruptions.

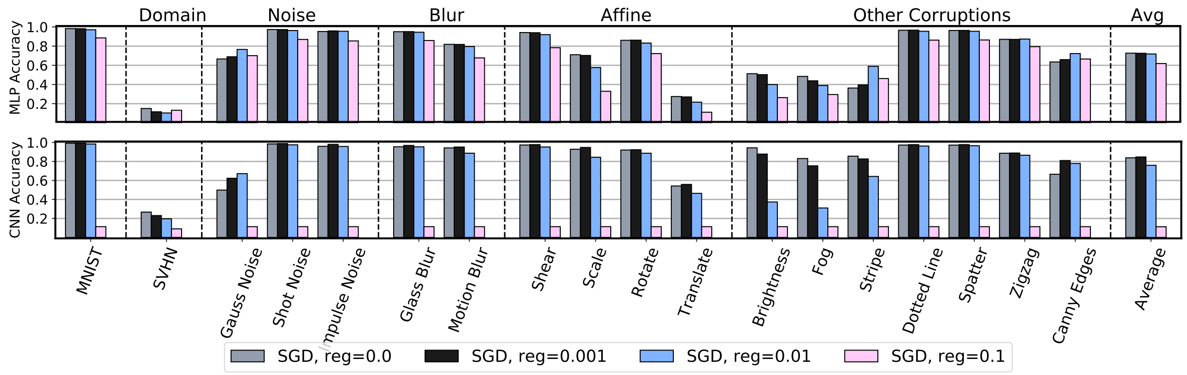


**Figure A.37: Effect of regularization on SGD's performance on corrupted MNIST.** Accuracy for the following values of the regularization parameter: 0.0, 0.001, 0.01, and 0.1. **Top**: Fully-connected network; **bottom**: Convolutional neural network. Regularization helps improve the performance on some corruptions, such as Gaussian noise, but its absence does not affect SGD's robustness under covariate shift because the weights are initialized at small values.

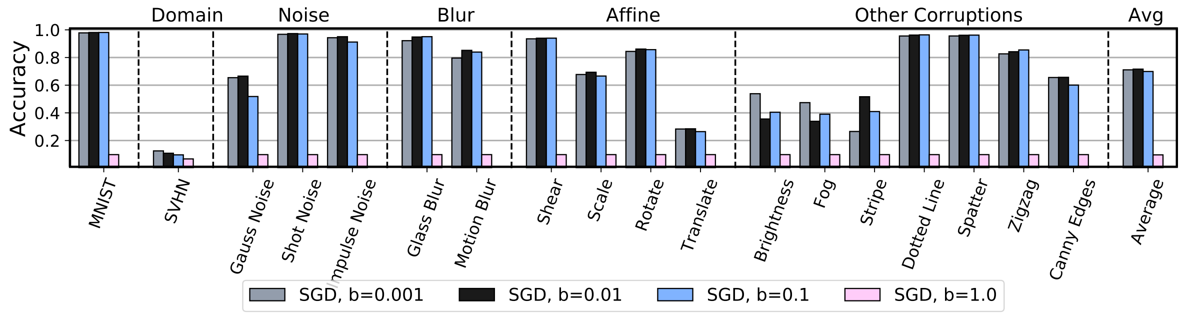


**Figure A.38: Effect of initialization on SGD's performance on corrupted MNIST with MLP.** The weights are initialized using a uniform distribution $\mathcal{U}(-b, b)$, and we consider the following values for $b$: 0.001, 0.01, 0.1, and 1.0. All experiments were run without regularization. For most corruptions, initializing the weights at smaller values leads to better robustness to covariate shift.

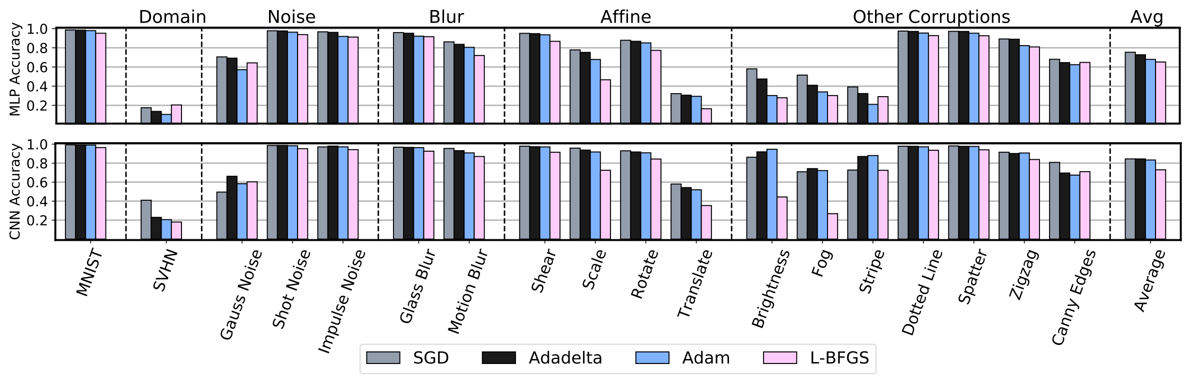


**Figure A.39: Robustness on MNIST for different stochastic optimizers.** Accuracy for SGD, Adadelta, Adam and L-BFGS on MNIST under covariate shift. **Top**: Fully-connected network; **bottom**: Convolutional neural network. Adam and Adadelta provide competitive performance with SGD for most corruptions. However, SGD is better on the MLP architecture for some corruptions whereas Adam and Adadelta are better on the *same* corruptions with the CNN architecture.

#### A.6.16.2 Other stochastic optimizers

In addition to SGD, we examine the performance of Adam [Kingma and Ba 2014], Adadelta [Zeiler 2012], L-BFGS [Nocedal 1980; Liu and Nocedal 1989] on corrupted MNIST. Figure A.39 shows the results for all 4 algorithms on the MNIST dataset under covariate shift, for both fully-connected and convolutional neural networks. We see that SGD, Adam and Adadelta have comparable performance for convolutional neural networks, whereas SGD has an edge over both algorithms on MLP. L-BFGS provides a comparatively poor performance and we hypothesise that it is due to the lack of regularization. Naive regularization of the objective function does not improve the performance of L-BFGS.

#### A.6.16.3 Loss surface analysis

There have been several works that tried to characterize the geometric properties of the loss landscape and describe its connection to the generalization performance of neural networks. In particular, it is widely believed that flat minima are able to provide better generalization [Hochreiter and Schmidhuber 1997a; Keskar et al. 2016]. Intuitively, the test distribution introduces a horizontal shift in the loss landscape which makes minima that lie in flat regions of the loss surface perform well for both train and test datasets. From the other side, it is well-known that SGD produces flat minima. Hence, we would like to understand the type of distortions that corruptions in the corrupted CIFAR-10 dataset introduce in the loss surface, and evaluate the potential advantage of flat minima in this context.

In the same fashion as Li et al. [2017], we visualize the effect of the Gaussian noise corruption on the loss surface for different intensity levels as shown in Figure A.40. These plots are produced for two random directions of the parameter space for a ResNet-56 network. We observe that high levels of intensity make the loss surface more flat, but result in a worse test loss overall. We can see visually that the mode in the central flat region, that we denote $w_0$, is less affected by the corruption than a solution picked at random. Figure A.41 shows the loss difference between different solutions including $w_0$, that we call *optimal*, and the new mode for each corruption intensity. We can see that the mode is indeed less affected by the corruptions than other randomly selected solutions of the same loss region.

#### A.6.16.4 Licensing

The MNIST dataset is made available under the terms of the Creative Commons Attribution-Share Alike 3.0 license. The CIFAR-10 dataset is made available under the MIT license. Our code is a fork of the Google Research repository at https://github.com/google-research/google-research, which has source files released under the Apache 2.0 license.

## A.7 Appendix for Deep Feature Reweighting

This appendix is organized as follows. In Section A.7.1 we provide additional related work discussion. In Section A.7.2 we present details on the experiments on feature learning in the presence of spurious correlations. In Section A.7.3 we provide details on the results for the spurious correla-

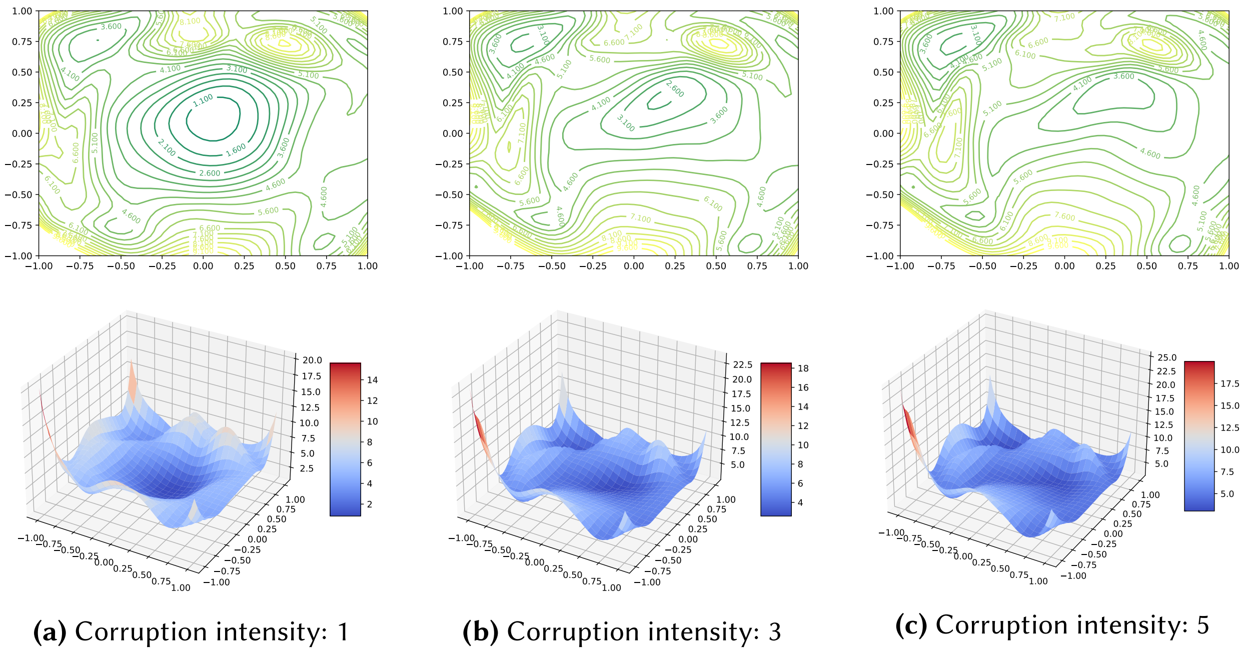

**(a)** Corruption intensity: 1 **(b)** Corruption intensity: 3 **(c)** Corruption intensity: 5

**Figure A.40:** 2D (**top**) and 3D **bottom** visualizations of the loss surface of a ResNet-56 network on corrupted CIFAR-10 with different intensity levels of the Gaussian noise corruption.

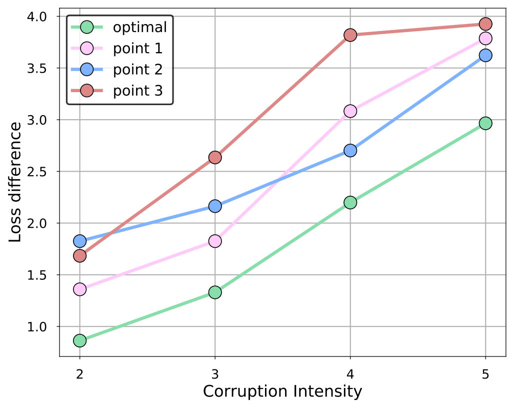


**Figure A.41:** Loss difference between different random solutions, including the mode found through standard SGD training (*optimal* in the legend), and the new mode for each corruption intensity.

tion benchmark datasets and perform ablation studies. We provide details on the experiments on the reliance of ImageNet-trained models on the background in Section A.7.5 and on the texture-bias in Section A.7.6. In Section **??** we provide further discussion of the results presented in the paper and future work. In Section A.7.7 we compare DFR to methods proposed in Kang et al. [2019] and other last layer retraining variations.

Code references. In addition to the experiment-specific packages that we discuss throughout the appendix, we used the following libraries and tools in this work: NumPy [Harris et al. 2020], SciPy [Virtanen et al. 2020b], PyTorch [Paszke et al. 2017], Jupyter notebooks [Kluyver et al. 2016], Matplotlib [Hunter 2007], Pandas [McKinney 2010], transformers [Wolf et al. 2019].

### A.7.1 Additional related work

Differences with Kang et al. [2019] . Our work and Kang et al. [2019] consider different settings: Kang et al. [2019] considers long-tail classification with class imbalance, while we consider spurious correlations and shortcut learning. In spurious correlation robustness, there is often no class imbalance, and the methods of Kang et al. [2019] cannot be directly applied. Our conceptual results, such as the ability to control the reliance on background or texture features in trained models are also orthogonal to the observations of Kang et al. [2019].

Algorithmically, DFR is related to the methods of Kang et al. [2019], but there are still important differences. In the Learning Weight Scaling (LWS) method, the authors rescale the logits of the classifier with scalar weights $f_i$: the weight of the $i$-th row of the weight matrix in the last layer is updated to $\hat{w}_i = f_i w_i$. The parameters $f_i$ are trained by minimizing the loss on the training set with class-balanced sampling. In particular, LWS is not full last layer retraining, as only one parameter per class is learned.

Kang et al. [2019] also proposed a Classifier Re-training (cRT) approach which retrains the last layer on the training set with class-balanced sampling, where we are equally likely to sample datapoints from each class. cRT is closer to DFR than LWS, as it retrains all parameters in the last layer.

The algorithmic differences of these methods with DFR are as follows:

- In DFR, we use a *group*-balanced subset of the data instead of *class*-balanced sampling. This distinction is important, as in the group robustness setting the classes are often balanced, so LWS and cRT are not immediately applicable to the spurious correlation setting.

- In DFR, we *subsample* the reweighting dataset to be group balanced instead of using class- or group-balanced *sampling*. Specifically, we produce a dataset where the number of examples in each group is the same, and only use these datapoints. This detail is hugely important, as group-balanced sampling does not produce classifiers robust to spurious correlations [e.g. see RWG and SUBG methods in Idrissi et al. 2021].

- In $\text{DFR}^{\text{Val}}_{\text{Tr}}$, we use *held-out data* for retraining the last layer, and not the training data. This is the important distinction between $\text{DFR}^{\text{Val}}_{\text{Tr}}$ and $\text{DFR}^{\text{Tr}}_{\text{Tr}}$. LWS and cRT both retrain the classifier on the training data.

- There are also technical differences in how we train the last layer in DFR: we average the weights of several independent runs of last layer retraining on different group-balanced subsets of the data, and use strong $\ell_1$ regularization.

To sum up, both cRT and LWS are not immediately applicable to spurious correlation robustness. Moreover, even if we adapt cRT to the spurious correlation setting, there are still major differences with DFR: subsampling vs group-balanced sampling, held-out data vs training data, and regularization. We present comparisons to LWS, cRT and related last layer retraining variations on Waterbirds and CelebA in Appendix A.7.7, where we show that both DFR versions outperform methods proposed in Kang et al. [2019] and other ablations. In Appendix A.7.3, we also show that regularization and multiple re-runs of last layer retraining are important for strong performance with DFR.

Differences with Rosenfeld et al. [2022] . While our works make similar high-level observations, they are actually complementary to each other. In particular, our works don't share any datasets, experiments or problem settings. Rosenfeld et al. [2022] focus on domain generalization, where the goal is to train a model that generalizes to unseen domains. In this setting, we know the domain labels on the train, and we have no data from the test domains. In spurious correlations, the goal is to train a model that does not rely on spurious features. We typically have examples from all the test groups in train, but the groups are highly imbalanced. As in domain generalization we do not have access to target domain data, Rosenfeld et al. [2022] refer to last layer retraining on the target domain as "cheating" (see e.g. the caption of Figure 1 in their paper). Consequently, they propose a different method, DARE, which is very different from DFR. DARE estimates means and covariance matrices for each domain to whiten out the features. On test, they apply approximate whitening to a new domain, and they don't use test domain data to retrain the last layer. On the other hand, DFR uses an ERM-trained feature extractor and simply retrains the last layer on a group-balanced reweighting dataset.

To sum up, the observations from our work and the concurrent work by Rosenfeld et al. [2022] are complimentary: we both show that ERM learns the core features well, but the settings, methods and experiments are different. In particular, it is not trivial to apply DARE to spurious correlations or DFR to domain generalization.

### A.7.2 Details: Understanding Representation Learning with Spurious Correlations

Here we provide details on the experiments in Section 5.3.

#### A.7.2.1 Feature learning on Waterbirds

Inverse problem. In Table A.25 we present the results on the inverted Waterbirds problem, where the goal is to predict the background type while the bird type serves as a spurious feature. We note that prior work did not consider this reverse Waterbirds problem. The results for the inverted problem are analogous to the results for the standard Waterbirds (Table 5.1): models trained on the Original data learn the background features sufficiently well to predict the background type with high accuracy when the spurious foreground feature is not present, but perform poorly when presented with conflicting background and foreground features. While it is often suggested than neural networks are biased to learn the background [Xiao et al. 2020], we see that in fact the network relies on the spurious foreground feature (bird) when trained to predict the background.

Data. We show examples of Original, FG-Only and BG-Only Waterbirds images in Figure A.42. To generate the data, we follow the instructions at `github.com/kohpangwei/group_DRO#waterbirds`, but in addition to the Original data we save the backgrounds and foregrounds separately. Consequently, the FG-Only data contains the same exact birds images as the Original data, and the BG-Only data contains the same exact places images as the Original data. For the 100% spurious correlation strength we simply discard all the minority groups data from the Original (95% spurious correlation) training dataset. For the Balanced data, we start with the Original data with 95% spurious correlation and replaced the background in the smallest possible number of images (chosen randomly) to achieve a 50% spurious correlation strength.

Hyper-parameters. For the experiments in this section we use a ResNet-50 model pretrained on ImageNet, imported from the `torchvision` package: `torchvision.models.resnet50(pretrained=True)` [?]. We train the models for 50 epochs

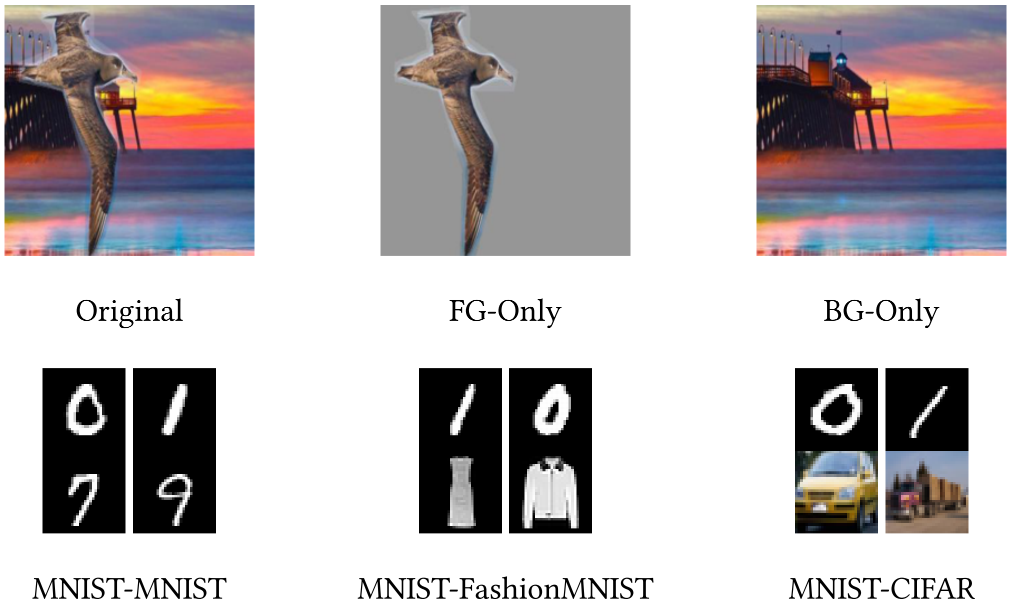


**Figure A.42: Data examples.** Variations of the waterbirds (Top) and Dominoes (Bottom) datasets generated for the experiment in Section 5.3.

with SGD with a constant learning rate of $10^{-3}$, momentum decay of 0.9, batch size 32 and a weight decay of $10^{-2}$. We use random crops (`RandomResizedCrop(224, scale=(0.7, 1.0), ratio=(0.75, 4./3.), interpolation=2)`) and horizontal flips (`RandomHorizontalFlip()`) implemented in `torchvision.transforms` as data augmentation.

#### A.7.2.2 Dominoes datasets

Data. We show data examples from Dominoes datasets (MNIST-MNIST, MNIST-FashionMNIST and MNIST-CIFAR) in Figure A.42. The top half of each image shows MNIST digits from classes $\{0, 1\}$, and the bottom half shows: MNIST images from classes {7, 9} for MNIST-MNIST, Fashion-MNIST images from classes {coat, dress} for MNIST-FashionMNIST, and CIFAR-10 images from classes {car, truck} for MNIST-CIFAR. The label corresponds to the more complex bottom part of the image, but the top and bottom parts are correlated (we consider 95%, 99% and 100% levels of spurious correlation strength in experiments). 20% of the training data was reserved for validation or *reweighting* dataset (see Section 5.4) where each group is equally represented.

| Train Data | Spurious Corr. (%) | Test Data (Worst/Mean, %) | |
|---|---|---|---|
| | | Original | BG-Only |
| Balanced | 50 | 93.2/95.6 | 93.6/96.0 |
| Original | 95 | 77.4/91.2 | 93.1/95.7 |
| Original | 100 | 36.1/77.5 | 92.7/94.8 |
| Place-Only | - | 91.8/94.2 | 92.4/95.2 |

**Table A.25: Feature learning on Inverted Waterbirds.** ERM classifiers trained on Inverted Waterbirds with Original and BG-Only images. Here the target is associated with the background type, and the foreground (bird type) serves as the spurious feature. All the models trained on the Original data including the model trained without any minority group examples (*Spurious corr.* 100%) underperform on the worst-group accuracy on the Original data, but perform well on the BG-Only data, almost matching the performance of the BG-Only trained model.

Hyper-parameters. We used a randomly initialized ResNet-20 architecture for this set of experiments. We trained the network for 500 epochs with SGD with batch size 32, weight decay $10^{-3}$, initial learning rate value $10^{-2}$ and a cosine annealing learning rate schedule. For the logistic regression model, we first extract the embeddings from the penultimate layer of the network, then use the logistic regression implementation (`sklearn.linear_model.LogisticRegression`) from the `scikit-learn` package [?]. We use $\ell_1$ regularization and tune the inverse regularization strength parameter $C$ in the range $\{0.1, 10, 100, 1000\}$. For more details on Deep Feature Reweighting implementation and tuning, see section A.7.3.

Transfer from simple to complex features. In addition to the results presented in Figure 5.2, we measure the decoded accuracy using the model trained just on the spurious simple features: the top half of the image showing an MNIST digit. After training, we retrain the last layer of the model using a validation split of the corresponding Dominoes dataset which has both top and bottom parts. In this case, we measure the transfer learning performance with the features

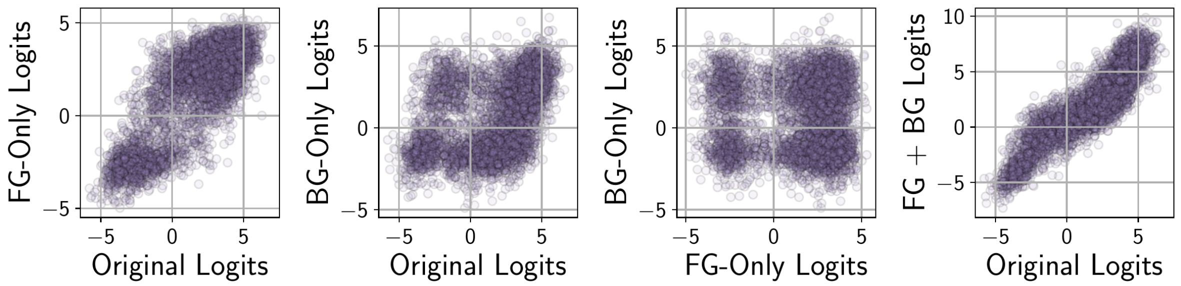


**Figure A.43: Logit additivity.** Distribution of logits for the negative class on the test data for a model trained on Waterbirds dataset. We show scatter plots of the logits for the Original, FG-Only and BG-Only images. The logits on Original images are well aligned with sums of logits on the corresponding FG and BG images (rightmost panel), suggesting that the foreground and background features are processed close to independently in the network.

learned on the binary MNIST classification problem applied to the more complex bottom half of the image. We obtain the following *transfer accuracy* results: 92.5% on MNIST-MNIST, 92.1% on MNIST-Fashion and 61.4% on MNIST-CIFAR. On all datasets, the decoded accuracy reported in Figure 5.2 for the spurious correlation levels 99% and 95% is better than the transfer accuracy. For the 100% spurious correlation, transfer achieves comparable results on MNIST-Fashion and MNIST-CIFAR, but on MNIST-MNIST the decoded accuracy with a model trained on the data with the core feature is significantly higher (99% vs 92%). These results confirm that for the spurious correlation strength below 100%, the model learns a high quality representation of the core features, which cannot be explained by transfer learning from the spurious feature.

#### A.7.2.3 ColorMNIST

**Data.** We follow Zhang et al. [2022] to generate ColorMNIST dataset: there are 5 classes corresponding to pairs of digits $(0, 1)$, $(2, 3)$, $(4, 5)$, $(6, 7)$, $(8, 9)$, and 5 fixed colors such that in train data each class $y$ has a color $s$ which is associated with it with correlation $p_{corr}$, and the remaining examples are colored with the remaining colors uniformly at random. Validation split is a randomly sampled 10% fraction of the original MNIST data, and validation and test data exmaples are colored with 5 colors uniformly at random.

| $p_{corr}$ | no corr. | 0.8 | 0.9 | 0.95 | 0.995 | 1.0 |
|---|---|---|---|---|---|---|
| ERM | $94.5_{\pm 1.0}$ | $85.1_{\pm 2.2}$ | $66.0_{\pm 7.2}$ | $40.8_{\pm 3.6}$ | $0.0_{\pm 0.0}$ | $0.0_{\pm 0.0}$ |
| $\text{DFR}^{\text{Val}}_{\text{Tr}}$ | – | $93.8_{\pm 0.9}$ | $92.6_{\pm 0.7}$ | $91.6_{\pm 0.8}$ | $80.4_{\pm 1.1}$ | $77.3_{\pm 1.3}$ |

**Table A.26: Results on ColorMNIST.** Worst-group accuracy of a small CNN model trained on 5-class ColorMNIST dataset, varying the spurious correlation strength $p_{corr}$ between classes and colors. DFR achieves strong performance even in the challenging settings with strong correlation between class labels and colors. We report mean ± std over 3 independent runs of the method.

**Hyper-parameters.** We train a small LeNet-like Convolutional Neural Network model on this dataset varying $p_{corr}$. The model is trained for 5 epochs, with batch size 32, learning rate $10^{-3}$, and weight decay $5 \times 10^{-4}$, following Zhang et al. [2022]. CNN consists of 2 convolutional layers with 6 and 16 output filters, followed by a max pooling layer, then another convolutional layer with 32 output filters and a fully-connected layer.

We report worst-group accuracy for ERM and $\text{DFR}^{\text{Val}}_{\text{Tr}}$ in Table A.26. The *no corr.* results for ERM correspond to the model trained on the dataset without correlation between colors and class labels. Notably, DFR recovers strong worst-group performance even in the cases when base model has 0% worst-group accuracy ($p_{corr} = 1.0$ and $p_{corr} = 0.995$). For $p_{corr} < 0.95$ DFR's worst-group accuracy is closely matching the optimal accuracy of the model trained on data without spurious correlations.

#### A.7.2.4 Logit additivity

To better understand why the models trained on the Original Waterbirds data perform well on FG-Only images, in Figure A.43 we inspect the logits of a trained model. We show scatter plots of logits for the negative class (logits for the positive class behave analogously) on the Original, FG-Only and BG-Only test data. Both logits on FG-Only and BG-Only data correlate with the logits on the Original images, and FG-Only show a higher correlation. The FG-Only and BG-Only logits are not correlated with each other, as in test data the groups are balanced and the foreground and background are independent.

We find that the sum of the logits for the BG-Only and the logits for the FG-Only images provides a good approximation of the logits on the corresponding Original image (combining the foreground and the background). We term this phenomenon *logit additivity*: on Waterbirds, logits for the different predictive features (both core and spurious) are computed close to independently and added together in the last classification layer.

### A.7.3 Details: Spurious Correlation Benchmarks

Here we provide details and additional ablations on the experiments in Section 5.5.

Data. We use the standard Waterbirds, CelebA and MultiNLI datasets, following e.g. [Sagawa et al. 2020; Liu et al. 2021; Idrissi et al. 2021]. We use CivilComments dataset as implemented in the WILDS benchmark [Koh et al. 2021]. See Figures A.44, A.45 for group descriptions and data examples.

Base model hyper-parameters. For the experiments on Waterbirds and CelebA we use a ResNet-50 model pretrained on ImageNet, imported from the `torchvision` package: `torchvision.models.resnet50(pretrained=True)`. We use random crops (`RandomResizedCrop(224, scale=(0.7, 1.0), ratio=(0.75, 4./3.), interpolation=2)`) and horizontal flips (`RandomHorizontalFlip()`) implemented in `torchvision.transforms` as data augmentation. We train all models with SGD with momentum decay of 0.9 and a constant learning rate. On Waterbirds, we train the models for 100 epochs with weight decay $10^{-3}$, learning rate $10^{-3}$ and batch size 32. On CelebA, we train the models for 50 epochs with weight decay $10^{-4}$, learning rate $10^{-3}$ and batch size 128. We do not use early stopping. On MultiNLI and CivilComments, we use the BERT for classification model from the `transformers` package: `BertForSequenceClassification.from_pretrained('bert-base-uncased',`

| Method | ImageNet | Dataset | Waterbirds | | CelebA | |
|---|---|---|---|---|---|---|
| | Pretrain | Fine-tune | Worst(%) | Mean(%) | Worst(%) | Mean(%) |
| Base Model | ✓ | ✓ | $74.9_{\pm 2.4}$ | $98.1_{\pm 0.1}$ | $46.9_{\pm 2.8}$ | $95.3_{\pm 0}$ |
| $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ | ✓ | ✓ | $90.2_{\pm 0.8}$ | $97.0_{\pm 0.3}$ | $80.7_{\pm 2.4}$ | $85.4_{\pm 0.4}$ |
| $\text{DFR}^{\text{Val}}_{\text{Tr}}$ | ✓ | ✓ | $92.9_{\pm 0.2}$ | $94.2_{\pm 0.4}$ | $88.3_{\pm 1.1}$ | $89.6_{\pm 0.4}$ |
| Base Model | ✗ | ✓ | $6.9_{\pm 3.0}$ | $88.0_{\pm 1.1}$ | $39.8_{\pm 2.0}$ | $95.7_{\pm 0.1}$ |
| $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ | ✗ | ✓ | $45.4_{\pm 4.1}$ | $69.8_{\pm 7.0}$ | $83.4_{\pm 2.6}$ | $87.5_{\pm 0.3}$ |
| $\text{DFR}^{\text{Val}}_{\text{Tr}}$ | ✗ | ✓ | $53.9_{\pm 1.8}$ | $62.6_{\pm 2.2}$ | $85.0_{\pm 2.1}$ | $87.6_{\pm 0.3}$ |
| $\text{DFR}^{\text{Val}}_{\text{IN}}$ | ✓ | ✗ | $88.7_{\pm 0.4}$ | $89.7_{\pm 0.1}$ | $73.1_{\pm 2.6}$ | $80.9_{\pm 0.5}$ |

**Table A.27: Effect of ImageNet pretraining and dataset fine-tuning.** Results for DFR on the Waterbirds and CelebA datasets when using an ImageNet-trained model as a feature extractor, training the feature extractor from random initialization or initializing the feature extractor with ImageNet-trained weights and fine-tuning on the target data. On Waterbirds, we can achieve surprisingly strong worst group accuracy of 88.7% without finetuning on the target data. On CelebA, we can achieve reasonable worst group accuracy of 85% without pretraining. However, on both datasets, both ImageNet pretraining and dataset finetuning are needed to achieve optimal performance. All methods in Table 5.2 use ImageNet-trained models as initialization and finetune on the target dataset.

`num_labels=num_classes`). We train the BERT models with the AdamW [Loshchilov and Hutter 2017a] with linear learning rate annealing with initial learning rate $10^{-5}$, batch size 16 and weight decay $10^{-4}$ for 5 epochs.

DFR DETAILS. For DFR, we first extract and save the embeddings (inputs to the classification layer of the base model) of the training, validation and testing data using the base model. We then preprocess the embeddings to have zero mean and unit standard deviation using the standard scaler `sklearn.preprocessing.StandardScaler` from the `scikit-learn` package. In each case, we compute the preprocessing statistics on the reweighting data used to train the last layer. To retrain the last layer, we use the logistic regression implementation

| Finetuned | WB | CelebA |
|---|---|---|
| Last Layer | $93.1_{\pm 0.2}$ | $88.3_{\pm 0.5}$ |
| Last 2 Layers | $93.1_{\pm 0.5}$ | $87.2_{\pm 0.9}$ |
| Last Block | $90.7_{\pm 0.4}$ | $83.0_{\pm 0.9}$ |
| All Layers | $35.6_{\pm 0.1}$ | $77.8_{\pm 0.1}$ |

**Table A.28: Retraining multiple layers.** We retrain the last linear layer, last two layers (linear, last batchnorm and conv), last residual block, and all layers from scratch on group-balanced validation data. Retraining the last layer is sufficient for optimal performance, and moreover retraining more layers can hurt the results.

(`sklearn.linear_model.LogisticRegression`) from the `scikit-learn` package. We use $\ell_1$ regularization. For $\text{DFR}^{\text{Val}}_{\text{Tr}}$ we only tune the inverse regularization strength parameter $C$: we consider the values in range $\{1., 0.7, 0.3, 0.1, 0.07, 0.03, 0.01\}$ and select the value that leads to the best worst-group performance on the available validation data (as described in Section 5.5, for $\text{DFR}^{\text{Val}}_{\text{Tr}}$ we use half of the validation to train the logistic regression model and the other half to tune the parameters at the tuning stage). For $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ we additionally tune the class weights: we set the weight for one of the classes to 1 and consider the weights for the other class in range $\{1, 2, 3, 10, 100, 300, 1000\}$; we then switch the classes and repeat the procedure. For the final evaluation, we use the best values of the hyper-parameters obtained during the tuning phase and train a logistic regression model on all of the available reweighting data. We train the logistic regression model on the reweighting data 10 times with random subsets of the data (we take all of the data from the smallest group, and subsample the other groups randomly to have the same number of datapoints) and average the weights of the learned models. We report the model with averaged weights.

#### A.7.3.1 Ablation studies

Is ImageNet pretraining necessary for image benchmarks? DFR relies on the ability of the feature extractor to learn diverse features, which may suggest that ImageNet pretraining is crucial. In Appendix Table A.27, we report the results on the same problems, but training the feature extractor from scratch. We find that on Waterbirds, ImageNet pretraining indeed has a dramatic effect on the performance of DFR as well as the base feature extractor model. However, on CelebA DFR shows strong performance regardless of pretraining. The difference between Waterbirds and CelebA is that Waterbirds contains only $4.8k$ training points, making it difficult to learn a meaningful feature extractor from scratch. Furthermore, on both datasets, finetuning the feature extractor on the target data is crucial: just using the features extracted by an ImageNet-trained model leads to poor results. We note that all the baselines considered in Table 5.2 use ImageNet pretraining.

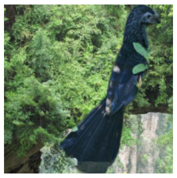
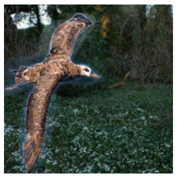
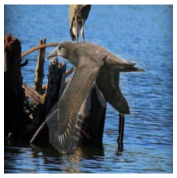
Retraining multiple layers. In Table A.28 we perform another ablation by retraining multiple layers from scratch on the group-balanced validation dataset on Waterbirds and CelebA. We find that retraining just the last layer provides optimal performance, retraining the last two layers (the last convolutional layer and the fully-connected classifier layer) or the last residual block is competitive (but worse), while retraining the full network is a lot worse.

| Base Model Hypers | | | | Method | Waterbirds | |
|---|---|---|---|---|---|---|
| lr | wd | batch size | aug | | Worst(%) | Mean(%) |
| $10^{-3}$ | $10^{-3}$ | 32 | ✓ | Base Model | $74.9_{\pm 2.4}$ | $98.1_{\pm 0.1}$ |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | $92.9_{\pm 0.2}$ | $94.2_{\pm 0.4}$ |
| $10^{-3}$ | $10^{-3}$ | 32 | ✗ | Base Model | 73.4 | 97.7 |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | 90.9 | 92.2 |
| $10^{-3}$ | $10^{-2}$ | 32 | ✓ | Base Model | 24.1 | 94.4 |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | 88.0 | 88.6 |
| $10^{-3}$ | $10^{-2}$ | 32 | ✗ | Base Model | 66.4 | 97.1 |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | 90.1 | 90.5 |
| $3 \cdot 10^{-3}$ | $10^{-3}$ | 32 | ✓ | Base Model | 71.5 | 98.1 |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | 91.9 | 93.5 |
| $3 \cdot 10^{-3}$ | $10^{-3}$ | 32 | ✗ | Base Model | 76.5 | 98.0 |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | 89.5 | 94.5 |
| $10^{-3}$ | $10^{-3}$ | 64 | ✓ | Base Model | 73.5 | 98.0 |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | 93.1 | 95.0 |
| $10^{-3}$ | $10^{-3}$ | 64 | ✗ | Base Model | 69.5 | 97.4 |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | 89.0 | 93.6 |

| Base Model Hypers | | | | Method | CelebA | |
|---|---|---|---|---|---|---|
| lr | wd | batch size | aug | | Worst(%) | Mean(%) |
| $10^{-3}$ | $10^{-4}$ | 128 | ✓ | Base Model | $46.9_{\pm 2.8}$ | $95.3_{\pm 0}$ |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | $88.3_{\pm 1.1}$ | $91.3_{\pm 0.3}$ |
| $10^{-3}$ | $10^{-3}$ | 128 | ✓ | Base Model | $44.3_{\pm 6.4}$ | $95.2_{\pm 0.1}$ |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | $86.2_{\pm 1.2}$ | $90.8_{\pm 0.7}$ |
| $10^{-3}$ | $10^{-4}$ | 128 | ✗ | Base Model | $46.7_{\pm 0.0}$ | $95.3_{\pm 0.1}$ |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | $86.9_{\pm 1.1}$ | $91.6_{\pm 0.2}$ |
| $10^{-3}$ | $10^{-3}$ | 128 | ✗ | Base Model | $40.6_{\pm 8.7}$ | $95.1_{\pm 0.2}$ |
| | | | | $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | $85.6_{\pm 1.4}$ | $91.8_{\pm 0.5}$ |

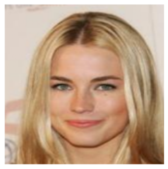
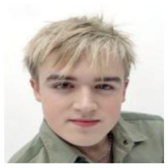
**Table A.29: Effect of base model hyper-parameters.** We report the results of $\text{DFR}_{\text{Tr}}^{\text{Val}}$ for a range of base model hyper-parameters on Waterbirds and CelebA as well as the performance of the corresponding base models. While the quality of the base model has an effect on DFR, the results are fairly robust. For a subset of configurations we report the mean and standard deviation over 3 independent runs of the base model and DFR.

**Waterbirds dataset**

| | $\mathcal{G}_1$ | $\mathcal{G}_2$ | $\mathcal{G}_3$ | $\mathcal{G}_4$ |
|---|---|---|---|---|
| **Image Examples** |  | |  |  |
| **Description** | landbird on land | landbird on water | waterbird on land | waterbird on water |
| **Class Label** | 0 | 0 | 1 | 1 |
| **# Train data** | 3498 (73%) | 184 (4%) | 56 (1%) | 1057 (22%) |
| **# Val data** | 467 | 466 | 133 | 133 |

**Target**: bird type; **Spurious feature**: background type; **Minority**: $\mathcal{G}_2, \mathcal{G}_3$

**CelebA hair color dataset**

| | $\mathcal{G}_1$ | $\mathcal{G}_2$ | $\mathcal{G}_3$ | $\mathcal{G}_4$ |
|---|---|---|---|---|
| **Image Examples** | | |  |  |
| **Description** | Non-blond woman | Non-blond man | Blond woman | Blond man |
| **Class Label** | 0 | 0 | 1 | 1 |
| **# Train data** | 71629 (44%) | 66874 (41%) | 22880 (14%) | 1387 (1%) |
| **# Val data** | 8535 | 8276 | 2874 | 182 |

**Target**: hair color; **Spurious feature**: gender; **Minority**: $\mathcal{G}_4$

**Figure A.44: Waterbirds and CelebA data.** Dataset descriptions and example images from each group on Waterbirds and CelebA datasets.

**MultiNLI**

| | Text examples | Class label | Description | # Train data | # Val data |
|---|---|---|---|---|---|
| $\mathcal{G}_1$ | *"if residents are unhappy, they can put wheels on their homes and go someplace else, she said. [SEP] residents are stuck here but they can't go anywhere else."* | 0 | contradiction no negations | 57498 (28%) | 22814 |
| $\mathcal{G}_2$ | *"within this conflict of values is a clash about art. [SEP] there is <u>no</u> clash about art."* | 0 | contradiction has negations | 11158 (5%) | 4634 |
| $\mathcal{G}_3$ | *"there was something like amusement in the old man's voice. [SEP] the old man showed amusement."* | 1 | entailment no negations | 67376 (32%) | 26949 |
| $\mathcal{G}_4$ | *"in 1988, the total cost for the postal service was about $36. [SEP] the postal service cost us citizens almost <u>nothing</u> in the late 80's. "* | 1 | entailment has negations | 1521 (1%) | 613 |
| $\mathcal{G}_5$ | *"yeah but even even cooking over an open fire is a little more fun isn't it [SEP] i like the flavour of the food."* | 2 | neutral no negations | 66630 (32%) | 26655 |
| $\mathcal{G}_6$ | *"that's not too bad [SEP] it's better than <u>nothing</u>"* | 2 | neutral has negations | 1992 (1%) | 797 |

**Target**: contradiction / entailment / neutral; **Spurious feature**: has negation words. **Minority**: $\mathcal{G}_4$, $\mathcal{G}_6$

**CivilComments**

| Text examples | Class label | Description | # Train data | # Val data |
|---|---|---|---|---|
| *"I'm quite surprised this worked for you. Infrared rays cannot penetrate tinfoil."* | 0 | non-toxic no identities | 148186 (55%) | 25159 |
| *"I think you may have misunderstood what 'straw <u>men</u>' are. But I'm glad that your gravy is good."* | 0 | non-toxic has identities | 90337 (33%) | 14966 |
| *"Hahahaha putting his faith in Snopes. Pathetic."* | 1 | toxic no identities | 12731 (5%) | 2111 |
| *"That sounds like something a <u>white person</u> would say."'* | 1 | toxic has identities | 17784 (7%) | 2944 |

**Target**: Toxic / not toxic comment; **Spurious feature**: mentions protected categories.

**Figure A.45: MultiNLI and CivilComments data.** Dataset descriptions and data examples from different groups on MultiNLI and CivilComments. We underline the words corresponding to the spurious feature. CivilComments contains 16 overlapping groups (corresponding to toxic / non-toxic comments and mentions of one of the protected identities: male, female, LGBT, black, white, Christian, Muslim, other religion. We only show examples with mentions of the male and white identities.

| Method | Group Info | Waterbirds | | CelebA | | MultiNLI | | CivilComments | |
|---|---|---|---|---|---|---|---|---|---|
| | Train / Val | Worst(%) | Mean(%) | Worst(%) | Mean(%) | Worst(%) | Mean(%) | Worst(%) | Mean(%) |
| Base (ERM) | ✗ / ✗ | $74.9_{\pm 2.4}$ | $98.1_{\pm 0.1}$ | $46.9_{\pm 2.8}$ | $95.3_{\pm 0}$ | $65.9_{\pm 0.3}$ | $82.8_{\pm 0.1}$ | $55.6_{\pm 0.6}$ | $92.1_{\pm 0.1}$ |
| $\text{DFR}_{\text{Tr}}^{\text{Tr}}$ | ✓/ ✓ | $90.2_{\pm 0.8}$ | $97.0_{\pm 0.3}$ | $80.7_{\pm 2.4}$ | $90.6_{\pm 0.7}$ | $71.5_{\pm 0.6}$ | $82.5_{\pm 0.2}$ | $58.0_{\pm 1.3}$ | $92.0_{\pm 0.1}$ |
| $\text{DFR}_{\text{Tr}}^{\text{Val}}$ | ✗ / ✓✓ | $\mathbf{92.9}_{\pm 0.2}$ | $94.2_{\pm 0.4}$ | $88.3_{\pm 1.1}$ | $91.3_{\pm 0.3}$ | $\mathbf{74.7}_{\pm 0.7}$ | $82.1_{\pm 0.2}$ | $\mathbf{70.1}_{\pm 0.8}$ | $87.2_{\pm 0.3}$ |
| Base Model trained without minority groups | | | | | | | | | |
| Base (ERM) | ✗ / ✗ | $31.9_{\pm 3.6}$ | $96.0_{\pm 0.2}$ | $21.7_{\pm 3.2}$ | $95.2_{\pm 0.1}$ | $66.0_{\pm 1.6}$ | $82.5_{\pm 0.1}$ | $8.4_{\pm 4.9}$ | $73.8_{\pm 0.3}$ |
| $\text{DFR}_{\text{Tr-NM}}^{\text{Tr}}$ | ✓/ ✓ | $89.9_{\pm 0.6}$ | $94.3_{\pm 0.5}$ | $\mathbf{89.0}_{\pm 1.1}$ | $91.6_{\pm 0.4}$ | $73.0_{\pm 0.6}$ | $82.2_{\pm 0.1}$ | $66.5_{\pm 0.7}$ | $85.9_{\pm 0.3}$ |

**Table A.30: DFR variations.** Worst-group and mean test accuracy of DFR variations on the benchmark datasets problems. $\text{DFR}_{\text{Tr}}^{\text{Val}}$ achieves the best performance across the board. Interestingly, $\text{DFR}_{\text{Tr-NM}}^{\text{Tr}}$ outperforms $\text{DFR}_{\text{Tr}}^{\text{Tr}}$ on all dataset and even outperforms $\text{DFR}_{\text{Tr}}^{\text{Val}}$ on CelebA. For all experiments, we report mean±std over 5 independent runs of the method.

What if we just use ImageNet features? As a baseline, we apply DFR to features extracted by a model pretrained on ImageNet with no fine-tuning on CelebA and Waterbirds data. We report the results in Table A.27 ($\text{DFR}^{\text{Tr}}_{\text{IN}}$ and $\text{DFR}^{\text{Val}}_{\text{IN}}$ lines). While on Waterbirds the performance is fairly good, on CelebA fine-tuning is needed to get reasonable performance. The results in Table A.27 suggest that both ImageNet pretraining and fine-tuning on the target data are needed to train the best feature extractor for DFR.

Robustness to base model hyper-parameters In Table A.29 we report the results of $\text{DFR}^{\text{Val}}_{\text{Tr}}$ for a range of configurations of the baseline model hyper-parameters. While the quality of the base model clearly has an effect on DFR performance, we achieve competitive results for all the hyper-parameter configurations that we consider, even when the base model performs poorly. For example, on Waterbirds with data augmentation, learning rate $10^{-3}$ and weight decay $10^{-2}$ the base model achieves worst group accuracy of 24.1%, but by retraining the last layer of this model with $\text{DFR}^{\text{Val}}_{\text{Tr}}$ we still achieve 88% worst group accuracy.

Ablation on the number of linear model retrains. We study the effect of the number of logistic regression retrains on different balanced subsets of the validation set in Table A.31 on Waterbirds and CelebA. We retrain the last linear layer multiple times and average the parameters of the resulting models, as described in Appendix A.7.3. In general, it is better to train more than 1 model and average their weights. This effect is especially prominent on CelebA, where a single model gets 85% worst group accuracy, while the average of 3 models gets 88.6%. As we increase the number of linear model retrains, the improvements saturate.

Ablation on the $\ell_1$ regularization. We emphasize that it is beneficial to use $\ell_1$ regularization in spurious correlations benchmarks where the number of last layer features is much higher than the reweighting dataset size (e.g. in Waterbirds we use approximately 500 examples for retraining the last layer in $\text{DFR}^{\text{Val}}_{\text{Tr}}$ while the dimensionality of the penultimate layer representations in

| | Number of retrains | | | | |
|---|---|---|---|---|---|
| | 1 | 3 | 5 | 10 | 20 |
| Waterbirds | $91.21_{\pm 1.82}$ | $92.88_{\pm 0.45}$ | $91.73_{\pm 1.25}$ | $93.13_{\pm 0.29}$ | $92.89_{\pm 0.19}$ |
| CelebA | $85.09_{\pm 1.49}$ | $88.64_{\pm 1.90}$ | $87.80_{\pm 1.17}$ | $88.02_{\pm 1.82}$ | $88.37_{\pm 2.02}$ |

**Table A.31: Ablation on the number of retrains in $\text{DFR}_{\text{Tr}}^{\text{Val}}$ .** Worst group accuracy of $\text{DFR}_{\text{Tr}}^{\text{Val}}$ varying the number of logistic regression retrains on Waterbirds and CelebA: we train the logistic regression model on the validation data several times with random subsets of the data (we take all of the data from the smallest group, and subsample the other groups randomly to have the same number of datapoints) and average the weights of the learned models. Averaging more than 1 linear model leads to improved performance.

ResNet-50 is 2048). Without $\ell_1$ regularization, $\text{DFR}_{\text{Tr}}^{\text{Val}}$ achieves $87.72 \pm 0.42\%$ worst group accuracy on Waterbirds and $86.03 \pm 0.42\%$ on CelebA, as opposed to $92.9 \pm 0.2\%$ and $88.3 \pm 1.1\%$ if we choose $\ell_1$ regularization strength through cross-validation as described in Appendix A.7.3.

Full model fine-tuning on validation. As an additional baseline, we finetune the full model (as opposed to just the last layer) on the group-balanced validation set for 10 epochs with SGD starting from the ResNet-50 checkpoint pre-trained on ImageNet and without training on the corresponding train splits of Waterbirds and CelebA. We achieve $89.3 \pm 1.3\%$ worst group accuracy on Waterbirds and $84.4 \pm 0.5\%$ on CelebA. While these results are good relative to ERM on the standard training set, they are still significantly worse than $\text{DFR}_{\text{Tr}}^{\text{Val}}$ (see Table 5.2).

### A.7.3.2 Prior work assumptions

In Section 5.5 we compare DFR to ERM, Group DRO [Sagawa et al. 2020], JTT [Liu et al. 2021], CnC [Zhang et al. 2022], SUBG [Idrissi et al. 2021] and SSA [Nam et al. 2022]. These methods differ in assumptions about the amount of group information available.

Group DRO and SUBG assume that both train and validation data have group labels, and the hyper-parameters are tuned using worst-group validation accuracy. $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ and $\text{DFR}^{\text{Tr}}_{\text{Tr-NM}}$ match the setting of these methods and use the same data and group information; in particular, these methods only use the validation set with group labels to tune the hyper-parameters.

A number of prior works [e.g. Liu et al. 2021; Creager et al. 2021; Zhang et al. 2022] sidestep the assumption of knowing the group labels on train data, but still rely on tuning hyper-parameters using worst-group accuracy on validation, and thus, having group labels on validation data. In fact, Idrissi et al. [2021] showed that ERM is a strong baseline when tuned with worst-group accuracy on validation.

Lee et al. [2022] explore the setting where they do not necessarily require group labels on validation data, but their method implicitly relies on the presence of sufficiently many minority examples in the validation set such that different prediction heads would disagree on those examples to choose the most reliable classifier.

Recent works Nam et al. [2022] and Sohoni et al. [2021] explore the setting where the group information is available on a small subset of the data (e.g. on the validation set), and the goal is to use the available data optimally, both to train the model and to tune the hyper-parameters. These methods use semi-supervised learning to extrapolate the available group labels to the rest of the training data. We consider this same setting with $\text{DFR}^{\text{Val}}_{\text{Tr}}$ , where we use the validation data to retrain the last layer of the model.

#### A.7.3.3 DFR variations

Here, we discuss two additional variations of $\text{DFR}^{\hat{\mathcal{D}}}_{\mathcal{D}}$. In $DFR^{Tr}_{Tr}$, we use a random group-balanced subset of the train data as $\hat{\mathcal{D}}$. In $DFR^{Tr}_{Tr\text{-}NM}$ (NM stands for "No Minority") we use a random group-balanced subset of the train data as $\hat{\mathcal{D}}$, but remove the minority groups ($\mathcal{G}_2, \mathcal{G}_3$ on Waterbirds and $\mathcal{G}_4$ on CelebA) from the data $\mathcal{D}$ used to train the feature extractor.

We compare different DFR versions in Table A.30. All three DFR variations obtain results competitive with state-of-the-art Group DRO results on the Waterbirds data. On CelebA, $\text{DFR}_{\text{Tr}}^{\text{Val}}$ and $\text{DFR}_{\text{Tr-NM}}^{\text{Tr}}$ match Group DRO, while $\text{DFR}_{\text{Tr}}^{\text{Tr}}$ performs slightly worse, but still on par with JTT. In particular, *$\text{DFR}_{\text{Tr-NM}}^{\text{Tr}}$ matches the state-of-the-art Group DRO by using the same data to train and tune the model.* Notably, $\text{DFR}_{\text{Tr-NM}}^{\text{Tr}}$ achieves these results with the features extracted by the network trained *without seeing any examples from the minority groups*! Even without minority groups, ERM models extract the core features sufficiently well to achieve state-of-the-art results on the image classification benchmarks.

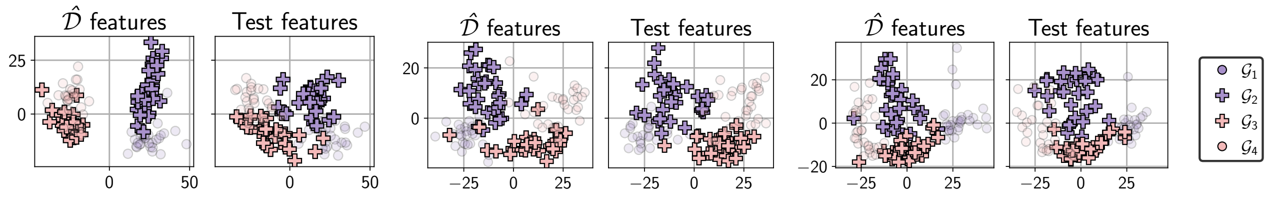
$\text{DFR}_{\text{Tr}}^{\text{Tr}}$ also significantly improves performance compared to the base model across the board, but underperforms compared to $\text{DFR}_{\text{Tr}}^{\text{Val}}$ : it is crucial to retrain the last layer on new data that was not used to train the feature extractor.

On the NLP datasets, $\text{DFR}_{\text{Tr}}^{\text{Val}}$ outperforms the other variations, but in all cases all DFR variations significantly improve performance compared to the base model. Notably, on CivilComments the no minority version group of the dataset only retains the toxic comments which mention the protected identities and non-toxic comments which do not mention these identities. As a result, the base model only achieves 8.4% worst group performance! $\text{DFR}_{\text{Tr-NM}}^{\text{Tr}}$ is still able to recover 66.5% worst group accuracy using the features extracted by this model.

#### A.7.3.4 Why is $\text{DFR}_{\text{Tr-NM}}^{\text{Tr}}$ better than $\text{DFR}_{\text{Tr}}^{\text{Tr}}$ ?

Let us consider the second stage of $\text{DFR}_{\mathcal{D}}^{\hat{\mathcal{D}}}$, where we fix the feature encoder $f(\cdot)$, and train a logistic regression model $\mathcal{L}$ on the dataset $f(\hat{\mathcal{D}})$, where by $f(\hat{\mathcal{D}})$ we denote the dataset with labels from the reweighting dataset $\hat{\mathcal{D}}$ and features from $\hat{\mathcal{D}}$ extracted by $f$. We then evaluate the logistic regression model $\mathcal{L}$ on the features extracted from the test data, $f(\mathcal{D}_{\text{Test}})$.

Let us use $\hat{\mathcal{M}}$ to denote a minority group in the reweighting dataset $\hat{\mathcal{D}}$, and $\mathcal{M}^{\text{Test}}$ to denote the same minority group in the test data. For $\text{DFR}_{\text{Tr-NM}}^{\text{Tr}}$ and $\text{DFR}_{\text{Tr}}^{\text{Val}}$, the model $f$ is trained without observing any data from $\hat{\mathcal{M}}$ or $\mathcal{M}^{\text{Test}}$. Assuming the datapoints in $\hat{\mathcal{M}}$ and $\mathcal{M}^{\text{Test}}$ are iid samples from the same distirbution, the distribution of features in $f(\hat{\mathcal{M}})$ and $f(\mathcal{M}^{\text{Test}})$ will also



(a) $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ (b) $\text{DFR}^{\text{Val}}_{\text{Tr}}$ (c) $\text{DFR}^{\text{Tr}}_{\text{Tr-NM}}$

**Figure A.46: DFR Variations.** Visualization of the features extracted from the reweighting dataset $\hat{D}$ and the test data for different variations of DFR on the Waterbirds data. We show projections of the 2048-dimensional features on the top-2 principal components extracted from $\hat{D}$. With $\text{DFR}^{\text{Val}}_{\text{Tr}}$ and $\text{DFR}^{\text{Tr}}_{\text{Tr-NM}}$, the distribution of the features for the minority groups $\mathcal{G}_2$ and $\mathcal{G}_3$ does not change between the reweighting and test data, while with $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ we see significant distribution shift.

be identical.

On the other hand, with $\text{DFR}^{\text{Tr}}_{\text{Tr}}$, the minority group datapoints $\hat{\mathcal{M}}$ are used to train the feature extractor $f$. In this case, we can no longer assume that the distribution of features $f(\hat{\mathcal{M}})$ will be the same as the distribution of $f(\mathcal{M}^{\text{Test}})$. Consequently, in $\text{DFR}^{\text{Tr}}_{\text{Tr}}$, the logistic regression model $\mathcal{L}$ will be evaluated under *distribution shift*, which makes the problem much more challenging and leads to inferior performance of $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ on the Waterbirds data. We verify this intuition in Figure A.46, where we visualize the feature embeddings for the reweighting dataset $\hat{D}$ and the test data. We see that as we predicted, the distribution of the minority group features coincides between $\hat{\mathcal{D}}$ and test data for $\text{DFR}^{\text{Tr}}_{\text{Tr-NM}}$ and $\text{DFR}^{\text{Val}}_{\text{Tr}}$, while $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ shows significant distribution shift.

What $\hat{\mathcal{D}}$ should be used in practice? In Table A.30, the best performance is achieved by $\text{DFR}^{\text{Tr}}_{\text{Tr-NM}}$ and $\text{DFR}^{\text{Val}}_{\text{Tr}}$, which retrain the last layer on data that was not used in training the feature extractor. In practice, we recommend collecting a group-balanced validation set, which can be used both to tune the hyper-parameters and re-train the last layer of the model, as we do in $\text{DFR}^{\text{Val}}_{\text{Tr}}$.

### A.7.4 Relation to transfer learning

Algorithmically, DFR is a special case of transfer learning, where $\mathcal{D}$ serves as the source data, and $\hat{\mathcal{D}}$ is the target data [Sharif Razavian et al. 2014]. However, the motivation of DFR is different from that of standard transfer learning: in DFR we are trying to correct the behavior of a pretrained model, and reduce the effect of spurious features, while in transfer learning the goal is to learn general features that generalize well to diverse downstream tasks. For example, we can use DFR to reduce the reliance of ImageNet-trained models on the background or texture and improve their robustness to covariate shift (see Section 5.6), while in standard transfer learning we would typically use a pretrained model as initialization or a feature extractor to learn a good solution on a *new dataset*. In the context of spurious correlations, one would not expect DFR to work as well as it does: the only reason DFR is successful is because, contrary to conventional wisdom, *neural network classifiers are in fact learning substantial information about core features, even when they seem to rely on spurious features to make predictions*. Moreover, in Appendix Table A.27 we show that transfer learning with features learned ImageNet does not work nearly as well as DFR on the spurious correlation benchmarks.

### A.7.5 Details: ImageNet Background Reliance

Here we provide details on the experiments on background reliance in Section 5.6.1 and additional evaluations verifying the main results.

Data. We use the ImageNet-9 dataset [Xiao et al. 2020]. To test whether our models can generalize to unusual backgrounds, we additionally generate *Paintings-BG* data shown in Figure A.49. For the Paintings-BG data we use the Original images and segmentation masks for ImageNet-9 data provided by Xiao et al. [2020], and combine them with random paintings from Kaggle's `painter-by-numbers` dataset available at kaggle.com/c/painter-by-numbers/ as backgrounds.

For the ImageNet-R dataset [Hendrycks et al. 2021], we only use the images that fall into one of the ImageNet-9 categories, and evaluate the accuracy with respect to these categories. We show examples of images from different dataset variations in Figure A.49.

Base model hyper-parameters. We use a ResNet-50 model pretrained on ImageNet and a VIT-B-16 model pretrained on ImageNet-21k and finetuned on ImageNet. The ResNet-50 model is imported from the `torchvision` package: `torchvision.models.resnet50(pretrained=True)`. The VIT-B-16 model is imported from the `lukemelas/PyTorch-Pretrained-ViT` package available at github.com/lukemelas/PyTorch-Pretrained-ViT; we use the command: `ViT('B_16_imagenet1k', pretrained=True)` to load the model. To extract the embeddings, we remove the last linear classification layer from each of the models. We preprocess the data using the `torchvision.transforms` package `Compose([ Resize(resize_size), CenterCrop(crop_size), ToTensor(), Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])`, where `(resize_size, crop_size)` are equal to (256, 224) for the ResNet-50 and (384, 384) for the VIT-B-16. We do not apply any data augmentation.

DFR details. We Train DFR on random subsets of the Mixed-Rand train data of different sizes ($\text{DFR}^{\text{MR}}$) or combinations of the Mixed-Rand and Original data ($\text{DFR}^{\text{OG+MR}}$). For $\text{DFR}^{\text{OG+MR}}$ we use the same number of Original and Mixed-Rand datapoints in all experiments. As the full ImageNet-9 contains $50k$ datapoints, we train the logistic regression on GPU with a simple implementation in PyTorch [Paszke et al. 2017]. We then preprocess the embeddings to have zero mean and unit standard deviation using the standard scaler `sklearn.preprocessing.StandardScaler` from the `scikit-learn` package. In each case, we compute the preprocessing statistics on the reweighting data used to train the last layer. For the experiments in this section we use $\ell_2$ regularization (as the number of datapoints is large relative to the number of observations, we do not have to use $\ell_1$). We set the regularization coefficient $\lambda$ to be 100 and train the logistic regression model with the loss $\sum_{x,y \in \mathcal{D}'} L(y, wx + b) + \frac{\lambda}{2}\|w\|^2$, where $L(\cdot, \cdot)$ is the cross-entropy loss. We use

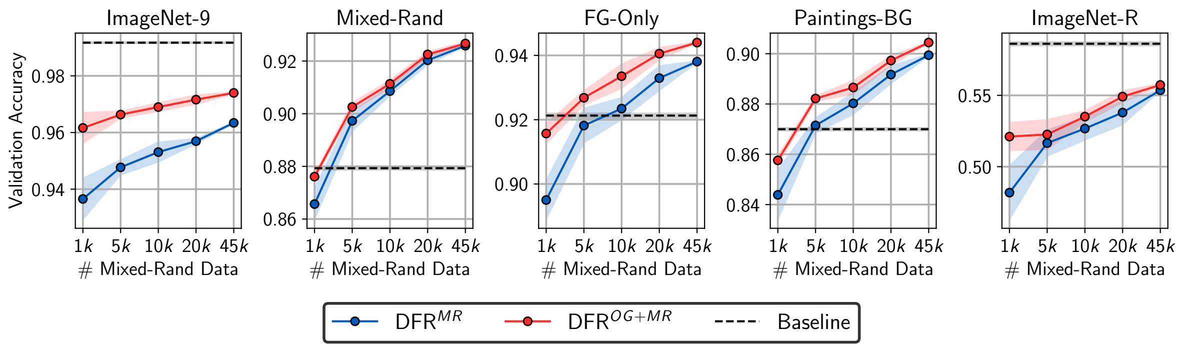


**Figure A.47: ImageNet background reliance (VIT-B-16).** Performance of DFR trained on MixedRand data and MixedRand + Original data on different ImageNet-9 validation splits. All methods use an VIT-B-16 feature extractor trained on ImageNet21k and finetuned on ImageNet. DFR can reduce background reliance with a minimal drop in performance on the Original data. See Figure 5.3 for analogous results with a ResNet-50 feature extractor.

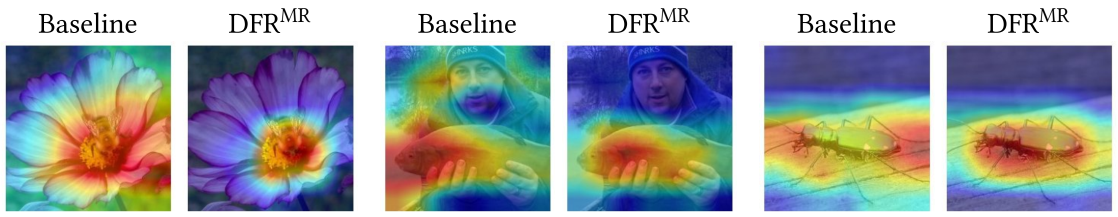


**Figure A.48:** GradCAM visualizations of the features used by the baseline model and DFR$^{MR}$ on ImageNet-9.

full-batch SGD with learning rate 1 and no momentum to train the model for 1000 epochs. We did not tune the $\lambda$ parameter or SGD hyper-parameters.

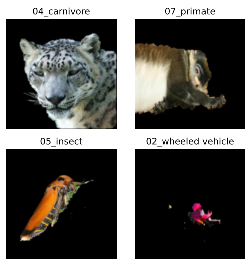
Results for VIT-B-16. We report the results for the VIT-B-16 base model in Figure A.47. The results are generally analogous to the results for ResNet-50: it is possible to significantly reduce the reliance of the model on the background by retraining the last layer with DFR. For the VIT, removing the background dependence hurts the performance on the Original data slightly, but greatly improves the performance on the images with unusual backgrounds (Mixed-Rand, FG-Only, Paintings-BG). Removing the background dependence does not improve the performance on ImageNet-R.

BG modification. For each test image we generate images with 7 modified backgrounds: MixRand (random Only-BG-T ImageNet-9 background, [1]), Paintings-BG, and constant black, white, red, green and blue backgrounds. Then, for each model from Section 5.6.1 we compute the percentage of the datapoints, on which changing the background with a fixed foreground does not change the predictions compared to the original image. We get 87.5% for the baseline model, 92.4% for DFR$^{\text{MR}}$and 93.1% for DFR$^{\text{OG+MR}}$, suggesting that the DFR$^{\text{MR}}$ and DFR$^{\text{OG+MR}}$ models are significantly more robust to modifying the background.

Prediction based on BG. Next, for each model we evaluate the percentage of test MixRand datapoints on which the model makes predictions that match the background class and not the foreground class. We get 14.8% for the baseline model, 11.2% for DFR$^{\text{MR}}$ and 11.7% for DFR$^{\text{OG+MR}}$. The baseline model predicts the background class significantly more frequently than the DFR models.

GradCAM. Finally, to gain a visual intuition into the features used by DFR$^{\text{MR}}$and the baseline models, in Figure A.48 we make GradCAM feature visualizations on three images from ImageNet-9. We use the `pytorch-grad-cam` package [Gildenblat and contributors 2021] available at github.com/jacobgil/pytorch-grad-cam. While the baseline model uses the background context, the DFR$^{\text{MR}}$ features are more compact and focus on the target object, again suggesting that DFR reduces the reliance on the background information.

(a) IN-9 Original [Xiao et al. 2020]

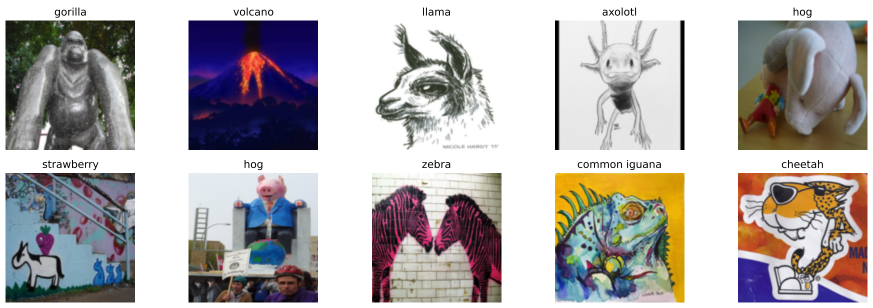


(b) IN-9 FG-Only [Xiao et al. 2020]

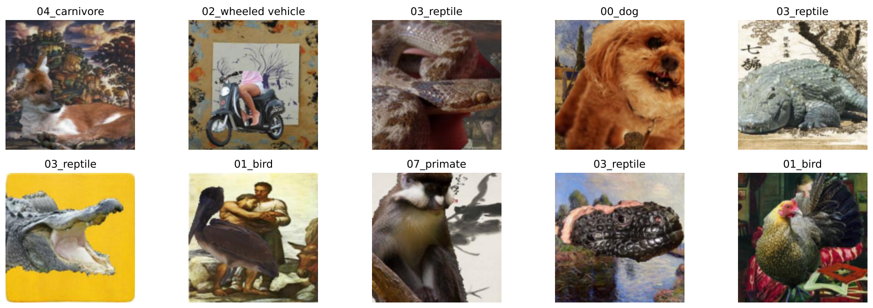
(c) IN-9 Mixed-Rand [Xiao et al. 2020]

marmoset
jay
bottlecap
red wine
water snake
jellyfish
greenhouse
guillotine
water jug
puffer

(d) Stylized ImageNet [Geirhos et al. 2018]



(e) ImageNet-R [Hendrycks et al. 2021]



(f) ImageNet-9, Paintings-BG (ours)

**Figure A.49: ImageNet variations.** Examples of datapoints from the ImageNet variations used in the experiments.

### A.7.6 Details: ImageNet Texture Bias Details

Here we provide details on the experiments on texture bias in Section 5.6.

Data. We generate the stylized ImageNet (SIN) data following the instructions at github.com/rgeirhos/Stylized-ImageNet [Geirhos et al. 2018]. For evaluation, we use ImageNet-C [Hendrycks and Dietterich 2019] and ImageNet-R datasets [Hendrycks et al. 2021]. For ImageNet-C we report the average performance across all 19 corruption types and 5 corruption intensities. We show examples of stylized ImageNet images in Figure A.49.

Base model hyper-parameters. We use the same ResNet-50 and VIT-B-16 models as described in Appendix A.7.5.

DFR details. We train DFR using the embeddings of the original ImageNet (IN), stylized ImageNet (SIN) and their combination (IN+SIN) as the reweighting dataset. We preprocess the base model embeddings by manually subtracting the mean and dividing by standard deviation computed on the reweighting data used to train DFR; we did not use `sklearn.preprocessing.StandardScaler` due to the large size of the datasets ($1.2M$ datapoints for IN and SIN; $2.4M$ datapoints for IN+SIN). We train the logistic regression for the last layer on a single GPU with a simple implementation in PyTorch. We use SGD with learning rate 1, no momentum, no regularization and batch size of $10^4$ to train the model for 100 epochs. We tuned the batch size (in the range $\{10^3, 10^4, 10^5\}$) and picked the batch size that leads to the best performance on the ImageNet validation set ($10^4$). We did not tune the other hyper-parameters.

Results for VIT-B-16. We report the results for the VIT-B-16 base model in Table A.32. For this model, baselines trained from scratch on IN+SIN and SIN are not available so we only report the results for the standard model trained on ImageNet21k and finetuned on ImageNet; for DFR we report the results using IN+SIN as the reweighting dataset. Despite the large-scale pretraining on ImageNet21k, we find that we can still improve the shape bias (36% $\longrightarrow$ 39.9%) as well as robustness to ImageNet-C corruptions (49.7% $\longrightarrow$ 52% Top-1 accuracy). On the original ImageNet and ImageNet-R the performance of DFR is similar to that of the baseline model.

paragraphDetailed texture bias evaluation. To provide further insight into the texture bias of the models trained with DFR, in Figure A.50 we report the fraction of shape and texture decisions for different classes following Geirhos et al. [2018]. We produce the figure using the `modelvshuman` codebase available at github.com/bethgelab/model-vs-human [Geirhos et al. 2021]. We report the results for both DFR models and models trained from scratch on IN, SIN and IN+SIN as well as the ShapeResNet-50 model and humans (results from Geirhos et al. [2018]). When trained on the same data, models trained from scratch achieve a higher shape bias than DFR models, but DFR can still significantly improve the shape bias compared to the base model trained on IN.

paragraphDetailed ImageNet-C results. In Figure A.51, we report the Top-1 accuracy for DFR models and models trained from scratch on IN, SIN and IN+SIN on individual ImageNet-C datasets. We use the ResNet-50 base model. The model trained from scratch on IN+SIN provides the best robustness across the board, but DFR trained on IN+SIN also provides an improvement over the baseline RN50(IN) model on many corruptions.

| **Method** | **Training Data** | **Shape bias (%)** | **Top-1 Acc (%) / Top-5 Acc (%)** | | |
|---|---|---|---|---|---|
| | | | ImageNet | ImageNet-R | ImageNet-C |
| VIT-B-16 | IN21k + IN | 36 | 79.2/95.0 | 29.1/42.0 | 49.7/69.3 |
| DFR | IN+SIN | 39.9 | 79.7/94.5 | 29.0/41.4 | 52.0/71.0 |

**Table A.32: Texture-vs-shape bias results for VIT-B-16.** Shape bias, top-1 and top-5 accuracy on ImageNet validation set variations for VIT-B-16 pretrained on ImageNet21k and finetuned on ImageNet and DFR using this model as a feature extractor and reweighting the features on combined ImageNet and Stylized ImageNet datasets. By retraining just the last layer with DFR, we can increase the shape bias compared to the feature extractor model and improve robustness to covariate shift on ImageNet-C.

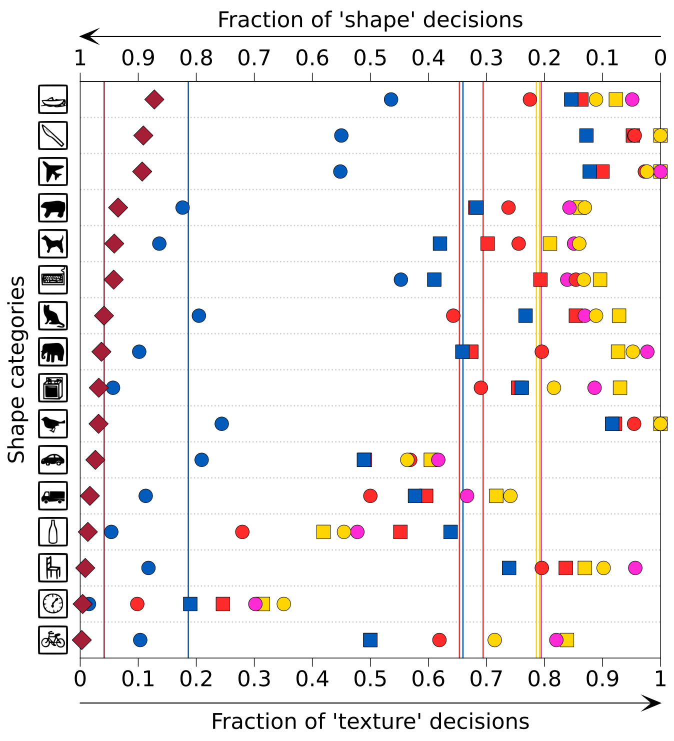


**Figure A.50: Shape-texture bias report.** Detailed report of the shape-texture bias generated using the `model-vs-human` codebase (`https://github.com/bethgelab/model-vs-human`) [Geirhos et al. 2021]. The plot shows the fraction of the decisions made based on shape and texture information respectively on examples with conflicting cues [Geirhos et al. 2018]. The brown diamonds (⋄) show human predictions and the circles (∘) show the performance of ResNet-50 models trained on different datasets: ImageNet (IN, yellow), Stylized ImageNet (SIN, blue), ImageNet + Stylized ImageNet (IN+SIN, red), ImageNet + Stylized ImageNet Finetuned on ImageNet (IN+SIN→IN, pink); these models are provided in the `model-vs-human` codebase. For each dataset (except for IN+SIN→IN) we report the results for DFR using an ImageNet-trained ResNet-50 model as a feature extractor with squares □ of the corresponding colors. Reweighting the features in a pretrained model with DFR we can significatly increase the shape bias: DFR trained on SIN (blue squares) virtually matches the shape bias of the model trained from scratch on IN+SIN (red circles). However, the model trained just on SIN (blue circles) from scratch still provides a significantly higher shape bias, that we cannot match with DFR.

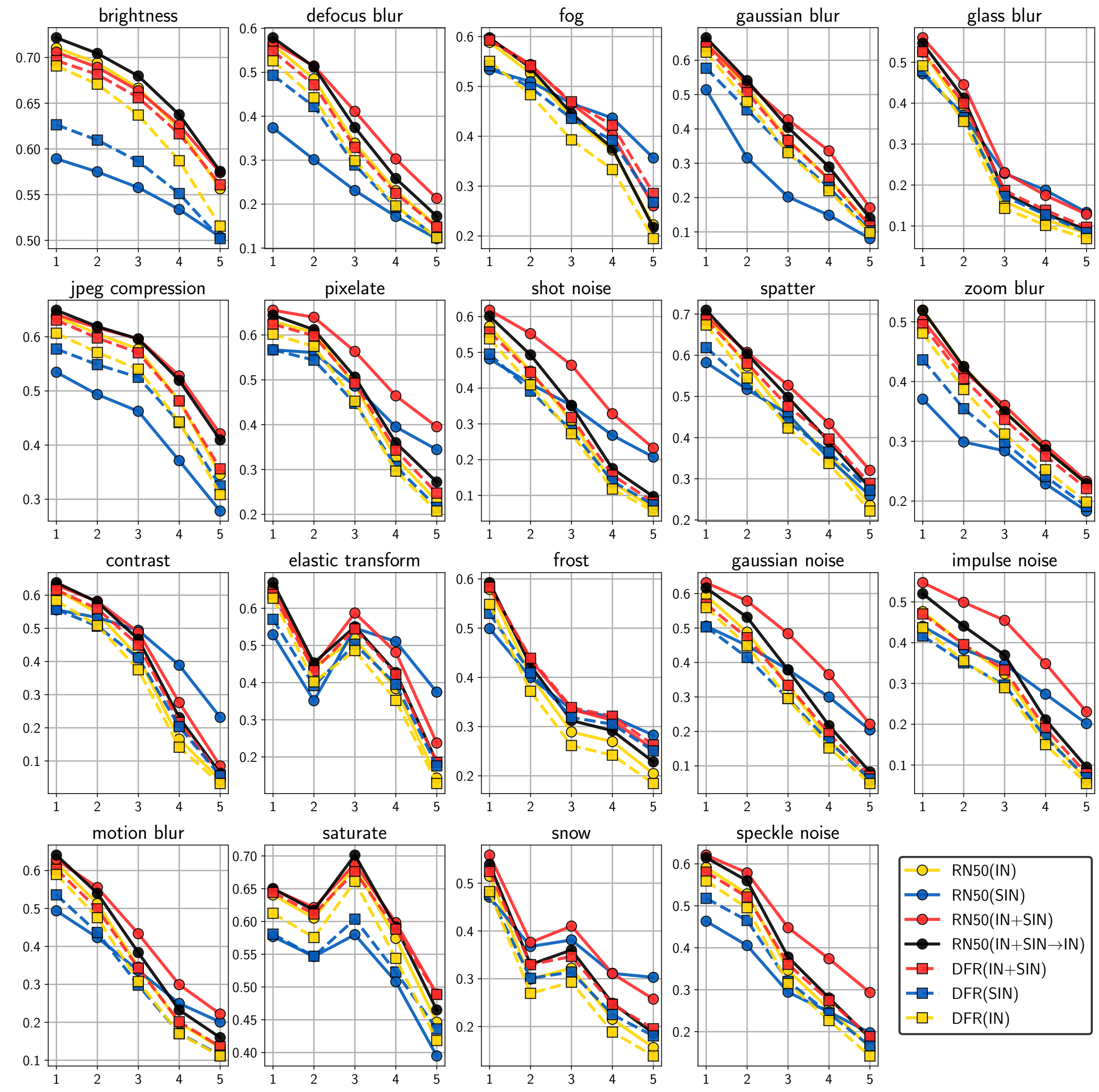


**Figure A.51: ImageNet-C results.** Results of ResNet-50 models trained on different ImageNet variations (shown in circles) and DFR using an ImageNet-trained ResNet-50 model as a feature extractor on ImageNet-C and ImageNet-R datasets. Each panel corresponds to a different corruption, and the horizontal axis represents the corruption intensity. Retraining the last layer on ImageNet-Stylized (DFR(SIN), red squares) improves robustness to ImageNet-C corruptions compared to the base model (RN50(IN), blue circles), but does not match the robustness of a model trained from scratch on SIN or IN+SIN.

### A.7.7 Comparison to Kang et al. [2019]

We compare $\text{DFR}^{\text{Val}}_{\text{Tr}}$ and $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ to LWS and cRT methods proposed in Kang et al. [2019], and perform other related ablations in Table A.33. We discuss the LWS and cRT methods in detail as well as their algorithmic differences with DFR in Section 5.7 and Appendix A.7.1. We adapt the original implementation for LWS and cRT[8].

In addition to LWS and cRT, we evaluate last layer re-training on the training and validation data with group-balanced data sampling. These methods serve as intermediate variations between DFR and cRT, as they use balanced sampling instead of subsampling compared to DFR, but use group information instead of class information compared to cRT. For DFR, we report the performance of $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ and $\text{DFR}^{\text{Val}}_{\text{Tr}}$ .

We train LWS, cRT and last layer re-training variations for 10 epochs with SGD, using cosine learning rate schedule decaying from 0.2 to 0; DFR implementation details can be found in Appendix A.7.3.

Since the original LWS and cRT methods were proposed to address the class imbalance problem, they perform poorly in terms of the worst group accuracy in the spurious correlation setting. LWS performs especially poorly, as it only retrains a single scaling parameter per class, which is not sufficient to remove the reliance on spurious features. Re-training the last layer with balanced data sampling with respect to the group labels does improve performance compared to these original methods from Kang et al. [2019] as well as ERM, but underperforms compared to both DFR versions. This ablation highlights the importance of subsampling compared to balanced sampling [see also Idrissi et al. 2021][9].

$\text{DFR}^{\text{Val}}_{\text{Tr}}$ achieves the best performance across the board. To sum up, for optimal performance, it is important to use held-out data (as opposed to re-using the train data) and to perform data

[8] https://github.com/facebookresearch/classifier-balancing

[9] The difference is less pronounced on Waterbirds with retraining on the validation set because the validation set is relatively group-balanced. In CelebA the group distribution is the same in validation and training sets, so subsampling performs much better than group-balanced sampling.

| Method | Class or group balancing | Data | Balancing | Waterbirds WGA | CelebA WGA |
|---|---|---|---|---|---|
| LWS [Kang et al. 2019] | Class | Train | Balanced sampling | $40.03_{\pm 8.07}$ | $35.55_{\pm 14.4}$ |
| cRT [Kang et al. 2019] | Class | Train | Balanced sampling | $74.48_{\pm 1.5}$ | $52.88_{\pm 5.96}$ |
| Re-training last layer | Group | Train | Balanced sampling | $76.48_{\pm 1.24}$ | $56.11_{\pm 4.77}$ |
| Re-training last layer | Group | Validation | Balanced sampling | $89.21_{\pm 1.04}$ | $67.66_{\pm 2.14}$ |
| $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ | Group | Train | Subsampling | $90.2_{\pm 0.8}$ | $80.7_{\pm 2.4}$ |
| $\text{DFR}^{\text{Val}}_{\text{Tr}}$ | Group | Validation | Subsampling | $\mathbf{92.9_{\pm 0.2}}$ | $\mathbf{88.3_{\pm 1.1}}$ |

**Table A.33: Comparison to LWS and cRT from Kang et al. [2019] and ablations.** We compare $\text{DFR}^{\text{Val}}_{\text{Tr}}$ and $\text{DFR}^{\text{Tr}}_{\text{Tr}}$ to LWS and cRT proposed in Kang et al. [2019] for long-tail classification, as well as variations of last layer retraining on Waterbirds and CelebA dataset. The methods differ in (1) the data (train or validation) used for retraining last layer or logits scales in LWS, (2) whether class or group labels are used for balancing the dataset, and (3) the type of balancing which is either subsampling the dataset or using a balanced dataloader which first selects the class or group label uniformly at random and then samples an example from that class or group. Notice the significant improvement that we gain in terms of WGA as we change the data from train to validation, and as we change the balancing from balanced sampling to subsampling. $\text{DFR}^{\text{Val}}_{\text{Tr}}$ performs best compared to other methods.

subsampling according to group labels (as opposed to using group-balanced data sampling).

# Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*. (Cited on 166)

Abnar, S., Dehghani, M., Neyshabur, B., and Sedghi, H. (2021). Exploring the limits of large scale pre-training. *arXiv preprint arXiv:2110.02095*. (Cited on 158)

Achille, A. and Soatto, S. (2018). Emergence of invariance and disentanglement in deep representations. *The Journal of Machine Learning Research*, 19(1):1947–1980. (Cited on 81)

Agarwal, A., Beygelzimer, A., Dudík, M., Langford, J., and Wallach, H. (2018). A reductions approach to fair classification. In *International Conference on Machine Learning*, pages 60–69. PMLR. (Cited on 158)

Ahn, S., Korattikara, A., and Welling, M. (2012). Bayesian posterior sampling via stochastic gradient fisher scoring. *arXiv preprint arXiv:1206.6380*. (Cited on 133)

Ahn, S., Shahbaba, B., and Welling, M. (2014). Distributed stochastic gradient mcmc. In *International conference on machine learning*, pages 1044–1052. PMLR. (Cited on 133)

Ainsworth, S. K., Hayase, J., and Srinivasa, S. (2022). Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836*. (Cited on 33)

Aitchison, L. (2020). A statistical theory of cold posteriors in deep neural networks. *arXiv preprint arXiv:2008.05912*. (Cited on 122)

Al-Shedivat, M., Wilson, A. G., Saatchi, Y., Hu, Z., and Xing, E. P. (2017). Learning scalable deep kernels with recurrent structure. *The Journal of Machine Learning Research*, 18(1):2850–2886. (Cited on 107)

Alcorn, M. A., Li, Q., Gong, Z., Wang, C., Mai, L., Ku, W.-S., and Nguyen, A. (2019). Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4845–4854. (Cited on 157)

Alquier, P. (2021). User-friendly introduction to pac-bayes bounds. *arXiv preprint arXiv:2110.11216*. (Cited on 81)

Arjovsky, M. (2021). Out of distribution generalization in machine learning. *arXiv preprint arXiv:2103.02667*. (Cited on 107, 109, 135)

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. (2019). Invariant risk minimization. *arXiv preprint arXiv:1907.02893*. (Cited on 158, 159)

Ashukha, A., Lyzhov, A., Molchanov, D., and Vetrov, D. (2020). Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. *arXiv preprint arXiv:2002.06470*. (Cited on 45, 122)

Asmussen, S. and Glynn, P. W. (2007). *Stochastic simulation: algorithms and analysis*, volume 57. Springer. (Cited on 180)

Athiwaratkun, B., Finzi, M., Izmailov, P., and Wilson, A. G. (2019). There are many consistent explanations of unlabeled data: Why you should average. *International Conference on Learning Representations*. (Cited on 35, 134)

Aubin, B., Słowik, A., Arjovsky, M., Bottou, L., and Lopez-Paz, D. (2021). Linear unit-tests for invariance discovery. *arXiv preprint arXiv:2102.10867*. (Cited on 158)

Auer, P., Herbster, M., and Warmuth, M. K. (1996). Exponentially many local minima for single neurons. In *Advances in Neural Information Processing Systems*, pages 316–322. (Cited on 5)

Babichev, D. and Bach, F. (2018). Constant step size stochastic gradient descent for probabilistic modeling. *arXiv preprint arXiv:1804.05567*. (Cited on 180)

Bahng, H., Chun, S., Yun, S., Choo, J., and Oh, S. J. (2020). Learning de-biased representations with biased representations. In *International Conference on Machine Learning*, pages 528–539. PMLR. (Cited on 159)

Barron, A. R. and Cover, T. M. (1991). Minimum complexity density estimation. *IEEE transactions on information theory*, 37(4):1034–1054. (Cited on 71, 72, 83)

Beal, M. J. (2003). *Variational algorithms for approximate Bayesian inference*. university of London. (Cited on 44)

Beery, S., Van Horn, G., and Perona, P. (2018). Recognition in terra incognita. In *Proceedings of the European conference on computer vision (ECCV)*, pages 456–473. (Cited on 139)

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854. (Cited on 68, 83)

Ben-Tal, A., Den Hertog, D., De Waegenaere, A., Melenberg, B., and Rennen, G. (2013). Robust solutions of optimization problems affected by uncertain probabilities. *Management Science*, 59(2):341–357. (Cited on 156)

Benton, G. W., Maddox, W. J., Lotfi, S., and Wilson, A. G. (2021). Loss surface simplexes for mode connecting volumes and fast ensembling. *arXiv preprint arXiv:2102.13042*. (Cited on 33, 107, 134)

Berger, J. O. (2013). *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media. (Cited on 182)

Betancourt, M. (2015). The fundamental incompatibility of hamiltonian monte carlo and data subsampling. *arXiv preprint arXiv:1502.01510*. (Cited on 133)

Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 4. Springer. (Cited on 74, 121)

Bissiri, P. G., Holmes, C. C., and Walker, S. G. (2016). A general framework for updating belief distributions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78(5):1103–1130. (Cited on 71, 72, 83)

Blanchard, G., Lee, G., and Scott, C. (2011). Generalizing from several related classification tasks to a new unlabeled sample. *Advances in neural information processing systems*, 24. (Cited on 158)

Blier, L. and Ollivier, Y. (2018). The Description Length of Deep Learning models. In *Advances in Neural Information Processing Systems*, page 11. (Cited on 84)

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*. (Cited on 46, 48, 80, 84, 88, 119, 132, 259)

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*. (Cited on 107)

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*. (Cited on 158)

Borkan, D., Dixon, L., Sorensen, J., Thain, N., and Vasserman, L. (2019). Nuanced Metrics For Measuring Unintended Bias With Real Data For Text Classification. *Companion proceedings of the 2019 world wide web conference*, pages 491–500. (Cited on 148)

Box, G. E. and Tiao, G. C. (1973). Bayesian inference in statistical analysis, addision-wesley. *Reading, MA*. (Cited on 79)

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs. (Cited on 92)

Brendel, W. and Bethge, M. (2019). Approximating CNNs With Bag-of-local-Features Models Works Surprisingly Well On ImageNet. *arXiv preprint arXiv:1904.00760*. (Cited on 139)

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., et al. (2023). Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*. (Cited on 1)

Bui, T., Hernández-Lobato, D., Hernandez-Lobato, J., Li, Y., and Turner, R. (2016). Deep gaussian processes for regression using approximate expectation propagation. In *International Conference on Machine Learning*, pages 1472–1481. (Cited on xli, 58, 197, 198)

Cadene, R., Dancette, C., Cord, M., Parikh, D., et al. (2019). Rubi: Reducing unimodal biases for visual question answering. *Advances in neural information processing systems*, 32. (Cited on 157)

Cao, K., Chen, Y., Lu, J., Arechiga, N., Gaidon, A., and Ma, T. (2020). Heteroskedastic and imbalanced deep learning with adaptive regularization. *arXiv preprint arXiv:2006.15766*. (Cited on 157)

Cao, K., Wei, C., Gaidon, A., Arechiga, N., and Ma, T. (2019). Learning imbalanced datasets with label-distribution-aware margin loss. *Advances in neural information processing systems*, 32. (Cited on 157)

Carbone, G., Wicker, M., Laurenti, L., Patane, A., Bortolussi, L., and Sanguinetti, G. (2020). Robustness of bayesian neural networks to gradient-based attacks. *arXiv preprint arXiv:2002.04359*. (Cited on 134)

Carvalho, C. M., Polson, N. G., and Scott, J. G. (2009). Handling sparsity via the horseshoe. In *Artificial Intelligence and Statistics*, pages 73–80. PMLR. (Cited on 135)

Cha, J. et al. (2021). Swad: Domain generalization by seeking flat minima. *NeurIPS*. (Cited on 35)

Chang, O., Yao, Y., Williams-King, D., and Lipson, H. (2019). Ensemble model patching: A parameter-efficient variational bayesian neural network. *arXiv preprint arXiv:1905.09453*. (Cited on 107)

Chaudhari, P., Choromanska, A., Soatto, S., and LeCun, Y. (2016). Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*. (Cited on 22, 80)

Chaudhari, P. and Soatto, S. (2018). Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. In *International Conference on Learning Representations*. arXiv: 1710.11029. (Cited on 184)

Chen, T., Fox, E., and Guestrin, C. (2014). Stochastic gradient Hamiltonian Monte Carlo. In *International Conference on Machine Learning (ICML)*. (Cited on 46, 48, 83, 89, 130, 133)

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR. (Cited on 160)

Chen, X., Lee, J. D., Tong, X. T., and Zhang, Y. (2016). Statistical Inference for Model Parameters in Stochastic Gradient Descent. arXiv: 1610.08637. (Cited on 49, 85, 180, 181, 182)

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015). The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204. (Cited on 5, 27, 31)

Coates, A., Ng, A., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings. (Cited on 55, 113)

Cobb, A. D., Baydin, A. G., Markham, A., and Roberts, S. J. (2019). Introducing an explicit symplectic integration scheme for riemannian manifold hamiltonian monte carlo. *arXiv preprint arXiv:1910.06243*. (Cited on 204)

Cobb, A. D. and Jalaian, B. (2020). Scaling hamiltonian monte carlo inference for bayesian neural networks with symmetric splitting. *arXiv preprint arXiv:2010.06772*. (Cited on 133)

Cranmer, M., Tamayo, D., Rein, H., Battaglia, P., Hadden, S., Armitage, P., Ho, S., and Spergel, D. N. (2021). A bayesian neural network predicts the dissolution of compact planetary systems. (Cited on 88)

Creager, E., Jacobsen, J.-H., and Zemel, R. (2021). Environment inference for invariant learning. In *International Conference on Machine Learning*, pages 2189–2200. PMLR. (Cited on 156, 160, 293)

Cui, T., Havulinna, A., Marttinen, P., and Kaski, S. (2020). Informative gaussian scale mixture priors for bayesian neural networks. *arXiv e-prints*, pages arXiv–2002. (Cited on 135)

Dagaev, N., Roads, B. D., Luo, X., Barry, D. N., Patil, K. R., and Love, B. C. (2021). A too-good-to-be-true prior to reduce shortcut reliance. *arXiv preprint arXiv:2102.06406*. (Cited on 156)

Darnieder, W. F. (2011). *Bayesian methods for data-dependent priors*. PhD thesis, The Ohio State University. (Cited on 74)

Dasgupta, S. and Hsu, D. (2007). On-line estimation with the multivariate gaussian distribution. In *Learning Theory: 20th Annual Conference on Learning Theory, COLT 2007, San Diego, CA, USA; June 13-15, 2007. Proceedings 20*, pages 278–292. Springer. (Cited on 49)

Daumé III, H. (2009). Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*. (Cited on 134)

Daume III, H. and Marcu, D. (2006). Domain adaptation for statistical classifiers. *Journal of artificial Intelligence research*, 26:101–126. (Cited on 134)

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, pages 2933–2941. (Cited on 5)

Daxberger, E., Nalisnick, E., Allingham, J. U., Antorán, J., and Hernández-Lobato, J. M. (2020). Expressive yet tractable bayesian deep learning via subnetwork inference. *arXiv preprint arXiv:2010.14689*. (Cited on 88, 132, 241)

de Heide, R., Kirichenko, A., Mehta, N., and Grünwald, P. (2019). Safe-Bayesian generalized linear regression. *arXiv preprint arXiv:1910.09227*. (Cited on 71, 72, 73, 83)

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training Of Deep Bidirectional Transformers For Language Understanding. *arXiv preprint arXiv:1810.04805*. (Cited on 149)

Ding, N., Fang, Y., Babbush, R., Chen, C., Skeel, R., and Neven, H. (2014). Bayesian sampling using stochastic gradient thermostats. (Cited on 133)

Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. (2017a). Sharp minima can generalize for deep nets. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1019–1028, International Convention Centre, Sydney, Australia. PMLR. (Cited on 31)

Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. (2017b). Sharp minima can generalize for deep nets. *arXiv preprint arXiv:1703.04933*. (Cited on 43, 80)

Domingos, P. (2000). Bayesian averaging of classifiers and the overfitting problem. In *ICML*, volume 747, pages 223–230. Citeseer. (Cited on 135)

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*. (Cited on 152)

Draxler, F., Veschgini, K., Salmhofer, M., and Hamprecht, F. (2018a). Essentially no barriers in neural network energy landscape. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1309–1318, Stockholmsmässan, Stockholm Sweden. PMLR. (Cited on 32, 186)

Draxler, F., Veschgini, K., Salmhofer, M., and Hamprecht, F. A. (2018b). Essentially no barriers in neural network energy landscape. *arXiv preprint arXiv:1803.00885*. (Cited on 99, 201)

Dusenberry, M., Jerfel, G., Wen, Y., Ma, Y., Snoek, J., Heller, K., Lakshminarayanan, B., and Tran, D. (2020). Efficient and scalable bayesian neural nets with rank-1 factors. In *International conference on machine learning*, pages 2782–2792. PMLR. (Cited on 105, 107, 132, 134, 233, 241, 242)

Dwork, C., Hardt, M., Pitassi, T., Reingold, O., and Zemel, R. (2012). Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pages 214–226. (Cited on 158)

Dziugaite, G. K. and Roy, D. M. (2017). Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*. (Cited on 43, 67, 81)

Eisenstein, J. (2022). Informativeness and invariance: Two perspectives on spurious correlations in natural language. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4326–4331. (Cited on 158)

Elfwing, S., Uchibe, E., and Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11. (Cited on 93)

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639):115–118. (Cited on 107)

Farquhar, S., Smith, L., and Gal, Y. (2020). Liberty or depth: Deep bayesian neural nets do not need complex weight posterior approximations. *arXiv preprint arXiv:2002.03704*. (Cited on 132)

Filos, A., Farquhar, S., Gomez, A. N., Rudner, T. G., Kenton, Z., Smith, L., Alizadeh, M., de Kroon, A., and Gal, Y. (2019). A systematic comparison of bayesian deep learning robustness in diabetic retinopathy tasks. *arXiv preprint arXiv:1912.10481*. (Cited on 88, 107)

Foong, A. Y., Burt, D. R., Li, Y., and Turner, R. E. (2019). On the expressiveness of approximate inference in bayesian neural networks. *arXiv preprint arXiv:1909.00719*. (Cited on 132)

Fort, S., Hu, H., and Lakshminarayanan, B. (2019). Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*. (Cited on 82, 88)

Fort, S. and Jastrzebski, S. (2019). Large scale structure of neural network loss landscapes. In *Advances in Neural Information Processing Systems*, pages 6706–6714. (Cited on 33, 201)

Fortuin, V. (2021). Priors in bayesian deep learning: A review. *arXiv preprint arXiv:2105.06868*. (Cited on 135)

Fortuin, V., Garriga-Alonso, A., Wenzel, F., Rätsch, G., Turner, R., van der Wilk, M., and Aitchison, L. (2021). Bayesian neural network priors revisited. *arXiv preprint arXiv:2102.06571*. (Cited on 110, 125, 134, 239)

Freeman, C. D. and Bruna, J. (2017). Topology and geometry of half-rectified network optimization. *International Conference on Learning Representations*. (Cited on 32)

Gal, Y. (2016). *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge. (Cited on 88)

Gal, Y. and Ghahramani, Z. (2016). Dropout as a Bayesian Approximation. In *International Conference on Machine Learning*. (Cited on 53, 54, 80, 84, 88, 107, 119, 132, 134)

Gal, Y., Hron, J., and Kendall, A. (2017). Concrete Dropout. In *Advances in Neural Information Processing Systems*. arXiv: 1705.07832. (Cited on 84, 132)

Ganin, Y. and Lempitsky, V. (2015). Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR. (Cited on 158)

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030. (Cited on 158)

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. (2018). Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, pages 7587–7597. (Cited on 185, 209)

Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P., and Wilson, A. G. (2018). Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in Neural Information Processing Systems*, 31. (Cited on xxxviii, 4, 17, 18, 20, 28, 42, 43, 80, 99, 182, 186, 201, 210, 229)

Garriga-Alonso, A. and Fortuin, V. (2021). Exact langevin dynamics with stochastic gradients. *arXiv preprint arXiv:2102.01691*. (Cited on 133)

Gastaldi, X. (2017). Shake-shake regularization. *arXiv preprint arXiv:1705.07485*. (Cited on 28, 179)

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. (2020). Shortcut Learning In Deep Neural Networks. *Nature Machine Intelligence*, 2(11):665–673. (Cited on 139, 151, 155)

Geirhos, R., Narayanappa, K., Mitzkus, B., Thieringer, T., Bethge, M., Wichmann, F. A., and Brendel, W. (2021). Partial success in closing the gap between human and machine vision. In *Advances in Neural Information Processing Systems 34*. (Cited on xxxvii, 302, 304)

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. (2018). ImageNet-trained CNNs Are Biased Towards Texture; Increasing Shape Bias Improves Accuracy And Robustness. *arXiv preprint arXiv:1811.12231*. (Cited on xxxvii, 1, 151, 153, 154, 157, 300, 301, 302, 304)

Gelman, A., Rubin, D. B., et al. (1992). Inference from iterative simulation using multiple sequences. *Statistical science*, 7(4):457–472. (Cited on 97, 225)

Geneva, N. and Zabaras, N. (2020). Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics*. (Cited on 88)

Ghadimi, S. and Lan, G. (2013). Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368. (Cited on 27)

Gildenblat, J. and contributors (2021). Pytorch library for cam methods. `https://github.com/jacobgil/pytorch-grad-cam`. (Cited on 299)

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448. (Cited on 158)

Goodfellow, I. J., Vinyals, O., and Saxe, A. M. (2015). Qualitatively characterizing neural network optimization problems. *International Conference on Learning Representations*. (Cited on 5, 27)

Gotmare, A., Shirish Keskar, N., Xiong, C., and Socher, R. (2018). Using mode connectivity for loss landscape analysis. *arXiv preprint arXiv:1806.06977*. (Cited on 10)

Graves, A. (2011). Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356. (Cited on 84, 132)

Grünwald, P. (2012). The safe Bayesian. In *International Conference on Algorithmic Learning Theory*, pages 169–183. Springer. (Cited on 71, 72, 73, 83)

Grünwald, P., Van Ommen, T., et al. (2017). Inconsistency of Bayesian inference for misspecified linear models, and a proposal for repairing it. *Bayesian Analysis*, 12(4):1069–1103. (Cited on 71, 72, 75, 83)

Guedj, B. (2019). A primer on pac-bayesian learning. *arXiv preprint arXiv:1901.05353*. (Cited on 81)

Gulrajani, I. and Lopez-Paz, D. (2020). In search of lost domain generalization. *arXiv preprint arXiv:2007.01434*. (Cited on 158)

Gulshan, V., Peng, L., Coram, M., Stumpe, M. C., Wu, D., Narayanaswamy, A., Venugopalan, S., Widner, K., Madams, T., Cuadros, J., et al. (2016). Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410. (Cited on 107)

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. (Cited on 1, 43, 53, 54, 55, 85, 132, 170, 190, 193)

Gustafsson, F. K., Danelljan, M., and Schön, T. B. (2019). Evaluating scalable bayesian deep learning methods for robust computer vision. *arXiv preprint arXiv:1906.01620*. (Cited on 82)

Hafner, D., Tran, D., Irpan, A., Lillicrap, T., and Davidson, J. (2018). Reliable uncertainty estimates in deep neural networks using noise contrastive priors. *arXiv preprint arXiv:1807.09289*. (Cited on 80)

Halko, N., Martinsson, P.-G., and Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288. (Cited on 51)

Han, D., Kim, J., and Kim, J. (2016). Deep pyramidal residual networks. *arXiv preprint arXiv:1610.02915*. (Cited on 28, 179)

Hardt, M., Price, E., and Srebro, N. (2016). Equality of opportunity in supervised learning. *Advances in neural information processing systems*, 29. (Cited on 158)

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. (2021). Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*. (Cited on 160)

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969. (Cited on 158)

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. (Cited on 9, 25, 28, 29, 55, 65, 84, 93, 149, 199)

Hendrycks, D., Basart, S., Mu, N., Kadavath, S., Wang, F., Dorundo, E., Desai, R., Zhu, T., Parajuli, S., Guo, M., Song, D., Steinhardt, J., and Gilmer, J. (2021). The many faces of robustness: A critical analysis of out-of-distribution generalization. *ICCV*. (Cited on 151, 297, 300, 301)

Hendrycks, D. and Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*. (Cited on 1, 61, 105, 107, 112, 154, 204, 205, 208, 214, 301)

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*. (Cited on 93)

Hermann, K., Chen, T., and Kornblith, S. (2020). The origins and prevalence of texture bias in convolutional neural networks. *Advances in Neural Information Processing Systems*, 33:19000–19015. (Cited on 153, 158)

Hermann, K. and Lampinen, A. (2020). What shapes feature representations? exploring datasets, architectures, and training. *Advances in Neural Information Processing Systems*, 33:9995–10006. (Cited on 140, 146, 155, 159)

Hernández-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869. PMLR. (Cited on 132)

Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. (Cited on 43, 80)

Hochreiter, S. and Schmidhuber, J. (1997a). Flat minima. *Neural Computation*, 9(1):1–42. (Cited on 15, 31, 43, 80, 272)

Hochreiter, S. and Schmidhuber, J. (1997b). Long short-term memory. *Neural computation*, 9(8):1735–1780. (Cited on 93)

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347. (Cited on 132)

Hu, W., Niu, G., Sato, I., and Sugiyama, M. (2018). Does distributionally robust supervised learning give robust classifiers? In *International Conference on Machine Learning*, pages 2029–2037. PMLR. (Cited on 156)

Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., and Weinberger, K. Q. (2017a). Snapshot ensembles: Train 1, get m for free. *International Conference on Learning Representations*. (Cited on 9, 13, 32, 173, 174, 176)

Huang, G., Liu, Z., Weinberger, K. Q., and van der Maaten, L. (2017b). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3. (Cited on 29, 55)

Huang, W. R., Emam, Z., Goldblum, M., Fowl, L., Terry, J. K., Huang, F., and Goldstein, T. (2019). Understanding generalization through visualizations. *arXiv preprint arXiv:1906.03291*. (Cited on 43, 81)

Huh, M., Agrawal, P., and Efros, A. A. (2016). What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*. (Cited on 158)

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Idrissi, B. Y., Arjovsky, M., Pezeshki, M., and Lopez-Paz, D. (2021). Simple Data Balancing Achieves Competitive Worst-Group-Accuracy. *arXiv preprint arXiv:2110.14503*. (Cited on 143, 149, 156, 159, 276, 283, 292, 293, 306)

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. (Cited on 21, 49, 93, 165)

Islam, M. A., Kowal, M., Esser, P., Jia, S., Ommer, B., Derpanis, K. G., and Bruce, N. (2021). Shape or texture: Understanding discriminative features in cnns. *arXiv preprint arXiv:2101.11604*. (Cited on 158)

Izmailov, P., Maddox, W. J., Kirichenko, P., Garipov, T., Vetrov, D., and Wilson, A. G. (2019).

Subspace inference for Bayesian deep learning. *Uncertainty in Artificial Intelligence*. (Cited on xxii, xxix, 42, 43, 44, 88, 102, 103, 104, 201, 229)

Izmailov, P., Nicholson, P., Lotfi, S., and Wilson, A. G. (2021a). Dangers of bayesian model averaging under covariate shift. *Advances in Neural Information Processing Systems*, 34:3309–3322. (Cited on 90)

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. (2018). Averaging weights leads to wider optima and better generalization. *Uncertainty in Artificial Intelligence*. (Cited on 4, 43, 49, 54, 62, 80, 83, 132, 177, 189, 198, 199, 260)

Izmailov, P., Vikram, S., Hoffman, M. D., and Wilson, A. G. (2021b). What are Bayesian neural network posteriors really like? In *International Conference on Machine Learning*. (Cited on xxiii, 72, 90, 109, 110, 111, 112, 119, 135, 239, 241, 243, 249)

Izmailov, P., Wilson, A. G., and Queneneville-Belair, V. (2020). Pytorch 1.6 now includes stochastic weight. (Cited on 35)

Jacobsen, J.-H., Behrmann, J., Zemel, R., and Bethge, M. (2018). Excessive invariance causes adversarial vulnerability. *arXiv preprint arXiv:1811.00401*. (Cited on 155, 246)

Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580. (Cited on 79, 202)

Jiang, Y., Neyshabur, B., Mobahi, H., Krishnan, D., and Bengio, S. (2019). Fantastic generalization measures and where to find them. *arXiv preprint arXiv:1912.02178*. (Cited on 81, 82)

Jonsson, H., Mills, G., and Jacobsen, K. W. (1998). Nudged elastic band method for finding minimum energy paths of transitions. *Classical and quantum dynamics in condensed phase simulations*. (Cited on 33)

Jouppi, N. P., Yoon, D. H., Kurian, G., Li, S., Patil, N., Laudon, J., Young, C., and Patterson, D. (2020). A domain-specific supercomputer for training deep neural networks. *Communications of the ACM*, 63(7):67–78. (Cited on 92, 111)

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589. (Cited on 1)

Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Dodds, Z. H., DasSarma, N., Tran-Johnson, E., et al. (2022). Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*. (Cited on 1)

Kang, B., Xie, S., Rohrbach, M., Yan, Z., Gordo, A., Feng, J., and Kalantidis, Y. (2019). Decoupling representation and classifier for long-tailed recognition. *arXiv preprint arXiv:1910.09217*. (Cited on xiii, xlv, 157, 274, 275, 276, 306, 307)

Kapoor, S., Maddox, W. J., Izmailov, P., and Wilson, A. G. (2022). On uncertainty, tempering, and data augmentation in bayesian classification. *Advances in Neural Information Processing Systems*. (Cited on 72, 76)

Kass, R. E. and Raftery, A. E. (1995). Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795. (Cited on 80)

Kaushik, D., Hovy, E., and Lipton, Z. (2020). Learning the difference that makes a difference with counterfactually-augmented data. In *International Conference on Learning Representations*. (Cited on 158)

Kaushik, D., Setlur, A., Hovy, E., and Lipton, Z. C. (2021). Explaining the efficacy of counterfactually augmented data. *International Conference on Learning Representations (ICLR)*. (Cited on 158)

Kendall, A. and Gal, Y. (2017). What uncertainties do we need in Bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584. (Cited on 80, 84, 107, 132)

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*. (Cited on 43, 80, 272)

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. *International Conference on Learning Representations*. (Cited on 5, 15, 22, 24, 31)

Kessler, S., Nguyen, V., Zohren, S., and Roberts, S. (2019). Hierarchical indian buffet neural networks for bayesian continual learning. *arXiv preprint arXiv:1912.02290*. (Cited on 135)

Khan, M. E., Nielsen, D., Tangkaratt, V., Lin, W., Gal, Y., and Srivastava, A. (2018). Fast and scalable Bayesian deep learning by weight-perturbation in adam. *arXiv preprint arXiv:1806.04854*. (Cited on 80, 84, 132)

Khani, F., Raghunathan, A., and Liang, P. (2019). Maximum weighted loss discrepancy. *arXiv preprint arXiv:1906.03518*. (Cited on 158)

Kim, B., Kim, H., Kim, K., Kim, S., and Kim, J. (2019). Learning not to learn: Training deep neural networks with biased data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9012–9020. (Cited on 157)

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28. (Cited on xli, 46, 84, 132, 198)

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. In *International Conference on Learning Representations*. (Cited on 58, 84, 259)

Kirichenko, P., Izmailov, P., and Wilson, A. G. (2023). Last Layer Re-Training Is Sufficient For Robustness To Spurious Correlations. *International Conference on Learning Representations*. (Cited on 138)

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526. (Cited on 48, 85, 88, 132)

Kleinberg, J., Mullainathan, S., and Raghavan, M. (2016). Inherent trade-offs in the fair determination of risk scores. *arXiv preprint arXiv:1609.05807*. (Cited on 158)

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. In Loizides, F. and Schmidt, B., editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press.

Koh, P. W., Sagawa, S., Marklund, H., Xie, S. M., Zhang, M., Balsubramani, A., Hu, W., Yasunaga, M., Phillips, R. L., Gao, I., et al. (2021). Wilds: A Benchmark Of In-The-Wild Distribution Shifts. *International Conference on Machine Learning (ICML)*. (Cited on 148, 283)

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., and Houlsby, N. (2020). Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pages 491–507. Springer. (Cited on 158)

Kolesnikov, A. and Lampert, C. H. (2016). Improving weakly-supervised object localization by micro-annotation. *arXiv preprint arXiv:1605.05538*. (Cited on 155, 157)

Kornblith, S., Shlens, J., and Le, Q. V. (2019). Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2661–2671. (Cited on 158)

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. (Cited on 144)

Krizhevsky, A., Nair, V., and Hinton, G. (2014). The CIFAR-10 dataset. `http://www.cs.toronto.edu/kriz/cifar.html`. (Cited on 61, 93, 110)

Krueger, D., Caballero, E., Jacobsen, J.-H., Zhang, A., Binas, J., Zhang, D., Le Priol, R., and Courville, A. (2021). Out-of-distribution generalization via risk extrapolation (rex). In *International Conference on Machine Learning*, pages 5815–5826. PMLR. (Cited on 158)

Kuditipudi, R., Wang, X., Lee, H., Zhang, Y., Li, Z., Hu, W., Ge, R., and Arora, S. (2019). Explaining landscape connectivity of low-cost solutions for multilayer nets. *Advances in neural information processing systems*, 32. (Cited on 33)

Kuleshov, V., Fenner, N., and Ermon, S. (2018). Accurate Uncertainties for Deep Learning Using Calibrated Regression. In *International Conference on Machine Learning*, page 9. (Cited on 85, 190)

Kumar, A., Raghunathan, A., Jones, R., Ma, T., and Liang, P. (2022). Fine-tuning can distort pre-trained features and underperform out-of-distribution. *arXiv preprint arXiv:2202.10054*. (Cited on 158)

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413. (Cited on 40, 44, 45, 53, 56, 85, 88, 107, 110, 119, 134, 190)

Langford, J. and Caruana, R. (2002). (not) bounding the true error. In *Advances in Neural Information Processing Systems*, pages 809–816. (Cited on 81)

Lao, J., Suter, C., Langmore, I., Chimisov, C., Saxena, A., Sountsov, P., Moore, D., Saurous, R. A., Hoffman, M. D., and Dillon, J. V. (2020). tfp. mcmc: Modern markov chain monte carlo tools built for modern hardware. *arXiv preprint arXiv:2002.01184*. (Cited on 97)

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*. (Cited on 110)

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. (Cited on 64, 110, 144)

Lee, J. D., Simchowitz, M., Jordan, M. I., and Recht, B. (2016a). Gradient descent only converges to minimizers. In *Conference on Learning Theory*, pages 1246–1257. (Cited on 32)

Lee, K., Lee, K., Lee, H., and Shin, J. (2018). A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*, pages 7167–7177. (Cited on xxxviii, 104, 105)

Lee, S., Prakash, S. P. S., Cogswell, M., Ranjan, V., Crandall, D., and Batra, D. (2016b). Stochastic multiple choice learning for training diverse deep ensembles. In *Advances in Neural Information Processing Systems*, pages 2119–2127. (Cited on 175)

Lee, Y., Yao, H., and Finn, C. (2022). Diversify and disambiguate: Learning from underspecified data. *arXiv preprint arXiv:2202.03418*. (Cited on 157, 159, 293)

Lemos, P. et al. (2022). Robust simulation-based inference in cosmology with bayesian neural networks. (Cited on 35)

Li, D. X. (2000). On default correlation: A copula function approach. *Journal of Fixed Income*, 9(4):43–54. (Cited on 132)

Li, H., Xu, Z., Taylor, G., and Goldstein, T. (2017). Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*. (Cited on 10, 31, 186, 273)

Li, M. et al. (2022). Branch-train-merge: Embarrassingly parallel training of expert language models. (Cited on 35)

Li, Y., Li, Y., and Vasconcelos, N. (2018a). Resound: Towards action recognition without representation bias. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 513–528. (Cited on 157)

Li, Y., Tian, X., Gong, M., Liu, Y., Liu, T., Zhang, K., and Tao, D. (2018b). Deep domain generalization via conditional invariant adversarial networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 624–639. (Cited on 158)

Li, Y. and Vasconcelos, N. (2019). Repair: Removing representation bias by dataset resampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9572–9581. (Cited on 157)

Liang, S., Li, Y., and Srikant, R. (2017). Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*. (Cited on xxxviii, 104, 105)

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528.

Liu, E. Z., Haghgoo, B., Chen, A. S., Raghunathan, A., Koh, P. W., Sagawa, S., Liang, P., and Finn, C. (2021). Just train twice: Improving group robustness without training group information. In *International Conference on Machine Learning*, pages 6781–6792. PMLR. (Cited on 143, 148, 149, 156, 159, 160, 283, 292, 293)

Liu, X., Xue, W., Xiao, L., and Zhang, B. (2017). Pbodl: Parallel bayesian online deep learning for click-through rate prediction in tencent advertising system. *arXiv preprint arXiv:1707.00802.* (Cited on 88)

Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep Learning Face Attributes In The Wild. *Proceedings of the IEEE international conference on computer vision.* (Cited on 148)

Loshchilov, I. and Hutter, F. (2016). Sgdr: stochastic gradient descent with restarts. *arxiv preprint arxiv:1608.03983.* (Cited on 15)

Loshchilov, I. and Hutter, F. (2017a). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101.* (Cited on 51, 283)

Loshchilov, I. and Hutter, F. (2017b). Sgdr: Stochastic gradient descent with warm restarts. *International Conference on Learning Representations.* (Cited on 32)

Louizos, C., Shi, X., Schutte, K., and Welling, M. (2019). The functional neural process. In *Advances in Neural Information Processing Systems.* (Cited on 80)

Louizos, C. and Welling, M. (2017). Multiplicative normalizing flows for variational bayesian neural networks. In *International Conference on Machine Learning.* (Cited on 84, 132)

Lovering, C., Jha, R., Linzen, T., and Pavlick, E. (2020). Predicting inductive biases of pre-trained models. In *International Conference on Learning Representations.* (Cited on 158)

Ma, Y.-A., Chen, T., and Fox, E. B. (2015). A complete recipe for stochastic gradient mcmc. *arXiv preprint arXiv:1506.04696.* (Cited on 133)

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics. (Cited on 93)

MacKay, D. J. (1992a). *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology. (Cited on 39, 74, 79)

MacKay, D. J. (1995). Probable networks and plausible predictions?a review of practical Bayesian methods for supervised neural networks. *Network: computation in neural systems*, 6(3):469–505. (Cited on 44, 74, 79, 80, 86, 88, 121, 212)

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press. (Cited on 74, 80, 81, 182, 188)

MacKay, D. J. C. (1992b). A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3):448–472. (Cited on 85, 132)

Maddox, W., Tang, S., Moreno, P., Wilson, A. G., and Damianou, A. (2021). Fast adaptation with linearized neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 2737–2745. PMLR. (Cited on 158)

Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. (2019). A simple baseline for Bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems*. (Cited on xxii, 35, 36, 59, 65, 88, 102, 103, 104, 107, 119, 132, 134, 205, 207, 209, 241, 260)

Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., and Van Der Maaten, L. (2018). Exploring the limits of weakly supervised pretraining. In *Proceedings of the European conference on computer vision (ECCV)*, pages 181–196. (Cited on 158)

Mandt, S., Hoffman, M. D., and Blei, D. M. (2017a). Stochastic gradient descent as approximate bayesian inference. *arXiv preprint arXiv:1704.04289*. (Cited on xi, 47, 49, 84, 85, 180, 181, 182, 183, 184, 185, 189)

Mandt, S., Hoffman, M. D., and Blei, D. M. (2017b). Stochastic gradient descent as approximate bayesian inference. *arXiv preprint arXiv:1704.04289*. (Cited on 26, 33, 132)

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330. (Cited on 168)

Masegosa, A. R. (2019). Learning under model misspecification: Applications to variational and ensemble methods. (Cited on 81, 82, 135)

McAllester, D. A. (1999). Pac-bayesian model averaging. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 164–170. (Cited on 81)

McCoy, R. T., Pavlick, E., and Linzen, T. (2019). Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. *arXiv preprint arXiv:1902.01007*. (Cited on 158)

McKinney, W. (2010). Data structures for statistical computing in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

Menon, A. K., Rawat, A. S., and Kumar, S. (2020). Overparameterisation and worst-case generalisation: friend or foe? In *International Conference on Learning Representations*. (Cited on 156)

Merity, S., Keskar, N. S., and Socher, R. (2017). Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*. (Cited on 57, 58, 194)

Michelmore, R., Wicker, M., Laurenti, L., Cardelli, L., Gal, Y., and Kwiatkowska, M. (2020). Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7344–7350. IEEE. (Cited on 107)

Miller, J. P., Taori, R., Raghunathan, A., Sagawa, S., Koh, P. W., Shankar, V., Liang, P., Carmon, Y., and Schmidt, L. (2021). Accuracy on the line: On the strong correlation between out-of-distribution and in-distribution generalization. In *International Conference on Machine Learning*, pages 7721–7735. PMLR. (Cited on 242)

Minderer, M., Djolonga, J., Romijnders, R., Hubis, F., Zhai, X., Houlsby, N., Tran, D., and Lucic, M. (2021). Revisiting the calibration of modern neural networks. *Advances in Neural Information Processing Systems*, 34:15682–15694. (Cited on 1)

Minka, T. (2000a). Automatic choice of dimensionality for pca. *Advances in neural information processing systems*, 13. (Cited on 74)

Minka, T. (2001). Expectation propagation for approximate Bayesian inference. In *Uncertainty in Artificial Intelligence*, volume 17, pages 362–369. (Cited on 44)

Minka, T. P. (2000b). Bayesian model averaging is not model combination. (Cited on 45, 135)

Moayeri, M., Pope, P., Balaji, Y., and Feizi, S. (2022). A Comprehensive Study Of Image Classification Model Sensitivity To Foregrounds, Backgrounds, And Visual Attributes. *arXiv preprint arXiv:2201.10766*. (Cited on 157)

Mohri, M. and Rostamizadeh, A. (2009). Rademacher complexity bounds for non-iid processes. In *Advances in Neural Information Processing Systems*, pages 1097–1104. (Cited on 40)

Molchanov, D., Ashukha, A., and Vetrov, D. (2017). Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*. (Cited on 84, 132, 135)

Morningstar, W. R., Alemi, A. A., and Dillon, J. V. (2020). Pac m-bayes: Narrowing the empirical risk gap in the misspecified bayesian regime. *arXiv preprint arXiv:2010.09629*. (Cited on 135)

Mu, N. and Gilmer, J. (2019). Mnist-c: A robustness benchmark for computer vision. *arXiv preprint arXiv:1906.02337*. (Cited on 111)

Muandet, K., Balduzzi, D., and Schölkopf, B. (2013). Domain generalization via invariant feature representation. In *International Conference on Machine Learning*, pages 10–18. PMLR. (Cited on 158)

Mukhoti, J. and Gal, Y. (2018). Evaluating Bayesian Deep Learning Methods for Semantic Segmentation. (Cited on 84)

Müller, U. K. (2013). Risk of bayesian inference in misspecified models, and the sandwich covariance matrix. *Econometrica*, 81(5):1805–1849. (Cited on 182)

Murphy, K. P. (2023). *Probabilistic Machine Learning: Advanced Topics*. MIT Press. (Cited on 35)

Naeini, M. P., Cooper, G. F., and Hauskrecht, M. (2015). Obtaining well calibrated probabilities using bayesian binning. In *AAAI*, pages 2901–2907. (Cited on 193)

Nagarajan, V., Andreassen, A., and Neyshabur, B. (2020). Understanding the failure modes of out-of-distribution generalization. *arXiv preprint arXiv:2010.15775*. (Cited on 134, 155)

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., and Sutskever, I. (2019). Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*. (Cited on 68, 83, 209)

Nakkiran, P., Venkat, P., Kakade, S., and Ma, T. (2020). Optimal regularization can mitigate double descent. *arXiv preprint arXiv:2003.01897*. (Cited on 70, 83)

Nalisnick, E. (2018). *On priors for Bayesian neural networks*. PhD thesis, University of California, Irvine. (Cited on 80)

Nam, J., Cha, H., Ahn, S., Lee, J., and Shin, J. (2020). Learning from failure: De-biasing classifier from biased classifier. *Advances in Neural Information Processing Systems*, 33:20673–20684. (Cited on 156)

Nam, J., Kim, J., Lee, J., and Shin, J. (2022). Spread Spurious Attribute: Improving Worst-group Accuracy With Spurious Attribute Estimation. *International Conference on Learning Representations*. (Cited on 149, 156, 292, 293)

Neal, R. (1996). *Bayesian Learning for Neural Networks*. Springer Verlag. (Cited on 46, 79, 83, 88, 133, 202)

Neal, R. M. et al. (2011). Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2. (Cited on 89, 91, 92)

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. (Cited on 113)

Neyshabur, B., Bhojanapalli, S., McAllester, D., and Srebro, N. (2017). Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956. (Cited on 81)

Neyshabur, B., Bhojanapalli, S., and Srebro, N. (2018). A PAC-bayesian approach to spectrally-normalized margin bounds for neural networks. In *International Conference on Learning Representations*. (Cited on 81)

Neyshabur, B., Sedghi, H., and Zhang, C. (2020). What is being transferred in transfer learning? *arXiv preprint arXiv:2008.11687*. (Cited on 134, 158)

Niculescu-Mizil, A. and Caruana, R. (2005). Predicting good probabilities with supervised learning. In *International Conference on Machine Learning*, pages 625–632, Bonn, Germany. ACM Press. (Cited on 54)

Nikishin, E., Izmailov, P., Athiwaratkun, B., Podoprikhin, D., Garipov, T., Shvechikov, P., Vetrov, D., and Wilson, A. G. (2018). Improving stability in deep reinforcement learning with weight averaging. *UAI Workshop*. (Cited on 35)

Nocedal, J. (1980). Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782.

OpenAI (2023). Gpt-4 technical report. (Cited on 1)

Opper, M., Kinzel, W., Kleinz, J., and Nehl, R. (1990). On the ability of the optimal perceptron to generalise. *Journal of Physics A: Mathematical and General*, 23(11):L581. (Cited on 83)

Oren, Y., Sagawa, S., Hashimoto, T. B., and Liang, P. (2019). Distributionally robust language modeling. *arXiv preprint arXiv:1909.02060*. (Cited on 156)

Osawa, K., Swaroop, S., Jain, A., Eschenhagen, R., Turner, R. E., Yokota, R., and Khan, M. E. (2019). Practical deep learning with bayesian principles. *arXiv preprint arXiv:1906.02506*. (Cited on 132)

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J. V., Lakshminarayanan, B., and Snoek, J. (2019). Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *arXiv preprint arXiv:1906.02530*. (Cited on xxxi, 45, 46, 60, 61, 105, 107, 119, 134, 203, 204, 219, 242)

Pagliardini, M., Jaggi, M., Fleuret, F., and Karimireddy, S. P. (2022). Agree to disagree: Diversity through disagreement for better transferability. *arXiv preprint arXiv:2202.04414*. (Cited on 144, 145, 157)

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359. (Cited on 158)

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. (Cited on 275, 297)

Pearce, T., Zaki, M., Brintrup, A., Anastassacos, N., and Neely, A. (2018). Uncertainty in neural networks: Bayesian ensembling. *arXiv preprint arXiv:1810.05546*. (Cited on 82)

Peters, J., Bühlmann, P., and Meinshausen, N. (2016). Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78(5):947–1012. (Cited on 158)

Pezeshki, M., Kaba, O., Bengio, Y., Courville, A. C., Precup, D., and Lajoie, G. (2021). Gradient starvation: A learning proclivity in neural networks. *Advances in Neural Information Processing Systems*, 34. (Cited on 155, 157, 159, 160)

Pleiss, G., Raghavan, M., Wu, F., Kleinberg, J., and Weinberger, K. Q. (2017). On fairness and calibration. *Advances in neural information processing systems*, 30. (Cited on 158)

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17. (Cited on 110)

Polyak, B. T. and Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855. (Cited on 26, 33, 85, 180)

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., and Courville, A. (2019). On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR. (Cited on 155)

Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*. (Cited on 93)

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. (2021). Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR. (Cited on 1)

Rasmussen, C. E. and Ghahramani, Z. (2001). Occam's razor. In *Neural Information Processing Systems (NIPS)*. (Cited on 80)

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian processes for Machine Learning*. The MIT Press. (Cited on 37, 67, 74, 121, 125, 202)

Ren, M., Zeng, W., Yang, B., and Urtasun, R. (2018). Learning to reweight examples for robust deep learning. In *International conference on machine learning*, pages 4334–4343. PMLR. (Cited on 157)

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. (Cited on 151)

Ritter, H., Botev, A., and Barber, D. (2018a). Online structured laplace approximations for overcoming catastrophic forgetting. *Advances in Neural Information Processing Systems*, 31. (Cited on 85)

Ritter, H., Botev, A., and Barber, D. (2018b). A Scalable Laplace Approximation for Neural Networks. In *International Conference on Learning Representations*. (Cited on 53, 54, 80, 85, 132, 199, 209)

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407. (Cited on 110)

Rosenfeld, A., Zemel, R., and Tsotsos, J. K. (2018). The elephant in the room. *arXiv preprint arXiv:1808.03305*. (Cited on 139, 151)

Rosenfeld, E., Ravikumar, P., and Risteski, A. (2022). Domain-adjusted regression or: Erm may already learn features sufficient for out-of-distribution generalization. *arXiv preprint arXiv:2202.06856*. (Cited on 156, 157, 276, 277)

Roy, A. G., Ren, J., Azizi, S., Loh, A., Natarajan, V., Mustafa, B., Pawlowski, N., Freyberg, J., Liu, Y., Beaver, Z., et al. (2021). Does your dermatology classifier know what it doesn't know? detecting the long-tail of unseen conditions. *arXiv preprint arXiv:2104.03829*. (Cited on 107)

Ruan, Y., Dubois, Y., and Maddison, C. J. (2021). Optimal representations for covariate shift. *arXiv preprint arXiv:2201.00057*. (Cited on 158, 159)

Rue, H., Martino, S., and Chopin, N. (2009). Approximate Bayesian inference for latent gaussian models by using integrated nested laplace approximations. *Journal of the royal statistical society: Series b (statistical methodology)*, 71(2):319–392. (Cited on 44)

Ruppert, D. (1988). Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering. (Cited on 15, 33, 85, 180)

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2012). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252. (Cited on 27, 53, 175)

Sagawa, S., Koh, P. W., Hashimoto, T. B., and Liang, P. (2020). Distributionally Robust Neural Networks For Group Shifts: On The Importance Of Regularization For Worst-Case Generalization. *International Conference on Learning Representations (ICLR)*. (Cited on xl, 141, 142, 143, 148, 150, 156, 159, 160, 283, 292)

Sagawa, S., Koh, P. W., Lee, T., Gao, I., Xie, S. M., Shen, K., Kumar, A., Hu, W., Yasunaga, M., Marklund, H., et al. (2021). Extending the wilds benchmark for unsupervised adaptation. *arXiv preprint arXiv:2112.05090*. (Cited on 148)

Sagun, L., Evci, U., Guney, V. U., Dauphin, Y., and Bottou, L. (2018). Empirical Analysis of the Hessian of Over-Parametrized Neural Networks. In *International Conference on Learning Representations Workshop Track*. arXiv: 1706.04454. (Cited on 186)

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, K., Gontijo Lopes, R., Karagol Ayan, B., Salimans, T., et al. (2022). Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494. (Cited on 1)

Scimeca, L., Oh, S. J., Chun, S., Poli, M., and Yun, S. (2021). Which shortcut cues will dnns choose? a study from the parameter-space perspective. *arXiv preprint arXiv:2110.03095*. (Cited on 155)

Shah, H., Tamuly, K., Raghunathan, A., Jain, P., and Netrapalli, P. (2020). The pitfalls of simplicity bias in neural networks. *Advances in Neural Information Processing Systems*, 33:9573–9585. (Cited on 140, 143, 144, 145, 146, 155)

Sharif Razavian, A., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813. (Cited on 158, 296)

Shetty, R., Schiele, B., and Fritz, M. (2019). Not using the car to see the sidewalk–quantifying and controlling the effects of context in classification and segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8218–8226. (Cited on 151, 157)

Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244. (Cited on 134)

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489. (Cited on 1)

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. (Cited on 9, 28, 65)

Singh, S. and Krishnan, S. (2020). Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11237–11246. (Cited on 93)

Singla, S. and Feizi, S. (2021). Salient imagenet: How to discover spurious features in deep learning? *arXiv preprint arXiv:2110.04301*. (Cited on 139, 151, 157, 160)

Singla, S., Nushi, B., Shah, S., Kamar, E., and Horvitz, E. (2021). Understanding failures of deep networks via robust feature extraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12853–12862. (Cited on 157)

Skorokhodov, I. and Burtsev, M. (2019). Loss landscape sightseeing with multi-point optimization. *arXiv preprint arXiv:1910.03867*. (Cited on 33)

Smith, L. N. and Topin, N. (2017). Exploring loss function topology with cyclical learning rates. *arXiv preprint arXiv:1702.04283*. (Cited on 13, 14, 17, 18)

Smith, S. L. and Le, Q. V. (2018). A Bayesian perspective on generalization and stochastic gradient descent. In *International Conference on Learning Representations*. (Cited on 67, 68, 81)

Sohoni, N., Dunnmon, J., Angus, G., Gu, A., and Ré, C. (2020). No subclass left behind: Fine-grained robustness in coarse-grained classification problems. *Advances in Neural Information Processing Systems*, 33:19339–19352. (Cited on 157)

Sohoni, N., Sanjabi, M., Ballas, N., Grover, A., Nie, S., Firooz, H., and Ré, C. (2021). BARACK: Partially Supervised Group Robustness With Guarantees. *arXiv preprint arXiv:2201.00072*. (Cited on 156, 159, 160, 293)

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958. (Cited on 34, 132)

Storkey, A. (2009). When training and test sets are different: characterizing learning transfer. *Dataset shift in machine learning*, 30:3–28. (Cited on 134)

Storkey, A. J. and Sugiyama, M. (2007). Mixture regression for covariate shift. *Advances in neural information processing systems*, 19:1337. (Cited on 134)

Sugiyama, M., Blankertz, B., Krauledat, M., Dornhege, G., and Müller, K.-R. (2006). Importance-weighted cross-validation for covariate shift. In *Joint Pattern Recognition Symposium*, pages 354–363. Springer. (Cited on 134)

Sugiyama, M., Krauledat, M., and Müller, K.-R. (2007). Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8(5). (Cited on 134)

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852. (Cited on 158)

Sun, S., Zhang, G., Shi, J., and Grosse, R. (2019). Functional variational Bayesian neural networks. *arXiv preprint arXiv:1903.05779*. (Cited on 80)

Tang, P. et al. (2019). Efficient skin lesion segmentation using separable-unet with stochastic weight averaging. *Computer methods and programs in biomedicine*. (Cited on 35)

Tartaglione, E., Barbano, C. A., and Grangetto, M. (2021). End: Entangling and disentangling deep representations for bias correction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13508–13517. (Cited on 157)

Teney, D., Abbasnejad, E., Lucey, S., and Hengel, A. v. d. (2021a). Evading the simplicity bias: Training a diverse set of models discovers solutions with superior ood generalization. *arXiv preprint arXiv:2105.05612*. (Cited on 157)

Teney, D., Abbasnejad, E., and van den Hengel, A. (2021b). Unshuffling data for improved generalization in visual question answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1417–1427. (Cited on 157)

Tran, B.-H., Rossi, S., Milios, D., and Filippone, M. (2020). All you need is a good functional prior for bayesian deep learning. *arXiv preprint arXiv:2011.12829*. (Cited on 125)

Trask, A., Hill, F., Reed, S., Rae, J., Dyer, C., and Blunsom, P. (2018). Neural arithmetic logic units. *arXiv preprint arXiv:1808.00508*.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2018). Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9446–9454. (Cited on 63)

Utama, P. A., Moosavi, N. S., and Gurevych, I. (2020). Towards debiasing nlu models from unknown biases. *arXiv preprint arXiv:2009.12303*. (Cited on 156)

Van der Vaart, A. W. (2000). *Asymptotic statistics*, volume 3. Cambridge university press. (Cited on 180)

Vapnik, V. N. (1998). Adaptive and learning systems for signal processing communications, and control. *Statistical learning theory*. (Cited on 40)

Veitch, V., D'Amour, A., Yadlowsky, S., and Eisenstein, J. (2021). Counterfactual invariance to spurious correlations: Why and how to pass stress tests. *arXiv preprint arXiv:2106.00545*. (Cited on 158)

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020a). Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature Methods*, pages 1–12. (Cited on 204)

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman,

K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020b). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. (2011). The caltech-ucsd birds-200-2011 dataset. (Cited on 142)

Walker, S. and Hjort, N. L. (2001). On Bayesian consistency. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(4):811–821. (Cited on 71, 72, 83)

Wang, H., He, Z., Lipton, Z. C., and Xing, E. P. (2019). Learning robust representations by projecting superficial statistics out. *arXiv preprint arXiv:1903.06256*. (Cited on 157)

Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. (Cited on 48, 53, 54, 58, 83, 88, 89, 130, 133, 200, 207)

Wenzel, F., Roth, K., Veeling, B. S., Światkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., and Nowozin, S. (2020). How good is the Bayes posterior in deep neural networks really? *arXiv preprint arXiv:2002.02405*. (Cited on xlii, 45, 71, 72, 73, 75, 76, 79, 83, 89, 93, 110, 122, 123, 124, 125, 130, 133, 207, 235, 236, 249)

Wicker, M., Laurenti, L., Patane, A., Chen, Z., Zhang, Z., and Kwiatkowska, M. (2021). Bayesian inference with certifiable adversarial robustness. In *International Conference on Artificial Intelligence and Statistics*, pages 2431–2439. PMLR. (Cited on 134)

Wilson, A. G. (2020). The case for Bayesian deep learning. *arXiv preprint arXiv:2001.10995*. (Cited on 80)

Wilson, A. G. and Izmailov, P. (2020). Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708. (Cited on 36, 89, 91, 105, 107, 122, 124, 125, 131, 132, 134, 229, 233, 239, 242)

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2019). Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Wortsman, M. et al. (2022). Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*. (Cited on 35)

Wu, A., Nowozin, S., Meeds, E., Turner, R. E., Hernández-Lobato, J. M., and Gaunt, A. L. (2018). Deterministic variational inference for robust Bayesian neural networks. *arXiv preprint arXiv:1810.03958*. (Cited on xxii, xli, 58, 84, 102, 103, 132, 197, 198)

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*. (Cited on 144)

Xiao, K., Engstrom, L., Ilyas, A., and Madry, A. (2020). Noise Or Signal: The Role Of Image Backgrounds In Object Recognition. *arXiv preprint arXiv:2006.09994*. (Cited on 151, 157, 278, 296, 300)

Xie, J., Xu, B., and Chuang, Z. (2013). Horizontal and vertical ensemble with deep representation for classification. *arXiv preprint arXiv:1306.2759*. (Cited on 32)

Xu, X., Lu, P., MacEachern, S., and Xu, R. (2019). Calibrated bayes factors for model comparison. *Journal of Statistical Computation and Simulation*, 89(4):591–614. (Cited on 74)

Xu, Y., He, H., Shen, T., and Jaakkola, T. S. (2022). Controlling directions orthogonal to a classifier. In *International Conference on Learning Representations*. (Cited on 157)

Yaghoobzadeh, Y., Mehri, S., Tachet, R., Hazen, T. J., and Sordoni, A. (2019). Increasing robustness to spurious correlations using forgettable examples. *arXiv preprint arXiv:1911.03861*. (Cited on 156)

Yang, G., Zhang, T., Kirichenko, P., Bai, J., Wilson, A. G., and De Sa, C. (2019a). Swalp: Stochastic weight averaging in low precision training. In *International Conference on Machine Learning (ICML)*. (Cited on 35)

Yang, W., Lorch, L., Graule, M. A., Srinivasan, S., Suresh, A., Yao, J., Pradier, M. F., and Doshi-Velez, F. (2019b). Output-constrained Bayesian neural networks. *arXiv preprint arXiv:1905.06287*. (Cited on 80)

Yao, J., Pan, W., Ghosh, S., and Doshi-Velez, F. (2019). Quality of uncertainty quantification for bayesian neural network inference. *arXiv preprint arXiv:1906.09686*. (Cited on 231)

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*. (Cited on 9, 28, 179, 188)

Zech, J. R., Badgeley, M. A., Liu, M., Costa, A. B., Titano, J. J., and Oermann, E. K. (2018). Variable Generalization Performance Of A Deep Learning Model To Detect Pneumonia In Chest Radiographs: A Cross-Sectional Study. *PLoS medicine*, 15(11):e1002683. (Cited on 139)

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhai, X., Puigcerver, J., Kolesnikov, A., Ruyssen, P., Riquelme, C., Lucic, M., Djolonga, J., Pinto, A. S., Neumann, M., Dosovitskiy, A., et al. (2019). A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*. (Cited on 158)

Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2021). Dive into deep learning. *arXiv preprint*. (Cited on 35)

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017a). Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations.* (Cited on 39, 40, 41, 63, 64, 65, 66, 67, 81, 208)

Zhang, G., Sun, S., Duvenaud, D., and Grosse, R. (2017b). Noisy Natural Gradient as Variational Inference. *arXiv:1712.02390 [cs, stat].* arXiv: 1712.02390. (Cited on 84)

Zhang, G., Sun, S., Duvenaud, D., and Grosse, R. (2018). Noisy natural gradient as variational inference. In *International Conference on Machine Learning*, pages 5852–5861. PMLR. (Cited on 132)

Zhang, J., Marszałek, M., Lazebnik, S., and Schmid, C. (2007). Local features and kernels for classification of texture and object categories: A comprehensive study. *International journal of computer vision*, 73(2):213–238. (Cited on 151)

Zhang, J., Menon, A., Veit, A., Bhojanapalli, S., Kumar, S., and Sra, S. (2020a). Coping with label shift via distributionally robust optimisation. *arXiv preprint arXiv:2010.12230.* (Cited on 156)

Zhang, M., Sohoni, N. S., Zhang, H. R., Finn, C., and Ré, C. (2022). Correct-n-contrast: A contrastive approach for improving robustness to spurious correlations. *arXiv preprint arXiv:2203.01517.* (Cited on 148, 156, 159, 281, 292, 293)

Zhang, R., Cooper, A. F., and De Sa, C. (2020b). Amagold: Amortized metropolis adjustment for efficient stochastic gradient mcmc. In *International Conference on Artificial Intelligence and Statistics*, pages 2142–2152. PMLR. (Cited on 133)

Zhang, R., Li, C., Zhang, J., Chen, C., and Wilson, A. G. (2020c). Cyclical stochastic gradient MCMC for Bayesian deep learning. In *International Conference on Learning Representations.* (Cited on 89, 122, 130, 133, 200)

Zhang, T. (2006). Information-theoretic upper and lower bounds for statistical estimation. *IEEE Transactions on Information Theory*, 52(4):1307–1321. (Cited on 71, 72, 83)

Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., and Torralba, A. (2017). Places: A 10 million image database for scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1452–1464. (Cited on 142)

Zhu, W., Zheng, H., Liao, H., Li, W., and Luo, J. (2021). Learning bias-invariant representation by cross-sample mutual information minimization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15002–15012. (Cited on 157)