Function Space Reasoning for Gaussian Processes and Neural Networks

by

Gregory Benton

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January, 2023

Professor Andrew Gordon Wilson

Dedication

For Emma, as the adventure continues.

Acknowledgements

The complete list of people to thank is much too long to fit here, and I am deeply grateful to all of them. Getting from part time night classes after work to completing a PhD has been a long journey and I have been supported by countless friends and mentors along the way.

First let me thank my advisor, Andrew, for his continued support. His insight, persistence, and motivation helped push me and my research further than I thought possible. Being a member of Andrew's group and learning from both him and the environment he has built has been an extreme privilege.

Working with such a talented and driven group of researchers has helped me grow tremendously over the last several years, and I would like to thank the members of the group for fostering such an incredible environment. Thank you to Marc, Pavel, Nate, and Sanae, whose efforts are reflected throughout this thesis, for being amazing coauthors and colleagues that can always be relied upon. A special thank you to Wesley, for both his love of espresso and his willingness to both find and pursue the good direction within any idea. You taught me a great deal, and thank you being an overall exceptional collaborator.

I would also like to thank my parents, for always backing me regardless of the direction I was headed. They have believed in me from the beginning, and without their support none of this would have been possible. Thank you to my siblings, nieces, and nephews for always having open doors and reminding me of the value of play.

Finally, my fiancée Emma and our dog, Thomas, both of whom I have the absolute joy of

sharing every day with. Thomas, thank you for being a perfect boy and continuing to prove that there is always time to go for a walk. Emma, I am eternally indebted to you for your boundless love and encouragement. You have been there for every early morning, late night, push for a deadline and did not waver in support for a moment. We met at the start of my PhD, and this journey has been a shared one.

Abstract

In a typical modeling setting we have prior notions of what types of functions we want to learn. For example, in regression we may want to learn a smooth function or a periodic function and in image classification we may want to learn a function that is invariant to rotations. While function space provides us the benefit of being able to reason about traits like invariance or smoothness, it is often difficult to directly quantify the functional properties of models.

In this thesis we leverage our ability to reason about function space to build more powerful models in both Gaussian processes (GPs) and neural networks. By generating GP kernels as functions themselves of latent processes, we introduce methods for providing uncertainty over what *types* of functions we produce, not just over the functions themselves in GP models. We also introduce methods for learning levels of invariance and equivariance in neural networks, enabling us to imbue the functions our models produce with soft inductive biases as opposed to hard constraints. Finally, we show how we can leverage our understanding of parameter space in neural networks to efficiently ensemble diverse collections of functions to improve the accuracy and robustness of our models. Through the introduction of these methods we show that by carefully considering the types of functions we are producing we can describe models with a range of desirable properties. These properties include more flexible models, models that better align with domain knowledge, and models that are both accurate and robust. We demonstrate these results on a broad range of problems, including time series forecasting, image classification, and reinforcement learning.

Contents

List of Figures

List of Tables

List of Appendices

1 Introduction

In many modeling contexts it is far easier to be prescriptive about what types of functions we want to produce than it is to reason about the parametric forms or the parameters of the functions themselves. For instance, with time series we may be able to simply look at our data and determine we want, e.g., some sort of quasi-periodic function with an upward trend. Conversely given a sufficiently complex parametric regression model and the same data, absent some numerical optimization routine it may be a priori impossible to determine reasonable values for the parameters in order to fit the data. Although it can be easier to reason about the types of functions we want to produce, in practice it is challenging to directly produce these functions, and our efforts instead become focused on learning parameters.

While a function space perspective provides us more direct contact with the data we aim to model than the parameter space perspective, it also introduces a number of new and exciting challenges. Although Gaussian processes (GPs) provide a method for modeling functions and even performing Bayesian inference in function space, there are limited methods for accounting for uncertainty over the GP models themselves. For example, in kernel learning we may wish to marginalize over a distribution of kernels, each of which may produce a different type of function. By placing function space priors over the kernels themselves, we are able to provide uncertainty over the types of functions our GP models produce, not just over the functions themselves.

Another growing area of interest that deals directly in function space is equivariance and invariance in neural networks. If, for example, we are seeking to model a function that is invariant

or to only a subset of rotations, meaning our predictions should not change as the inputs are rotated, then we may aim to learn a distribution over rotations that reflects the range of rotations to which we expect our function to be invariant. Through simple distributional assumptions over transformations we enable models to learn approximate invariances to both the correct transformations, and at the correct amount of the those transformations.

This thesis is composed of three parts, each concerned with a distinct component of function space modeling focused on either Gaussian processes models or neural networks. In Chapter 2 we discuss methods for forming distributions over covariance functions in Gaussian process models. First from a spectral representation perspective by modeling the Fourier transform of a kernel function with a latent GP, then via stochastic volatility models by using a latent GP to model a time varying volatility term.

In Chapter 3 we introduce methods for building distributions over symmetries in neural networks. We first examine approximate symmetries to a limited range of transformations, such as invariance to only a subset of rotations. Then we examine learning distributions over symmetries that are only approximately satisfied, such as physical systems where reflections about an axis may nearly, but not perfectly, preserve quantities like energy and momentum.

Finally, in Chapter 4, we explore the connections between parameter space and function space in neural networks. We conclude by describing a general approach for aggregating and ensembling collections of training solutions in neural networks. This approach is centered around cases where we cannot directly address function space quantities like symmetries and instead wish to ensemble diverse sets of functions. In these cases where we cannot efficiently measure functional diversity, we rely on loss surface inference to collect diverse sets of parameters as proxy for collecting diverse functions.

In aggregate, these approaches reflect the strength of modeling with a function space perspective, or with a viewpoint that enables us to establish a connection between the parameters in our models and the functions they produce.

1.1 Background: The Function Space Perspective

In this section we provide an introduction to general Gaussian process models, and the function space perspective of modeling both with Gaussian processes and neural networks as it is seen throughout the thesis. For a complete introduction to GPs and the function space perspective of modeling, we refer the reader to Chapter 2 of Rasmussen and Williams [2006].

Gaussian Processes

In Chapter 2 we focus on *Gaussian processes*. Simply put, a Gaussian process (GP) is a collection of random variables, any finite number of which are multivariate normal. Since the joint distribution of a Gaussian process observed at a finite number of points is multivariate normal, in order to fully specify the distribution of a Gaussian process we need only specify the mean and covariance functions of the process. Given a mean function $\mu(x)$ and a covariance function $k(x, x')$, we say $f(x)$ is a Gaussian process:

$$f(x) \sim \mathcal{GP}\left(\mu(x), k(x, x')\right). \tag{1.1}$$

With the mean and covariance specified, given some observations we can compute a posterior distribution at any input test points through an application of conditional multivariate normal identities [Bishop and Nasrabadi 2006]. For a set of input points $\mathbf{x} = \{x_1, \ldots, x_n\}$, the corresponding observations $f(\mathbf{x})$, and some test inputs $\mathbf{x}^*$, we can compute the posterior distribution over the test values as

$$\begin{aligned} f(\mathbf{x}^*)|\mathbf{x}, f(\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}^*) + k(\mathbf{x}^*, \mathbf{x}^*)k(\mathbf{x}, \mathbf{x})^{-1}(f(\mathbf{x}) - \mu(\mathbf{x})), \\ k(\mathbf{x}^*, \mathbf{x}^*) - k(\mathbf{x}^*, \mathbf{x})k(\mathbf{x}, \mathbf{x})^{-1}k(\mathbf{x}, \mathbf{x}^*)). \end{aligned} \tag{1.2}$$

For clarity, if we consider a partitioned multivariate normal distribution,

$$\begin{bmatrix} y_a \\ y_b \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m_a \\ m_b \end{bmatrix}, \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}\right), \tag{1.3}$$

we find that the posterior distribution of y_b given y_a is simply a new multivariate normal distribution with mean $m_b + \Sigma_{ba}\Sigma_{aa}^{-1}(x_a - m_a)$ and covariance $\Sigma_{bb} - \Sigma_{ba}\Sigma_{aa}^{-1}\Sigma_{ab}$. Recognizing the correspondence between $\mu(\cdot)$ in Equation 1.2 and the mean terms in Equation 1.3, and $k(\cdot, \cdot)$ and the covariance terms, we can see that the posterior distribution in Equation 1.2 is simply the conditional form of a partitioned multivariate normal distribution.

Typically, when building predictive models with Gaussian processes we select a straightforward mean, such as a constant or a linear function, and it is the covariance function, or *kernel*, that dictates the generalization properties. In Figure 1.1 we show examples of Gaussian processes posteriors with various kernels. Despite these kernels all being relatively simple covariance functions controlled by a small number of hyperparameters, they lead to GP models with very different behavior. Given the importance of the kernel function, the bulk of the efforts in Gaussian process research have been focused on developing increasingly expressive kernels that can model a wide range of functions [Rasmussen and Williams 2006; Wilson 2014a].

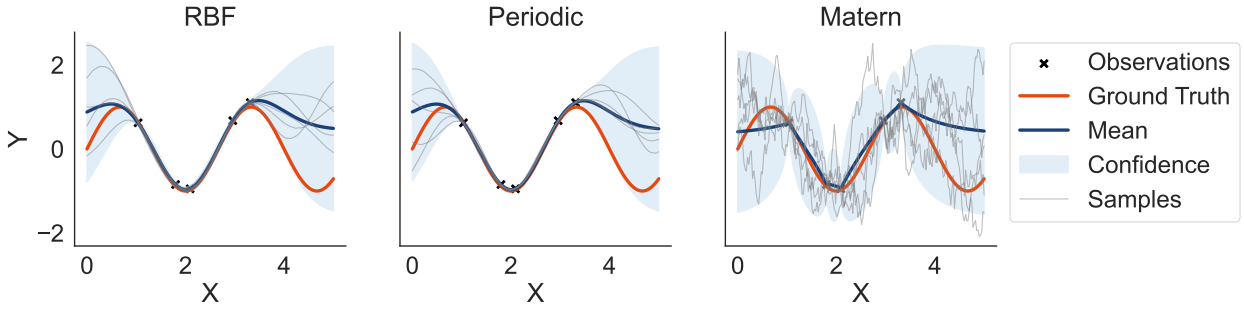


Figure 1.1: A Gaussian process on simple sinusoidal data with various kernels. Despite these kernels all being simple stationary covariance functions, they produce very different functions.

One key trait of GPs is that they provide a *function space* perspective. Rather than forming a

distribution over parameters which induces a distribution over functions, as we do in parametric modeling, GPs directly model a distribution over functions themselves. Through providing function space distributions, with GPs we can directly reason about the types of functions we want to produce, and through careful choice of kernels we can directly control the inductive biases of our models. Furthermore, since GPs provide closed form distributions in function space, we can easily perform Bayesian inference over the functions themselves.

In Chapter 2 we leverage the function space perspective of GPs to induce distributions over kernels. Since a GP provides a distribution over functions, and a kernel is itself just a function, we are able to use transformations of GPs as kernels themselves. Such approaches take the Bayesian framework of GPs one step further, enabling us to perform inference not just over the functions our GP models, but over the structure of the GP itself.

Neural Networks

As opposed to GPs, in neural network modeling we are generally concerned with *parameter space*. The prior distributions, the learning algorithms, and the posteriors (or their approximations) are usually all centered around the values of the parameters. We consider the functions as a consequence of those parameters and their combination with the architecture of the network.

For a neural network $f(x; \mathbf{w})$ with parameters $\mathbf{w}$, even with a closed form approximation of a posterior in parameter space, $p(\mathbf{w}|\mathcal{D})$, such as a Laplace approximation, the posterior induced over functions $p(f(x; \mathbf{w})|\mathcal{D})$ is still intractable [MacKay 2003].

While the functional distributions of neural networks are typically intractable and can only be approximated, there have been increasing efforts to connect the networks we build with the types of functions they produce. Such efforts can be seen with the wide adoption of convolutional neural networks, which encode translation invariance into our models, or recurrent neural networks, which encode temporal dependence structure into our models.

In Chapter 3 we consider equivariant and invariant neural networks. Equivariance and in-

variance, informally, are functional properties related to how our models change under transformations to the inputs [Cohen and Welling 2016a]. Where $f(x; \mathbf{w})$ is a neural network function, and g is a transformation, we can describe equivariance and invariance of the network to g as:

$$gf(x; \mathbf{w}) = f(gx; \mathbf{w}) \qquad \text{Equivariance,}$$
$$f(x; \mathbf{w}) = f(gx; \mathbf{w}) \qquad \text{Invariance.}$$

Equivariant functions change accordingly to transformations of the inputs, and invariant functions are unaffected by the transformation. Equivariance and invariance are special cases of functional constraints that allows us to reason about the functions our models generate and, with appropriate parameterizations, allow us to control the functional inductive biases of our models.

In cases lacking specific functional constraints like equivariance and invariance, neural networks are often treated as *black boxes* and little can be said about the function $f(\cdot; \mathbf{w})$ without querying that function at specific inputs. Chapter 4 explores these cases more fully, focusing on what we can say about the functions produced by neural networks under specific perturbations to the parameters. Namely, through understanding the how the training loss changes as a function of the parameters, what can be said about functions of the form $f(\cdot; \mathbf{w} + \Delta_{\mathbf{w}})$, where $\mathbf{w}$ are the parameters found through standard training procedures and $\Delta_{\mathbf{w}}$ is perturbation to those parameters.

We show that many conclusions about the functions produced under a parameter perturbation can be made by the curvature of training loss in the direction of that perturbation. The connections between loss surface curvature and the functional properties of neural networks help us better understand the connections between parameter and function space in neural networks, and give prescriptive guidance on how to build neural networks that produce accurate functions.

2 | Function-Space Kernels in Gaussian Processes

In practice modeling typically follows a two-step procedure: (1) choosing the functional form of a model, such as a neural network; (2) focusing learning efforts on training the parameters of that model. While inference of these parameters consume our efforts, they are rarely interpretable, and are only of interest insomuch as they combine with the functional form of the model to make predictions. Gaussian processes (GPs) provide an alternative *function space* approach to machine learning, directly placing a distribution over functions that could fit data [Rasmussen and Williams 2006]. This approach enables great flexibility, and also provides a compelling framework for controlling the inductive biases of the model, such as whether we expect the solutions to be smooth, periodic, or have conditional independence properties.

These inductive biases, and thus the generalization properties of the GP, are determined by a kernel function. The performance of the GP, and what representations it can learn, therefore crucially depend on what we can learn about the kernel function itself. Accordingly, kernel functions are becoming increasingly expressive and parametrized [Jang et al. 2017; Tobar et al. 2015; Wilson and Adams 2013].

In many modeling cases we have no a priori reason to believe that the data are generated from a single parametric family of kernels. In other cases, particularly those explored starting in Section 2.6, we may have good reason to believe that the data are generated from a family of

kernels, but we do not know which kernel is the correct one. In either setting, we should aim to take a Bayesian approach to kernel learning itself, and marginalize over a distribution over kernels, rather than try to estimate a single kernel with which to generate all of our predictions.

This chapter is adapted from the papers "Function-Space Distributions over Kernels", which originally appeared at Neurips 2019 and is joint work with Wesley Maddox, Jayson Salkey, Julio Albinati, and Andrew Gordon Wilson, and "Volatility Based Kernels and Moving Average Means for Accurate Forecasting with Gaussian Processes", which originally appeared at ICML 2022 and is joint work with Wesley Maddox, and Andrew Gordon Wilson.

2.1 Function-Space Distributions over Kernels

In the following sections we propose a method extending the function-space view to kernel learning itself – to represent uncertainty over the kernel function, and to reflect the belief that the kernel does not have a simple parametric form. Just as one uses GPs to directly specify a prior and infer a posterior over functions that can fit data, we propose to directly reason about priors and posteriors over kernels. In Figure 2.1, we illustrate the shift from standard function-space GP regression, to a function-space view of kernel learning.

Specifically, our contributions are as follows:

- We model a spectral density as a transformed Gaussian process, providing a non-parametric function-space distribution over kernels. Our approach, *functional kernel learning* (FKL), has several key properties: (1) it is highly flexible, with support for any stationary covariance function; (2) it naturally represents uncertainty over all values of the kernel; (3) it can easily be used to incorporate intuitions about what types of kernels are *a priori* likely; (4) despite its flexibility, it does not require sophisticated initialization or manual intervention; (5) it provides a conceptually appealing approach to kernel learning, where we reason directly about prior and posterior kernels, rather than about parameters of these kernels.

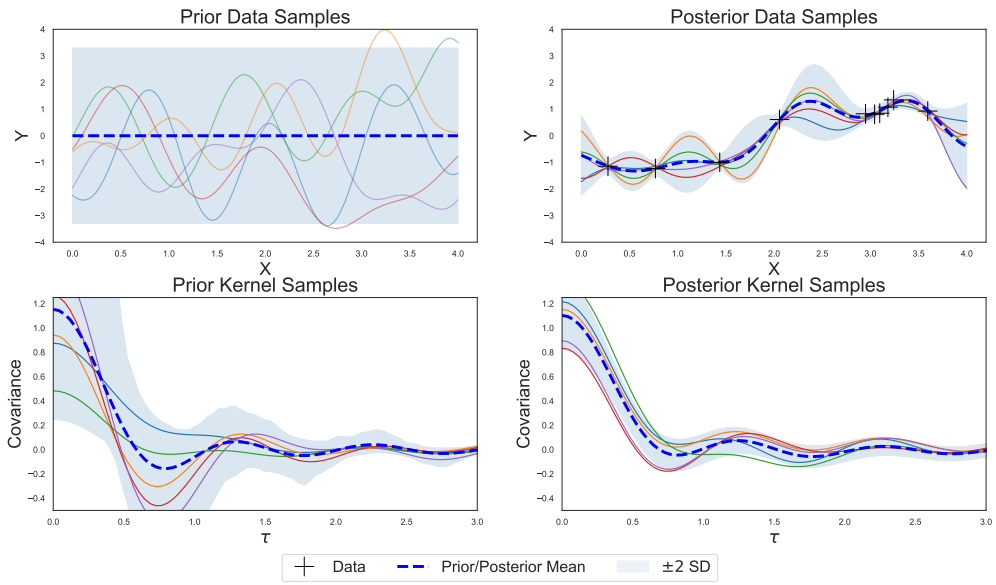


Figure 2.1: Above: A function-space view of regression on data. We show draws from a GP prior and posterior over functions in the left and right panels, respectively. **Below:** With FKL, we apply the function-space view to *kernels*, showing prior kernel draws on the left, and posterior kernel draws on the right. In both cases, prior and posterior means are in thick black, two standard deviations about the mean in grey shade, and data points given by crosses. With FKL, one can specify the prior mean over kernels to be any parametric family, such an RBF kernel, to provide a useful *inductive bias*, while still containing support for *any* stationary kernel.

- We further develop FKL to handle multidimensional and irregularly spaced data, and multi-task learning.

- We demonstrate the effectiveness of FKL in a wide range of settings, including interpolation, extrapolation, and kernel recovery experiments, demonstrating strong performance compared to state-of-the-art methods.

Our work is intended as a step towards developing Gaussian processes for *representation learning*. By pursuing a function-space approach to kernel learning, we can discover rich representations of data, enabling strong predictive performance, and new interpretable insights into our modeling problems.

2.2 Functional Kernel Learning Related Work

We assume some familiarity with Gaussian processes [e.g., Rasmussen and Williams 2006]. A vast majority of kernels and kernel learning methods are parametric. Popular kernels include the parametric RBF, Matérn, and periodic kernels. The standard multiple kernel learning [Genton 2001; Gönen and Alpaydın 2011; Lanckriet et al. 2004; Rakotomamonjy et al. 2007] approaches typically involve additive compositions of RBF kernels with different bandwidths. More recent methods model the spectral density (the Fourier transform) of stationary kernels to construct kernel learning procedures. Lázaro-Gredilla et al. [2010] models the spectrum as independent point masses. Wilson and Adams [2013] models the spectrum as a scale-location mixture of Gaussians, referred to as a *spectral mixture kernel* (SM). Yang et al. [2015] combine these approaches, using a random feature expansion for a spectral mixture kernel, for scalability. Oliva et al. [2016] consider a Bayesian non-parametric extension of Yang et al. [2015], using a random feature expansion for a Dirichlet process mixture. Alternatively, Jang et al. [2017] model the parameters of a SM kernel with prior distributions, and infer the number of mixture components. While these approaches provide strong performance improvements over standard kernels, they often struggle with difficulty specifying a prior expectation over the value of the kernel, and multi-modal learning objectives, requiring sophisticated manual intervention and initialization procedures [Herlands et al. 2018].

A small collection of pioneering works [Tobar 2018; Tobar et al. 2015; Wilson 2014b] have considered various approaches to modeling the spectral density of a kernel with a Gaussian process. Unlike FKL, these methods are constrained to one-dimensional time series, and still require significant intervention to achieve strong performance, such as choices of windows for convolutional kernels. Moreover, we demonstrate that even in this constrained setting, FKL provides improved performance over these state-of-the-art methods.

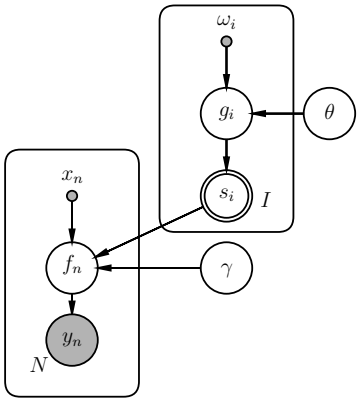


Figure 2.2: Graphical model for the FKL framework. Observed data is y_n, corresponding to the GP output f_n. The spectral density is S_i for observed frequencies ω_i., and hyper-parameters are $\phi = \{\theta, \gamma\}$.

2.3 Functional Kernel Learning

In this section, we introduce the prior model for *functional kernel learning* (FKL). FKL induces a distribution over kernels by modeling a spectral density (Section 2.3.1) with a transformed Gaussian process (Section 2.3.2). Initially we consider one dimensional inputs x and outputs y, and then generalize the approach to multiple input dimensions (Section 2.3.3), and multiple output dimensions (multi-task) (Section 2.3.4). We consider inference within this model in Section 2.8.2.

2.3.1 Spectral Transformations of Kernel Functions

Bochner's Theorem [Bochner 1959; Rasmussen and Williams 2006] specifies that $k(\cdot)$ is the covariance of a stationary process on $\mathbb{R}$ if and only if

$$k(\tau) = \int_{\mathbb{R}} e^{2\pi i \omega \tau} S(\omega) d\omega, \tag{2.1}$$

where $\tau = |x - x'|$ is the difference between any pair of inputs x and x', for a positive, finite *spectral density* $S(\omega)$. This relationship is reversible: if $S(\omega)$ is known, $k(\tau)$ can be computed via inverse Fourier transformation.

For $k(\tau)$ to be real-valued, $S(\omega)$ must be symmetric. Furthermore, for finitely sampled τ we are only able to identify angular frequencies up to $2\pi/\Delta$ where Δ is the minimum absolute difference between any two inputs. Equation 2.1 simplifies to

$$k(\tau) = \int_{[0,2\pi/\Delta)} \cos(2\pi\tau\omega)S(\omega)d\omega, \tag{2.2}$$

by expanding the complex exponential and using the oddness of sine (see Eqs. 4.7 and 4.8 in Rasmussen and Williams [2006]) and then truncating the integral to the point of identifiability.

For an arbitrary function, $S(\omega)$, Fourier inversion does not produce an analytic form for $k(\tau)$, however we can use simple numerical integration schemes like the trapezoid rule to approximate the integral in Equation 2.2 as

$$k(\tau) \approx \frac{\Delta_\omega}{2} \sum_{i=1}^{I} \cos(2\pi\tau\omega_i)S(\omega_i) + \cos(2\pi\tau\omega_{i-1})S(\omega_{i-1}), \tag{2.3}$$

where the spectrum is sampled at I evenly spaced frequencies ω_i that are Δ_ω units apart in the frequency domain.

The covariance $k(\tau)$ in Equation (2.3) is periodic. In practice, frequencies can be chosen such that the period is beyond the bounds that would need to be evaluated in τ. As a simple heuristic we choose P to be $8\tau_{max}$, where τ_{max} is the maximum distance between training inputs. We then choose frequencies so that $\omega_n = 2\pi n/P$ to ensure $k(\tau)$ is P-periodic. We have found choosing 100 frequencies ($n = 0, \ldots, 99$) in this way leads to good performance over a range of experiments in Section 2.5.

2.3.2 Specification of Latent Density Model

Uniqueness of the relationship in Equation 2.1 is guaranteed by the Wiener-Khintchine Theorem (see Eq. 4.6 of Rasmussen and Williams [2006]), thus learning the spectral density of a kernel is sufficient to learn the kernel. We propose modeling the log-spectral density of kernels using GPs. The log-transformation ensures that the spectral representation is non-negative. We let $\phi = \{\theta, \gamma\}$ be the set of *all* hyper-parameters (including those in both the data, γ, and latent spaces, θ), to simplify the notation of Section 2.8.2.

Using Equation 2.3 to produce a kernel $k(\tau)$ through $S(\omega)$, the hierarchical model over the data is

$$\begin{aligned}
&\{\text{Hyperprior}\} && p(\phi) = p(\theta, \gamma) \\
&\{\text{Latent GP}\} && g(\omega)|\theta \sim \mathcal{GP}\left(\mu(\omega;\theta), k_g(\omega, \omega';\theta)\right) \\
&\{\text{Spectral Density}\} && S(\omega) = \exp\{g(\omega)\} \\
&\{\text{Data GP}\} && f(x_n)|S(\omega), \gamma \sim \mathcal{GP}(\gamma_0, k(\tau; S(\omega))).
\end{aligned} \tag{2.4}$$

We let $f(x)$ be a noise free function that forms part of an observation model. For regression, we can let $y(x) = f(x) + \epsilon(x)$, $\epsilon \sim \mathcal{N}(0, \alpha^2)$ (in future equations we implicitly condition on hyper-parameters of the noise model, e.g., α^2, for succinctness, but learn these as part of ϕ). The approach can easily be adapted to classification through a different observation model; e.g., $p(y(x)) = \sigma(y(x)f(x))$ for binary classification with labels $y \in \{-1, 1\}$. Full hyper-parameter prior specification is given in Appendix A.1.2. Note that unlike logistic Gaussian process density estimators [Adams et al. 2009; Tokdar and Ghosh 2007] we need not worry about the normalization factor of $S(\omega)$, since it is absorbed by the scale of the kernel over data, $k(0)$. The hierarchical model in Equation 2.4 defines the functional kernel learning (FKL) prior, with corresponding graphical model in Figure 2.2. Figure 2.3 displays the hierarchical model, showing the connection between spectral and data spaces.

A compelling feature of FKL is the ability to conveniently specify a prior expectation for the

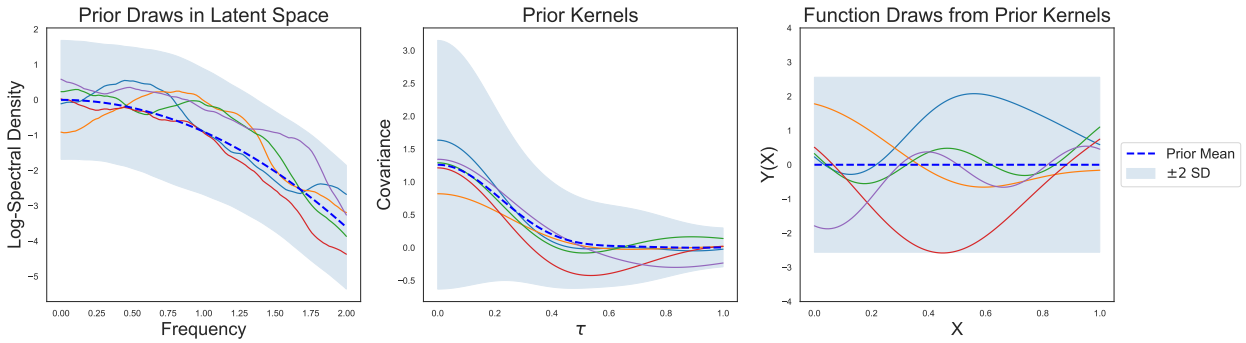


Figure 2.3: Forward sampling from the hierarchical FKL model of Equation (2.4). **Left**: Using randomly initialized hyper-parameters ϕ, we draw functions $g(\omega)$ from the latent GP modeling the log spectral density. **Center**: We use the latent realizations of $g(\omega)$ with Bochner's Theorem and Eq. (2.3) to compose kernels. **Right**: We sample from a mean-zero Gaussian process with a kernel given by each of the kernel samples. Shaded regions show 2 standard deviations above and below the mean in dashed blue. Notice that the shapes of the prior kernel samples have significant variation but are clearly influenced by the prior mean, providing a controllable inductive bias.

kernel by specifying a mean function for $g(\omega)$, and to encode smoothness assumptions by the choice of covariance function. For example, if we choose the mean of the latent process $g(\omega)$ to be negative quadratic, then prior kernels are concentrated around RBF kernels, encoding the inductive bias that function values close in input space are likely to have high covariance. In many cases the spectral density contains sharp peaks around dominant frequencies, so we choose a Matérn 3/2 kernel for the covariance of $g(\omega)$ to capture this behaviour.

2.3.3 Multiple Input Dimensions

We extend FKL to multiple input dimensions by either corresponding each one-dimensional kernel in a product of kernels with its own latent GP with distinct hyper-parameters (FKL separate) or having all one-dimensional kernels be draws from a single latent process with one set of hyper-parameters (FKL shared). The hierarchical Bayesian model over the d dimensions is described in the following manner:

$$
\begin{aligned}
&\{\text{Hyperprior}\} && p(\phi) = p(\theta, \gamma) \\
&\{\text{Latent GP } \forall d \in \{1, \ldots D\}\} && g_d(\omega_d)|\theta \sim \mathcal{GP}\left(\mu(\omega_d; \theta), k_{g_d}(\omega_d, \omega_d'; \theta)\right) \\
&\{\text{Product Kernel GP}\} && f(x)|\{g_d(\omega_d)\}_{d=1}^{D}, \gamma \sim \mathcal{GP}(\gamma_0, \prod_{d=1}^{D} k(\tau_d; S(\omega_d)))
\end{aligned}
\tag{2.5}
$$

Tying the kernels over each dimension while considering their spectral densities to be draws from the same latent process (FKL shared) provides multiple benefits. Under these assumptions, we have more information to learn the underlying latent GP $g(\omega)$. We also have the helpful inductive bias that the covariance functions across each dimension have some shared high-order properties, and enables linear time scaling with dimensionality.

2.3.4 Multiple Output Dimensions

FKL additionally provides a natural way to view multi-task GPs. We assume that each task (or output), indexed by $t \in \{1, \ldots, T\}$, is generated by a GP with a distinct kernel. The kernels are tied together by assuming each of those T kernels are constructed from realizations of a single *shared* latent GP. Notationally, we let $g(\omega)$ denote the latent GP, and use subscripts $g_t(\omega)$ to indicate independent realizations of this latent GP. The hierarchical model can then be described in the following manner:

$$
\begin{aligned}
&\{\text{Hyperprior}\} && p(\phi) = p(\theta, \gamma) \\
&\{\text{Latent GP}\} && g(\omega)|\theta \sim \mathcal{GP}\left(\mu(\omega; \theta), k_g(\omega, \omega'; \theta)\right) \\
&\{\text{Task GP } \forall t \in \{1, \ldots T\}\} && f_t(x)|g_t(\omega), \gamma \sim \mathcal{GP}(\gamma_{0,t}, k(\tau; S_t(\omega)))
\end{aligned}
\tag{2.6}
$$

In this setup, rather than having to learn the kernel from a single realization of a process (a single task), we can learn the kernel from multiple realizations, which provides a wealth of information for kernel learning [Wilson et al. 2015]. While sharing individual hyper-parameters across multiple tasks is standard (see e.g. Section 9.2 of MacKay [1998]), these approaches can only learn

limited structure. The information provided by multiple tasks is distinctly amenable to FKL, which shares a flexible *process over kernels* across tasks. FKL can use this information to discover unconventional structure in data, while retaining computational efficiency (see Appendix A.1.1).

2.4 Inference and Prediction

When considering the hierarchical model defined in Equation 2.4, one needs to learn both the hyper-parameters, ϕ, and an instance of the latent Gaussian process, $g(\omega)$. We employ alternating updates in which the hyper-parameters ϕ and draws of the latent GP are updated separately. A full description of the method is in Algorithm 2 in Appendix A.1.2.

Updating Hyper-Parameters: Considering the model specification in Eq. 2.4, we can define a loss as a function of $\phi = \{\theta, \gamma\}$ for an observation of the density, $\tilde{g}(\omega)$, and data observations $y(x)$. This loss corresponds to the entropy, marginal log-likelihood of the latent GP with fixed data GP, and the marginal log-likelihood of the data GP.

$$\mathcal{L}(\phi) = -\left(\log p(\phi) + \log p(\tilde{g}(\omega)|\theta, \omega) + \log p(y(x)|\tilde{g}(\omega), \gamma, x)\right). \tag{2.7}$$

This objective can be optimized using any procedure; we use the AMSGRAD variant of Adam as implemented in PyTorch [Reddi et al. 2019]. For GPs with D input dimensions (and similarly for D output dimensions), we extend Eq. 2.7 as

$$\mathcal{L}(\phi) = -\left(\log p(\phi) + \sum_{d=1}^{D} \left[\log p(\tilde{g}_d(\omega_d)|\theta, \omega)\right] + \log p(y(x)|\{\tilde{g}_d(\omega_d)\}_{d=1}^{D}, \gamma, x)\right). \tag{2.8}$$

Updating Latent Gaussian Process: With fixed hyper-parameters ϕ, the posterior of the latent GP is

$$p(g(\omega)|\phi, x, y(x), f(x)) \propto \mathcal{N}(\mu(\omega; \theta), k_g(\omega; \theta))p(f(x)|g(\omega), \gamma). \tag{2.9}$$

We sample from this posterior using elliptical slice sampling (ESS) [Murray et al. 2010; Murray and Adams 2010], which is specifically designed to sample from posteriors with highly correlated Gaussian priors. Note that we must reparametrize the prior by removing the mean before using ESS; we then consider it part of the likelihood afterwards.

Taken together, these two updates can be viewed as a single sample Monte Carlo expectation maximization (EM) algorithm [Wei and Tanner 1990] where only the final $g(\omega)$ sample is used in the Monte Carlo expectation. Using the alternating updates (following Algorithm 2) and transforming the spectral densities into kernels, samples of predictions on the training and testing data can be taken. We generate posterior estimates of kernels by fixing ϕ after updating and drawing samples from the posterior distribution, $p(g(\omega)|f, y, \phi)$, taken from ESS (using y as short for $y(x)$, the training data indexed by inputs x).

Prediction: The predictive distribution for any test input x^* is given by

$$p(f^*|x^*, x, y, \phi) = \int p(f^*|x^*, x, y, \phi, k)p(k|x^*, x, y, \phi)dk \tag{2.10}$$

where we are only conditioning on data x, y, and hyper-parameters ϕ determined from optimization, by *marginalizing* the whole posterior distribution over kernels k given by FKL. We use simple Monte Carlo to approximate this integral as

$$p(f^*|x^*, x, y, \phi) \approx \frac{1}{J}\sum_{j=1}^{J} p(f^*|x^*, x, y, \phi, k_j), \quad k_j \sim p(k|x^*, x, y, \phi). \tag{2.11}$$

We sample from the posterior over $g(\omega)$ using elliptical slice sampling as above. We then transform these samples $S(\omega) = \exp\{g(\omega)\}$ to form posterior samples from the spectral density. We then sample $k_j \sim p(k|x^*, x, y, \phi)$ by evaluating the trapezoidal approximation in Eq. (2.3) (at a collection of frequencies ω) for each sample of the spectral density. For regression with Gaussian

noise $p(f^*|x^*, x, y, \phi, k)$ is Gaussian, and our expression for the predictive distribution becomes

$$
\begin{aligned}
p(f^*|x^*, x, y, \phi, \omega) &= \frac{1}{J}\sum_{j=1}^{J} \mathcal{N}(\bar{f}^*(x^*)_j, \mathrm{Cov}(f^*)_j) \\
\bar{f}^*(x^*)_j &= k_{f_j}(x^*, x; \gamma) k_{f_j}(x, x; \theta)^{-1} y \\
\mathrm{Cov}(f^*)_j &= k_{f_j}(x^*, x^*; \gamma) - k_{f_j}(x^*, x; \gamma) k_{f_j}(x, x; \theta)^{-1} k_{f_j}(x, x^*; \gamma),
\end{aligned}
\tag{2.12}
$$

where k_{f_j} is the kernel associated with sample g_j from the posterior over g after transformation to a spectral density and then evaluation of the trapezoidal approximation (suppressing dependence on ω used in Eq. (2.3)). y is an $n \times 1$ vector of training data. $k_{f_j}(x, x; \theta)$ is an $n \times n$ matrix formed by evaluating k_{f_j} at all pairs of n training inputs x. Similarly $k_{f_j}(x^*, x^*; \theta)$ is a scalar and $k_{f_j}(x^*, x)$ is $1 \times n$ for a single test input x^*. This distribution is a mixture of Gaussians with J components. Following the above procedure, we obtain J samples from the unconditional distribution in Eq. (2.12). We can compute the sample mean for point predictions and twice the sample standard deviation for a credible set. Alternatively, we can use the mixture of Gaussians representation in conjunction with the laws of total mean and variance to approximate the moments of the predictive distribution in Eq. (2.12), which is what we do for the experiments.

2.5 Experiments

We demonstrate the practicality of FKL over a wide range of experiments: (1) recovering known kernels from data (Section 2.5.1); (2) extrapolation (Section 2.5.2); (3) multi-dimensional inputs and irregularly spaced data (section 2.5.3); (4) multi-task precipitation data (Section 2.5.4); and (5) multidimensional pattern extrapolation (Section 2.5.5). We compare to the standard RBF and Matérn kernels, as well as spectral mixture kernels [Wilson and Adams 2013], and the Bayesian nonparametric spectral estimation (BNSE) of Tobar [2018].

For FKL experiments, we use $g(\omega)$ with a negative quadratic mean function (to induce an

RBF-like prior mean in the distribution over kernels), and a Matérn kernel with $\nu = \frac{3}{2}$ (to capture the typical sharpness of spectral densities). We use the heuristic for frequencies in the trapezoid rule described in Section 2.3.1. Using $J = 10$ samples from the posterior over kernels, we evaluate the sample mean and twice the sample standard deviation from the unconditional predictive distribution in Eq. (2.12) for point predictions and credible sets. We perform all experiments in GPyTorch [Gardner et al. 2018a].

2.5.1 Recovery of Spectral Mixture Kernels

Here we test the ability of FKL to recover known ground truth kernels. We generate 150 data points, $x_i \sim U(-7., 7)$ randomly and then draw a random function from a GP with a two component spectral mixture kernel with weights 1 and 0.5, spectral means of 0.2 and 0.9 and standard deviations of 0.05. As shown in Figure 2.4, FKL accurately reconstructs the underlying spectral density, which enables accurate in-filling of data in a held out test region, alongside reliable credible sets. A GP with a spectral mixture kernel is suited for this task and closely matches with withheld data. GP regression with the RBF or Matérn kernels is unable to predict accurately very far from the training points. BNSE similarly interpolates the training data well but performs poorly on the extrapolation region away from the data. In Appendix A.1.5.1 we illustrate an additional kernel recovery experiment, with similar results.

2.5.2 Interpolation and Extrapolation

Airline Passenger Data We next consider the airline passenger dataset [Hyndman 2005] consisting of 96 monthly observations of numbers of airline passengers from 1949 to 1961, and attempt to extrapolate the next 48 observations. We standardize the dataset to have zero mean and unit standard deviation before modeling. The dataset is difficult for Gaussian processes with standard stationary kernels, due to the rising trend, and difficulty in extrapolating quasi-periodic

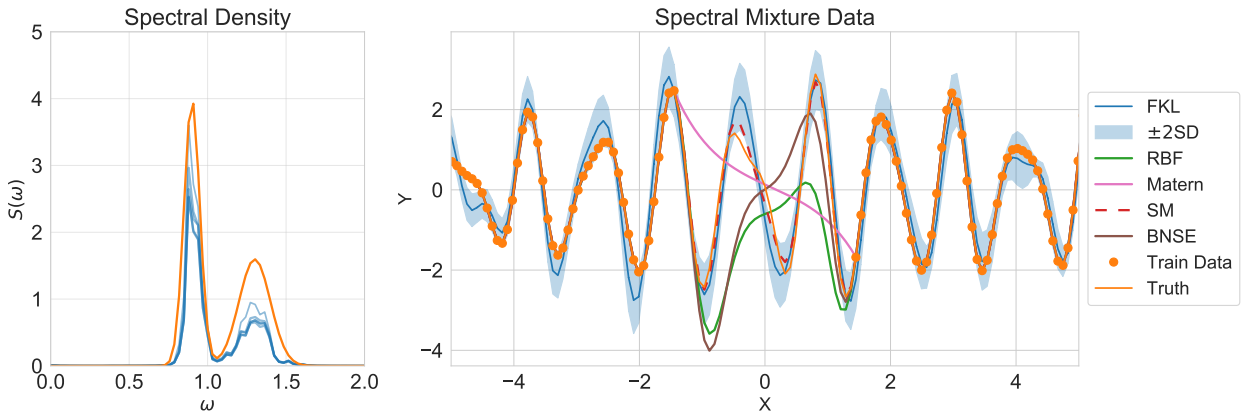


Figure 2.4: Left: Samples from the FKL posterior over the spectral density capture the shape of the true spectrum. **Right**: Many of the FKL predictions on the held out data are nearly on par with the ground-truth model (SM in dashed red). GPs using the other kernels perform poorly on extrapolation away from the training points.

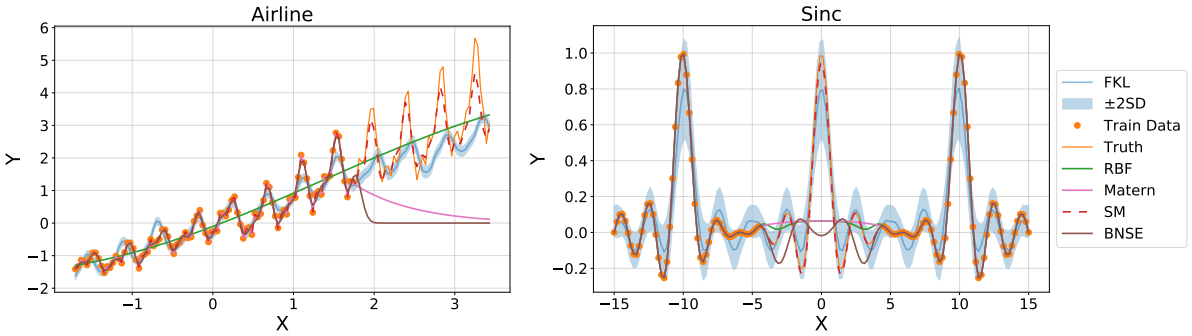
structure.

Sinc We model a pattern of three sinc functions replicating the experiment of Wilson and Adams [2013]. Here $y(x) = \text{sinc}(x+10) + \text{sinc}(x) + \text{sinc}(x-10)$ with $\text{sinc}(x) = \sin(\pi x)/(\pi x)$. This has been shown previously [Wilson and Adams 2013] to be a case for which parametric kernels fail to pick up on the correct periodic structure of the data.

Figures 2.5(a) and 2.5(b) show that FKL outperforms simple parametric kernels on complex datasets. Performance of FKL is on par with that of SM kernels while requiring less manual tuning and being more robust to initialization.

2.5.3 Multiple Dimensions: Interpolation on UCI datasets

We use the product kernel described in Section 2.5.3 with both separate and shared latent GPs for regression tasks on UCI datasets. Figure 2.6 visually depicts the model with respect to prior and posterior products of kernels. We standardize the data to zero mean and unit variance and randomly split the training and test sets, corresponding to 90% and 10% of the full data, respectively. We conduct experiments over 10 random splits and show the average RMSE and standard



(a) Extrapolation on the airlines dataset [Hyndman 2005].

(b) Interpolation on the sinc function.

Figure 2.5: (a): Extrapolation on the airline passenger dataset. **(b)**: Prediction on sinc data. FKL is on par with a carefully tuned SM kernel (dashed pink) in **(a)** and shows best performance in **(b)**, BNSE (brown) performs well on the training data, but quickly reverts to the mean in the testing set.

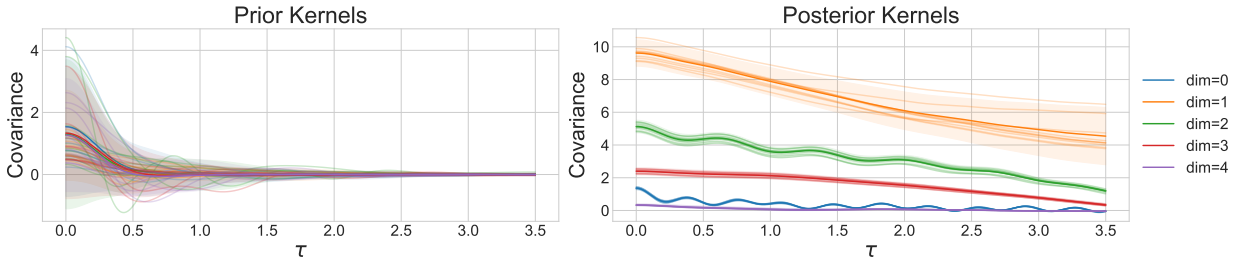


Figure 2.6: Samples of prior **(a)** and posterior **(b)** kernels displayed alongside the sample mean (thick lines) and ± 2 standard deviations (shade). Each color corresponds to a kernel, $k(\cdot)$, for a dimension of the airfoil dataset.

deviation. We compare to the RBF, ARD, and ARD Matérn. Furthermore, we compare the results of sharing a single latent GP across the kernels of the product decomposition(Eq. 2.5) with independent latent GPs for each kernel in the decomposition.

2.5.4 Multi-Task Extrapolation

We use the multi-task version of FKL in Section 2.3.4 to model precipitation data sourced from the United States Historical Climatology Network [Menne et al. 2015]. The data contain daily precipitation measurements over 115 years collected at 1218 locations in the US. Average positive precipitation by day of the year is taken for three climatologically similar recording locations in

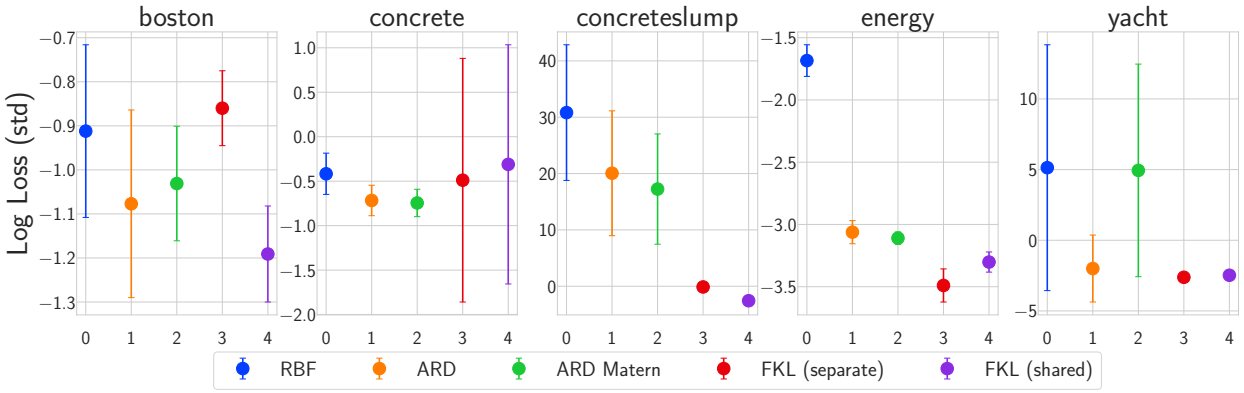


Figure 2.7: Standardized log losses on five of the 12 UCI datasets used. Here, we can see that FKL typically outperforms parametric kernels, even with a shared latent GP. See Table A.2 for the full results in the Appendix.

Colorado: Boulder, Telluride, and Steamboat Springs, as shown in Figure 2.8. The data for these locations have similar seasonal variations, motivating a shared latent GP across tasks, with a flexible kernel process capable of learning this structure. Following the procedure outlined in Section 2.8.2 and detailed in Algorithm 3 in the Appendix, FKL provides predictive distributions that accurately interpolates and extrapolates the data with appropriate credible sets. In Appendix A.1.6 we extend these multi-task precipitation results to large scale experimentation with datasets containing tens of thousands of points.

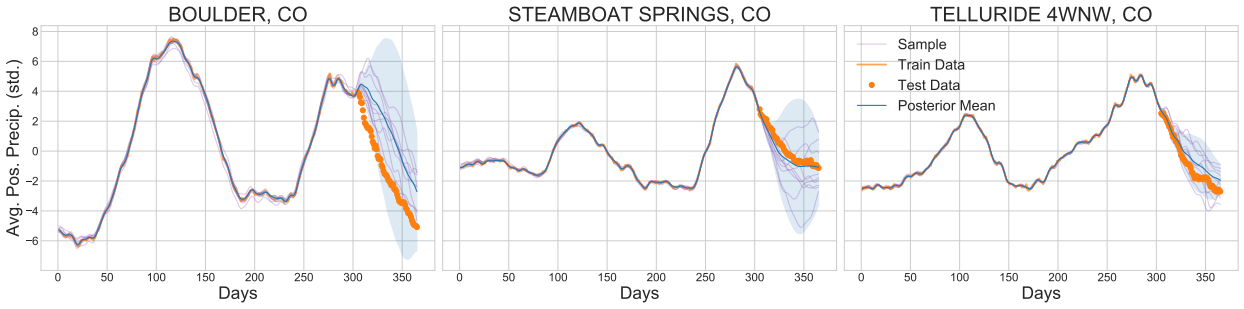


Figure 2.8: Posterior predictions generated using latent GP samples. 10 samples of the latent GP for each site are used to construct covariance matrices and posterior predictions of the GPs over the data.

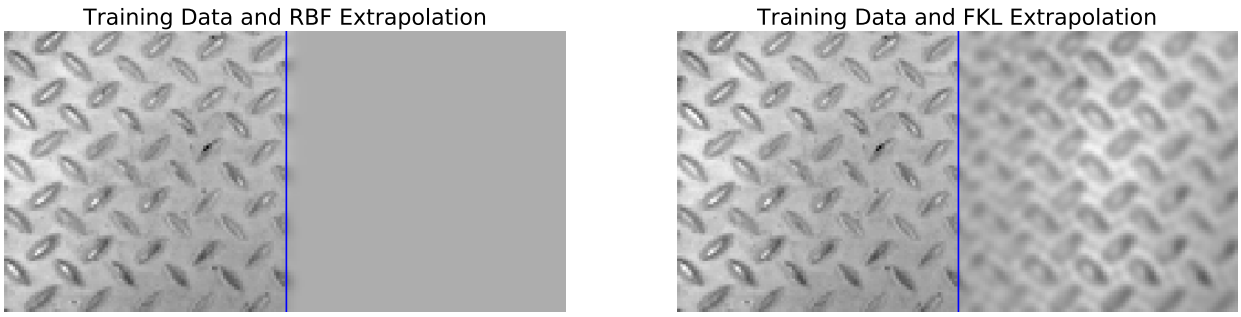


Figure 2.9: Texture Extrapolation: training data is shown to the left of the blue line and predicted extrapolations according to each model are to the right.

2.5.5 Scalability and Texture Extrapolation

Large datasets typically provide additional information to learn rich covariance structure. Following the setup in [Wilson et al. 2014], we exploit the underlying structure in images and scale FKL to learn such a rich covariance — enabling extrapolation on textures. When the inputs, X, form a Cartesian product multidimensional grid, the covariance matrix decomposes as the Kronecker product of the covariance matrices over each input dimension, i.e. $K(X,X) = K(X_1,X_1) \otimes K(X_2,X_2) \otimes \cdots \otimes K(X_P,X_P)$ where X_i are the elements of the grid in the i^{th} dimension [Saatçi 2012]. Using the eigendecompositions of Kronecker matrices, solutions to linear systems and log determinants of covariance matrices that have Kronecker structure can be computed exactly in $O(PN^{P/2})$ time, instead of the standard cubic scaling in N [Wilson et al. 2014].

We train FKL on a 10,000 pixel image of a steel tread-plate and extrapolate the pattern beyond the training domain. As shown in Figure 2.9, FKL uncovers the underlying structure, with no sophisticated initialization procedure. While the spectral mixture kernel performs well on these tasks [Wilson et al. 2014], it requires involved initialization procedures. By contrast, standard kernels, such as the RBF kernel, are unable to discover the covariance structure to extrapolate on these tasks.

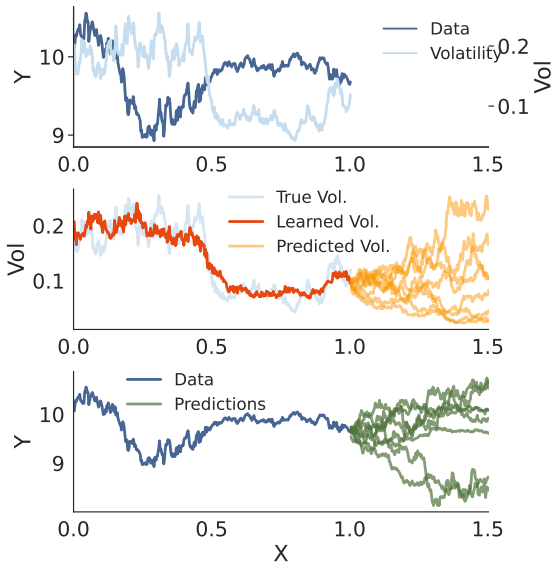


Figure 2.10: An overview of stochastic volatility, Volt, and forecasting. **Top:** the observed data over and the corresponding volatility path. **Middle:** the learned volatility from the data, and volatility forecasts. **Bottom:** the data over 1 year and forecasts, with each sample path corresponding to a distinct sample from the volatility forecast.

2.6 Volatility Based Kernels and Moving Average Means for Accurate Forecasting with Gaussian Processes

With Functional Kernel Learning we introduced one method for forming distributions over kernel functions in Gaussian process (GP) models. While FKL is an effective method for GP prediction problems, it relies on making minimal assumptions regarding the kernel and instead opts for providing support over all stationary covariance functions. In the following sections we introduce *Volt* as a method for forming distributions over GP kernels, but rather than begin with the highly flexible and kernel-agnostic approach of FKL, with Volt we begin with widely used stochastic volatility models and work backwards to derive an associated Gaussian process model to be used for prediction. In this way, Volt provides a principled approach to GP modeling that is grounded in the underlying stochastic volatility model.

Both financial and climatological time series are nonstationary, and are characterized by hav-

ing time-varying and stochastic *volatilities*, or degrees of variation, making them compelling use cases for Volt. Volt uses forecasts of volatility to specify the covariance structure over future data observations. By considering not only a single volatility forecast, but a distribution of volatility forecasts, we induce a distribution over covariance functions in the data domain. Accounting for uncertainty in volatility and propagating it to our data forecasts yields projected distributions that are well calibrated to the data, providing critical tools for understanding risk levels and simulating potential outcomes. For further information on stochastic volatility see Appendix A.2.1.

Figure 2.10 provides a graphical representation of the hierarchical GP model described by Volt. Given a set of observations (top row) we infer a volatility path over those returns (middle row), and form a hierarchical GP model where the first GP models volatility, and the second GP is used to forecast distributions over the data given samples from the volatility GP (bottom row).

The covariance structure described by Volt provides a faithful representation of the uncertainty in forecasts, but overlooks the mean function of the data space GP, which is a powerful tool for capturing trends in data. To that end, we jointly introduce Moving Average Gaussian Processes, or *Magpie*, in which we replace the standard parametric mean function in GPs with a moving average. Moving averages are a widely used technique in domains such as climatology and finance [Nau 2014]. By joining the trend fitting capabilities of moving averages with the probabilistic framework of GPs we can produce forecasts that are both accurate and have calibrated uncertainties.

While Volt and Magpie can be used separately, we present them as a single work because it is specifically their *combination* that solves challenging forecasting problems. In time evolving domains like stock prices or wind speeds, the inherent randomness of the processes prevents us from producing accurate point estimates far out into the future, and we need just accuracy, but uncertainty that is faithful to the stochasticity of the data. For this reason one needs both the accurate trend capture provided by Magpie, and the accurate uncertainty representation provided

by Volt

Our key contributions are as follows:

- Deriving a hierarchical GP model, Volt, inspired by stochastic volatility models that produces calibrated forecasts of stochastic time series (Section 2.8).
- Describing a simple but powerful mean function, Magpie, that enables Gaussian process models to accurately forecast trends (Section 2.8).
- Using Volt and Magpie to produce highly calibrated forecasts in financial and climatological domains (Section 2.9).
- Extending our procedure to multitask problems by accounting for correlations in both volatility and price across different financial assets and different spatial locations (Section 2.10).

2.7 Volt Related Work

Early autoregressive approaches to modeling the volatility of time series returns such as GARCH have seen widespread success [Bollerslev 1986]. These approaches typically view the volatility process as a time-evolving series, and are effective for inferring and forecasting volatility, but do not typically interface directly with a model over data as we have with Volt.

Volatility models have been extended to use both neural networks or Gaussian processes as their base components. For example, Cao et al. [2020] use a multi-layer perceptron to estimate volatility surfaces while Luo et al. [2018] use RNNs with rollouts to forecast volatility into the future but only considered one-step lookahead price forecasts. Wilson and Ghahramani [2010] use Gaussian processes to parameterize the volatility using Laplace approximations and MCMC sampling introducing the Gaussian process copula volatility model (GPCV), while Wu et al. [2014] used GP state space models and particle filters to estimate volatility. Similar to

Wilson and Ghahramani [2010], Lázaro-Gredilla and Titsias [2011] used Gaussian processes to parameterize volatility models with an exponential link, but used a highly structured variational approximation for inference. Liu et al. [2020a] used multi-task Gaussian processes to forecast volatility into the future, applying their models to foreign exchange currency returns, again with one-step lookahead forecasts in price. Crucially, predicting volatility alone does yield a straightforward path to forecast data, which is our central aim with Volt. Furthermore, Volt builds off of the GPCV, but other volatility estimation methods such as the ones described here could also be used.

Stochastic volatility models such as the Heston model [Heston 1993] and SABR [Hagan et al. 2002], treat the evolution of the price of a security and the associated volatility as a coupled system of SDEs. Such SDEs are commonly used as methods for pricing financial derivatives. Differing from our viewpoint, these models are typically used to price stock options under risk-neutral measures, with Volt and Magpie we are focused on performing predictive inference by conditioning on observations.

The connection between Gaussian processes and SDEs has been extensively studied by Särkkä and Solin [2019] who suggest Kalman filtering based approaches for estimating GP hyperparameters in SDE-inspired GP models, which we do not consider here, preferring simply marginal likelihood based estimation. Systems of linear differential equations have been integrated into GP models previously via latent force models both for ordinary differential equations [Alvarez et al. 2009] and partial differential equations [Särkkä 2011]. To perform inference, Alvarez et al. [2009] derive covariances corresponding to the linear projection of the differential operator onto a specific kernel, while Särkkä [2011]; Särkkä and Solin [2019] use the projection operator explicitly to develop kernel functions to emulate systems of SDEs. Similarly, Zhu and Dunson [2013] use SDEs to derive a nested GP, but their approach produces a standard GP with a non-deep, but structured, covariance function. Autoregressive mean functions for GPs have been explored in Gonzalvez et al. [2019], however in their approach they use autoregressive features as inputs to

a GP model, rather than as a way to specify the prior functions.

While many of the references above are focused specifically on finance, Volt and Magpie are applicable to a broad set of domains including climate modeling. Autoregressive and volatility models have successfully been applied to domains such as wind and precipitation forecasting as in Mehdizadeh et al. [2020]; Liu et al. [2011] and Tian et al. [2018].

The Gaussian process autoregressive model [Requeima et al. 2019b] stacks Gaussian processes of different tasks, using the GP for one task as the mean function for the next. It thus bears only slight resemblance to our moving average or multi-task approaches. Furthermore, many well-studied autoregressive models, e.g. the AR(p) family, can be written as Gaussian processes [Whilliams 2010]. As an alternative to developing domai n specific kernel functions, one could alternatively construct manual combinations of generic kernels, which either requires significant amounts of hand-tuning as in Rasmussen and Williams [2008, Ch 5.4,] or solving discrete optimization problems [Lloyd et al. 2014; Sun et al. 2018]. As we wish to develop our models efficiently and succinctly, we also do not consider these models.

2.8 Methods

We first begin with a brief overview of Gaussian process regression models, before deriving the Volt kernel and Magpie mean functions in Section 2.8.1. After deriving the Volt kernel and Magpie mean, we explain the inference procedure in Section 2.8.2 and how we perform forecasting in Section 2.8.3.

Gaussian Processes Please see Rasmussen and Williams [2008] for a more detailed introduction to Gaussian processes (GPs). We assume noisy observations $y(t) \sim \mathcal{N}(f(t), \sigma^2)$, where $f \sim \mathcal{GP}(\mu(t), k(t, t'))$, so that σ is the observation noise and f is drawn from a GP with mean function $\mu(t)$ and $k(t, t')$ as the covariance function. When using GPs, we can compute the posterior

predictive distribution, $p(f(\mathbf{t}^*)|\mathcal{D})$, $\mathcal{D} := \{\mathbf{t}, \mathbf{y}\}$, over new data points $\mathbf{t}^*$ is given by $p(f(\mathbf{t}^*)|\mathcal{D}, \theta) = \mathcal{N}(\mu^*_{f|\mathcal{D}}, \Sigma^*_{f|\mathcal{D}})$ where $\mu^*_{f|\mathcal{D}} = K_{\mathbf{t}^*\mathbf{t}}(K_{\mathbf{tt}}+\sigma^2 I)^{-1}(\mathbf{y}-\mu(\mathbf{t}))+\mu(\mathbf{t}^*)$ and $\Sigma^*_{f|\mathcal{D}} = K_{\mathbf{t}^*\mathbf{t}^*}-K_{\mathbf{t}^*\mathbf{t}}(K_{\mathbf{tt}}+\sigma^2 I)^{-1}K_{\mathbf{tt}^*}$ with $K_{A,B} := k(A, B)$.

2.8.1 Volt and Magpie

We make the common assumption that both the data $S(t)$, and volatility $V(t)$, have paths with log-normal marginal distributions. We therefore place the following joint SDE structure over $s(t) = \log S(t)$ and $v(t) = \log V(t)$,

$$\begin{aligned} ds(t) &= \mu_s dt + V(t)dW(t) \\ dv(t) &= -\frac{\sigma^2}{2}dt + \sigma dZ(t). \end{aligned} \tag{2.13}$$

The drift term in Equation (2.13), $-\frac{\sigma^2}{2}dt$, arises from the log-transformation of the volatility, and ensures that forecast distributions over volatility have a constant mean (for further details see Appendix A.2.2.2). Furthermore, this structure allows us to derive closed form expressions for and auto-covariance functions associated with both log-data and log-volatility, allowing us to define the Volt model.

Equation (2.13) gives a relationship between the log-price and log-volatility that is mirrored by many stochastic volatility models, including GARCH and SABR, where the volatility of the price is itself governed by an SDE [Bollerslev 1986; Hagan et al. 2002]. By recasting Equation (2.13) as a system of GPs we can move from an SDE sampling approach to a proper forecasting system based on historical observations.

A Gaussian Process Perspective Since for any finite collection of time points, $\mathbf{t} = \{t_i\}_{i=1}^N$, the observations $v = v(\mathbf{t})$ and $s = s(\mathbf{t})$ each have a multivariate normal distribution, v and s now correspond to Gaussian processes. Therefore we only need to derive the mean and covariance

functions of the two processes to fully cast our problem as one of forming predictive distributions from GPs.

As $v(t)$ is a scaled Wiener process with constant drift term, the autocovariance function is

$$K_v(t, t') = \sigma^2 \min\{t, t'\} \tag{2.14}$$

and the mean is $\mu_v(t) = -t\frac{\sigma^2}{2}$ so that, $v(t) \sim \mathcal{GP}(\mu_v(t), K_v(t, t'))$. Conditional on a realization of $V(t) = \exp v(t)$, $s(t)$ is also described by a Gaussian process with $\mathbb{E}[s(t)] = \int_0^t \mu_s dt = t\mu_s$ and ,

$$Cov(s(t), s(t')) = \int_0^{\min\{t,t'\}} V(t)^2 dt = K_s(t, t'; V(t)), \tag{2.15}$$

producing our model over log-data:

$$s(t) \sim \mathcal{GP}(t\mu_s + s(0), K_s(t, t'; V(t))). \tag{2.16}$$

The final Volt model is then a hierarchical composition of Gaussian processes:

$$\begin{aligned} v(t) &\sim \mathcal{GP}(m_v(t), K_v(t, t')) \\ V(t) &= \exp(v(t)) \\ s(t) &\sim \mathcal{GP}(m_s(t), K_s(t, t'; V(t))) \\ S(t) &= \exp(s(t)), \end{aligned} \tag{2.17}$$

The log-volatility is distributed as a Gaussian process dependent on the the time inputs, the mean m_v, and the *volvol* hyperparameter σ and has a Brownian motion covariance (Eq. 2.14). Given a realization of a volatility path over time and the parameters of the log-linear mean, the log-price is also distributed as a Gaussian process with covariance given by Eq. 2.15. To generate predictions using the log-volatility and log-price GPs we first must infer both a volatility path

from the observed time series, $S = S(\mathbf{t})$, and the hyperparameters of both the data and volatility models. A complete derivation of the GPs in Equation (2.17) is in Appendix A.2.2.2.

MAGPIE For the sake of deriving the covariance functions associated with the log-data and log-volatility processes, we have left the mean functions of the data GP in Equation (2.17) as a simple linear function. While we may believe that there are nontrivial trends in the data over time, we also believe that these trends may be more complex than simple polynomial or periodic functions, and in the context of applications like finance and climatology are likely to change over time with evolving market or climatological conditions.

To address these deficiencies in using simple mean functions in modeling nonstationary signals we replace the simple mean functions typically found in GP models with exponential moving averages (EMA) [Nau 2014]. We use the EMA with a limited number of terms, defined as

$$\begin{aligned} EMA(\mathbf{s})_{i+1} =& \alpha[s_i + (1-\alpha)s_{i-1} + (1-\alpha)^2 s_{i-2} \\ &+ \cdots + (1-\alpha)^{k-1} s_{i-(k-1)}] \end{aligned} \tag{2.18}$$

where $\alpha = 2/(k+1)$ is a hyperparameter governing the smoothing of the moving average. A smaller value of k uses only more recent observations, enabling a closer match of the data, whereas a larger value of k uses more data and smooths the data more.

While we focus on the EMA in Equation (2.18), Magpie naturally extends to alternate moving averages, such as lag-corrected moving averages. We provide comparisons of these alternate moving averages, as well as the effect of the k hyperparamter in Appendix Figure A.11 and in the extended results of Section 2.9. With Equation (2.18) we can define the Magpie mean function as $m_{EMA}(t_{i+1}, \mathbf{s}) = EMA(\mathbf{s})_{i+1}$.

We close this section by noting that moving from a linear to a exponential moving average mean for the GPs breaks the connection with the SDEs described in Section 2.8.1, making the combination of Volt and Magpie necessarily a practical approach, rather than an entirely theo-

retically motivated approach.

2.8.2 Inference

Here we outline the procedure for using a series of price observations to train the hyperparameters of the GPs in Equation 2.17, and form the associated posterior predictive distributions. In general the training procedure can be thought of as a three step process: a) use a Gaussian Process Copula Volatility (GPCV) model to infer a volatility path, V, given a sequence of observations S, b) learn the hyperparameters of the GP in log-volatility space by maximizing the Marginal Log-Likelihood (MLL) with respect to the GPCV inferred volatility, c) learn the hyperparameters of the GP in log-data space by maximizing the MLL with respect to the observed prices, using the kernel generated by the GPCV inferred volatility path. Note that our use of the GPCV to estimate volatility is a modelling choice and we could alternatively have used any other volatility estimation model such as GARCH.

Inferring Volatility from Training Data One challenge in formulating the model outlined in Equation (2.17) is the need to have both data and volatility observations for some range of training observations. To estimate the volatility, we use a variant of Gaussian copula process volatility (GPCV) model first proposed by Wilson and Ghahramani [2010]. Our GPCV model uses a warped Gaussian process to model the variability of the responses, $w(t)$, according to:

$$\begin{aligned} f(t) &\sim \mathcal{GP}(c, K_v(t, t')) \\ \gamma(f(t)) &= \exp\{f(t)\} \\ w(t) &\sim \mathcal{N}(0, \gamma^2(f(t))). \end{aligned} \tag{2.19}$$

We use the kernel derived from log-volatility SDE in Equation (2.17) to infer the latent function $f(t)$, and use variational inference [Hensman et al. 2013, 2015] to train the model. See Appendix A.2.2.3 for further details.

Following Wilson and Ghahramani [2010], we consider the responses as the log-returns of the data, that is: $w(t_i) = \log S(t_i) - \log S(t_{i-1})$. We construct a volatility prediction over times $0, \cdots, t-1$ by drawing posterior samples from $f(t)$ and passing them through the warping function $\sigma(\cdot)$; so our estimate for $V(t)$ is

$$\hat{V}(t) := \frac{1}{J} \sum_{j=1}^{J} \gamma(f_j(t)), \;\; f_j(t) \sim q(f(t)|w(t), \upsilon, \theta), \tag{2.20}$$

where $q(f(t)|w(t), \upsilon, \theta)$ is our approximate posterior distribution over the latent function $f(t)$. We demonstrate that our approach is able to correctly estimate the true volatility in Figure 2.10, where the volatility and price are drawn from a SABR volatility model [Hagan et al. 2002].

Training the Gaussian Processes Given the volatility path associated with the training data learned using a GPCV, we assume a Gaussian process priors over the log-volatility and log-data according to Equation (2.17). Given the volatility over the training data, the single hyperparameter of the log-volatility model is the σ^2 term describing the *volvol*. The hyperparameters of the log-data model are just the parameters of the mean in Equation (2.17), of which there are none if we are using a non-parametric mean like Magpie. To train we maximize the MLL of the models with respect to their hyperparameters using gradient based optimization [Rasmussen and Williams 2008, Chapter 5]. The total computational cost for inference in Volt, regardless of the use of a Magpie mean, is just the cost of training *one* variational GP and two standard GP models on evenly spaced data, which can be done efficiently via exploiting the (Toeplitz) structure of the data [Wilson and Nickisch 2015].

2.8.3 Predictions

In Volt, we condition the log-volatility GP on a log-path inferred by GPCV and the log-data GP on historical observations of log-price and draw samples from the *posterior* distributions, producing

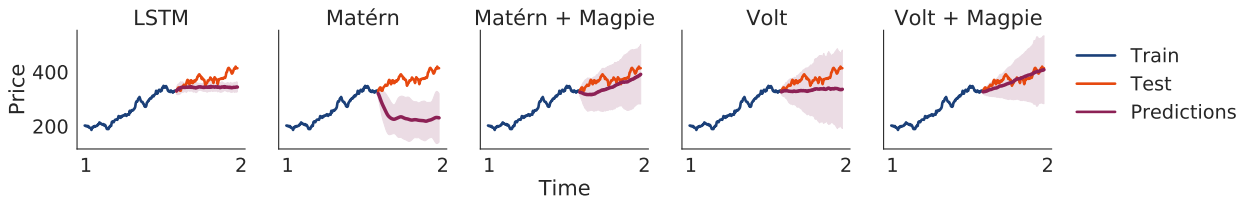


Figure 2.11: Simulations and forecasts showing the mean and 95% confidence region for various model choices. Probabilistic LSTMs perform well on the training data, but do not extrapolate far from observed data well. Matérn forecasts quickly revert to constant level of uncertainty which leads to overconfidence far away from observations, whereas Volt's increase in uncertainty as we move away from training data produces well calibrated to the data. The constant mean forecasts in both Matérn and Volt fail to pick up the long term trend in the data, which Magpie means accurately capture. The combination of Volt and Magpie, with correct inductive biases in both the kernel *and* mean functions produces forecasts consistent with trends in the data and with well calibrated uncertainty.

a mixture of log-normal distributions over data. Sampling the posterior requires sample N_v log-volatility paths, v^*, over the test inputs and for each of these we generate a kernel $K_s(t, t', V^*)$, and sample N_s data paths, $S^* = \exp(s^*)$, producing $N_v \times N_s$ samples.

In the Gaussian process viewpoint of Section 2.8.1, standard Monte Carlo simulation of an SDE procedure is equivalent to sampling log-volatility paths, $v^* = v(\mathbf{t}^*)$, from the *prior* distributions of Equation (2.17) up to time T rather than the *posterior* distribution [Sauer 2012]. With the prior samples of $S_T^* = \exp(s_T^*)$, we can form a Monte Carlo estimate of future distributions over price. However, the distinction between this type of approach and our approach for sampling with Volt is that Volt samples from the *posterior* distributions over volatility and data conditional on observations, while the SDE based approaches sample from the prior distribution over volatility.

Rollout Predictions The Magpie mean only allows for predictions one step ahead, so we do our forecasting in a *rollout* fashion. That is, we use observations $s_0, \ldots, s_t$ to sample $\hat{s}_{t+1}$ from the GP posterior $p(s_{t+1}|s_0, \ldots, s_t)$, then condition our GP (and Magpie mean) on $\hat{s}_{t+1}$ in order to sample $\hat{s}_{t+2}$ from the updated GP posterior $p(s_{t+2}|s_0, \ldots, s_t, \hat{s}_{t+1})$, and so on. These rollout forecasts are critical to the Magpie framework. By sequentially sampling the price forecasts and updating the GP with each observation we allow for trend reversals in the moving average mean in a way

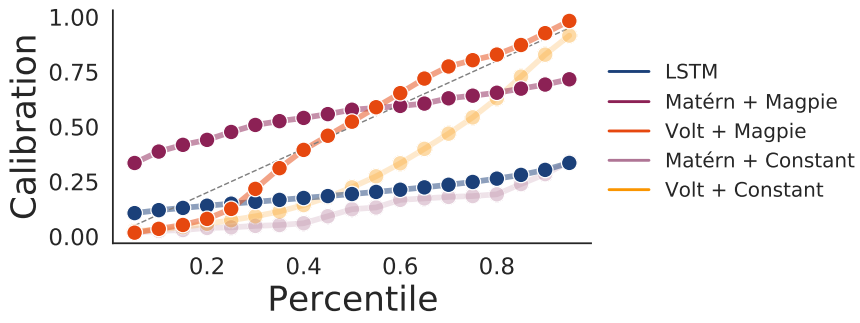


Figure 2.12: Calibration of various approaches on the 2 years of data from the NASDAQ 100. The forecasts generated by standard kernels and probabilistic LSTMs are significantly overconfident, leading to very poor calibration.

that is not possible with other GPs. Rollouts are unnecessary for traditional means because the conditional means over each time step factorize into a single multivariate Gaussian distribution.

2.9 Forecasting

In both financial and climatological applications we are considering the data as stochastically evolving and are thus interested in forecasting *distributions* over outcomes, rather than point estimates. For this reason we use *calibration* and negative log likelihood as our primary measures of interest, rather than an accuracy metric like mean squared error.

We compute the calibration at percentile p by computing the frequency with which the true observation is less than the empirically computed p^{th} quantile of the forecast distribution. More specifically, for a forecast of the price stock S at time T and percentile p we compute the empirical quantile of the forecast q_T where $\hat{\mathbb{P}}(S_T < q_T) = p$. We can then compute the calibration at p as the empirically observed frequency of the event $S_T < q_T$ by calculating $C_p = \frac{1}{K}\sum_k \mathbb{I}_{\{S_{T_k} < q_{T_k}\}}$ as the average frequency of $S_T < q_T$ over K different forecasts. If our forecasts are well calibrated then this empirical frequency will be close to p for each value of p; therefore by computing the calibration of our forecasts at a range of percentiles, p, we can determine the overall calibration of the forecast distribution. Such a calibration metric is similar to those explored for regression

in Kuleshov et al. [2018].

Note that for accurate calibration to occur in this setting our forecast distribution must match the empirical observations at all quantiles. We could not, for example, just forecast that a price increases some fixed percentage of the time that matches the observed frequency of the price increased and expect to achieve accurate calibration.

2.9.1 Stock Price and Foreign Exchange Rate Forecasting

As Volt and Magpie are primarily inspired by financial time series models, forecasting distributions over stock prices is a core application of our approach. We compare Volt and Magpie to baseline models of GPs with standard kernel and mean functions. Along with these GP models, we include probabilistic LSTMs where we optimize a predicted mean and variance at each time step with respect to the negative log-likelihood (NLL) which have been previously used in a quantitative finance setting [Chauhan et al. 2020]. All models assume the marginal distributions of the observations are normally distributed, thus we model the log-price of stocks in each case.

Figure 2.11 provides a representative comparison of forecasts generated by GPs both with and without Volt and Magpie, and the probabilistic LSTMs used here. Simpler probabilistic models like standard GPs and probabilistic neural networks generally provide overconfident forecasts, and more traditional mean functions in GP models do not capture the long range trends that are commonly present in financial time-series data.

Figure 2.12 shows the calibration of the compared methods aggregated over thousands of forecasts. We consider stocks in the NASDAQ 100 collection, and a history of two years of daily observations leading up to January 2022. For 25 evenly spaced days we forecast 1000 paths 100 days into the future and compute the calibration curves of the forecasts for the days 75 to 100 days out.

We see in Figure 2.12 that Volt is able to remedy a significant overconfidence that is present in alternative methods such as standard GP kernels or probabilistic LSTMs. Furthermore, it is

	Stock Prices	Wind Speeds
Volt + Magpie	5.88 ± 0.02	4.28 ± 0.16
Volt + Con.	**4.69 ± 0.03**	**3.38 ± 0.05**
Matérn + Magpie	9.80 ± 0.27	12.13 ± 0.81
Matérn + Con.	7.74 ± 0.21	18.03 ± 1.90
SM + Magpie	147.84 ± 1.84	110.07 ± 7.81
SM + Con.	80.43 ± 0.57	70.14 ± 5.03
LSTM	49.95 ± 0.59	45.13 ± 1.82
Volt-VHGP + Con.	4.76 ± 3.05	5.75 ± 0.44
Volt-VHGP + Magpie	6.97 ± 1.24	5.91 ± 0.34
GPCV	5.45 ± 1.51	4.89 ± 0.04

Table 2.1: Negative log likelihoods (NLLs) per test point with 2 standard deviations for the methods compared on both the stock forecasting and wind speed tasks. By accounting for uncertainty in both the volatility and the data forecasts, Volt provides highly accurate test distributions relative to baseline approaches. Volt-VHGP indicates a Volt model where we use variational heteroscedastic GPs from Lázaro-Gredilla and Titsias [2011] in place of GPCV. We provide expanded results including foreign exchange data in Appendix A.2.3. In each case the mean and standard deviation are computed over approximately 2 thousand time series 75 to 100 time steps into the future, yielding tens of thousands of individual forecasts.

the Magpie mean function that enables the distributions to be centered at the correct values, which is why we see the LSTMs and constant mean GPs showing bias in the calibration plots. Note that for accurate calibration to occur in this setting our forecast distribution must match the empirical observations at *all* quantiles. We could not, for example, just forecast that a price increases some fixed percentage of the time that matches the observed frequency of the price increased and expect to achieve accurate calibration.

Table 2.1 gives the average test negative log likelihood (NLL) values on the stock forecasting task and on the foreign exchange data from Lai et al. [2018]. Both variants of the Volt model outperform competing methods such as LSTMs, and GPs with Matérn and Spectral Mixture (SM) kernels [Wilson and Nickisch 2015]. Volt with a constant mean is slightly better than with a Magpie mean in terms of NLL, the Magpie mean is key to achieving high calibration, as we see in Figure 2.12. SM kernels are a highly class of flexible kernel, but rely on there being frequency components in the data, with non-stationary data such as those studied here, the lack of regularity in the data leads to weak performance.

2.9.2 Wind Speed Forecasting

Probabilistic forecasting models play an important role in statistical climatology in providing forecast distributions over quantities of interest, such as rainfall or wind speed, that can be used to generate synthetic data or estimate the risk of extreme events. Stochastic volatility models have a history of use in modeling wind speed, but typically these models have been limited to GARCH based approaches [Liu et al. 2011; Tian et al. 2018], and are thus focused on the *volatility* of wind, rather than forecasting distributions of wind speed itself.

Here we apply Volt and Magpie to the problem of developing a stochastic weather model for wind speed. We source historical wind data from the U.S. Climate Reference Network (USCRN), with observations taken at 5 minute intervals over the 2021 calendar year at 154 spatial locations in the United States [Diamond et al. 2013]. Figure 2.13 provides an example of what the wind speed observations look like, as well as example Volt forecasts in comparison to ground truth held out data. Each forecast path represents a realistic scenario drawn from a distribution over paths from which the true data would, hypothetically, be a representative candidate. By sampling paths from the forecast distribution over wind speeds we can simulate future observations with accurate probability enabling us to estimate statistics of interest, such as expected wind speed or the probability of extreme events.

As with stock price forecasting, we are interested in producing forecast distributions that match the ground truth of the data, rather than attempting to generate point predictions. In Figure 2.14 we compare the calibration of forecasts against the ground truth wind speeds in the forecast windows. Table 2.1 provides the NLL values of the various approaches. As with stock forecasting, we see that constant means do provide slightly better NLL values than Magpie means, but Volt models are key to producing accurate forecasts. Distinct from stock price forecasting however, is the bounded nature of wind speed. As we do not expect wind speed to grow indefinitely (as we may see with stocks) we forecast with a small amount of mean reversion applied to the

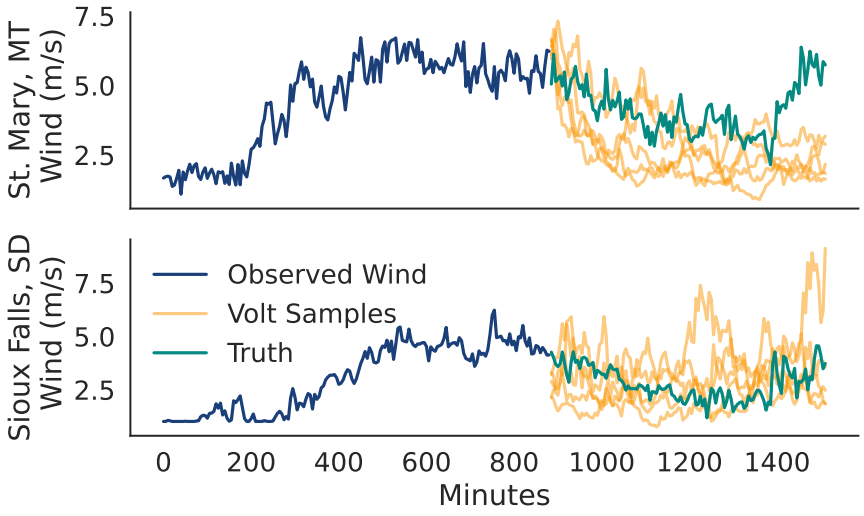


Figure 2.13: A representative example of observed wind speed and samples of multitask Volt forecasts for two related observation stations. While none of the Volt forecasts perfectly fit the true future observations of wind, each individual roll is a realistic potential realization of wind speed. By generating many plausible outcomes we are able to forecast distributions over wind speed that are highly calibrated to held out test observations.

GP models. Experimental details, including a sensitivity to the mean reversion can be found in Appendix A.2.3.2.

2.10 Multi-Task Volatility Modelling

Finally, we extend Volt to model several asset prices at once by using multi-task Gaussian processes with the goal of jointly modelling different time series at once, such as the wind speeds for the continental United States.

First, we extend the GPCV of Wilson and Ghahramani [2010] to several tasks before then placing a multi-task model over volatility in the hierarchical GP formulation. Our approach enables simultaneous estimation of the time series, its volatility, and the relationships between the time series themselves.

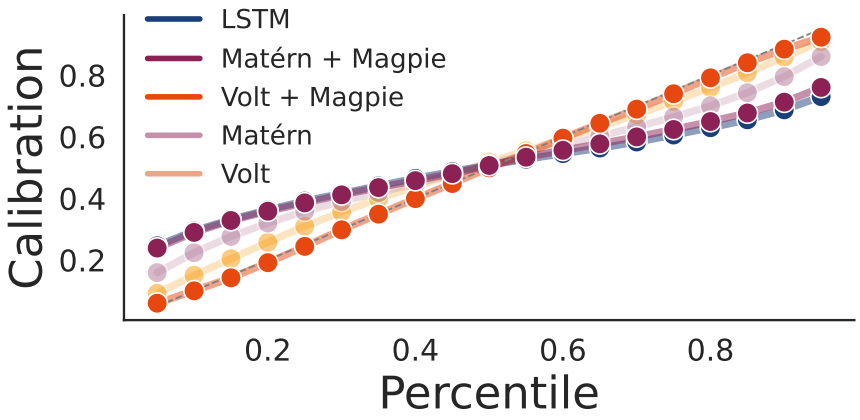


Figure 2.14: A calibration plot for wind speed, aggregated over hundreds of thousands of distinct forecasts. While probabilistic LSTMs and standard GP models provide competitive baselines, the Volt and Magpie model generates wind speed distribution forecasts that are extremely well calibrated. These calibrated distributions enable us to quickly simulate thousands of scenarios that can be trusted to faithfully represent potential outcomes.

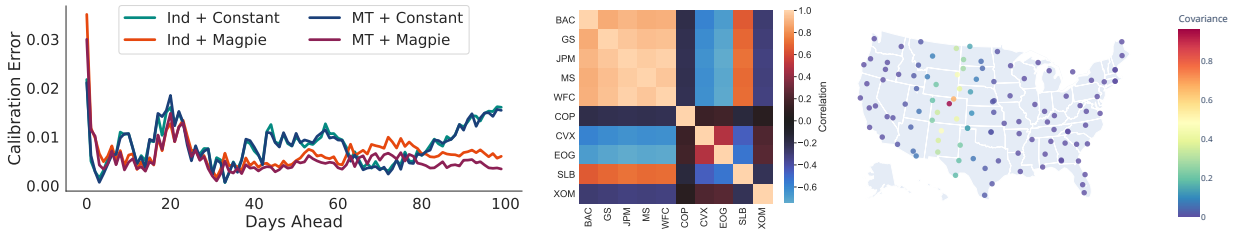


Figure 2.15: Left panel: Calibration error of both Volt and MT Volt at the 95% confidence level as a function of time step lookahead over 30 stocks from an entire sector ETF (XLF). While both are well-calibrated, MT Volt preserves well-calibration to longer time steps. **Center panel:** Estimated correlation matrix of stocks from two different sectors. MT-Volt successfully learns the high volatility correlation amongst the finance sector stocks (first five) with lower correlations between the energy sector stocks (second five). **Right panel:** Estimated volatility covariance with Boulder, CO. Correlations decrease as the stations go further away.

2.10.1 A Multitask GPCV

We extend the Gaussian process copula volatility model (GPCV) described in Section 2.8.2 to model several jointly related volatilities at once by using multi-task Gaussian processes [Bonilla et al. 2007; Alvarez et al. 2012]. We assume that all returns and volatilities are observed at once, with P different responses, so that the covariance between the pth and p'th latent Gaussian process is given by: $k([t,p],[t',p']) = K_v(t,t')K_i(p,p')$ where $K_i(p,p')$ is a lookup table describing the intertask covariance. The intertask covariance is a $P \times P$ matrix; we can regularize it with a LKJ prior [Lewandowski et al. 2009] or to incorporate side information such as geographic coordinates.

We have the multi-task probabilistic model:

$$\begin{aligned} f_p(t) &\sim \mathcal{GP}(c, K_v(t,t')K_i(p,p')) \\ w_p(t) &\sim \mathcal{N}(0, \exp^2(f_p(t)). \end{aligned} \tag{2.21}$$

Again, we use variational inference to infer the latent posterior distribution over each price's latent Gaussian process, see Appendix A.2.2.3 for more details. We also use posterior samples from this multi-task GPCV to estimate volatility, $\hat{V}_p(t)$, for each stock price p by following Eq. 2.20.

In Appendix Figure A.14, we simulate price data from a correlated SABR volatility model and use our multi-task GPCV to recover both the volatility as well as the latent correlation structures. This suggests that our inference scheme enables us to accurately recover latent correlations.

2.10.2 Multi-Task Stock Modeling

After using a multi-task GPCV to estimate the volatility for us, we then use a multi-task Gaussian process model [Bonilla et al. 2007] to estimate volatility, producing the following probabilistic

model:

$$
\begin{aligned}
v_p(t) &\sim \mathcal{GP}(m_v, K_v(t,t')K_p(t,t')) \\
V_p(t) &= \exp\left(v_p(t)\right) \\
s_p(t) &\sim \mathcal{GP}(m_{s,p}(t), K_{s,p}(t,t';V_p(t))) \\
S_p(t) &= \exp\left(s_p(t)\right).
\end{aligned}
\tag{2.22}
$$

Conditional on the correlated volatility paths, the prices themselves are independent, so we use P independent Gaussian process models to model the prices. Intuitively, this dependency structure makes sense as we expect exogenous shocks (for example, large scale macroeconomic trends) to affect variability in an asset prices, rather than just directly producing an increase or decrease.

2.10.3 Multitask Stock Price Prediction

In Figure 2.15 left panel, we consider the calibration of both Volt and MT Volt on predictions across five different groupings of stocks each with between 5 and 30 different stocks in each group, finding that all models are fairly well calibrated in terms of the calibration error, which is the squared error of the average calibration across bins of the empirical observed calibration of the foreast [Kuleshov et al. 2018]. The mult-task models tend to improve calibration over independent models, especially when using Magpie means. We display the results for calibration across time steps, mean absolute error (MAE) and negative log likelihood (NLL) in Appendix A.2.4.

In Figure 2.15 center, we showcase how the multi-task Volt model of volatility can be used to measure the relationships between assets. We considered 10 stocks, five from the financial sector and five from the energy sector. Volt learns strong correlations amongst the stocks in the financial sector and much weaker cross-correlations with the energy stocks.

2.10.4 Spatiotemporal Wind Modelling

Finally, we consider multi-task modelling for stochastic weather generation. Here, as we have longitude and latitude coordinates for each of the weather locations, we can incorporate this information into the inter-task covariance matrix by using a geodesic exponential kernel, which is given as $k(x, y) = \exp\{-\arccos(x^\top y)/2\sigma^2\}$ for $x, y \in \mathbb{S}^2$, that is points on the unit sphere [Jayasumana et al. 2013]. Note that we have no restrictions on kernel choice and could alternatively consider non-stationary kernels here instead.

We model 110 stations across the United States in the year 2021 again at 5 minute intervals, estimating the relationship between each station using the geodesic exponential kernel described above, and learning the lengthscale. We display the results in Figure 2.15 right panel with the stations described on a map of the United States. Further experimental results are shown in Appendix A.2.4.

2.11 Discussion

In this chapter we have proposed both Functional Kernel learning, as well as Volt and Magpie, for building accurate Gaussian process models. Underlying both of these methods is the use of a latent GP to imply distributions over covariance functions. While FKL and Volt approach this modeling perspective from different angles, both methods are capable of producing accurate forecasts with well calibrated predictive uncertainties.

FKL relies on the use of Bochner's theorem to learn a functional distribution over kernels, providing support for *any* stationary kernel. With this property we have shown that FKL is capable of high performance extrapolation in a range of domains, and is capable of recovering the underlying kernel structure from data when the generative process is known.

Volt deviates from the usual assumptions of stochastic differential equation (SDE) models for

financial and climatological models, and incorporates historical data through GPs, allowing us to better estimate expectations and forecast distributions. Magpie allows us to replace the often over-simplified mean functions in Gaussian process models with a nonstationary mean leading to forecasts that more closely represent the data.

Both FKL and Volt lead to novel multitask GP models, where rather than modeling the relationship between tasks in dataspace, we share information in the latent space (the spectral domain in FKL, and the volatility domain in Volt). These multitask approaches are particularly compelling in cases like wind speed or precipitation forecasting, where the highly stochastic nature of the data make it such that relationships in data space themselves may be weak. However, for these same cases the relationships in the latent space may be much stronger, and thus we can leverage this information to produce more accurate multitask forecasts.

The potential applications of our approach are broad, with potential uses in financial domains such as automated trading and strategy development, and climatological research in which Volt and Magpie could serve as a backbone for large spatiotemporal climate models. In the future, it would be useful to extend both the single and multi-task models to use online variational inference [Bui et al. 2017; Maddox et al. 2021b] to enable online deployment of scalable forecasting strategies. We hope our work will catalyze further development of kernel distributions for Gaussian processes, as well as applications of probabilistic machine learning to financial climatological data.

3 | Approximate Equivariance and Invariance in Neural Networks

In the previous chapters we examined solutions for learning distributions over covariance functions in Gaussian processes. In this chapter we shift our focus to function space considerations in neural networks, with particular focus on invariance and equivariance. Invariance and equivariance are functional properties that help us in formalizing the idea of symmetries in our models. In equivariant models we can use symmetries when the outputs of our models change in the same way as our inputs. For example if we are modeling the momentum of a pendulum, our model should be equivariant to rotations — rotating the pendulum should also rotate the momentum. Invariance merely represents the other side of the coin, where the outputs of our model do not change with respect to the inputs. For example, if we are labeling images of cars, our model should be invariant to reflections — reflecting the image should not change the label.

The ability to learn these constraints or symmetries is a foundational property of intelligent systems. Humans are able to discover patterns and regularities in data that provide compressed representations of reality, such as translation, rotation, intensity, or scale symmetries. Indeed, we see the value of such constraints in deep learning. Fully connected networks are more flexible than convolutional networks, but convolutional networks are more broadly impactful because they enforce the *translation equivariance* symmetry: when we translate an image, the outputs of a convolutional layer translate in the same way [LeCun et al. 1998a; Cohen and Welling 2016a].

Further gains have been achieved by recent work hard-coding additional symmetries, such as rotation equivariance, into convolutional neural networks (CNNs) [e.g., Cohen and Welling 2016a; Worrall et al. 2017; Zhou et al. 2017; Marcos et al. 2017].

While hard constraints like perfect invariance and equivariance are conceptually appealing and have proven success, they are not always appropriate for a world that is frequently more complex than we imagine. For example, a pendulum may have rotational symmetry, but adding wind breaks the symmetry, or a robot may have translational symmetry, but bumpy or tilted terrain breaks the symmetry. Similarly, in handwriting recognition if we rotate a 6 too far it becomes indistinguishable from a 9. To accurately engage with problems like these, we require models where equivariances are limited, either in their scope, where equivariance holds only to limited ranges of transformations, or in their strength, where equivariance is only approximate.

Since equivariance and invariance are functional properties of a model, to address either the scope or strength of these constraints we first require a model construction that allows us to engage with the function space properties of the model. Through this chapter we explore two distinct avenues for controlling the functional properties of neural networks. In the beginning sections of this chapter we focus on the *strength* of equivariance in neural networks, and control the types of functions we produce in a theoretically motivated way by incorporating and modifying equivariant layers into our models. The result is a model where the prior provably favors more equivariant solutions, but is not strictly constrained.

In the later portion of the chapter, beginning in Section 3.8, we address the issue of *scope* in invariant models, and rather than use a theoretical construction, we instead take a well motivated empirical approach through the use of carefully designed data augmentation procedures. By augmenting inputs with transformations sampled from just a range of transformations, and averaging the model outputs over these transformed inputs, we are able to produce models that are invariant over just that range of transformations.

The two approaches discussed in this chapter are distinct, but highly complementary. Both

allow to control the functional properties of our models, free us from the use of hard constraints, and incorporate a prior that biases us towards more equivariant or invariant functions either in scope or strength. These functional priors not only encode our beliefs about the world, that equivariance is desirable but possibly overly restrictive, but also allow us to learn levels of equivariance from the data alone.

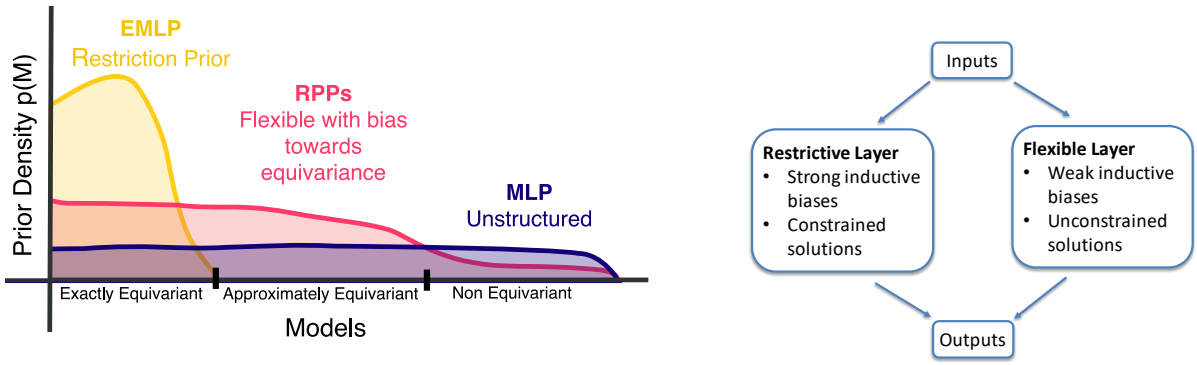
This chapter is adapted from the papers "Residual Pathway Priors for Soft Equvariance Constraints" which originally appeared at Neurips 2021, and "Learning Invariances in Neural Networks" which originally appeared at Neurips 2020. "Residual Pathway Priors for Soft Equvariance Constraints" is joint work with Marc Finzi and Andrew Gordon Wilson, and "Learning Invariances in Neural Networks" is joint work with Marc Finzi, Pavel Izmailov, and Andrew Gordon Wilson.

3.1 Residual Pathway Priors

In addressing the strength of equivariance in neural networks, we can think about a prior over the types of functions that our model produces. A highly flexible and unconstrained prior will have no preference for equivariant functions over non-equivariant functions, while a highly constrained prior will only produce equivariant functions and place no prior mass on non-equivariant functions. In this section we explore a prior that is flexible enough to allow for non-equivariant functions, but is biased towards equivariant functions.

To address the need for more interpretable priors we introduce *Residual Pathway Priors* (RPPs), a method for converting hard architectural constraints into soft priors. Practically, RPPs allow us to tackle problems in which perfect symmetry has been violated, but approximate symmetry is still present, as is the case for most real world physical systems. By favoring functions that are more equivariant, RPPs will automatically tune the strength of equivariance to match the data.

We use the schematic in Figure 3.1(a) as an approach to model construction [Wilson and Iz-



(a) Priors over Equivariant Solutions **(b)** Structure of RPP Models

Figure 3.1: Left: RPPs encode an Occam's razor approach to modeling. Highly flexible models like MLPs lack the inductive biases to assign high prior mass to relevant solutions for a given problem, while models with strict constraints are not flexible enough to support solutions with only approximate symmetry. For a given problem, we want to use the most constrained model that is consistent with our observations. **Right:** The structure of RPPs. Expanding the layers into a sum of the constrained and unconstrained solutions, while setting the prior to favor the constrained solution, leads to the more flexible layer explaining only the *residual* of what is already explained by the constrained layer.

mailov 2020; MacKay 2003]. The flexibility of our model is described by what solutions have non-zero prior probability density. The *inductive biases* are described by the distribution of support over solutions. We wish to construct models with inductive biases that assign significant prior mass for solutions we believe to be a priori likely, but without ruling out other solutions we believe to be possible. For example, models constrained to exact symmetries could not fully represent many problems, such as the motion of a pendulum in the presence of wind. Flexible models with poor inductive biases, spread thinly across possible solutions, could express an approximate symmetry, but such solutions are unlikely to be found because of the low prior density. In this sense, we wish to embrace a notion of Occam's razor such that "everything should be made as simple as possible, but no simpler".

As we find with problems in which symmetries exist, highly flexible models with weak inductive biases like MLPs fail to concentrate prior mass around solutions that exhibit any symmetry. On the other hand when symmetries are only approximate, the strong restriction biases of constrained models like Equivariant Multi-Layer Perceptrons (EMLP) [Finzi et al. 2021] fail to provide

support for the observations. As a middle ground between these two extremes, RPPs combine the inductive biases of constrained models with the flexibility of MLPs to define a model class which excels when data show approximate symmetries, as shown in Figure 3.1(b).

In the following sections we introduce our method and show results across a variety of domains. We list our contributions and the accompanying sections below:

1. We propose *Residual Pathway Priors* as a mechanism to imbue models with soft inductive biases, without constraining flexibility.
2. While our approach is general, we use RPPs to show how to turn hard architectural constraints into soft equivariance priors (Section 3.4).
3. We demonstrate that RPPs are robust to varying degrees of symmetry (Section 3.5). RPPs perform well under exact, approximate, or misspecified symmetries.
4. Using RPP on the approximate symmetries in the complex state spaces of the Mujoco locomotion tasks, we improve the performance of model free RL agents (Section 3.6).

3.2 Residual Pathway Priors Related Work

The challenge of equivariant models not being able to fully fit the data has been identified in a number of different contexts, and with different application specific adjustments to mitigate the problem. Liu et al. [2018b] observe that convolutional networks can be extremely poor at tasks that require identifying or outputting spatial locations in an image as a result of the translation symmetry. The authors solve the problem by concatenating a coordinate grid to the input of the convolution layer. Constructing translation and rotation equivariant GCNNs, Weiler and Cesa [2019] find that in order to get the best performance on CIFAR-10 and STL-10 datasets which have a preferred camera orientation, they must break the symmetry, which they do by using equivariance to progressively smaller subgroups in the later layers. Bogatskiy et al. [2020] go to

great lengths to construct Lorentz group equivariant networks for tagging collisions in particle colliders only to break the symmetry by introducing dummy inputs that identify the collision axis. van der Wilk et al. [2018] use the marginal likelihood to learn approximate invariances in Gaussian processes from data. In a related procedure, Benton et al. [2020] learn the *extent* of symmetries in neural networks using the reparametrization trick and test time augmentation. While sharing some commonalities with RPP, this method is not aimed at achieving approximate equivariance and cannot bake equivariance into the model architecture.

A separate line of work has attempted to combine the extreme flexibility of the Vision Transformer (ViT) [Dosovitskiy et al. 2021] with the better sample efficiency of convolutional networks, by incorporating convolutions at early layers [Xiao et al. 2021] or making the self attention layer more like a convolution [d'Ascoli et al. 2021; Dai et al. 2021]. Most similar to our work, ConViT [d'Ascoli et al. 2021] uses a gating mechanism for adding a soft locality bias to the self attention mechanism in Vision Transformers. ConViT and RPP share the same motivation, but while ConViT is designed specifically for biasing towards locality in the self attention layer, RPP is a general approach that we can apply broadly with other kinds of layers, symmetries, or architectural constraints.

Outside of equivariance, adding model outputs to a much more restrictive base model has been a fruitful idea employed in multiple contexts. The original ResNet [He et al. 2016a,b] drew on this motivation, with shortcut connections. Johannink et al. [2019] and Silver et al. [2018] proposed Residual Reinforcement Learning, whereby the RL problem is split into a user designed controller using engineering principles and a flexible neural network policy learned with RL. Similarly, in modeling dynamical systems, one approach is to incorporate a base parametric form informed by models from physics or biology, and only learn a neural network to fit the delta between the simple model and reality [Kashinath et al. 2021; Liu et al. 2021].

There have been several works tackling symmetries and equivariance in RL, such as permutation equivariance for multi-agent RL [Sukhbaatar et al. 2016; Jiang et al. 2018; Liu et al. 2020b], as

well exploring reflection symmetry for continuous control tasks [Abdolhosseini et al. 2019], and discrete symmetries in the more general framework of MDP homomorphisms [van der Pol et al. 2020b]. However, in each of these applications the symmetries need to be exact, and the complexities of real data often require violating those symmetries. Although not constructed with this purpose, some methods which use regularizers to enforce equivariance [van der Pol et al. 2020a] could be used for approximate symmetries. Interestingly, the value of approximate symmetries of MDPs has been explored in some theoretical work [Ravindran and Barto 2004; Taylor et al. 2008], but without architectures that can make use of it. Additionally, data augmentation, while not able to bake in architectural equivariance, has been successfully applied to encouraging equivariance on image tasks [Kostrikov et al. 2020] and recently even on tabular state vectors [Lin et al. 2020; Mavalankar 2020].

3.3 Background

In order develop our method, we first review the concept of group symmetries, how representations formalize the way these symmetries act on different objects.

Group Symmetries In the machine learning context, a symmetry group G can be understood as a set of invertible transformations under which an object is the same, such as reflections or rotations. These symmetries can act on many different kinds of objects. A rotation could act on a simple vector, a 2d array like an image, a complex collection objects like the state space of a robot, or more abstractly on an entire classification problem or Markov Decision Process (MDP).

Representations The way that symmetries act on objects is described by a *representation*. Given an object in an n-dimensional vector space V, a group representation is a mapping $\rho : G \to \mathbb{R}^{n \times n}$, yielding a matrix which acts on V. Vectors $v \in V$ are transformed $v \mapsto \rho(g)v$. In deep learning, each of the inputs and outputs to our models can be embedded in some vector space:

an $m \times m$ sized rgb image exists in $\mathbb{R}^{3m^2}$, and a node valued function on a graph of m elements exists within $\mathbb{R}^m$. The representation ρ specifies how each of these objects transform under the symmetry group G.

These representations can be composed of multiple simpler subrepresentations, describing how each object within a collection transforms. For example given the representation ρ_1 of rotations acting on a vector in $\mathbb{R}^3$, and a representation ρ_2 of how rotations act on a 3×3 matrix, the two objects concatenated together have a representation given by $\rho_1(g) \oplus \rho_2(g) = \begin{bmatrix} \rho_1(g) & 0 \\ 0 & \rho_2(g) \end{bmatrix}$, where the two matrices are concatenated along the diagonal. Practically this means we can represent intricate and multifaceted structures by breaking them down into their component parts and defining how each part transforms. For example, we may know that the velocity vector, an orientation quaternion, a joint angle, and a control torque all transform in different ways under a left-right reflection, and one can accommodate this information into the representation.

EQUIVARIANCE Given some data X with representation ρ_{in}, and Y with representation ρ_{out}, we may wish to learn some mapping $f : X \to Y$. A model f is equivariant [Cohen and Welling 2016a], if applying the symmetry transformation to the input is equivalent to applying it to the output

$$f(\rho_{\text{in}}(g)x) = \rho_{\text{out}}(g)f(x).$$

In other words, it is not the symmetry of X or Y that is relevant, but the symmetry of the function f mapping from X to Y. If the true relationship in the data has a symmetry, then constraining the hypothesis space to functions f that also have the symmetry makes learning easier and improves generalization [Elesedy and Zaidi 2021]. Equivariant models have been developed for a wide variety of symmetries and data types like images [Cohen and Welling 2016a; Worrall et al. 2017; Zhou et al. 2017; Weiler and Cesa 2019], sets [Zaheer et al. 2017; Maron et al. 2020], graphs [Maron et al. 2018], point clouds [Anderson et al. 2019; Fuchs et al. 2020; Satorras et al. 2021], dynamical

systems [Finzi et al. 2020], jets [Bogatskiy et al. 2020], and other objects [Wang et al. 2020; Finzi et al. 2021].

3.4 Residual Pathway Priors Methodology

In this section, we introduce Residual Pathway Priors (RPPs). The core implementation of the RPP is to expand each layer in model into a sum of both a restrictive layer that encodes the hard architectural constraints and a generic more flexible layer, but penalize the more flexible path via a lower prior probability. Through the difference in prior probability, explanations of the data using only the constrained solutions are prioritized by the model; however, if the data are more complex the residual between the target and the constrained layer will be explained using the flexible layer. We can apply this procedure to any restriction priors, such as linearity, locality, Markovian structure, and, of course, equivariance.

Provided we can represent the r dimensional orthogonal basis of the k weights (A) in a constrained model as $Q \in \mathbb{R}^{k\times r}$, then we can define a Gaussian prior over the weights in that basis as $A \sim \mathcal{N}(0, \sigma_a^2 QQ^\top)$. Since Q is orthogonal we can define it's orthogonal complement as P, then a Gaussian prior over unconstrained weights (B) can be written $B \sim \mathcal{N}(0, \sigma_b^2 I) = \mathcal{N}(0, \sigma_b^2 QQ^\top + \sigma_b^2 PP^\top)$. Thus the prior over the sum of the weights of the constrained and unconstrained layers is

$$A + B \sim \mathcal{N}(0, (\sigma_a^2 + \sigma_b^2)QQ^T + \sigma_b^2 PP^T). \tag{3.1}$$

Regardless of the values of the prior variances σ_a^2 and σ_b^2, solutions in the constrained subspace $QQ^\top$ are automatically favored by the model and assigned higher prior probability mass than those in the subspace $PP^\top$ that violate the constraint. Even if $\sigma_b > \sigma_a$, the model still favors equivariance because the equivariance solutions are contained in the more flexible layer A. We show in Section 3.5.2 that RPPs are insensitive to the choice of σ_a and σ_b, provided that σ_a is large enough to be able to fit the data.

The Residual Pathway Prior draws inspiration from the residual connections in ResNets [He et al. 2016a,b], whereby training stability and generalization improves by providing multiple paths for gradients to flow through the network that have different properties. One way of interpreting a residual block and shortcut connection $f(x) = x + h(x)$ in combination with l2 regularization, either explicitly from weight decay or implicitly from the training dynamics [Neyshabur et al. 2014], is as a prior that places higher prior likelihood on the much simpler identity mapping than on the more flexible function $h(x)$. In this way, $h(x)$ need only explain the the difference between what is explained in the previous layer (passed through by I) and the target.

Under the prior of Equation 3.2, a MAP optimized model will favor explanations of the data using the more structured layer A, and only resort to using layer B to explain the *difference* between the target and what is already explained by the more structured model A. Adding these unconstrained residual pathways to each layer of an constrained model, we have a model that has the same expressivity of a network formed entirely of B layers, but with the inductive bias towards a model formed entirely with the constrained A layers. We term this model a *Residual Pathway Prior*.

To make the approach concrete, we first consider constructing equivariance priors using the constraint solving approach known as Equivariant Multi-Layer Perceptrons (EMLP) from Finzi et al. [2021].

Equivariant MLPs EMLPs provide a method for automatically constructing exactly equivariant layers for any given group and representation by solving a set of constraints. The way in which the vectors are equivariant is given by a formal specification of the types of the input and output through defining their representations. Given some input vector space V_{in} with representation ρ_{in} and some output space V_{out} with representation ρ_{out} the space of all equivariant linear

layers mapping $V_{\text{in}} \to V_{\text{out}}$ satisfies

$$\forall g \in G: \quad \rho_{\text{out}}(g)W = W\rho_{\text{in}}(g).$$

These solutions to the constraint form a subspace of matrices $\mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$ which can be solved for and described by a r dimensional orthonormal basis $Q \in \mathbb{R}^{n_{\text{out}} n_{\text{in}} \times r}$. Linear layers can then be parametrized in this equivariant basis. The elements of W can be parametrized $\text{vec}(W) = Q\beta$ for $\beta \in \mathbb{R}^r$ for the linear layer $v \mapsto Wv$, and symmetric biases can be parametrized similarly.

Equivariance Priors with EMLP In order to convert the hard equivariance constraints in EMLP into a soft prior over equivariance that can accommodate approximate symmetries, we can apply the RPP procedure from above to each these linear layers in the network. Instead of parametrizing the weights W directly in the equivariant basis $\text{vec}(W) = Q\beta$, we can instead define W as the sum $W = A + B$ of an equivariant weight matrix $\text{vec}(A) = Q\beta$ an unconstrained weight matrix B. Placing Gaussian priors over both A and B yields the RPP prior in Equation (3.1) with $A + B = W \sim \mathcal{N}(0, (\sigma_a^2 + \sigma_b^2)QQ^\top + \sigma_b^2 PP^\top)$.

By replacing each of the equivariant linear layers in an EMLP with a sum of an equivariant layer and an unconstrained layer and adding in the negative prior likelihood to the loss function, we produce an RPP-EMLP that can accommodate approximate or incorrectly specified symmetries. [1]

RPPs With Other Equivariant Models While in EMLP equivariant bases are solved for explicitly, the RPP can be applied to the linear layers in other equivariant networks in precisely the same way. A good example is the translationally equivariant convolutional neural network (CNN), which can be viewed as a restricted subset of a fully connected network. Though the layers are parametrized as convolutions, the convolution operation can be expressed as a Toeplitz

[1] For the EMLP that uses gated nonlinearities which do not always reduce to a standard Swish, we likewise add a more general Swish weighted by a parameter with prior variance σ_b^2.

matrix residing within the space of dense matrices. Adding the convolution to a fully connected layer and choosing a prior variance σ_a^2 and σ_b^2 over each, we have the same RPP prior

$$W \sim N(0, \sigma_a^2 QQ^\top + \sigma_b^2 I) \tag{3.2}$$

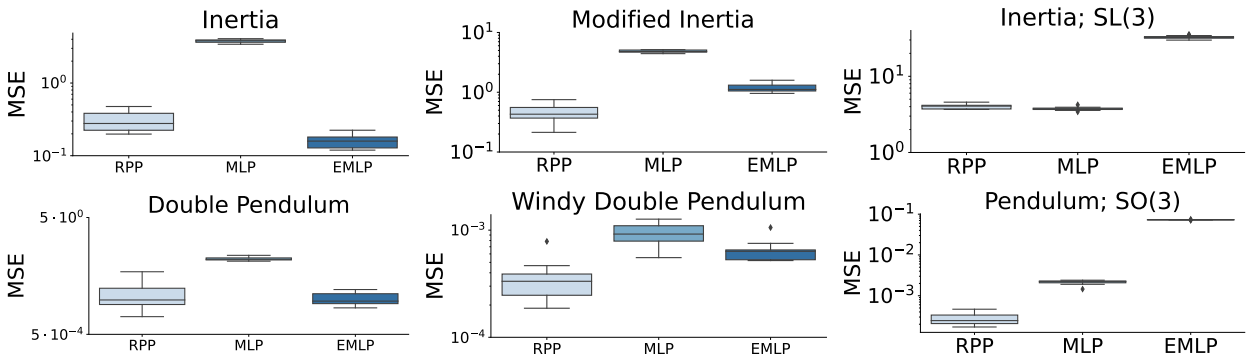
where Q is the basis of (bi-)Toeplitz matrices corresponding to 3×3 filters. This RPP CNN has the biases of convolution but can readily fit non translationally equivariant data. We can similarly create priors with the biases of other equivariant models like GCNNs [Cohen and Welling 2016a], without any hard constraints. We can even apply the RPP principle to the breaking of a given symmetry group to a subgroup.

3.5 How and Why RPPs Work

We explore how and why RPPs work on a variety of domains, applying RPPs where (1) constraints are known to be helpful, (2) cannot fully describe the problem, and (3) are misspecified.

3.5.1 Dynamical Systems and Levels of Equivariance

In order to better understand how and why residual pathway priors interact with the symmetries of the problem we move to settings in which we can directly control both the type of symmetry and the level to which the symmetries are violated. We examine how RPPs coupled with EMLP networks (RPP-EMLP) perform on the inertia and double pendulum datasets featured in Finzi et al. [2021] in 3 experimental settings: (*i*) the original inertia and double pendulum datasets which preserve exact symmetries with respect to the to O(3) and O(2) groups respectively; (*ii*) modified versions of these datasets with additional factors (such as wind on the double pendulum) that lead to approximate symmetries; and (*iii*) versions with misspecified symmetry groups that break the symmetries entirely (described in subsection A.3.3).



(a) Exact Symmetries **(b)** Approximate Symmetries **(c)** Mis-specified Symmetries

Figure 3.2: A comparison of test performance over 10 independent trials using RPP-EMLP and equivalent EMLP and MLP models on the inertia (**top**) and double pendulum (**bottom**) datasets in which we have three varying levels of symmetries. The boxes represent the interquartile range, and the whiskers the remainder of the distribution. **Left:** perfect symmetries in which EMLP and the equivariant components of RPP-EMLP exactly capture the symmetries in the data. **Center:** approximate symmetries in which the perfectly symmetric systems have been modified to include some non-equivariant components. **Right:** mis-specified symmetries in which the symmetric components of EMLP and RPP-EMLP do not reflect the symmetries present in the data.

The results for these 3 settings are given in Figure 3.4. Across all settings RPP-EMLP match the performance of EMLP when symmetries are exact, perform as well as an MLP when the symmetry is misspecified and better than both when the symmetry is approximate. For these experiments we use a prior variance of $\sigma_a^2 = 10^5$ on the EMLP weights and $\sigma_b^2 = 1$ on the MLP weights.

Exact Symmetries As part of the motivation, RPPs should properly allocate prior mass to both constrained and unconstrained solutions, we test cases in which symmetries are exact, and show that RPP-EMLP is capable of performing on par with EMLP which only admits solutions with perfect symmetry. The results in Figure 3.4(a) show that although the prior over models as described RPP-EMLP is broader than that of EMLP (as we can admit non-equivariant solutions) in the presence of perfectly equivariant data RPP-EMLP do not hinder performance, and we are able to generalize nearly as well as the perfectly prescribed EMLP model.

Approximate symmetries To better showcase the ideas of Figure 3.1 we compare RPP-EMLPs to EMLPs and MLPs on the modified inertia and windy pendulum datasets. In these datasets we can think about the systems as primarily equivariant but containing non-equivariant contributions. As shown in 3.4(b) these problems are best suited for RPP-EMLP as MLPs have no bias towards the approximately symmetry present in the data, and EMLPs are overly constrained in this setting.

Misspecified symmetries In contrast to working with perfect symmetries and showing that RPP-EMLPs are competitive with EMLPs, we also show that when symmetries are *mis*specified the bias towards equivariant solutions does not hinder the performance of RPP-EMLPs. For the inertia dataset we substitute the group equivariance in EMLP from O(3) to the overly large group SL(3) consisting of all volume and orientation preserving linear transformations, not just the orthogonal ones. For the double pendulum dataset, we substitute O(2) symmetry acting on $\mathbb{R}^3$ with the larger SO(3) rotation group that contains it but is not a symmetry of the dataset.

By purposefully misspecifying the symmetry in these datasets we intentionally construct EMLP and RPP-EMLP models with incorrect inductive biases. In this setting EMLP is incapable of making accurate predictions as it has a hard constraint on an incorrect symmetry. Figure 3.4(c) shows that even in cases where the model is intentionally mis-specified that RPPs can overcome a poorly aligned inductive bias and recover solutions that perform as well as standard MLPs, even where EMLPs fail.

3.5.2 Prior Levels of Equivariance

To test the effect of prior variances we use the modified inertia dataset, which represents a version of a problem in which perfect equivariance has been broken by adding new external forces to the dynamical system. Shown in Figure 3.3 (right) is a comparison of mean squared error on test data as a function of the prior precision terms on both the equivariance and basic weights. As a general

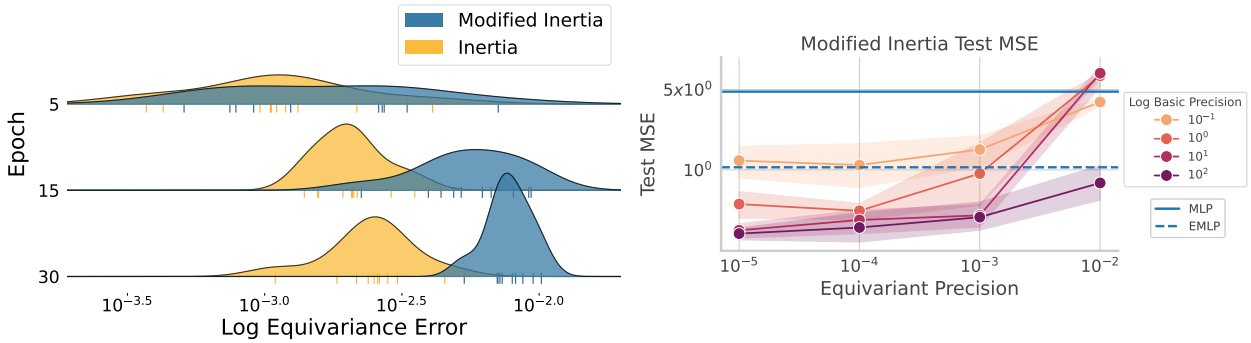


Figure 3.3: Left: Kernel density estimators of log equivariance error across training epochs for 10 independently trained networks. Here the color denotes the dataset these models were trained on. Treating these samples as a proxy for posterior density, we see that on the non-equivariant Modified Inertia dataset, the posterior is shifted upward to match the level of equivariance in the data during training. **Right:** Test MSE as a function of the weight decay parameters on the equivariant and basic weights on the modified inertia dataset. We observe that so long as the prior in the basis of equivariant weights is broad enough, we can achieve low test error with RPPs.

trend we see that when the regularization on the equivariant weights is too high (equivalent to a concentrated prior around 0) we find instability in test performance, yet when we apply a broad prior to the equivariant weights performance is typically both better in terms of MSE, and more stable to the choice of prior on the basic model weights.

As the prior variances over the equivariant basis Q and the non-equivariant basis P describe our bias towards or away from equivariant solutions we investigate how the choice of prior variance relates to the level of symmetry present in a given dataset. In the windy pendulum dataset we have control over the level of wind and thus how far our system is from perfect equivariance.

3.5.3 Posterior Levels of Equivariance

RPPs describe a method for setting a prior over equivariance, and in the presence of new data we expect the posterior distribution over equivariance to change accordingly. Using samples from a deep ensemble to query points of high density in the posterior we estimate how the distribution over equivariance error progresses through training. Recalling that with an equivariant function

f we have $\rho_2(g)f(x) = f(\rho_1(g)x)$, we compute equivariance error as

$$\text{EquivErr}(f, x) = \text{RelErr}(\rho_2(g)f(x), f(\rho_1(g)x)) \text{ where } \text{RelErr}(a, b) = \frac{\|a - b\|}{\|a\| + \|b\|}. \tag{3.3}$$

We train one deep ensemble on the inertia dataset which exhibits perfect symmetry, and another on the modified inertia dataset which has only partial symmetry, with each deep ensemble being comprised of 10 individual models using the same procedure as in Section 3.5.1. In Figure 3.3 (left) we see that throughout training the models trained on the modified inertia concentrate around solutions with substantially higher equivariance error than models trained on the dataset with the exact symmetry. This figure demonstrates one of the core desiderata of RPPs: that we are able to converge to solutions with an appropriate level of equivariance for the data.

3.5.4 RPPs and Convolutional Structure

Using the RPP-Conv specified by the prior in Eqn 3.2 we apply the model to CIFAR-10 classification and UCI regression tasks where the inputs are reshaped to zero-padded two dimensional arrays and treated as images. Notably, the model is still an MLP and merely has larger prior variance in the convolutional subspace. As a result it can perform well on image datasets where the inductive bias is aligned, as well as on the UCI data despite not being an image dataset as shown in Table 3.1. While retaining the flexibility of an MLP, the RPP performs better than the locally connected MLPs trained with β-lasso in Neyshabur [2020] which get 14% error on CIFAR-10. The full details for the architectures and training procedure are given in Appendix A.3.3.

3.6 Approximate Symmetries in Reinforcement Learning

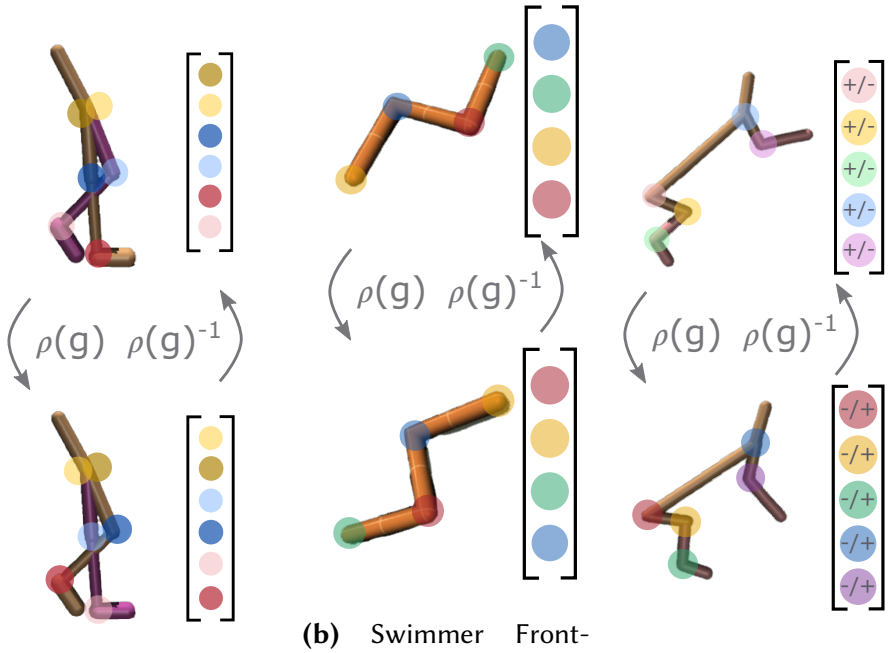
Both model free and model based reinforcement learning present opportunities to take advantage of structure in the data for predictive power and data efficiency. On the one hand stands the

	CIFAR-10	Energy	Fertility	Pendulum	Wine
MLP	37.61 ± 0.56	0.39 ± 0.48	0.049 ± 0.0044	4.65 ± 0.50	0.66 ± 0.058
RPP	12.62 ± 0.34	0.73 ± 0.44	0.060 ± 0.0097	4.25 ± 0.50	0.69 ± 0.031
Conv	12.03 ± 0.46	1.34 ± 0.38	0.076 ± 0.0157	4.63 ± 0.36	0.79 ± 0.092

Table 3.1: Mean test classification error on CIFAR-10 and MSE on 4 UCI regression tasks, with one standard deviation errors taken over 10 trials. Similar to Figure 3.4, we find that whether the constrained convolutional structure is helpful (CIFAR) or not (UCI), RPP-Conv performs similarly to the model with the correct level of complexity.

use of model predictive control in the engineering community where finely specified dynamics models are constructed by engineers and only a small number of parameters are fit with system identification to determine mass, inertia, joint stiffness, etc. On the other side of things stands the hands off approach taken in the RL community, where general and unstructured neural networks are used for both transition models [Chua et al. 2018; Wang and Ba 2019; Janner et al. 2019] as well as policies and value functions [Haarnoja et al. 2018a]. The state and action spaces for these systems are highly complex with many diverse inputs like quaternions, joint angles, forces, torques that each transform in different ways under a symmetry transformation like a left-right reflection or a rotation. As a result, most RL methods treat these spaces a black box ignoring all of this structure, and as a result they tend to require tremendous amounts of training data, making it difficult to apply to real systems without the use of simulators.

We can make use of this information about what kinds of objects populate the state and action spaces to encode approximate symmetries of the RL environments. As shown in van der Pol et al. [2020b], exploiting symmetries in MDPs by using equivariant networks can yield substantial improvements in data efficiency. But symmetries are brittle, and minor effects like rewards for moving in one direction, gravity, or even perturbations like wind, a minor tilt angle in CartPole, or other environment imperfections can break otherwise perfectly good symmetries. As shown in Table 3.2, broadening the scope to approximate symmetries allows for leveraging a lot more structure in the data which we can exploit with RPP. While Walker2d, Swimmer, Ant, and



(a) Walker2d Left-Right **(b)** Swimmer Front-Back **(c)** HalfCheetah In-Out

Figure 3.4: Example illustrations of symmetries and representations from the Mujoco environments. **Left:** left-right symmetry in the *Walker2d* environment, **center:** front-back symmetry in the *Swimmer* environment, and **right:** In-out similarity in the *HalfCheetah* environment

Humanoid have exact left/right reflection symmetries, Hopper, HalfCheetah, and Swimmer have approximate front/back reflection symmetries. Ant and Humanoid have an even more diverse set, with the D_4 dihedral symmetry by reflecting and cyclicly permuting the legs of the ant, as well as continuous rotations of the Ant and Humanoid within the environment which can be broken by external forces or rewards. Identifying this structure in the data, we are able to use the generality of EMLP to construct an equivariant model for this data, and then turn it into a soft prior using RPP.

Symmetries	Walker2d	Hopper	HalfCheetah	Swimmer	Ant	Humanoid
Exact	$\mathbb{Z}_2$	✗	✗	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$
Approximate	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2 \times \mathbb{Z}_2$	$D_4 \times O(2)$	$\mathbb{Z}_2 \times O(2)$
This work	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2 \times \mathbb{Z}_2$	$\mathbb{Z}_4$	$SO(2)$

Table 3.2: Exact and approximate symmetries of Mujoco locomotion environments of which we use the subgroups in the bottom row, see subsection A.3.4 for the detailed action and state representations.

3.6.1 Approximate Symmetries in Model Free Reinforcement Learning

We evaluate RPPs on the standard suite of Mujoco continuous control tasks in the context of model-free reinforcement learning. With the appropriately specified action and state representations detailed in subsection A.3.4, we construct RPP-EMLPs which we use as a drop-in replacement for both the policy and Q-function in the Soft Actor Critic (SAC) algorithm [Haarnoja et al. 2018a], using the same number of layers and channels. In contrast with van der Pol et al. [2020b] where equivariance is used just for policies, we find that using RPP-EMLP for the policy function alone is not very helpful with Actor Critic (see Figure 3.5). With the exception of the Humanoid-v2 environment where the RPP-EMLP destabilizes SAC, we find that incorporating the exact and approximate equivariance with RPP yields consistent improvements in the data efficiency of the RL agent as shown in Figure 3.5.

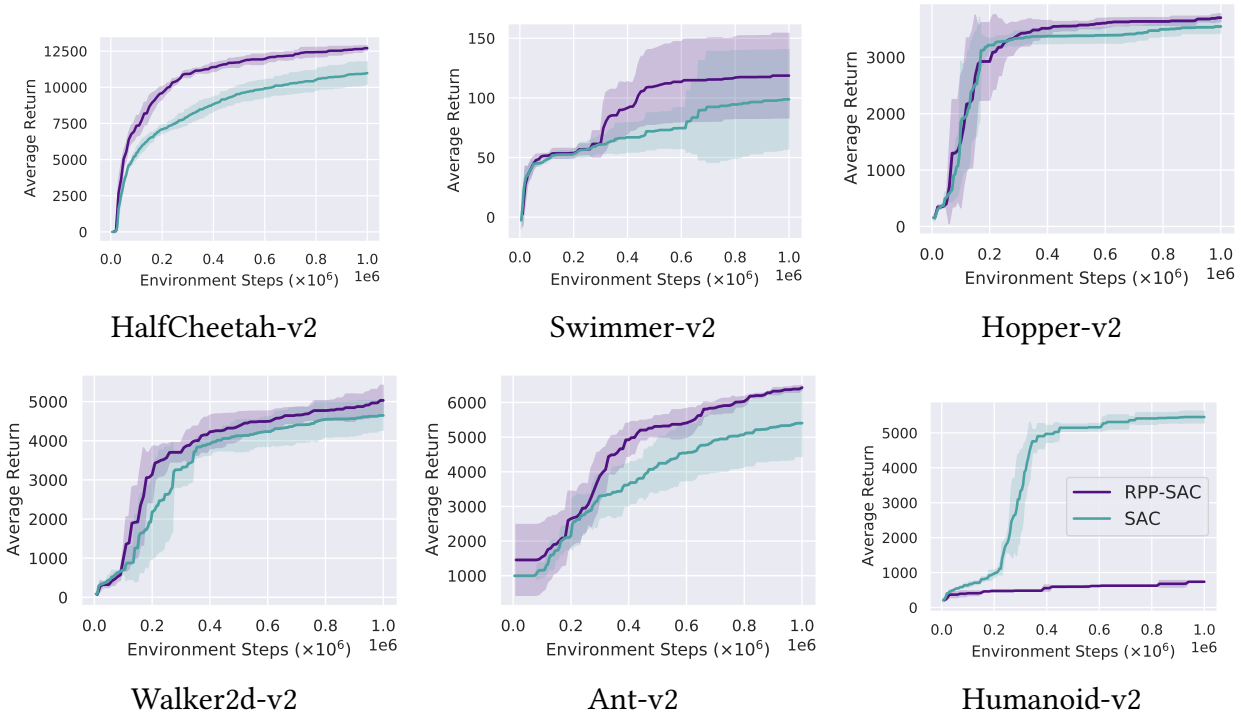


Figure 3.5: Average reward curve of RPP-SAC and SAC trained on Mujoco locomotion environments (max average reward attained at each step). Mean and one standard deviation taken over 4 trials shown in the shaded region. Incorporating approximate symmetries in the environments improves the efficiency of the model free RL agents.

3.6.2 Better Transition Models for Model Based Reinforcement Learning

	Swimmer-v2		Hopper-v2		Ant-v2	
Rollout	MLP	**RPP**	MLP	**RPP**	**MLP**	RPP
10 Steps	0.51 ± 0.02	$\mathbf{0.40 \pm 0.04}$	1.1 ± 0.1	$\mathbf{0.9 \pm 0.1}$	$\mathbf{4.2 \pm 0.1}$	5.2 ± 0.3
30 Steps	1.6 ± 0.2	$\mathbf{1.26 \pm 0.14}$	3.8 ± 0.3	$\mathbf{3.1 \pm 0.5}$	$\mathbf{11.3 \pm 0.2}$	13.9 ± 0.7
100 Steps	3.9 ± 1.0	$\mathbf{2.75 \pm 0.31}$	9.8 ± 0.5	$\mathbf{7.0 \pm 0.7}$	$\mathbf{16.0 \pm 0.3}$	20.0 ± 1.1
Equiv Err	46%	19%	98%	32%	36%	31%

Table 3.3: Transition model rollout relative error in percent % averaged over 10, 30, and 100 step rollouts (geometric mean over trajectory). Errorbars are 1 standard deviation taken over 3 random seeds. Equivariance error is computed from as the geometric mean averaged over the 100 step rollout.

We also investigate whether the equivariance prior of RPP can improve the quality of the predictions for transition models in the context of model based RL. To evaluate this in a way decoupled from the complex interactions between policy, model, and value function in MBRL, we instead construct a static dataset of $50,000$ state transitions sampled uniformly from the replay buffer of a trained SAC agent. Since the trajectories in the replay buffer come from different times, they capture the varied dynamics MBRL transition models often encounter during training.

State of the art model based approaches on Mujoco tend to use an ensemble of small MLPs that predict the state transitions [Chua et al. 2018; Wang and Ba 2019; Janner et al. 2019; Amos et al. 2020], without exploiting any structure of the state space. We evaluate test rollout predictions via the relative error of the state over different length horizons for the RPP model against an MLP, the method of choice. As shown in Table 3.3, RPP transition models outperform MLPs on the Swimmer and Hopper environments, especially for long rollouts showing promise for use in MBRL. On these environments, RPP learns a smaller but non-negligible equivariance error that still enables it to fit the data.

3.7 RPP Limitations

Using RPP-EMLP for the state and action spaces of the Mujoco environments required identifying the meaning of each of the components in terms of whether they are scalars, velocity vectors, joint angles, or orientation quaternions, and also which part of the robot they correspond to. This can be an error-prone process. While RPPs are fairly robust to such mistakes, the need to identify components makes using RPP more challenging than standard MLP. Additionally, due to the bilinear layers within EMLP, the Lipschitz constant of the network is unbounded which can lead to training instabilities when the inputs are not well normalized. We hypothesize these factors may contribute to the training instability we experienced using RPP-EMLP on Humanoid-v2.

3.8 Learning Invariances in Neural Networks: Augerino

With Residual Pathway Priors we examined cases where equivariance is only held approximately. Namely, we developed a model which yields functions such that $gf(x) \approx f(gx)$ for some transformation g. In the following sections we explore a highly related approach, but rather than approximate equivariances we focus on limited invariances, where $f(x) = f(gx)$ for some $g \subseteq G$. For these cases we aim to learn functions where the invariance is held as close to exactly as possible, but where the transformations g are restricted to a subset of the full symmetry group G. Such limited symmetries are important for many visual tasks, especially where labels depend not only on image content but also pose. For example, 6's can be rotated to become 9's, or an n can become a u if the image is rotated. In these cases invariance to some rotation is present, but not to the full range of rotations.

This approach is driven by training with data augmentations, leading to the name *Augerino*. Since the approach relies on data augmentation during training, rather than the underlying ar-

chitecture of the model, Augerino is compatible with any standard neural network architecture. This reliance on augmentations, not architectures, means that Augerino is able to influence the functional properties of the model while still treating the base architecture as a black box.

Augerino (1) can learn both invariances and equivariances over a wide range of symmetry groups, including translations, rotations, scalings, and shears; (2) can discover partial symmetries, such as rotations not spanning the full range from $[-\pi, \pi]$; (3) can be combined with any standard architectures, loss functions, or optimization algorithm with little overhead; (4) performs well on regression, classification, and segmentation tasks, for both image and molecular data.

3.9 Augerino Related Work

There is a large body of work constructing convolutional neural networks that have *hard-coded* invariance or equivariance to a set of transformations, such as rotation [Cohen and Welling 2016a; Worrall et al. 2017; Zhou et al. 2017; Marcos et al. 2017] and scaling [Worrall and Welling 2019; Sosnovik et al. 2019]. While recent methods use a representation theoretic approach to find a basis of equivariant convolutional kernels [Cohen and Welling 2016b; Worrall et al. 2017; Weiler and Cesa 2019], the older method of Laptev et al. [2016] pools network outputs over many hard-coded transformations of the input for fixed invariances, but does not consider equivariances or learning the transformations.

van der Wilk et al. [2018] learn transformations for learning invariances in kernel methods from training data, using the marginal likelihood of a Gaussian process. The marginal likelihood, which is the integral of the product of the likelihood with a parameter prior, automatically selects for constraints [e.g., MacKay 2003]. They propose a similar pipeline of learning the parameters of a transformation directly by backpropagation and the reparametrization trick. In contrast to their work, we develop a framework that can be easily applied to deep neural networks with standard loss functions, without needing to compute a marginal likelihood (which is typically intractable).

Our framework can also learn more general transformations through the exponential map, as well as *equivariant* models.

With a desire to automate the machine learning pipeline, Cubuk et al. [2019] introduced *AutoAugment* in which reinforcement learning is used to find an optimal augmentation policy within a discrete search space. At the expense of a massive computational budget for the search, AutoAugment brought substantial gains in image classification performance, including state-of-the-art results on ImageNet. The AutoAugment framework was extended first to *Fast AutoAugment* in Lim et al. [2019], improving both the speed and accuracy of AutoAugment by using Bayesian data augmentation [Tran et al. 2017]. Both Cubuk et al. [2019] and Lim et al. [2019] apply a reinforcement learning approach to searching the space of augmentations, significantly differing from our work which directly optimizes distributions over augmentations with respect to the training loss.

Faster AutoAugment [Hataya et al. 2019], which uses a GAN framework to match augmentations to the data distribution, and *Differentiable Automatic Data Augmentation* [Li et al. 2020] which applies a DARTS [Liu et al. 2018a] bi-level optimization procedure to learn augmentation from the validation loss are most similar to Augerino in the discovery of distributions over augmentations. Both methods learn augmentations from data using the reparametrization trick; however unlike Li et al. [2020] and Liu et al. [2018a], we learn augmentations directly from the training loss without need for GAN training or the complex DARTS procedure [Liu et al. 2018a; Xu et al. 2019; Liang et al. 2019], and are specifically learning degrees of invariances and equivariances.

To the best of our knowledge, Augerino is the first work to *learn* invariances and equivariances in neural networks from training data alone. The ability to automatically discover symmetries enables us to uncover interpretable salient structure in data, and provide better generalization.

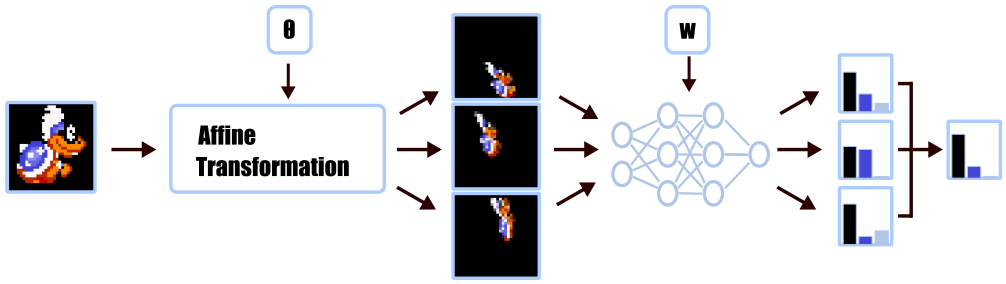


Figure 3.6: The Augerino framework. Augmentations are sampled from a distribution governed by parameters θ, and applied to an input to produce multiple augmented inputs. These augmented inputs are then passed to a neural network with weights w, and the final prediction is generated by averaging over the multiple outputs. Augerino discovers invariances by learning θ from training data alone.

3.10 Augerino: Learning Invariances through Augmentation

A simple way of constructing a model invariant to a given group of transformations is to average the outputs of an arbitrary model for the inputs transformed with all the transformations in the group. For example, if we wish to make a given image classifier invariant to horizontal reflections, we can average the predictions of the network for the original and reflected input.

Augerino functions by sampling multiple augmentations from a parametrized distribution then applying these augmentations to an input to acquire multiple augmented samples of the input. The augmented input samples are each then passed through the model, with the final prediction being generated by averaging over the individual outputs. We present the Augerino framework in Figure 3.6.

Now, suppose we are working with a set $\mathcal{S}$ of transformations. Relevant transformations may not always form a group structure, such as rotations R_ϕ by limited angles in the range $\phi \in [-\theta, \theta]$. Given a neural network f_w, with parameters w, we can make a new model $\bar{f}$ which is approximately invariant to transformations $\mathcal{S}$ by averaging the outputs over a uniform distribution $\mu_\theta(\cdot)$ over the transformations $g \in S$ with $\text{supp}(\mu_\theta) = \mathcal{S}$[2] [e.g., Laptev et al. 2016; Raj et al. 2017; van der

[2]See Appendix A.4.1 for further discussion on forming the invariant model.

Wilk et al. 2018] :

$$\bar{f}_w(x) = \mathrm{E}_{g\sim\mu_\theta} f_w(gx). \tag{3.4}$$

For cross-entropy loss we can use Jensen's inequality to bound the loss of $\bar{f}$ by the expected loss of f:

$$\ell(\bar{f}_w(x)) = \ell(\mathrm{E}_{g\sim\mu_\theta} f_w(gx)) \leq \mathrm{E}_{g\sim\mu_\theta} \ell(f_w(gx)). \tag{3.5}$$

Note that here we are considering $f_w(x)$ to be the log-probabilities of the classes, i.e. the post-softmax outputs of the network. We then train the augmentation averaged model $\bar{f}$ by minimizing the upper bound on the $\ell(\bar{f}_w(x))$, the loss of $f_w(gx)$ averaged over a finite number of samples from $g \sim \mu_\theta$ at training time, using a Monte Carlo estimator.

To learn the invariances we can also backpropagate through to the parameters θ of the distribution μ_θ by using the reparametrization trick [Kingma and Welling 2013]. For example, for a uniform distribution over rotations with angles $U[-\theta, \theta]$, we can parametrize the rotation angle by $\phi = \theta\epsilon$ with $\epsilon \sim U[-1, 1]$. The loss $L(\cdot)$ for the augmentation-averaged model on an input x can be computed as

$$L_x(\theta, w) = \mathrm{E}_{\phi\sim U[-\theta,\theta]} \ell\big(f_w(R_\phi x)\big) = \mathrm{E}_{\epsilon\sim U[-1,1]} \ell\big(f_w(R_{\epsilon\theta} x)\big). \tag{3.6}$$

Specifically, during training we can use a single sample from the augmentation distribution to estimate the gradients. The learned range of rotations $[-\theta, \theta]$ would correspond to the extent rotational invariance is present in the data. With a more general set of k transformations, we can similarly define a distribution $\mu_\theta(\cdot)$ over the transformation elements using the reparametrization trick $g = g_\epsilon = \epsilon \odot \theta$, with $\epsilon \sim U[-1, 1]^k$ and $\theta \in \mathbb{R}^k$. The reparametrized loss is then

$$L_x(\theta, w) = \mathrm{E}_{\epsilon\sim U[-1,1]^k} \ell\big(f_w(g_\epsilon x)\big). \tag{3.7}$$

In Section 3.10.2 we describe a parameterization of the set of affine transformations which in-

cludes translations, rotations, and scalings of the input as special cases. In this fashion, we can train both the parameters of the augmentation averaged model $\bar{f}$ consisting both of the weights w of f_w and the parameters θ of the augmentation distribution μ_θ.

Test-time Augmentation At test time we sample multiple transformations $g \sim \mu_\theta$ and make a prediction by averaging over the predictions generated for each transformed input, approximating the expectation in Equation (3.4). We further discuss train and test time augmentation in Appendix A.4.4.

Regularized Loss Invariances correspond to constraints on the model, and in general the most unconstrained model may be able to achieve the lowest training loss. However, we have a prior belief that a model should preserve *some* level of invariance, even if standard losses cannot account for this preference. To bias training towards solutions that incorporate invariances, we add a regularization penalty to the network loss function that promotes broader distributions over augmentations. Our final loss function is given by

$$L_x(\theta, w) = \mathrm{E}_{g\sim\mu_\theta}\ell\big(f_w(gx)\big) + \lambda R(\theta), \tag{3.8}$$

where R is a regularization function encouraging coverage of a larger volume of transformations and λ is the regularization weight (the form of $R(\theta)$ is discussed in Section 3.10.2). In practice we find that the choice of λ is *largely unimportant*; the insensitivity to the choice of λ is demonstrated throughout Sections 3.11 and 3.13 in which performance is consistent for various values of λ. This is due to the fact that there is essentially no gradient signal for θ over the range of augmentations consistent with the data, so even a small push is sufficient. We discuss further why Augerino is able to learn the correct level of invariance — *without sensitivity to λ, and from training data alone* — in Section 3.12.

We refer to the introduced method as *Augerino*. We summarize the method in Algorithm 1.

Algorithm 1: Learning Invariances with Augerino

Inputs:

Dataset $\mathcal{D}$; parametric family g of data augmentations and a distribution μ_θ over the parameters θ; neural network f_w with parameters w; number n_{copies} of augmented inputs to use during training; number of training steps N.

for $i = 1, \ldots, N$ **do**

Sample a mini-batch $\tilde{x}$ from $\mathcal{D}$;

For each datapoint in $\tilde{x}$ sample n_{copies} transformations from μ_θ;

Average predictions of the network f_w over n_{copies} data transformations of $\tilde{x}$;

Compute the loss (3.8), $L_{\tilde{x}}(\theta, w)$ using the averaged predictions;

Take the gradient step to update the parameters w and θ;

end

3.10.1 Extension to Equivariant Predictions

We now generalize Augerino to problems where the targets are *equivariant* rather than invariant to a certain set of transformations. We say that target values are equivariant to a set of input transformations if the targets for a transformed input are transformed in the same way as the input. Formally, a function f is equivariant to a symmetry transformation g, if applying g to the input of the function is the same as applying g to the output, such that $f(gx) = gf(x)$. For example, in image segmentation if the input image is rotated the target segmentation mask should also be rotated by the same angle, rather than being unchanged.

To make the Augerino model equivariant to transformations sampled from $\mu_\theta(\cdot)$, we can average the inversely transformed outputs of the network for transformed inputs:

$$f_{\text{aug-eq}}(x) = \mathrm{E}_{g\sim\mu_\theta} g^{-1} f(gx). \tag{3.9}$$

Supposing that g acts linearly on the image then the model is equivariant:

$$f_{\text{aug-eq}}(hx) = \mathrm{E}_{g\sim\mu_\theta} g^{-1} f(ghx) = \mathrm{E}_{g\sim\mu_\theta} h(gh)^{-1} f(ghx) = h\mathrm{E}_{u\sim\mu_\theta} u^{-1} f(ux) \tag{3.10}$$

$$= h f_{\text{aug-eq}}(x) \tag{3.11}$$

where $u = gh$ and the distribution is right invariant: for any measurable set S, $\forall h \in G : \mu_\theta(S) = \mu_\theta(hS)$. If the distribution over the transformations is uniform then the model is equivariant.

3.10.2 Parameterizing Affine Transformations

We now show how to parametrize a distribution over the set of affine transformations of $2d$ data (e.g. images). With this parameterization, Augerino can learn from a broad variety of augmentations including translations, rotations, scalings and shears.

The set of affine transformations form an algebraic structure known as a Lie Group. To apply the reparametrization trick, we can parametrize elements of this Lie Group in terms of its Lie Algebra via the exponential map [Falorsi et al. 2019]. With a very simple approach, we can define bounds θ_i on a uniform distribution over the different exponential generators G_i in the Lie Algebra:

$$g_\epsilon = \exp\left(\sum_i \epsilon_i \theta_i G_i\right) \quad \epsilon \sim U[-1, 1]^k, \tag{3.12}$$

where exp is the matrix exponential function: $\exp(A) = \sum_{n=0}^{\infty} \frac{1}{n!} A^n$. [3]

The generators of the affine transformations in $2d$, $G_1, \ldots, G_6$, correspond to translation in x, translation in y, rotation, scaling in x, scaling in y, and shearing; we write out these generators in Appendix A.4.2. The exponential map of each generating matrix produces an affine matrix that can be used to transform the coordinate grid points of the input like in Jaderberg et al. [2015]. To ensure that the parameters θ_i are positive, we learn parameters $\tilde{\theta}_i$ where $\theta_i = \log(1 + \exp\tilde{\theta}_i)$. In

[3] Mathematically speaking, this distribution is a *pushforward* by the exp map of a scaled cube with side lengths θ_i of a cube $\mu_\theta(\cdot) = \exp_* \mathrm{Cube}_\theta(\cdot)$.

maximizing the volume of transformations covered, it would be geometrically sensible to maximize the Haar measure $\mu_H(S)$ of the set of transformations $S = \exp(\text{Cube}_\theta)$ that are covered by Augerino, which is similar to the volume covered in the Lie Algebra $\text{Vol}(\text{Cube}_\theta) = \Pi_{i=1}^{k} \theta_i$. However, we find that even the negative L_2 regularization $R(\theta) = -\|\theta\|^2$ on the bounds θ_i is sufficient to bias the model towards invariance. More intuitively, the regularization penalty biases solutions towards values of θ which induce broad distributions over affine transformations, μ_θ.

We apply the L_2 regularization penalty on both classification and regression problems, using cross entropy and mean squared error loss, respectively. This regularization method is effective, interpretable, and leads to the discovery of the correct level of invariance for a wide range of λ.

3.11 Shades of Invariance

We can broadly classify invariances in three distinct ways: first there are cases in which we wish to be completely invariant to transformations in the data, such as to rotations on the rotMNIST dataset. There are also cases in which we want to be only partially invariant to transformations, i.e. *soft* invariance, such as if we are asking if a picture is right side up or upside down. Lastly, there are cases in which we wish there to be no invariance to transformations, such as when we wish to predict the rotations themselves. We show that Augerino can learn full invariance, soft invariance, and no invariance to rotations. We then explain in Section 3.12 why Augerino is able to discover the correct level of invariance from training data alone. Incidentally, soft invariances are the most representative of real-world problems, and also the most difficult to correctly encode a priori — where we most need to learn invariances.

For the experiments in this and all following sections we use a 13-layer CNN architecture from Laine and Aila [2016]. We compare Augerino trained with three values of λ from Equation 3.8; $\lambda = \{0.01, 0.05, 0.1\}$ corresponding to low, standard, and high levels of regularization. To further emphasize the need for invariance to be *learned* as opposed to just embedded in a model we

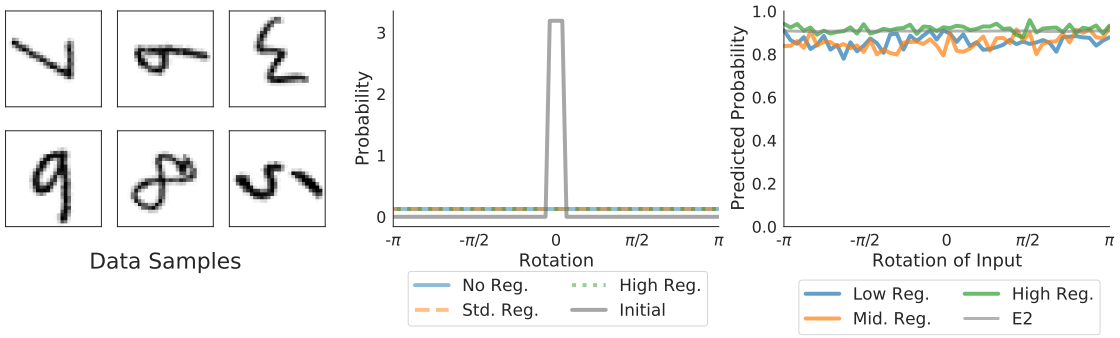


Figure 3.7: Left: Samples of the rotated digits in the data. **Center:** The initial and learned distributions over rotations. **Right:** The prediction probabilities of the correct class label over rotated versions of an image; the model learns to be approximately invariant to rotations under all levels of regularization.

also show predictions generated from an invariant $E(2)$-steerable network [Cohen and Welling 2016b]. Specific experimental and training details are in Appendix A.4.4.

3.11.1 Full Rotational Invariance: rotMNIST

The rotated MNIST dataset (rotMNIST) consists of the MNIST dataset with the input images randomly rotated. As the dataset has an inherent augmentation present (random rotations), we desire a model that is invariant to such augmentations. With Augerino, we aim to approximate invariance to rotations by learning an augmentation distribution that is uniform over all rotations in $[0, 2\pi]$.

Figure 3.7 shows the learned distribution over rotations to apply to images input into the model. On top of learning the correct augmentation through automatic differentiation using *only* the training data, we achieve 98.9% test accuracy. We also see the level of regularization has little effect on performance. To our knowledge, only Weiler and Cesa [2019] achieve better performance on the rotMNIST dataset, using the correct equivariance already hard-coded into the network.

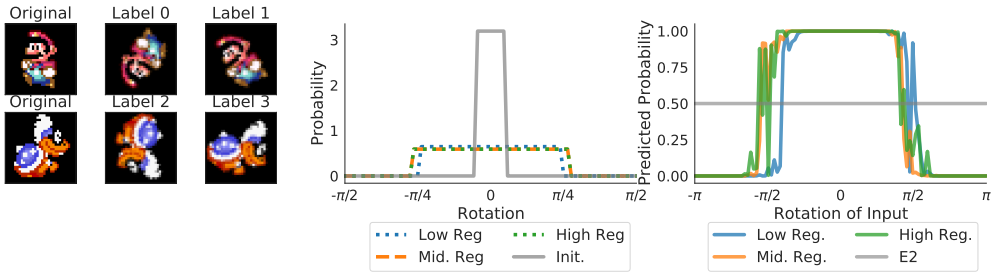


Figure 3.8: Left: Example data from the constructed Mario dataset. Labels are dependent on both the character, Mario or Iggy, and the rotation, upper half- or lower half-plane. **Center:** The initial and learned distribution over rotations. Rotations in the data are limited to $[-\pi/4, \pi/4]$ and $[-\pi, -3\pi/4] \cup [3\pi/4, \pi]$, meaning that augmenting an image by no more than $\pi/4$ radians will keep the rotation in the same half of the plane as where it started. The learned distributions approximate the invariance to rotations in $[-\pi/4, \pi/4]$ that is present in the data. **Right:** The predicted probability of label 1 for input images of Mario rotated at various angles. $E2$-steerable model is invariant, and incapable of distinguishing between inputs of different rotations.

3.11.2 Soft Invariance: Mario & Iggy

We show that Augerino can learn *soft* invariances — e.g. invariance to a subset of transformations such as only partial rotations. To this end, we consider a dataset in which the labels are dependent on both image and pose. We use the sprites for the characters Mario and Iggy from Super Mario World, randomly rotated in the intervals of $[-\pi/4, \pi/4]$ and $[-\pi, -3\pi/4] \cup [3\pi/4, \pi]$ [Nintendo 1990]. There are 4 labels in the dataset, one for the Mario sprite in the upper half plane, one for the Mario sprite in the lower half plane, one for the Iggy sprite in the upper half plane, and one for the Iggy sprite in the lower half plane; we show an example demonstrating each potential label in Figure 3.8.

In Figure 3.8, the limited rotations present in the data give that the labels are invariant to rotations of up to $\pi/4$ radians. Augerino learns the correct augmentation distribution, and the predicted labels follow the desired invariances to rotations in $[-\pi/4, \pi/4]$.

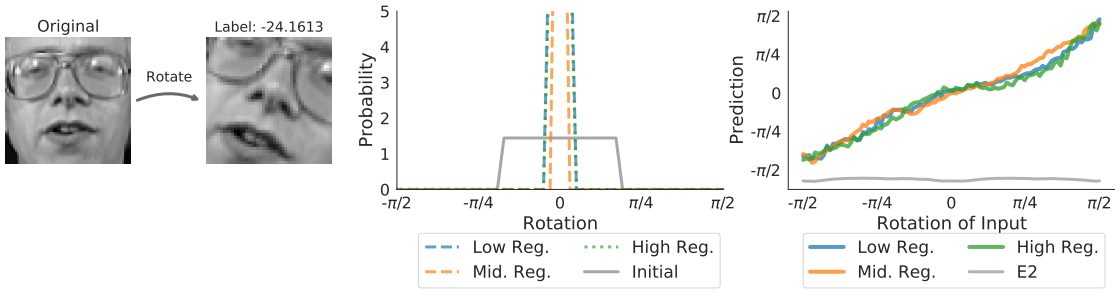
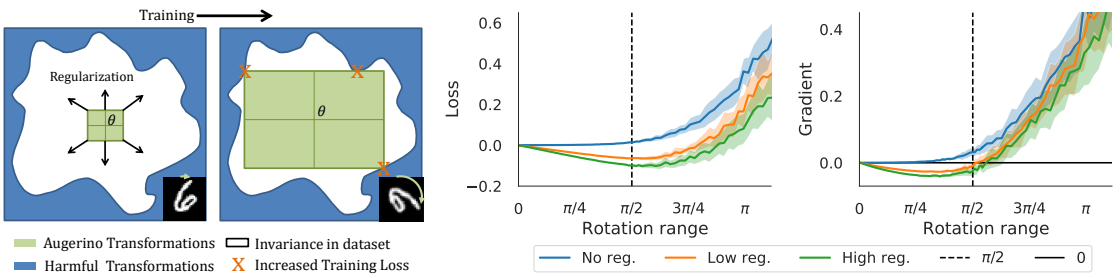


Figure 3.9: Left: The data generating process for the Olivetti faces dataset. The labels correspond to the rotation of the input image. **Center:** The initialized and learned distributions over rotations. **Right:** The predictions generated as an input is rotated. Here we see that there is no invariance present for any level of regularization - as the image rotates the predicted label changes accordingly. The $E2$-steerable network fails for this task, as the invariance to rotations prevents us from being able to predict the rotation of the image.

3.11.3 Avoiding Invariance: Olivetti Faces

To test that Augerino can avoid unwanted invariances we train the model on the rotated Olivetti faces dataset [Hinton and Salakhutdinov 2008]. This dataset consists of 10 distinct images of 40 different people. We select the images of 30 people to generate the training set, randomly rotating each image in $[-\pi/2, \pi/2]$, retaining the angle of rotation as the new label. We then crop the result to 45×45 pixel square images. We repeat the process 30 times for each image, generating 9000 training images. Figure 3.9 shows the data generating process and the corresponding label. Augmenting the image with any rotation would make it impossible to learn the angle by which the original image was rotated.

We find experimentally in Figure 3.9 that when we initialize the Augerino model such that the distribution over the rotation generating matrix G_3 is uniform $[0, 1]$, training for 200 epochs reduces the distribution on the rotational augmentation to have domain of support 0.003 radians wide. The model learns a nearly fixed transformation in each of the 5 other spaces of affine transformation, all with domains of support for the weights w_i under 0.1 units wide.



(a) Augerino training

(b) Loss function and Gradient

Figure 3.10: (a): A visualization of the space of possible transformations. Augerino expands to fill out the invariances in the dataset but is halted at the boundary where harmful transformations increase the training loss like rotating a 6 to a 9. **(b):** Loss value as a function of the rotation range applied to the input on the Mario and Iggy classification problem of Section 3.11.2 and its derivative. Without regularization the loss is flat for augmentations within the range $[0, \pi/2]$ corresponding to the true rotational invariance range in the data, and grows sharply beyond this range.

3.12 Why Augerino Works

The conventional wisdom is that it is impossible to learn invariances directly from the training loss as invariances are constraints on the model which make it harder to fit the data. Given data that has invariance to some augmentation, the training loss will not be improved by widening our distribution over this augmentation, even if it helps generalization: we would want a model to be invariant to rotations of a '6' up until it looks more like a '9', but no invariance will achieve the same training loss. However, it is sufficient to add a simple regularization term to encourage the model to discover invariances. In practice we find that the final distribution over augmentations is insensitive to the level of regularization, and that even a small amount of regularization will enable Augerino to find wide distributions over augmentations that are consistent with the precise level of invariances in the data.

We illustrate the learning of invariances with Augerino in panel (a) of Figure 3.10. Suppose only a limited degree of invariance is present in the data, as in Section 3.11.2. Then the training loss for the augmentation parameters will be flat for augmentations within the range of invariance present in the data (shown in white), and then will increase sharply beyond this range

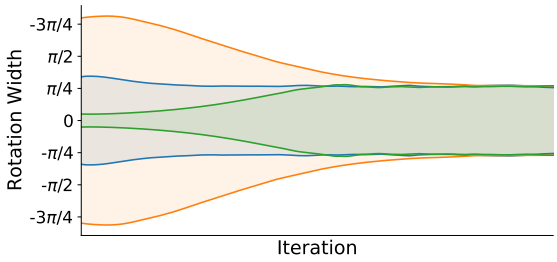


Figure 3.11: The distribution over rotation augmentations for the Mario and Iggy dataset over training iterations for various initializations. Regardless of whether we start with too wide, too narrow, or approximately the correct distribution over rotations, Augerino converges to the appropriate width.

(corresponding region of Augerino parameters is shown in blue). The regularized loss in Eq. (3.8) will push the model to increase the level of invariance within the flat region of the training loss, but will not push it beyond the degree of invariance present in the data unless the regularization strength is extreme.

We demonstrate the effect described above for the Mario and Iggy classification problem of Section 3.11.2 in panel (b) of Figure 3.10. We use a network trained with Augerino and visualize the loss and gradient with respect to the range of rotations applied to the input with and without regularization. Without regularization, the loss is almost completely flat until the value of $\pi/2$ which is the true degree of rotational invariance in the data. With regularization we add an incentive for the model to learn larger values of the rotation range. Consequently, the loss achieves its optimum close to the optimal value of the parameter at $\pi/2$ and then quickly grows beyond that value. Figure 3.11 displays the results of panel (b) of Figure 3.10 in action; gradient signals push augmentation distributions that are too wide down and too narrow up to the correct width.

Incidentally, the Augerino solutions are substantially flatter than those obtained by standard training, as shown in Appendix A.4.7, Figure A.22, which may also make them more easily discoverable by procedures such as SGD. We also see that these solutions indeed provide better generalization. We provide further discussion of learning partial invariances with Augerino in Appendix A.4.1.

3.13 Image Recognition

As Augerino learns a set of augmentations specific to a given dataset, we expect to see that Augerino is capable of boosting performance over applying any level of fixed augmentation. Using the CIFAR-10 dataset, we compare Augerino to training on data with *i*) no augmentation, *ii*) fixed, commonly applied augmentations, and *iii*) the augmentations as given by Fast AutoAugment Lim et al. [2019].

Table 3.4: Test accuracy for models trained on CIFAR-10 with different augmentations applied to the training data.

	No Aug.	Fixed Aug.	Augerino (4 copies)	Augerino (1 copy)	Fast AA
Test Accuracy	90.60	92.64	**93.81** ± 0.002	92.22 ± 0.002	92.65

We compare models trained with no augmentation, a fixed commonly applied set of augmentations (including flipping, cropping, and color-standardization), Augerino, and Fast AutoAugment [Lim et al. 2019]. Augerino with $n_{copies} = 4$ provides a boost in performance with minimal increased training time. Error bars are reported as the standard deviation in accuracy for Augerino trained over 10 trials.

Table 3.4 shows that Augerino is competitive with advanced models that seek data-based augmentation schemes. The gains in performance are accompanied by notable simplifications in setup: we do not require a validation set and the augmentation is learned concurrently with training (there is no pre-processing to search for an augmentation policy). In Appendix A.4.7 we show that Augerino find *flatter* solutions in the loss surface, which are known to generalize [Maddox et al. 2020]. To further address the choice of regularization parameter, we train a number of models on CIFAR-10 with varying levels of regularization. In Figure A.22 we present the test accuracy of models for different regularization parameters along with the corresponding effective dimensionalities of the networks as a measure of the *flatness* of the optimum found through training. [Maddox et al. 2020] shows that effective dimensionality can capture the flatness of optima in parameter space and is strongly correlated to generalization, with lower effective dimensionality implying flatter optima and better generalization.

The results of the experiment presented in Figure A.22 solidify Augerino's capability to boost performance on image recognition tasks as well as demonstrate that the inclusion of regularization is helpful, but not necessary to train accurate models. If the regularization parameter becomes too large, as can be seen in the rightmost violins of Figure A.22, training can become unstable with more variance in the accuracy achieved. We observe that while it is possible to achieve good results with no regularization, the inclusion of an inductive bias that we ought to include some invariances (by adding a regularization penalty) improves performance.

3.14 Molecular Property Prediction

We test out our method on the molecular property prediction dataset QM9 [Blum and Reymond 2009; Rupp et al. 2012] which consists of small inorganic molecules with features given by the coordinates of the atoms in 3D space and their charges. We focus on the HOMO task of predicting the energy of the highest occupied molecular orbital, and we learn Augerino augmentations in the space of affine transformations of the atomic coordinates in $\mathbb{R}^3$. We parametrize the transformation as before with a uniform distribution for each of the generators listed in Appendix A.4.2. We use the LieConv model introduced in Finzi et al. [2020], both with no equivariance (LieConv-Trivial) and 3D translational equivariance (LieConv-T(3)). We train the models for 500 epochs on MAE (additional training details are given in A.4.4) and report the test performance in Table 3.5. Augerino performs much better than using no augmentations and is competitive with the hand chosen random rotation and translation augmentation (SE(3)) that incorporates domain knowledge about the problem. We detail the learned distribution over affine transformations in Appendix A.4.6. Augerino is useful both for the non equivariant LieConv-Trivial model as well as the translationally equivariant LieConv-T(3) model, suggesting that Augerino can complement architectural equivariance.

Table 3.5: Test MAE (in meV) on QM9 tasks trained with specified augmentation.

	HOMO (meV)			LUMO (meV)		
	No Aug.	Augerino	SE(3)	No Aug.	Augerino	SE(3)
LieConv-Trivial	52.7	38.3	**36.5**	43.5	33.7	**29.8**
LieConv-T(3)	34.2	33.2	**30.2**	30.1	26.9	**25.1**

3.15 Semantic Segmentation

In Section 3.10.1 we showed how Augerino can be extended to equivariant problems. In Semantic Segmentation the targets are perfectly aligned with the inputs and the network should be equivariant to any transformations present in the data. To test Augerino in equivariant learning setting we construct rotCamVid, a variation of the CamVid dataset [Brostow et al. 2008b,a] where all the training and test points are rotated by a random angle (see Appendix Figure A.20). For any fixed image we always use the same rotation angle, so no two copies of the same image with different rotations are present in the data. We use the FC-Densenet segmentation architecture [Jégou et al. 2017]. We train Augerino with a Gaussian distribution over random rotations and translations.

In Appendix Figure A.20 we visualize the training data and learned augmentations for Augerino. Augerino is able to successfully recover rotational augmentation while matching the performance of the baseline. For further details, please see Appendix A.4.3.

3.16 Color-Space Augmentations

In the previous sections we have focused on learning spatial invariances with Augerino. Augerino is general and can be applied to arbitrary differentiable input transformations. In this section, we demonstrate that Augerino can learn color-space invariances.

We consider two color-space augmentations: brightness adjustments and contrast adjustments. Each of these can be implemented as simple differentiable transformations to the RGB values of the input image (for details, see Appendix A.4.5). We use Augerino to learn a uniform

distribution over the brightness and contrast adjustments on STL-10 [Coates et al. 2011] using the 13-layer CNN architecture (see Section 3.11). For both Augerino and the baseline model, we use standard spatial data augmentation: random translations, flips and cutout [DeVries and Taylor 2017]. The baseline model achieves 89.0 ± 0.35% accuracy where the mean and standard deviation are computed over 3 independent runs. The Augerino model achieves a slightly higher 89.7 ± 0.3% accuracy and learns to be invariant to noticeable brightness and contrast changes in the input image (see Appendix Figure A.21).

3.17 Discussion

The world is rife with equivariance and invariance, especially in real world settings where symmetries are not always perfectly preserved or present over a full range of transformations. In this chapter we have introduced *Residual Prior Priors* (RPPs), and *Augerino*, two frameworks for controlling the invariance and equivariance properties of neural network functions. RPPs and Augerino convert restrictive priors with hard constraints into priors that favor partially equivariant or invariant functions. With these functional inductive biases, Augerino and RPP allow us to utilize equivariance and invariance in a broader range of settings, including those where the exact symmetries are either unknown a priori, or only approximately held.

We have shown that RPPs and Augerino can improve the performance of neural networks in a variety of settings, including image recognition, regression, and reinforcement learning, while requiring little modification to the overall training procedure or optimization routine. The hope of this chapter is to explore further methods for engaging with the functional properties of neural networks, both through modifying the architecture of the network itself, as we do with RPPs, or through modifying the training procedure as we do with Augerino.

4 | Connecting Parameter and Function Spaces in Neural Networks

Chapter 3 explored methods for functional reasoning in neural networks to develop models with soft equivariance and invariance constraints. Equivariance and invariance represent a special case in which we are explicitly concerned with a *functional* property of our models. A more general case for neural networks is one where we are not concerned with a specific functional property, but rather with constructing a model that leads to accurate predictions and have limited insight into the functional form of the model.

Residual pathway priors and Augerino allowed us to specify priors that lead to desired functional properties, like equivariance, and posteriors that were interpretable with respect to those functional properties. The more general case, however, usually involves reasoning about parameter space priors and posteriors that are not interpretable with respect to functional properties. In this chapter we will expand on the connections between parameter and function space in neural networks, and explore how we can use these connections to build more accurate models.

By first exploring the types of solutions typically found in parameter space, and measuring the *posterior contraction* of these solutions, we can gain insight into how both parameter distributions, and the implied functional distributions, change through training. With this insight about parameter space, can then explore how we can use these connections to build more accurate models.

This chapter is adapted from the papers "Rethinking Parameter Counting in Deep Models: Effective Dimensionality Revisited" which was originally made available in 2020 and is joint work with Wesley Maddox, and Andrew Gordon Wilson, and "Loss Surface Simplexes for Mode Connecting Volumes and Fast Ensembling" which was originally published at ICML in 2021 and is joint work with Wesley Maddox, Sanae Lotfi, and Andrew Gordon Wilson.

4.1 Flatness and Functions in Neural Network Loss Landscapes

Parameter counting is often used as a proxy for model complexity to reason about generalization [e.g., Zhang et al. 2017; Shazeer et al. 2017; Belkin et al. 2019a], but it can be a poor description of both model flexibility and inductive biases. One can easily construct degenerate cases, such as predictions being generated by a sum of parameters, where the number of parameters is divorced from the statistical properties of the model. When reasoning about generalization, *overparametrization* is besides the point: what matters is how the parameters combine with the functional form of the model.

Indeed, the practical success of convolutional neural networks (CNNs) for image recognition tasks is almost entirely about the inductive biases of convolutional filters, depth, and sparsity, for extracting local similarities and hierarchical representations, rather than flexibility [LeCun et al. 1989; Szegedy et al. 2015]. Convolutional neural networks have far fewer parameters than fully connected networks, yet can provide much better generalization. Moreover, width can provide flexibility, but it is *depth* that has made neural networks distinctive in their generalization abilities.

In the following sections, we move beyond simple parameter counting, and show how the functional properties of neural networks become interpretable through the lens of *effective dimensionality* [MacKay 1992b]. Effective dimensionality was originally proposed to measure how many directions in parameter space had been determined in a Bayesian neural network, by com-

puting the eigenspectrum of the Hessian on the training loss (Eq. (4.3), Section 4.2). We provide explicit connections between effective dimensionality, posterior contraction, and loss surfaces in modern deep learning.

4.2 Posterior Contraction, Effective Dimensionality, and the Hessian

We consider a model, typically a neural network, $f(x;\theta)$, with inputs x and parameters $\theta \in \mathbb{R}^k$. We define the Hessian as the $k \times k$ matrix of second derivatives of the loss, $\mathcal{H}_\theta = -\nabla\nabla_\theta \mathcal{L}(\theta, \mathcal{D})$. Often the loss used to train a model by optimization is taken to be the negative log posterior $\mathcal{L} = -\log p(\theta|\mathcal{D})$.

To begin, we describe posterior contraction, effective dimensionality, and connections to the Hessian.

4.2.1 Posterior Contraction

Definition 4.1. We define *posterior contraction* of a set of parameters, θ, as the difference in the trace of prior and posterior covariance.

$$\Delta_{post}(\theta) = tr(Cov_{p(\theta)}(\theta)) - tr(Cov_{p(\theta|\mathcal{D})}(\theta)), \tag{4.1}$$

where $p(\theta)$ is the prior distribution and $p(\theta|\mathcal{D})$ is the posterior distribution given data, $\mathcal{D}$.

With increases in data the posterior distribution of parameters becomes increasingly concentrated around a single value [e.g., van der Vaart 1998, Chapter 10]. Therefore Eq. (4.1) serves to measure the increase in certainty about the parameters under the posterior as compared to the prior.

4.2.2 Parameter Space and Function Space

When combined with the functional form of a model, a distribution over parameters $p(\theta)$ induces a distribution over functions $p(f(x;\theta))$. The parameters are of little direct interest — what matters for generalization is the distribution over functions. Figure 4.1 provides both *parameter-* and *function-space* viewpoints. As parameter distributions concentrate around specific values, we expect to generate less diverse functions.

We show in Appendix A.5.3 that the posterior contraction for Bayesian linear regression, $y \sim \mathcal{N}(f = \Phi^\top \beta, \sigma^2 I)$, with isotropic Gaussian prior, $\beta \sim \mathcal{N}(0, \alpha^2 I_N)$, is given by

$$\Delta_{post}(\theta) = \alpha^2 \sum_{i=1}^{N} \frac{\lambda_i}{\lambda_i + \alpha^{-2}}, \tag{4.2}$$

where λ_i are the eigenvalues of $\Phi^\top \Phi$. This quantity is distinct from the posterior contraction in function space (also shown in Appendix A.5.3). We refer to the summation in Eq. (4.2) as the *effective dimensionality* of $\Phi^\top \Phi$.

4.2.3 Effective Dimensionality

Definition 4.2. The effective dimensionality of a symmetric matrix $A \in \mathbb{R}^{k \times k}$ is defined as

$$N_{eff}(A, z) = \sum_{i=1}^{k} \frac{\lambda_i}{\lambda_i + z}, \tag{4.3}$$

in which λ_i are the eigenvalues of A and $z > 0$ is a regularization constant [MacKay 1992b].

Typically as neural networks are trained we observe a gap in the eigenspectrum of the Hessian of the loss [Sagun et al. 2017]; a small number of eigenvalues become large while the rest take on values near 0. In this definition of effective dimensionality, eigenvalues much larger than z contribute a value of approximately 1 to the summation, and eigenvalues much smaller than z

contribute a value of approximately 0.

4.2.4 The Hessian and the Posterior Distribution

We provide a simple example involving posterior contraction, effective dimensionality, and their connections to the Hessian. Figure 4.1 shows the prior and posterior distribution for a Bayesian linear regression model with a single parameter, with predictions generated by parameters drawn from these distributions. As expected from Sections 4.2.1 and 4.2.3, we see that the variance of the posterior distribution is significantly reduced from that of the prior — what we refer to here as *posterior contraction*.

We can see from Figure 4.1 that the arrival of data increases the curvature of the loss (negative log posterior) at the optimum. This increase in curvature of the loss that accompanies certainty about the parameters leads to an increase in the eigenvalues of the Hessian of the loss in the multivariate case. Thus, growth in eigenvalues of the Hessian of the loss corresponds to increased certainty about parameters, leading to the use of the effective dimensionality of the Hessian of the loss as a proxy for the number of parameters that have been determined.[1]

We often desire models that are both consistent with data, but as simple as possible in function space, embodying Occam's razor and avoiding overfitting. The effective dimensionality explains the number of parameters that have been determined by the data, which corresponds to the number of parameters the model is using to make predictions. Therefore in comparing models of the same parameterization that achieve low loss on the training data, we expect models with *lower* effective dimensionality to generalize better, which is empirically verified in Maddox et al. [2020].

We can further connect the Hessian and the posterior distribution by considering a Laplace approximation as in MacKay [1992b,a]. Here we assume that the distribution of parameters θ is multivariate normal around the maximum a posteriori (MAP) estimate, $\theta_{\text{MAP}} = \text{argmax}_\theta p(\theta|\mathcal{D})$,

[1] Empirically described in Appendix A.5.1.

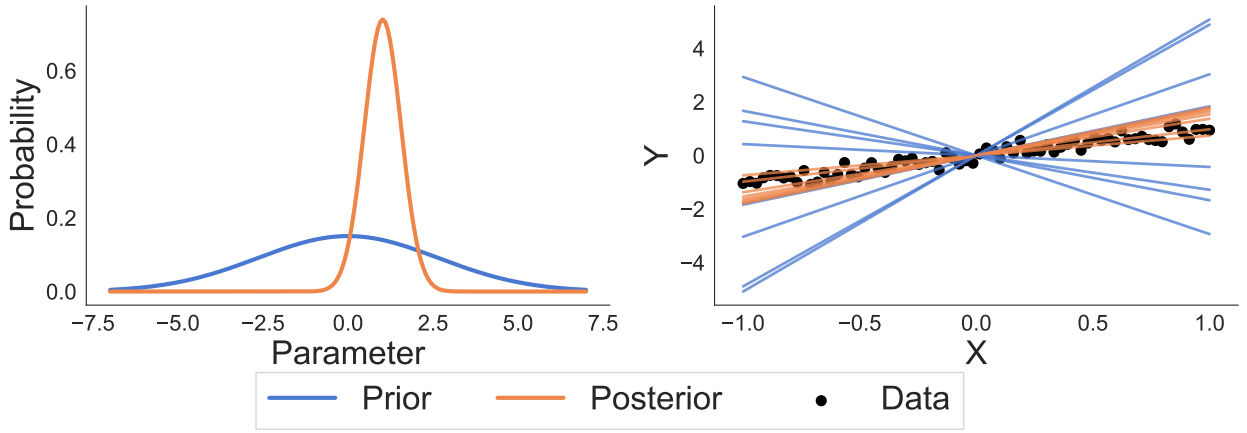


Figure 4.1: Left: A comparison of prior and posterior distributions in a Bayesian linear regression setting, demonstrating the decrease in variance referred to as *posterior contraction*. **Right:** Functions sampled from the prior and posterior distributions, along with the training data.

and the Hessian of the negative log posterior, $\mathcal{H}_\theta + A$,[2] serves as the precision matrix. The approximating distribution is then $\mathcal{N}(\theta_{\text{MAP}}, (\mathcal{H}_\theta + A)^{-1})$. The intuition built using Figure 4.1 carries through to this approximation: as the eigenvalues of the Hessian increase, the eigenvalues of the covariance matrix in our approximation to the posterior distribution shrink, further indicating contraction around the MAP estimate. We demonstrate this property algebraically in Appendix A.5.2, where we also connect the effective dimensionality to the bias-variance tradeoff [Dobriban and Wager 2018] and to the Hilbert space norm [Rasmussen and Williams 2008].

4.3 Effective Dimensionality Related Work

Cleveland [1979] introduced effective dimensionality into the splines literature as a measure of goodness of fit, while Hastie and Tibshirani [1990, Chapter 3] used it to assess generalized additive models. Gull [1989] first applied effective dimensionality in a Bayesian setting for an image reconstruction task, while MacKay [1992b,a] used it to compute posterior contraction in Bayesian neural networks. Moody [1992] argued for the usage of the effective dimensionality as a proxy for

[2] $A = -\nabla\nabla_\theta \log p(\theta)$ is the Hessian of the log prior.

generalization error, while Moody [1991] suggested that effective dimensionality could be used for neural network architecture selection. Zhang [2005] and Caponnetto and Vito [2007] studied the generalization abilities of kernel methods in terms of the effective dimensionality.

Friedman et al. [2001, Chapter 7] use the effective dimensionality (calling it the effective degrees of freedom) to compute the expected generalization gap for regularized linear models. Dobriban and Wager [2018] specifically tied the bias variance decomposition of predictive risk in ridge regression (e.g. the finite sample predictive risk under Gaussian priors) to the effective dimensionality of the feature matrix, $\Phi^\top \Phi$. Hastie et al. [2019], Muthukumar et al. [2019], Bartlett et al. [2019], Mei and Montanari [2019], and Belkin et al. [2019b] studied risk and generalization in over-parameterized linear models, including under model misspecification. Bartlett et al. [2019] also introduced the concept of effective rank of the feature matrix, which has a similar interpretation to effective dimensionality.

Sagun et al. [2017] found that the eigenvalues of the Hessian increase through training, while Papyan [2018] and Ghorbani et al. [2019] studied the eigenvalues of the Hessian for a range of modern neural networks. Suzuki [2018] produced generalization bounds on neural networks via the effective dimensionality of the covariance of the functions at each hidden layer. Fukumizu et al. [2019] embedded narrow neural networks into wider neural networks and studied the flatness of the resulting minima in terms of their Hessian via a PAC-Bayesian approach. Achille and Soatto [2018] argue that flat minima have low information content (many small magnitude eigenvalues of the Hessian) by connecting PAC-Bayesian approaches to information theoretic arguments, before demonstrating that low information functions learn invariant representations of the data. Dziugaite and Roy [2017] optimize a PAC-Bayesian bound to both encourage flatness and to compute non-vacuous generalization bounds, while Jiang et al. [2019] recently found that PAC-Bayesian measures of flatness, in the sense of insensitivity to random perturbations, perform well relative to other generalization bounds. Zhou et al. [2018] used PAC-Bayesian compression arguments to construct non-vacuous generalization bounds at the ImageNet scale.

Moreover, MacKay [2003] and Smith and Le [2017] provide an Occam factor perspective linking flatness and generalization. Related minimum description length perspectives can be found in MacKay [2003] and Hinton and Van Camp [1993]. Other works also link flatness and generalization [e.g., Hochreiter and Schmidhuber 1997a; Keskar et al. 2017; Chaudhari et al. 2019; Izmailov et al. 2018], with Izmailov et al. [2018] and Chaudhari et al. [2019] developing optimization procedures to select for flat regions of the loss.

4.4 Posterior Contraction and Function-Space Homogeneity in Bayesian Models

In this section, we demonstrate that effective dimensionality of both the posterior parameter covariance and the Hessian of the loss provides insights into how a model adapts to data during training. We derive an analytic relationship between effective dimensionality and posterior contraction for models where inference is exact, and demonstrate this relationship experimentally for deep neural networks.

4.4.1 Posterior Contraction of Bayesian Linear Models

Theorem 4.3 (Posterior Contraction in Bayesian Linear Models)**.** *Let $\Phi = \Phi(x) \in \mathbb{R}^{n \times k}$ be a feature map of n data observations, x, with $n < k$ and assign isotropic prior $\beta \sim \mathcal{N}(0_k, \alpha^2 I_k)$ for parameters $\beta \in \mathbb{R}^k$. Assuming a model of the form $y \sim \mathcal{N}(\Phi\beta, \sigma^2 I_n)$ the posterior distribution of β has a $k - n$ directional subspace in which the variance is identical to the prior variance.*

We prove Theorem 4.3 in Appendix A.5.4.1, in addition to an equivalent result for generalized linear models. Theorem 4.3 demonstrates why *parameter counting* often makes little sense: for a fixed data set of size n, only $\min(n, k)$ parameters can be determined, leaving many dimensions in which the posterior is unchanged from the prior when $k \gg n$.

Empirical Demonstration for Theorem 4.3. We construct $\Phi(x)$ with each row as an instance of a 200 dimensional feature vector consisting of sinusoidal terms for each of 500 observations: $\Phi(x) = [\cos(\pi x), \sin(\pi x), \cos(2\pi x), \sin(2\pi x), \dots]$. We assign the coefficient vector β a prior $\beta \sim \mathcal{N}(0, I)$, and draw ground truth parameters β^* from this distribution. The model takes the form $\beta \sim \mathcal{N}(0, I)$ and $y \sim \mathcal{N}\left(\Phi\beta, \sigma^2 I\right)$.

We randomly add data points one at a time, tracking the posterior covariance matrix at each step. We compute the effective dimensionality, $N_{eff}\left(\Sigma_{\beta|\mathcal{D},\sigma}, \alpha\right)$, where $\Sigma_{\beta|\mathcal{D},\sigma}$ is the posterior covariance of β.[3]

In Figure 4.2 we see that the effective dimensionality of the posterior covariance decreases linearly with an increase in available data until the model becomes overparameterized, at which point the effective dimensionality of the posterior covariance of the parameters slowly approaches 0, while the effective dimensionality of the Hessian of the loss increases towards an asymptotic limit. As the parameters become more determined (e.g. the effective dimensionality of the posterior covariance decreases), the curvature of the loss increases (the effective of the Hessian increases). In the Bayesian linear model setting, the Hessian of the loss is the inverse covariance matrix and the trade-off between the effective dimensionality of the Hessian and the parameter covariance can be determined algebraically (see Appendix A.5.2.1).

4.4.2 Posterior Contraction of Bayesian Neural Networks

While much effort has been spent grappling with the challenges of marginalizing a high dimensional parameter space for Bayesian neural networks, the practical existence of subspaces where the posterior variance has not collapsed from the prior suggests that both computational and approximation gains can be made from ignoring directions in which the posterior variance is unchanged from the prior. This observation helps explain the success of subspace based techniques that examine the loss in a lower dimensional space such as Izmailov et al. [2019a]. Alternatively,

[3]Here we use $\alpha = 5$, however the results remain qualitatively the same as this parameter changes.

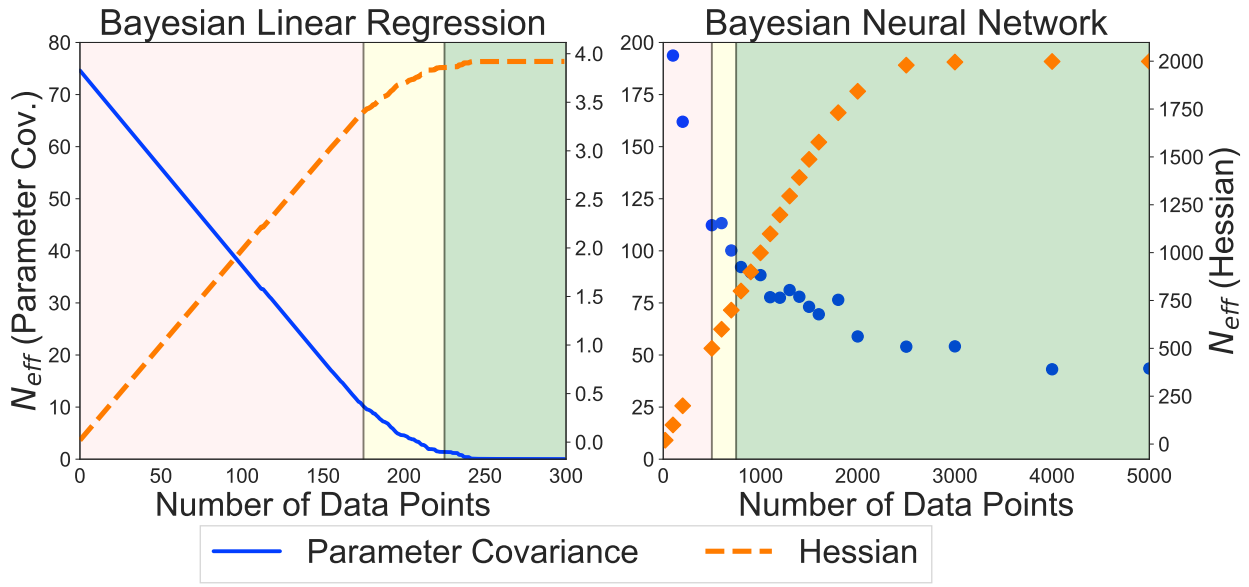


Figure 4.2: Left: Bayesian linear regression. **Right:** Bayesian neural network. **Both:** The effective dimensionality of the posterior covariance over parameters and the function-space posterior covariance. Red indicates the under-parameterized setting, yellow the critical regime with $p \approx n$, and green the over-parameterized regime. In both models we see the expected increase in effective dimensionality in parameter space and decrease in effective dimensionality of the Hessian.

by working directly in function space, as in Sun et al. [2019], the redundancy of many parameters could be avoided.

For Bayesian linear models, the effective dimensionality of the parameter covariance is the inverse of the Hessian, and as the effective dimensionality of the parameter covariance decreases the effective dimensionality of the Hessian increases. We hypothesize that a similar statement holds for Bayesian neural networks — as the number of data points grows, the effective dimensionality of the posterior covariance should decrease while the effective dimensionality of the Hessian should increase.

To test this hypothesis, we generate a nonlinear function of the form, $y = w_1 x + w_2 x^2 + w_3 x^3 + (0.5 + x^2)^2 + \sin(4x^2) + \epsilon$, with $w_i \sim \mathcal{N}(0, I)$ and $\epsilon \sim \mathcal{N}(0, 0.05^2)$, and de-mean and standardize the inputs.[4] We then construct a Bayesian neural network with two hidden layers each with 20 units,

[4]From the Bayesian neural network example in NumPyro [Phan et al. 2019; Bingham et al. 2019]: https://github.com/pyro-ppl/numpyro/blob/master/examples/bnn.py.

no biases, and *tanh* activations, placing independent Gaussian priors with variance 1 on all model parameters. We then run the No-U-Turn sampler [Hoffman and Gelman 2014] for 2000 burn-in iterations before saving the final 2000 samples from the approximated posterior distribution. Using these samples, we compute the effective dimensionality of the sample posterior covariance, $\mathrm{Cov}_{p(\theta|\mathcal{D})}(\theta)$, and Hessian of the loss at the MAP estimate in Figure 4.2. The trends of effective dimensionality for Bayesian neural networks are aligned with Bayesian linear regression, with the effective dimensionality of the Hessian (corresponding to function space) increasing while the effective dimensionality of the parameter space decreases.

4.4.3 Function-Space Homogeneity

In order to understand how the function-space representation varies as parameters are changed in directions *undetermined* by the data, we first consider Bayesian linear models.

Theorem 4.4 (Function-Space Homogeneity in Linear Models)**.** *Let $\Phi = \Phi(x) \in \mathbb{R}^{n\times k}$ be a feature map of n data observations, x, with $n < k$, and assign isotropic prior $\beta \sim \mathcal{N}(0_k, \alpha^2 I_k)$ for parameters $\beta \in \mathbb{R}^k$. The minimal eigenvectors of the Hessian define a $k - n$ dimensional subspace in which parameters can be perturbed without changing the training predictions in function space.*

We prove Theorem 4.4 and its extension to generalized linear models in Appendix A.5.4.2. This theorem suggests that although there may be large regions in parameter-space that lead to low-loss models, many of these models may be homogeneous in function space.

We can interpret Theorem 4.4 in terms of the eigenvectors of the Hessian indicating which directions in parameter space have and have not been determined by the data. The dominant eigenvectors of the Hessian (those with the largest eigenvalues) correspond to the directions in which the parameters have been determined from the data and the posterior has contracted significantly from the prior. The minimal eigenvectors (those with the smallest eigenvalues) correspond to the directions in parameter space in which the data has not determined the parameters.

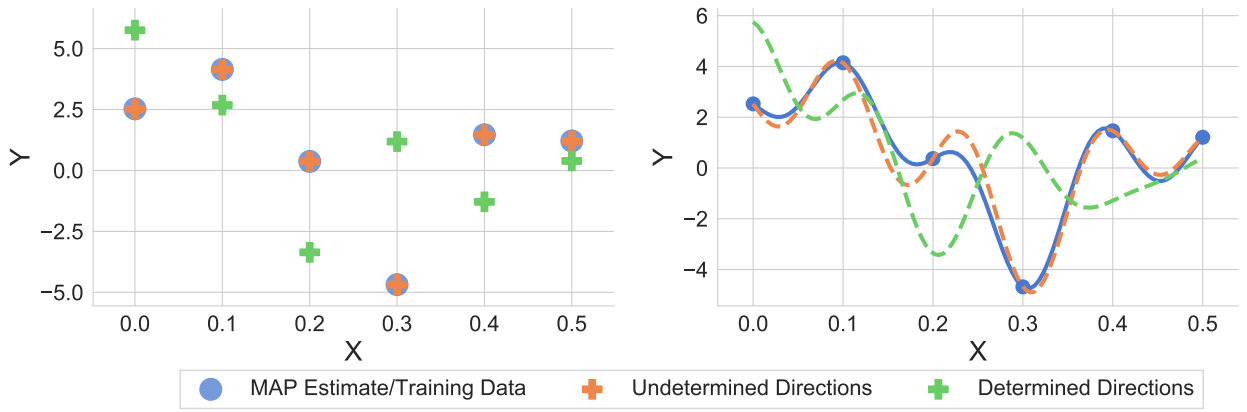


Figure 4.3: Left: The predictions on training data for a simple Bayesian linear regression model with sinusoidal features for various parameter settings. **Right:** The predictions over the entire test domain. **Both:** Blue represents the MAP estimate as well as the training points, orange represents the model after the parameters have been perturbed in a direction in which the posterior has not contracted, and green represents the model after parameters have been perturbed in a direction in which the posterior has contracted. Perturbing parameters in directions that have not been determined by the data gives not only identical predictions on training data, but the functions produced on the test set are nearly the same.

Figure 4.3 demonstrates the result of Theorem 4.4 for a Bayesian linear model with sinusoidal features. We compare predictions made using the MAP estimate of the parameters, $\theta^* = \text{argmax}_\theta p(\theta|\mathcal{D})$, to predictions generated using perturbed parameters. As parameters are perturbed in directions that have not been determined by the data (minimal eigenvectors of the Hessian), the predictions on both train and test remain nearly identical to those generated using the MAP estimate. Perturbations in determined directions (dominant eigenvectors of the Hessian) yield models that perform poorly on the training data and significantly deviate from the MAP estimate on the test set.

4.5 Loss Surfaces and Function Space Representations

Recent works have discussed the desirability of finding solutions corresponding to *flat* optima in the loss surface, arguing that such parameter settings lead to better generalization [Izmailov et al. 2018; Keskar et al. 2017]. There are multiple notions of flatness in loss surfaces, relating to both the volume of the basin in which the solution resides and the rate of increase in loss as

one moves away from the found solution. As both definitions correspond to low curvature in the loss surface, it is standard to use the Hessian of the loss to examine structure in the loss surface [Madras et al. 2019; Keskar et al. 2017].

The effective dimensionality of the Hessian of the loss indicates the number of parameters that have been determined by the data. In highly over-parameterized models we hypothesize that the effective dimensionality is substantially less than the number of parameters, i.e. $N_{eff}(\mathcal{H}_\theta, \alpha) \ll p$, since we should be unable to determine many more parameters than we have data observations.

Recall from Section 4.2.3 the large eigenvalues of the Hessian have eigenvectors corresponding to directions in which parameters are determined. Eq. (4.3) dictates that low effective dimensionality (in comparison to the total number of parameters) would imply that there are many directions in which parameters are not determined, and the Hessian has eigenvalues that are near zero, meaning that in many directions the loss surface is constant. We refer to directions in parameter space that have not been determined as *degenerate* for two reasons: (1) degenerate directions in parameter space provide minimal structure in the loss surface, shown in Section 4.5.1; (2) parameter perturbations in degenerate directions do not provide diversity in the function-space representation of the model, shown in Section 4.5.2. We refer to the directions in which parameters have been determined, directions of high curvature, as *determined*.

To empirically test our hypotheses regarding degenerate directions in loss surfaces and function space diversity, we train a neural network classifier on 1000 points generated from the two-dimensional Swiss roll data, with a similar setup to Huang et al. [2019], using Adam with a learning rate of 0.01 [Kingma and Ba 2014]. The network is fully connected, consisting of 5 hidden layers each 20 units wide (plus a bias term), and uses ELU activations with a total of 2181 parameters. We choose a small model with two-dimensional inputs so that we can both tractably compute all the eigenvectors and eigenvalues of the Hessian and visualize the functional form of the model. To demonstrate the breadth of these results, we provide comparable visualizations in the Appendix A.5.5, but for a convolutional network trained on CIFAR-10.

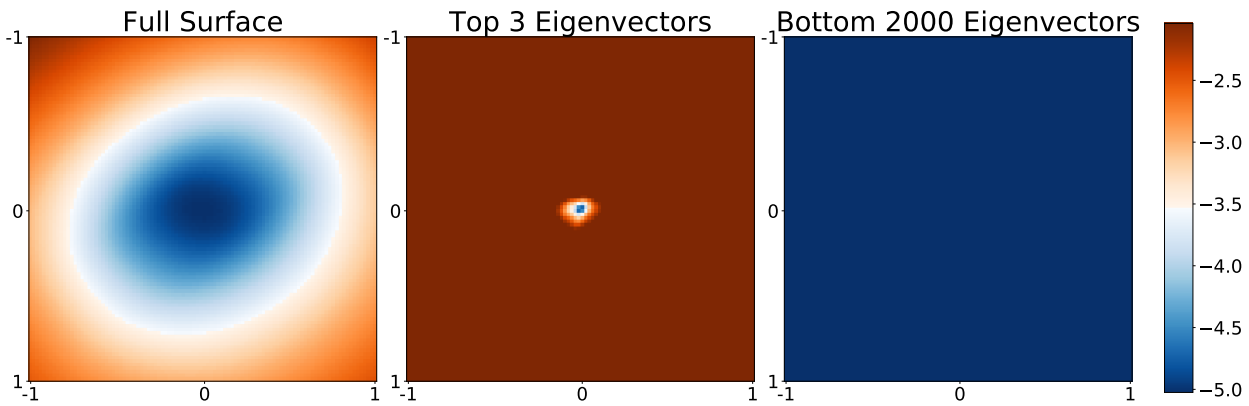


Figure 4.4: Left: A random projection of the loss surface. **Center:** A projection of the loss surface in the top 3 directions in which parameters have been determined. **Right:** A projection of the loss surface in the 2000 (out of 2181) directions in which parameters have been determined the least. The rightmost plot shows that in degenerate parameter directions the loss is constant.

4.5.1 Loss Surfaces as Determined by the Hessian

To examine the loss surface more closely, we visualize low dimensional projections. To create the visualizations, we first define a basis given by a set of vectors, then choose a two random vectors, u and $\widetilde{v}$, within the span of the basis. We use Gram-Schmidt to orthogonalize $\widetilde{v}$ with respect to u, ultimately giving u and v with $u \perp v$. We then compute the loss at parameter settings θ on a grid surrounding the optimal parameter set, θ^*, which are given by

$$\theta \leftarrow \theta^* + \alpha u + \beta v \tag{4.4}$$

for various α and β values such that all points on the grid are evaluated.

By selecting the basis in which u and v are defined we can specifically examine the loss in determined and degenerate directions. Figure 4.4 shows that in determined directions, the optimum appears extremely sharp. Conversely, in all but the most determined directions, the loss surface loses all structure and appears constant. Even in degenerate directions, if we deviate from the optimum far enough the loss will eventually become large. However to observe this increase in loss requires perturbations to the parameters that are significantly larger in norm than θ^*.

4.5.2 Degenerate Parameters Lead to Homogeneous Models

In this section we show that degenerate parameter directions do not contain diverse models. This result is not at odds with the notion that flat regions in the loss surface can lead to diverse but high performing models. Rather, we find that there is a subspace in which the loss is constant and one cannot find model diversity, noting that this subspace is distinct from those employed by works such as Izmailov et al. [2019a] and Huang et al. [2019]. This finding leads to an interpretation of effective dimensionality as *model compression*, since the undetermined directions do not contain additional functional information.

We wish to examine the functional form of models obtained by perturbing the parameters found through training, θ^*. Perturbed parameters are computed as

$$\theta \leftarrow \theta^* + s\frac{Bv}{||Bv||_2} \tag{4.5}$$

where $B \in \mathbb{R}^{k\times d}$ is a d dimensional basis in which we wish to perturb θ^*, and $v \sim \mathcal{N}(0, I_d)$, giving Bv as a random vector from within the span of some specified basis (i.e. the dominant or minimal eigenvectors). The value s is chosen to determine the scale of the perturbation, i.e. the length of the random vector by which the parameters are perturbed.

Experimentally, we find that in a region near the optimal parameters θ^*, i.e. $s \leq ||\theta^*||_2/2$ the function-space diversity of the model is contained within the subspace of determined directions. While the degenerate directions contain wide ranges of parameter settings with low loss, the models are equivalent in function space.

Figure 4.5 shows the trained classifier and the differences in function-space between the trained classifier and those generated from parameter perturbations. We compare perturbations of size $||\theta^*||_2/2 \approx 10$ in the direction of the 500 minimal eigenvectors and perturbations of size 0.1 in the directions of the 3 maximum eigenvectors. A perturbation from the trained parameters in the directions of low curvature (center plot in Figure 4.5) still leads to a classifier that labels

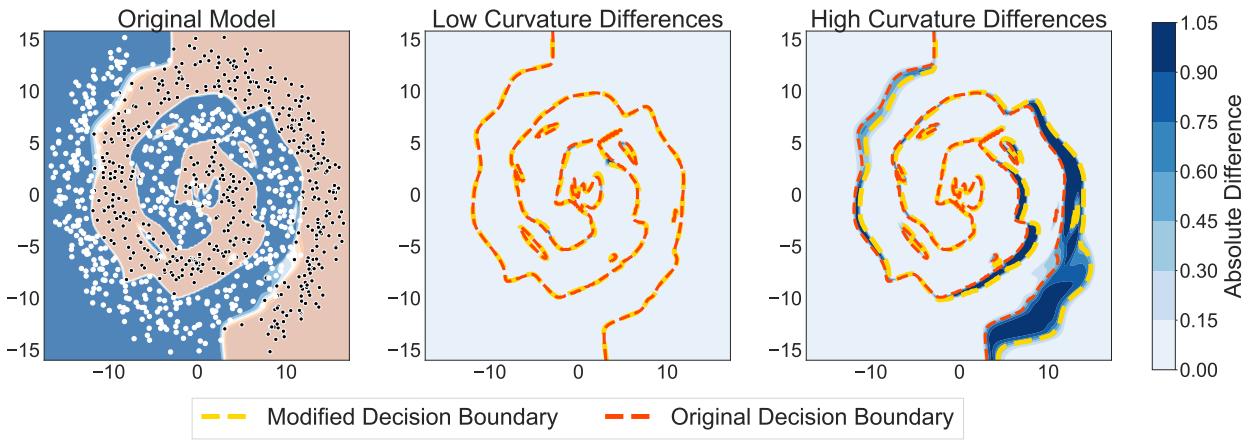


Figure 4.5: Swiss roll data. **Left:** Adam trained feed-forward, fully connected classifier. **Center:** Differences in original and perturbed classifier when parameters are perturbed by in low curvature, degenerate directions. **Right:** Differences in the original and perturbed classifier when parameters are perturbed in high curvature directions. **Note** the perturbation in the center plot is approximately 100 times the size of that of the plot on the right.

all points identically. A perturbation roughly 100 times smaller the size in directions in which parameters have been determined leads to a substantial change in the decision boundary of the classifier.

However, the change in the decision boundary resulting from perturbations in determined directions is not necessarily desirable. One need not perturb parameters in either determined or degenerate directions to perform a downstream task such as ensembling. Here, we are showcasing the degeneracy of the subspace of parameter directions that have not been determined by the data. This result highlights that despite having many parameters the network could be described by a relatively low dimensional subspace.

4.6 Loss Surface Simplexes for Mode Connecting Volumes and Fast Ensembling

So far in this chapter we have explored the ways that neural network parameter distributions contract through training, and how we can relate the axes of posterior contraction to stability

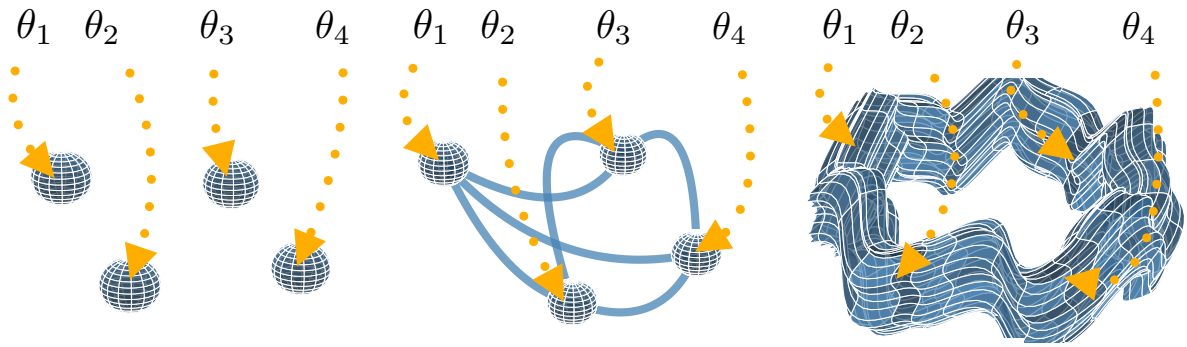


Figure 4.6: A progressive understanding of the loss surfaces of neural networks. **Left:** The traditional view of loss in parameter space, in which regions of low loss are disconnected [Goodfellow et al. 2015; Choromanska et al. 2015]. **Center:** The revised view of loss surfaces provided by work on mode connectivity; multiple SGD training solutions are connected by narrow tunnels of low loss [Garipov et al. 2018; Draxler et al. 2018; Fort and Jastrzebski 2019]. **Right:** The viewpoint introduced in this work; SGD training converges to different points on a connected *volume* of low loss. Paths between different training solutions exist within a large multi-dimensional manifold of low loss. We provide a two dimensional representation of these loss surfaces in Figure A.27.

in function space. Key to the findings in the previous sections are that in standard training we typically find flat optima in the loss surface. In the following sections we exploit this flatness to show that there are in fact large multi-dimensional simplicial complexes of low loss in the parameter space of neural networks that contain arbitrarily many modes independently trained SGD solutions.

The ability to find these large volumes of low loss that can connect any number of independent training solutions represents a natural progression in how we understand the loss landscapes of neural networks, as shown in Figure 4.6. In the left of Figure 4.6, we see the classical view of loss surface structure in neural networks, where there are many isolated low loss modes that can be found through training randomly initialized networks. In the center we have a more contemporary view, showing that there are paths that connect these modes. On the right we present a new view — that all modes found through standard training converge to points within a single connected *multi-dimensional* volume of low loss.

We introduce Simplicial Pointwise Random Optimization (SPRO) as a method of finding sim-

plexes and simplicial complexes that bound volumes of low loss in parameter space. With SPRO we are able to find mode connecting spaces that simultaneously connect many independently trained models through a a single well-defined multi-dimensional manifold. Furthermore, SPRO is able to explicitly define a space of low loss solutions through determining the bounding vertices of the simplicial complex, meaning that computing the dimensionality and volume of the space become straightforward, as does sampling models within the complex.

This enhanced understanding of loss surface structure enables practical methodological advances. Through the ability to rapidly sample models from within the simplex we can form Ensembled SPRO (ESPRO) models. ESPRO works by generating a simplicial complex around independently trained models and ensembling from within the simplexes, outperforming the gold standard deep ensemble combination of independently trained models [Lakshminarayanan et al. 2017]. We can view this ensemble as an approximation to a Bayesian model average, where the posterior is uniformly distributed over a simplicial complex.

The remaining sections of this chapter are structured as follows: in Section 4.8, we introduce a method to discover multi-dimensional mode connecting simplexes in the neural network loss surface. In Section 4.9, we show the existence of mode connecting volumes and provide a lower bound on the dimensionality of these volumes. Building on these insights, in Section 4.10 we introduce ESPRO, a state-of-the-art approach to ensembling with neural networks, which efficiently averages over simplexes. In Section 4.11, we show that ESPRO also provides well-calibrated representations of uncertainty. We emphasize that ESPRO can be used as a simple drop-in replacement for deep ensembles, with improvements in accuracy and uncertainty representations.

4.7 SPRO Related Work

The study of neural network loss surfaces has long been intertwined with an understanding of neural network generalization. Hochreiter and Schmidhuber [1997b] argued that *flat* minima

provide better generalization, and proposed an optimization algorithm to find such solutions. Keskar et al. [2017] and Li et al. [2018] reinvigorated this argument by visualizing loss surfaces and studying the geometric properties of deep neural networks at their minima. Izmailov et al. [2018] found that averaging SGD iterates with a modified learning rate finds flatter solutions that generalize better. Maddox et al. [2019a] leveraged these insights in the context of Bayesian deep learning to form posteriors in flat regions of the loss landscape. Moreover, Maddox et al. [2020] found many directions in parameter space that can be perturbed without changing the training or test loss.

Freeman and Bruna [2017] demonstrated that single layer ReLU neural networks can be connected along a low loss curves. Garipov et al. [2018] and Draxler et al. [2018] simultaneously demonstrated that it is possible to find low loss curves for ResNets and other deep networks. Skorokhodov and Burtsev [2019] used multi-point optimization to parameterize wider varieties of shapes in loss surfaces, when visualizing the value of the loss, including exotic shapes such as cows. Czarnecki et al. [2019] then showed that low dimensional spaces of nearly constant loss theoretically exist in the loss surfaces of deep ReLU networks, but did not provide an algorithm to find these loss surfaces.

Fort and Jastrzebski [2019] propose viewing the loss landscape as a series of potentially connected low-dimensional wedges in the much higher dimensional parameter space. They then demonstrate that sets of optima can be connected via low-loss connectors that are generalizations of Garipov et al. [2018]'s procedure. Our work generalizes these findings by discovering higher dimensional mode connecting volumes, which we then leverage for a highly efficient and practical ensembling procedure.

Also appearing at the same conference as this work, Wortsman et al. [2021] concurrently proposed a closely related technique to learning low dimensional neural network subspaces by extending the methods of Fort et al. [2019] and Garipov et al. [2018]. Wortsman et al. [2021] propose learning simplexes in parameter space with a regularization penalty to encourage diversity

in weight space.

4.8 Mode Connecting Volumes

We now show how to generalize the procedure of Garipov et al. [2018] to discover simplices of mode connecting *volumes*, containing infinitely many mode connecting curves. In Section 4.8, we then show how to use our procedure to demonstrate the existence of these volumes in modern neural networks, revising our understanding about the structure of their loss landscapes. In Sections 4.10 and 4.11 we show how to we can use these discoveries to build practical new methods which provide state of the art performance for both accuracy and uncertainty representation. We refer to our approach as SPRO (Simplicial Pointwise Random Optimization).

4.8.1 Simplicial Complexes of Low Loss

To find mode connecting volumes we seek *simplexes* and *simplicial complexes* of low loss. Two primary reasons we seek simplexes of low loss are that (i) simplexes are defined by only a few points, and (ii) simplexes are easily sampled. The first point means that to define a mode connecting simplicial complex of low loss we need only find a small number of vertices to fully determine the simplexes in the complex. The second point means that we have easy access to the models contained within the simplex, leading to the practical simplex-based ensembling methods presented later in the paper.

We consider data $\mathcal{D}$, and training objective $\mathcal{L}$. We refer to $S_{(a_0,a_1,\ldots,a_k)}$ as the k-simplex formed by vertices $a_0, a_1, \ldots, a_k$, and $\mathrm{V}(S_{(a_0,\ldots,a_k)})$ as the volume of the simplex.[5] Simplicial complexes are denoted $\mathcal{K}(S_{(a_0,a_1,\ldots,a_{N_a})}, S_{(b_0,b_1,\ldots,b_{N_b})}, \ldots, S_{(m_0,m_1,\ldots,m_{N_m})})$, and their volume is computed as the sum of the volume of their components. We use w_j to denote *modes*, or SGD training solutions, and θ_j to denote mode connecting points. For example, we could train two independent models to

[5] We use Cayley-Menger determinants to compute the volume of simplexes; for more information see Appendix A.6.2.

find parameter settings w_0 and w_1, and then find mode connecting point θ_0 such that the path $w_0 \rightarrow \theta_0 \rightarrow w_1$ traversed low loss parameter settings as in Fort and Jastrzebski [2019] and Garipov et al. [2018].

4.8.2 Simplicial Complexes With SPRO

To find a simplicial complex of low loss solutions, we first find a collection of modes $w_0, \ldots, w_k$ through standard training. This procedure gives the trivial simplicial complex $\mathcal{K}(S_{(w_0)}, \ldots, S_{(w_k)})$ (or $\mathcal{K}$), a complex containing k disjoint 0-simplexes. With these modes we can then iteratively add connecting points, θ_j, to join any number of the 0-simplexes in the complex, and train the parameters in θ_j such that the loss within the simplicial complex, $\mathcal{K}$, remains low. The procedure to train these connecting θ_j forms the core of the SPRO algorithm, given here.

To gain intuition, we first consider some examples before presenting the full SPRO training procedure. As we have discussed, we can take modes w_0 and w_1 and train θ_0 to find a complex $\mathcal{K}(S_{(w_0,\theta_0)}, S_{(w_1,\theta_0)})$, which recovers a mode connecting path as in Garipov et al. [2018]. Alternatively, we could connect θ_0 with more than two modes and build the complex $\mathcal{K}(S_{(w_0,\theta_0)}, \ldots, S_{(w_4,\theta_0)})$, connecting 5 modes through a single point, similar to the m-tunnels presented in Fort and Jastrzebski [2019]. SPRO can be taken further, however, and we could train (one at a time) a sequence of θ_j's to find the complex $\mathcal{K}(S_{(w_0,\theta_0,\theta_1,\theta_2)}, S_{(w_1,\theta_0,\theta_1,\theta_2)}, S_{(w_2,\theta_0,\theta_1,\theta_2)})$, describing a multi-dimensional volume that simultaneously connects 3 modes through 3 shared points.

We aim to train the θ_j's in $\mathcal{K}$ such that the expected loss for models in the simplicial complex is low and the volume of the simplicial complex is as large as possible. That is, as we train the j^{th} connecting point, θ_j, we wish to minimize $\mathbb{E}_{\phi\sim\mathcal{K}}\mathcal{L}(\mathcal{D}, \phi)$ while maximizing $\mathrm{V}(\mathcal{K})$, using $\phi \sim \mathcal{K}$ to indicate ϕ follows a uniform distribution over the simplicial complex $\mathcal{K}$.

Following Garipov et al. [2018], we use H parameter vectors randomly sampled from the simplex, $\phi_{h=1}^{H} \sim \mathcal{K}$, to compute $\frac{1}{H}\sum_{h=1}^{H}\mathcal{L}(\mathcal{D}, \phi_h)$ as an estimate of $\mathbb{E}_{\phi\sim\mathcal{K}}\mathcal{L}(\mathcal{D}, \phi)$.[6] In practice we

[6] We discuss the exact method for sampling, and the implications on bias in the loss estimate in Appendix A.6.2.

only need a small number of samples, H, and for all experiments use $H = 5$ to balance between avoiding significant slowdowns in the loss function and ensuring we have reasonable estimates of the loss over the simplex. Using this estimate we train θ_j by minimizing the regularized loss,

$$\mathcal{L}_{reg}(\mathcal{K}) = \frac{1}{H} \sum_{\phi_h \sim \mathcal{K}} \mathcal{L}(\mathcal{D}, \phi_h) - \lambda_j \log(\mathrm{V}(\mathcal{K})). \tag{4.6}$$

The regularization penalty λ_j balances the objective between seeking a smaller simplicial complex that contains strictly low loss parameter settings (small λ_j), and a larger complex that that may contain less accurate solutions but encompasses more volume in parameter space (large λ_j). In general only a small amount of regularization is needed, and results are not sensitive to the choice of λ_j. In Section 4.10 we explain how to adapt Eq. 4.6 to train simplexes of low loss using single independetly trained models.... We provide details about how we choose λ_j in Appendix A.6.2.1.

4.9 Volume Finding Experiments

In this section, we find volumes of low loss in a variety of settings. First, we show that the mode finding procedure of Garipov et al. [2018] can be extended to find distributions of modes. Then, we explore mode connecting simplicial complexes of low loss in a variety of settings, and finally provide an empirical upper bound on the dimensionality of the mode connecting spaces.

Loss Surface Plots. Throughout this section and the remainder of the paper we display two-dimensional visualizations of loss surfaces of neural networks. These plots represent the loss within the plane defined by the three points (representing parameter vectors) in each plot. More specifically, if the three points in question are, e.g., w_0, w_1, and w_2 then we define $c = \frac{1}{3}\sum_{i=0}^{2} w_i$ as the center of the points and use Gram-Schmidt to construct u and v, an orthonormal basis for the plane defined by the points. With the center and the basis chosen, we can sample the loss at parameter vectors of the form $w = c + r_u u + r_v v$ where r_u and r_v range from $-R$ to R, a range

parameter chosen such that all the points are within the surface with a reasonable boundary.

4.9.1 Volumes of Connecting Modes

In Bayesian deep learning, we wish to form a predictive distribution through a posterior weighted Bayesian model average:

$$p(y|x, \mathcal{D}) = \int p(y|w, x)p(w|\mathcal{D})dw\,, \tag{4.7}$$

where y is an output (e.g., a class label), x is an input (e.g., an image), $\mathcal{D}$ is the data, and w are the neural network weights. This integral is challenging to compute due to the complex structure of the posterior $p(w|\mathcal{D})$.

To help address this challenge, we can instead approximate the Bayesian model average in a subspace that contains many good solutions, as in Izmailov et al. [2019b]. Here, we generalize the mode connecting procedure of Garipov et al. [2018] to perform inference over subspaces that contain *volumes* of mode connecting curves.

In Garipov et al. [2018], a mode connecting curve is defined by its parameters θ. Treating the objective used to find θ in Garipov et al. [2018], $l(\theta)$, as a likelihood, we infer an approximate Gaussian posterior $q(\theta|\mathcal{D})$ using the SWAG procedure of Maddox et al. [2019a], which induces a distribution over mode connecting curves. Each sample from $q(\theta|D)$ provides a mode connecting curve, which itself contains a space of complementary solutions.

In Figure 4.7, we see that it is possible to move between different values of θ without leaving a region of low loss. We show samples from the SWAG posterior, projected into the plane formed by the endpoints of the curves, w_0 and w_1, and a mode connecting point θ_0. We show the induced connecting paths from SWAG samples with orange lines. All samples from the SWAG posterior lie in the region of low loss, as do the sampled connecting paths, indicating that there is indeed an entire volume of connected low loss solutions induced by the SWAG posterior over θ. We

provide training details in the Appendix A.6.3.

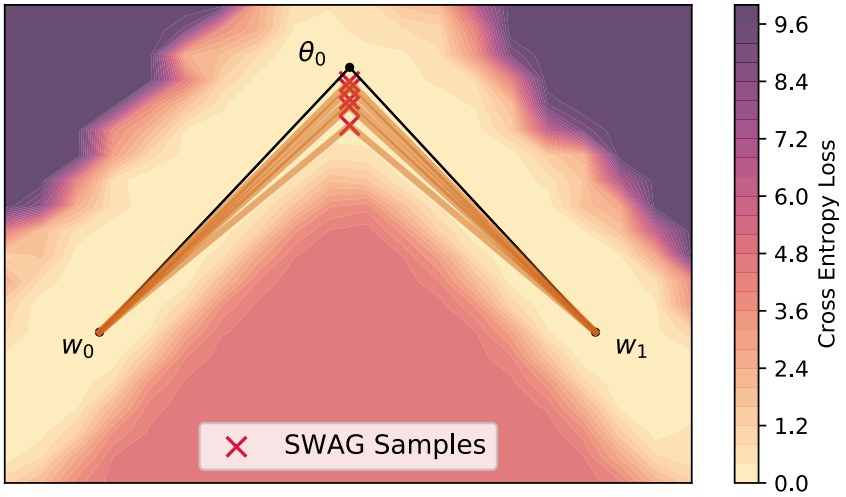


Figure 4.7: A loss surface in the basis spanned by the defining points of a connecting curve, w_0, w_1, θ_0. Using SWAG, we form a posterior distribution over mode connecting curves, representing a volume of low loss explanations for the data.

4.9.2 Simplicial Complex Mode Connectivity

The results of Section 4.9.1 suggest that modes might be connected by *multi-dimensional* paths. SPRO represents a natural generalization of the idea of learning a distribution over connecting paths. By construction, if we use SPRO to find the simplicial complex $\mathcal{K}(S_{(w_0,\theta_0,\ldots,\theta_k)}, \ldots, S_{(w_m,\theta_0,\ldots,\theta_k)})$ we have found a whole *space* of suitable vertices to connect the modes $w_0, \cdots, w_m$. Any θ sampled from the k-simplex $S_{(\theta_0,\ldots,\theta_k)}$ will induce a low-loss connecting path between any two vertices in the complex.

To demonstrate that SPRO finds volumes of low loss, we trained a simplicial complex using SPRO, $\mathcal{K}(S_{(w_0,\theta_0,\theta_1,\theta_2)}, S_{(w_1,\theta_0,\theta_1,\theta_2)})$, forming two simplexes containing three connecting vertices $\theta_0, \theta_1, \theta_2$ between the two fixed points, w_0 and w_1, which are pre-trained models.

Figure 4.8 shows loss surface visualizations of this simplicial complex in the parameter space of a VGG-16 network trained on CIFAR-10. We see that this complex contains not only standard

mode connecting paths, but also volumes of low loss that connect modes. Figure 4.8 is a straightforward representation of how the loss landscape of large neural networks should be understood as suggested in Figure 4.6; not only are all training solutions connected by paths of low loss, they are points on the same *multi-dimensional manifold* of low loss. In the bottom right panel of Figure 4.8, every point in the simplex corresponds to a different mode connecting curve.

In Figure 4.9, we show there exist manifolds of low loss that are vastly more intricate and high dimensional than a simple composition of 3-simplexes connecting two modes. In Figure 4.9(a), we connect 4 modes using 3 connecting points so that we have four different simplexes formed between the modes of low loss for VGG-16 networks [Simonyan and Zisserman 2015] on CIFAR-100. The structure becomes considerably more intricate as we expand the amount of modes used; Figure 4.9(b) uses 7 modes with 9 connecting points, forming 12 inter-connected simplexes. Note that in this case not all modes are in shared simplexes with all connecting points. These results clearly demonstrate that SPRO is capable of finding intricate and multi-dimensional structure within the loss surface. As a broader takeaway, *any* mode we find through standard training is a single point within a large and high dimensional structure of loss, as shown in the rightmost representation in Figure 4.6. We consider the accuracy of ensembles found via these mode connecting simplexes in Appendix A.6.4.4. In Section 4.10.4 we consider a particularly practical approach to ensembling with SPRO.

4.9.3 Dimensionality of Loss Valleys

We can estimate the highest dimensionality of the connecting space that SPRO can find, which provides a lower bound on the true dimensionality of these mode connecting subspaces for a given architecture and dataset. To measure dimensionality, we take two pre-trained modes, w_0 and w_1, and construct a connecting simplex with as many connecting points as possible, by finding the largest k such that $\mathcal{K}(S_{(w_0,\theta_0,\ldots,\theta_k)}, S_{(w_1,\theta_0,\ldots,\theta_k)})$ contains both low loss parameter settings and has non-zero volume. We could continue adding more degenerate points to the simplex; however,

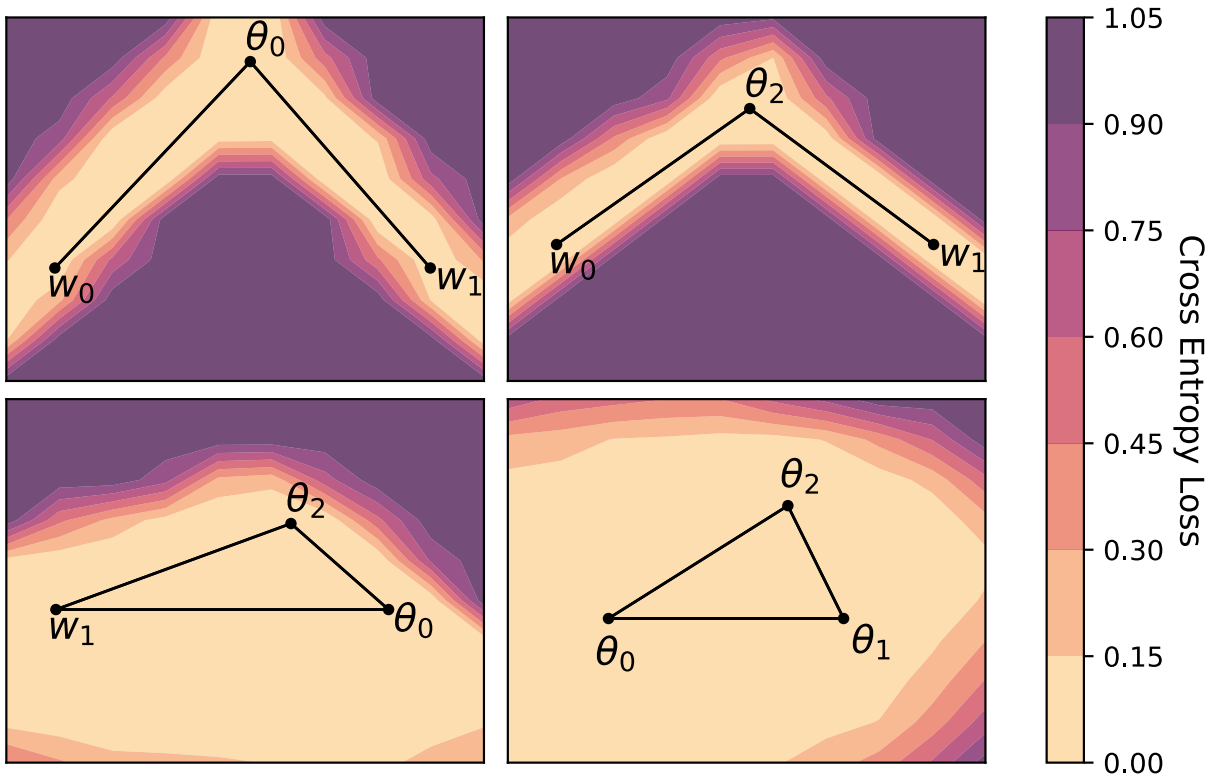
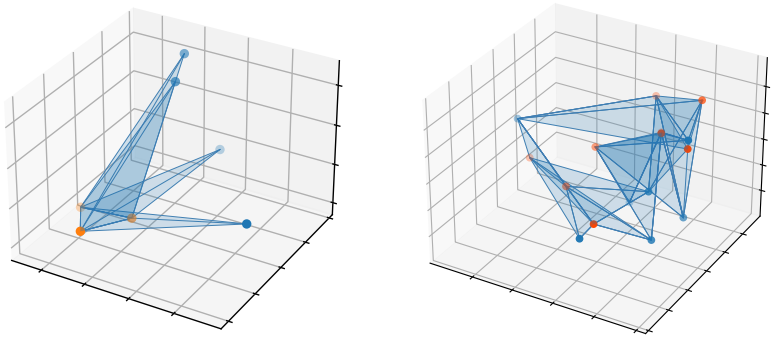


Figure 4.8: Loss surfaces for planes intersecting a mode connecting simplicial complex trained on CIFAR-10 using a VGG-16 network. **Top:** along any $w_0 \to \theta_j \to w_1$ path we recover a standard mode connecting path. **Bottom Left:** a face of one of the simplexes that contains one of the independently trained modes. We see that as we travel away from w_1 along any path within the simplex we retain low train loss. **Bottom Right:** the simplex defined by the three mode connecting points. Any point sampled from within this simplex defines a low-loss mode connecting path between w_0 and w_1.

the resulting simplicial complex has no volume.

Figure 4.10 shows the volume of a simplicial complex connecting two modes as a function of the number of connecting points, k, for a VGG-16 network on CIFAR-10. To ensure these are indeed low-loss complexes, we sample 25 models from each of these simplicial complexes and find that all sampled models achieve greater than 98% accuracy on the train set. We can continue adding new modes until we reach $k = 11$, when the volume collapses to approximately 10^{-4}, from a maximum of 10^5. Thus the dimensionality of the manifold of low loss solutions for this architecture and dataset is at least 10, as adding an eleventh point collapses the volume.



(a) 4 modes, 3 connectors. **(b)** 7 modes, 9 connectors.

Figure 4.9: (a,b) Three dimensional projections of mode connecting simplicial complexes with training modes shown in blue and connectors in orange. Blue shaded regions represent regions of low loss found via SPRO. **(a)** 4 modes and 3 connecting points found with a VGG-16 network on CIFAR-100. **(b)** 7 modes and a total of 9 connecting points found with a VGG-16 network on CIFAR-10.

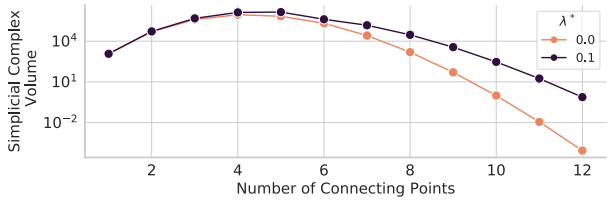


Figure 4.10: Volume of the simplicial complex as a function of the number of connectors for a VGG net on CIFAR-10 for two settings λ of SPRO regularization. After 10 connectors, the volume collapses, indicating that new points added to the simplicial complex are within the span of previously found vertices. The low-loss manifold must be at least 10 dimensions in this instance.

4.10 ESPRO: Ensembling with SPRO

The ability to find large regions of low loss solutions has significant practical implications: we show how to use SPRO to efficiently create ensembles of models either within a single simplex or by connecting an entire simplicial complex. We start by generalizing the methodology presented in Section 4.8.2, leading to a simplex based ensembling procedure, we call ESPRO (Ensembling SPRO). Crucially, our approach finds a low-loss simplex starting from only a *single* SGD solution. We show that the different parameters in these simplexes gives rise to a diverse set of functions, which is crucial for ensembling performance. Finally, we demonstrate that ESPRO outperforms

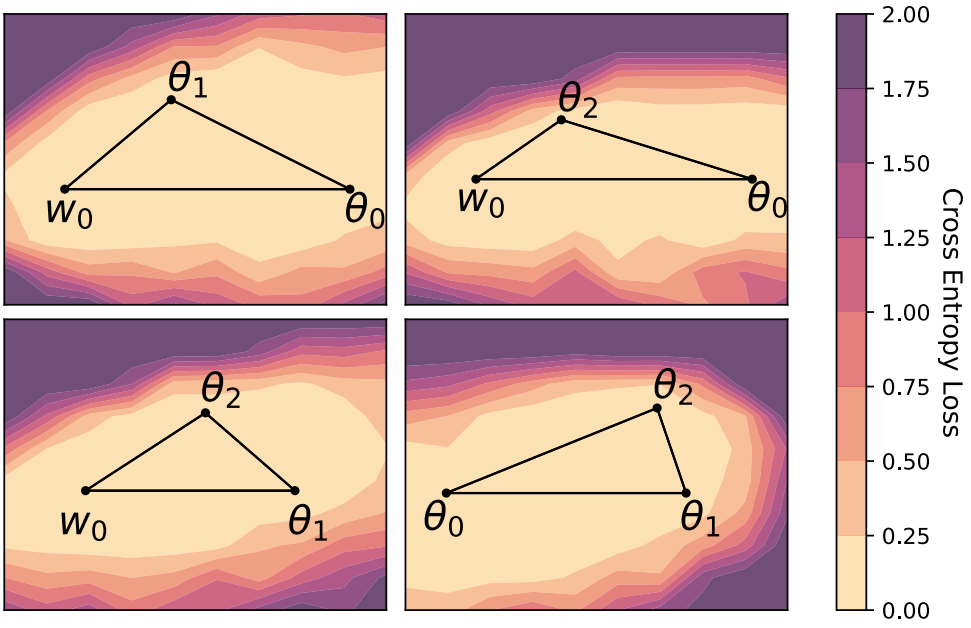


Figure 4.11: Loss surface visualizations of the faces of a sample ESPRO 3-simplex for a VGG network trained on CIFAR-100. The ability to find a low-loss simplex starting from only a *single* SGD solution, w_0, leads to an efficient ensembling procedure.

state-of-the-art *deep ensembles* [Lakshminarayanan et al. 2017], both as a function of ensemble components and total computational budget. In Section 4.11, we show ESPRO also provides state-of-the-art results for uncertainty representation.

4.10.1 Finding Simplexes from a Single Mode

In Section 4.8.2 we were concerned with finding a simplicial complex that connects multiple modes. We now describe how to adapt SPRO into a practical approach to ensembling by instead finding multiple simplexes of low loss, each — crucially — starting from a *single* pre-trained SGD solution.

Simplexes contain a single mode, and take the form $S_{(w_j,\theta_{j,0},\ldots,\theta_{j,k})}$ where the $\theta_{j,k}$ is the k^{th} vertex found with SPRO in a simplex where one of the vertices is mode w_j. We find SPRO simplexes one at a time, rather than as a complex. The associated loss function to find the k^{th} vertex in

association with mode w_j is

$$\mathcal{L}_{reg}(\mathcal{D}, S_{(w_j,\theta_{j,0},\ldots,\theta_{j,k})}) = \frac{1}{H}\sum_{\phi_h \sim S} \mathcal{L}(\mathcal{D}, \phi_h) - \lambda_i \log(\mathrm{V}(S_{(w_j,\theta_{j,0},\ldots,\theta_{j,k})})). \tag{4.8}$$

For compactness we write $\phi_h \sim S$ to indicate ϕ_h is sampled uniformly at random from simplex $S_{(w_j,\theta_{j,0},\ldots,\theta_{j,k})}$.

We can think of this training procedure as extending out from the pre-trained mode w_j. First, in finding $\theta_{j,0}$ we find a line segment of low loss solutions, where one end of the line is w_j. Next, with $\theta_{j,0}$ fixed, we seek $\theta_{j,1}$ such that the triangle formed by w_j, $\theta_{j,0}$, $\theta_{j,1}$ contains low loss solutions. We can continue adding vertices, constructing many dimensional simplexes.

With the resulting simplex $S_{(w_j,\theta_{j,0},\ldots,\theta_{j,k})}$, we can sample as many models from within the simplex as we need, and use them to form an ensemble. Functionally, ensembles sampled from SPRO form an approximation to Bayesian marginalization over the model parameters where we assume a posterior that is uniform over the simplex. We can define our prediction for a given input x as,

$$\hat{y} = \frac{1}{M}\sum_{\phi_m \sim S} f(x, \phi_m) \approx \int_{\phi_m \in S} f(x, \phi_h) d\phi_h, \tag{4.9}$$

where we write S as shorthand for $S_{(w_j,\theta_{j,0},\ldots,\theta_{j,k})}$. Specifically, the Bayesian model average and its approximation using approximate posteriors is

$$p(y^*|y, \mathcal{M}) = \int p(y^*|\phi)p(\phi|y)d\phi \approx \int p(y^*|\phi)q(\phi|y)d\phi \approx \frac{1}{M}\sum_{i=1}^{M} p(y^*|\phi_i); \quad \phi_i \sim q(\phi|y)$$

4.10.2 ESPRO: Ensembling over Multiple Independent Simplexes

We can significantly improve performance by ensembling from a simplicial complex containing multiple disjoint simplexes, which we refer to as ESPRO (Ensembling over SPRO simplexes). To form such an ensemble, we take a collection of j parameter vectors from independently trained models, $w_0, \ldots, w_j$, and train a k + 1-order simplex at each one using ESPRO. This procedure defines the simplicial complex $\mathcal{K}(S_{(w_0,\ldots,\theta_{0,k})}, \ldots, S_{(w_j,\ldots,\theta_{j,k})})$, which is composed of j disjoint simplexes in parameter space. Predictions with ESPRO are generated as,

$$\hat{y} = \frac{1}{J} \sum_{\phi_j \sim \mathcal{K}} f(x, \phi_j) \approx \int_{\mathcal{K}} f(x, \phi_j) d\phi_j \tag{4.10}$$

where $\mathcal{K}$ is shorthand for $\mathcal{K}(S_{(w_0,\ldots,\theta_{0,k})}, \ldots, S_{(w_j,\ldots,\theta_{j,k})})$. ESPRO can be considered a mixture of simplexes (e.g. a simplicial complex) to approximate a multimodal posterior, towards a more accurate Bayesian model average. This observation is similar to how Wilson and Izmailov [2020] show that deep ensembles provide a compelling approximation to a Bayesian model average (BMA), and improve on deep ensembles through the MultiSWAG procedure, which uses a mixture of Gaussians approximation to the posterior for a higher fidelity BMA. ESPRO further improves the approximation to the BMA, by covering a larger region of the posterior corresponding to low loss solutions with functional variability. This perspective helps explain why ESPRO improves both accuracy and calibration, through a richer representation of epistemic uncertainty.

We verify the ability of ESPRO to find a simplex of low loss starting from a single mode in Figure 4.11, which shows the loss surface in the planes defined by the faces of a 3-simplex found in the parameter space of a VGG-16 network trained on CIFAR-100. The ability to find these simplexes is core to forming ESPRO ensembles, as they only take a small number of epochs to find, typically less than 10% the cost of training a model from scratch, and they contain diverse solutions that can be ensembled to improve model performance. Notably, we can sweep out a volume of low loss in parameter space *without* needing to first find multiple modes, in contrast to prior

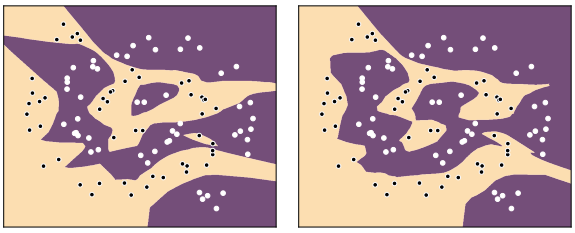


Figure 4.12: Functional diversity within a simplex. We show the decision boundaries for two classes, in the two spirals problem, with predictions in yellow and purple respectively. Both plots are independent solution samples drawn from a 3-simplex of an 8-layer feed forward classifier and demonstrate that the simplexes have considerable functional diversity, as illustrated by different decision boundaries. Significant differences are visible inside the data distribution (center of plots) and outside (around the edges).

work on mode connectivity [Draxler et al. 2018; Garipov et al. 2018; Fort and Jastrzebski 2019]. We show additional results with image transformers [Dosovitskiy et al. 2021] on CIFAR-100 in Appendix A.6.4.3, emphasizing that these simplexes are not specific to a particular architecture.

4.10.3 SPRO and Functional Diversity

In practice we want to incorporate as many diverse high accuracy classifiers as possible when making predictions to gain the benefits of ensembling, such as improved accuracy and calibration. SPRO gives us a way to sample diverse models in *parameter space*, and in this section we show, using a simple $2D$ dataset, that the parameter diversity found with SPRO is a reasonable proxy for the *functional diversity* we actually seek.

To better understand how the simplexes interact with the functional form of the model, we consider an illustrative example on the two-spirals classification dataset presented in Huang et al. [2019], in which predictions can be easily visualized. We find a 3-simplex (a tetrahedron) in the parameter space of a simple 8 layer deep feed forward classifier, and visualize the functional form of the model for both samples taken from within the simplex in parameter space. By examining the functional form of models sampled from simplexes in parameter space we can quickly see why ESPRO is beneficial. Figure 4.12 shows individual models sampled from a single 3-simplex in parameter space, corresponding to clear functional diversity. Models within the simplex all fit

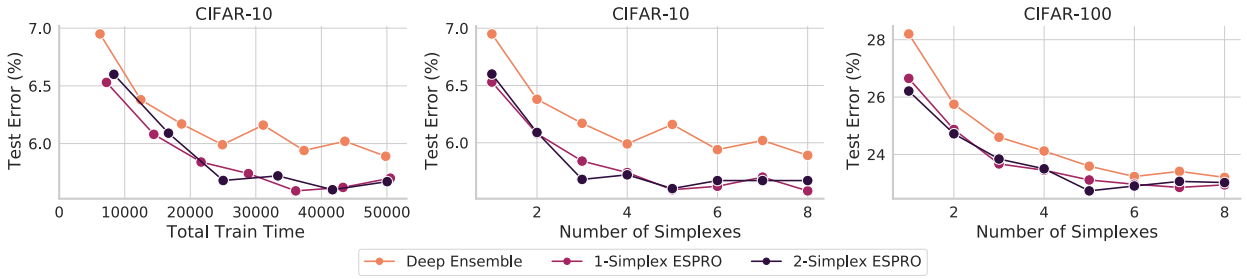


Figure 4.13: Performance of deep ensembles and ESPRO (with either a 1-simplex, e.g. a line or a 2-simplex, e.g. a triangle) using VGG-16 networks in terms of total train time and the number of simplexes (number of ensembles). **Left:** Test error as a function of total training budget on CIFAR-10. The number of components in the ensembles increases as curves move left to right. For any given training budget, ESPRO outperforms deep ensembles. **Center:** Test error as a function of the number of simplexes in the ensemble on CIFAR-10. A comparison of performance of ESPRO models on CIFAR-10 (**left**) and CIFAR-100 (**right**) of VGG-16 networks with various numbers of ensemble components along the x-axis, and various simplex orders indicated by color. For any fixed number of ensemble components we can outperform a standard deep ensemble using simplexes from ESPRO. Notably, expanding the number of vertices in a simplex takes *only* 10 *epochs of training* on CIFAR-10 compared to the 200 epochs of training required to train a model from scratch. On CIFAR-100 adding a vertex to an ESPRO simplex takes just 20 epochs of training compared to 300 to train from scratch.

the training data nearly perfectly but do so in distinct ways, such that we can improve our final predictions by averaging over these models.

4.10.4 Performance of Simplicial Complex Ensembles

Section 4.10.3 shows that we are able to discover simplexes in parameter space containing models that lead to diverse predictions, meaning that we can ensemble within a simplex and gain some of the benefits seen by deep ensembles [Lakshminarayanan et al. 2017]. We use SPRO to train simplicial complexes containing a number of disjoint simplexes, and ensemble over these complexes to form predictions, using Eq. 4.10. We fix the number of samples taken from the ESPRO ensemble, J, to 25 which provides the best trade off of accuracy vs test time compute cost.[7] For example, if we are training a deep ensemble of VGG-16 networks with 3 ensemble components on CIFAR-10, we can form a deep ensemble to achieve an error rate of approximately 6.2%; however,

[7] We show the relationship between samples from the simplex and test error in Appendix A.6.4.2.

by extending each base model to just a simple 2-simplex (3 vertices) we can achieve an error rate of approximately 5.7% — an improvement of nearly 10%!

After finding a mode through standard training, a low order simplex can be found in just a small fraction of the time it takes to train a model from scratch. For a fixed training budget, we find that we can achieve a much lower error rate through training fewer overall ensemble components, but training low order simplexes (order 0 to 2) at each mode using ESPRO. Figure 4.13 shows a comparison of test error rate for ensembles of VGG-16 models over different numbers of ensemble components and simplex sizes on CIFAR-10 and CIFAR-100. For any fixed ensemble size, we can gain performance by using a ESPRO ensemble rather than a standard deep ensemble. Furthermore, training these ESPRO models is generally inexpensive; the models in Figure 4.13 are trained on CIFAR-10 for 200 epochs and CIFAR-100 for 300 epochs. Adding a vertex takes only an additional 10 epochs of training on CIFAR-10, and 20 epochs of training on CIFAR-100. We show the CIFAR-100 time-accuracy tradeoff in Appendix A.6.4 finding a similar trend to CIFAR-10.

Figure 4.14 shows a comparison of test error for ensembles of ResNet-56 models over different ensemble and simplex sizes in Figure 4.13, providing more evidence for the general applicability of the ESPRO procedure. The main practical difference between ResNet-56's and the previous VGG networks is that the ResNet-56's use BatchNorm. BatchNorm statistics need to be adjusted when we sample a model from within a simplex, leading to an additional cost at test time. To generate predictions, we use 100 minibatches of train data to update the batch norm statistics before freezing the statistics and predicting on the test set.

4.11 Uncertainty and Robustness

We finish by investigating the uncertainty representation and robustness to dataset shift provided by ESPRO. We show qualitative results on a regression problem, before studying corruptions of CIFAR-10, comparing to deep ensembles, MultiSWA, and the state-of-the-art Bayesian approach

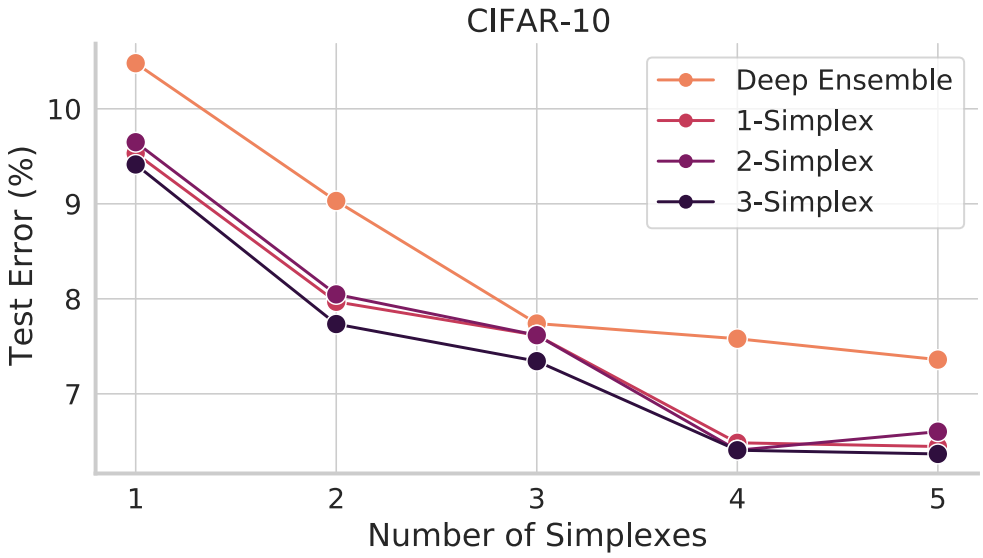


Figure 4.14: Performance of deep ensembles and ESPRO (1, 2, or 3-simplex) using ResNet-56 models on CIFAR-10. The ResNet-56s follow the same trend as VGG networks: more ensemble components increases accuracy, ESPRO significantly outperforms deep ensembles, and adding further simplex vertices to each ESPRO component provides additional improvements.

MultiSWAG [Wilson and Izmailov 2020].

4.11.1 Qualitative Regression Experiments

In general, a good representation of epistemic (model) uncertainty has the property that the uncertainty grows as we move away from the data. Visualizing the growth in uncertainty is most straightforward in simple one-dimensional regression problems.

Izmailov et al. [2019b] visualize one dimensional regression uncertainty by randomly initializing a two layer neural network, evaluating the neural network on three disjoint random inputs in one dimension: $(-7, -5)$, $(-1, 1)$, and $(5, 7)$, and adding noise of $\sigma^2 = 0.1$ to the net's outputs. The task is to recover the true noiseless function, f, given another randomly initialized two layer network, as well as to achieve reasonable confidence bands in the regions of missing data — we used a Gaussian likelihood with fixed $\sigma^2 = 0.1$ to train the networks, modelling the noisy data y. In Figure 4.15, we show ESPRO (top left) which recovers good qualitative uncertainty bands

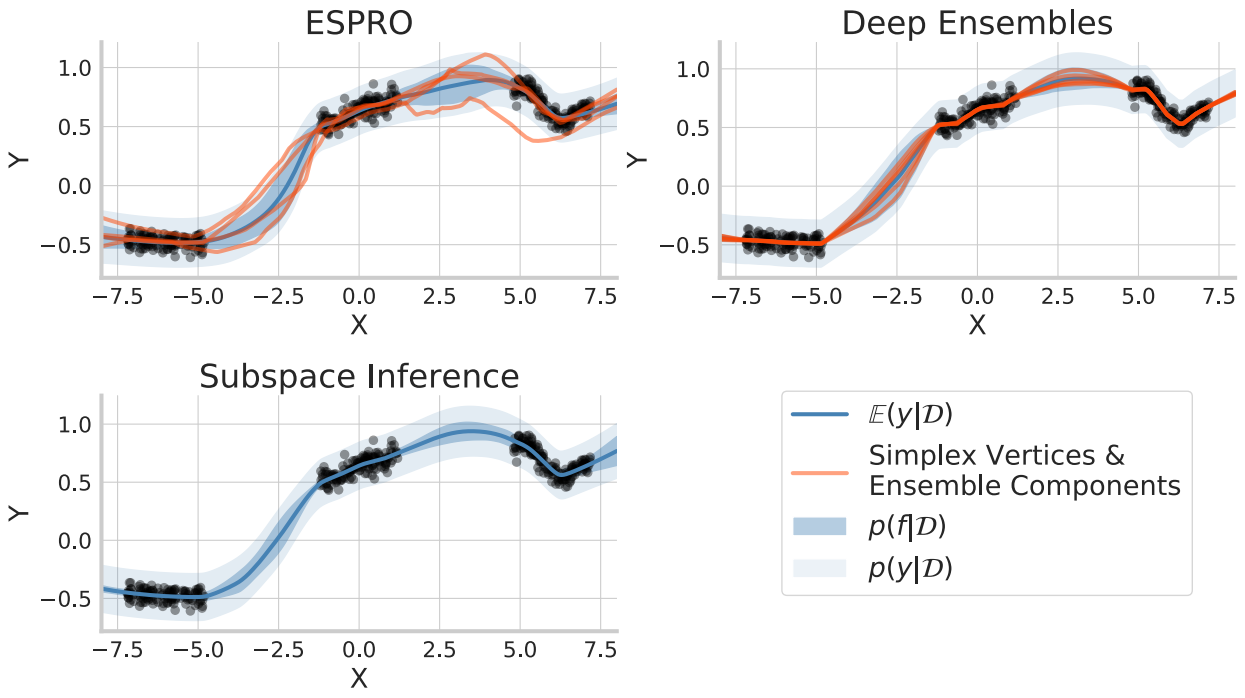
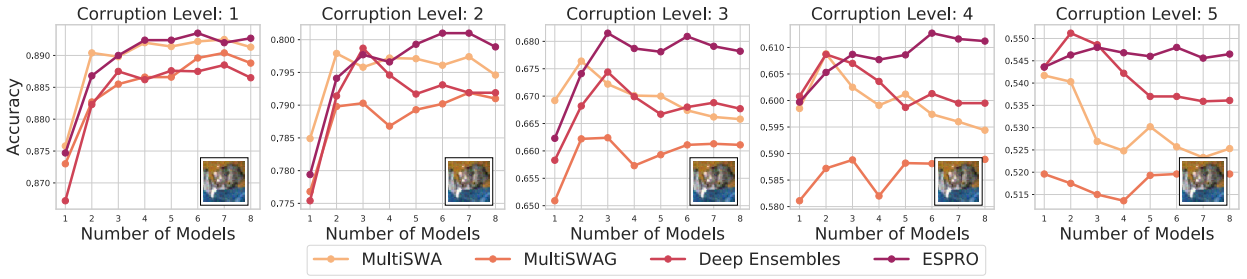
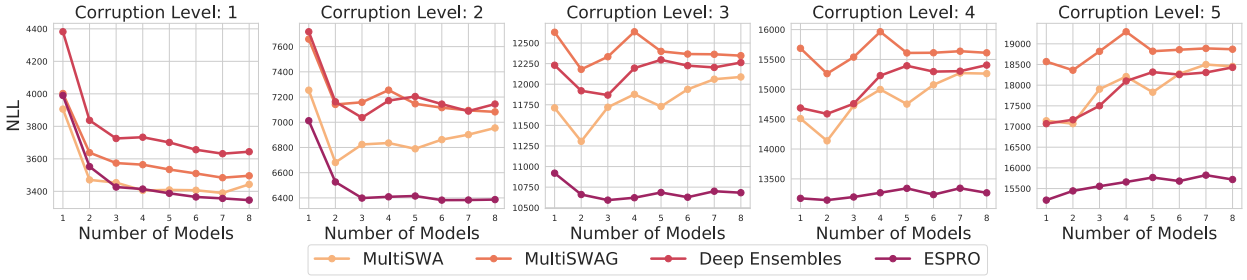


Figure 4.15: Qualitative uncertainty plots of $p(f|\mathcal{D})$ on a regression problem. We show both the 2σ confidence regions from $p(f|\mathcal{D})$ (the latent noise-free function) and $p(y|\mathcal{D})$, which includes the observed noise of the data (aleatoric uncertainty). **Top Left:** ESPRO, colored lines are the vertices in the simplex. First two are fixed points in the simplex. **Top Right:** Deep ensembles, colored lines are individual models. **Bottom Left:** Curve subspaces. ESPRO solutions produce functionally diverse solutions that have good in-between (between the data distribution) and extrapolation (outside of the data distribution) uncertainties; the ESPRO predictive distribution is broader and more realistic than deep ensembles and mode-connecting subspace inference, by containing a greater variety of high performing solutions.

on this task. We compare to deep ensembles (size 5) (top right) and the state of the art subspace inference method of Izmailov et al. [2019b] (bottom left), finding that ESPRO does a better job of recovering uncertainty about the latent function f than either competing method, as shown by the 2σ confidence region about $p(f|\mathcal{D})$. Indeed, after adding in the true noise, ESPRO complexes also do a better job of modelling the noisy responses, y, measured by $p(y|\mathcal{D})$ than either approach.



(a) Accuracy for Gaussian noise corruption



(b) NLL for Gaussian noise corruption

Figure 4.16: (a) Accuracy for Gaussian blur corruption for MultiSWA, MultiSWAG, deep ensembles and ESPRO. **(b)** NLL under the same corruption. All models were originally significantly over-confident so we use temperature scaling [Guo et al. 2017] to improve uncertainty; after temperature scaling ESPRO generally performs the best under varying levels of corruption.

4.11.2 Uncertainty and Accuracy under Dataset Shift

Modern neural networks are well known to be poorly calibrated and to result in overconfident predictions. Following Ovadia et al. [2019], we consider classification accuracy, the negative log likelihood (NLL), and expected calibration error (ECE), to asses model performance under varying amounts of dataset shift, comparing to deep ensembles [Lakshminarayanan et al. 2017], MultiSWA, and MultiSWAG [Wilson and Izmailov 2020], a state-of-the-art approach to Bayesian deep learning which generalizes deep ensembles. In Figure 4.16(a), we show results across all levels for the Gaussian noise corruption, where we see that ESPRO is most accurate across all levels. For NLL we use temperature scaling [Guo et al. 2017] on all methods to reduce the over-confidence and report the results in Figure 4.16(b). We see that ESPRO with temperature scaling outperforms all other methods for all corruption levels. We show ECE and results across other types of dataset corruption in Appendix A.6.5.1.

4.12 Discussion

Through this chapter we have explored and utilized a better understanding of the connection between the statistical properties of parameter space and function space in neural networks. Taking inspiration from theoretically driven results in simpler cases, such as linear models, we have empirically explored the ways in which training solutions in neural networks contract in the presence of data, as well as the implications of this contraction on the functional forms of the model.

Extending the results regarding posterior contraction in parameter space, we have also shown that the loss landscapes for deep neural networks contain large multi-dimensional simplexes of low loss solutions. We proposed a simple approach, which we term SPRO, to discover these simplexes. We show how this geometric discovery can be leveraged to develop a highly practical

approach to ensembling, which samples diverse and low loss solutions from the simplexes. Our approach improves upon state-of-the-art methods including deep ensembles and MultiSWAG, in accuracy and robustness. Overall, these results provide a new understanding of how the loss landscapes in deep learning are structured: rather than isolated modes, or basins of attraction connected by thin tunnels, there are large multidimensional manifolds of connected solutions.

This new understanding of neural network loss landscapes has many exciting practical implications and future directions. We have shown we can build state-of-the-art ensembling approaches from low less simplexes, which serve as a simple drop-in replacement for deep ensembles. Moving forward, we hope our work will help inspire a continued effort to capture the nuanced interplay between the statistical properties of parameter space and function space in understanding generalization behavior.

5 | Conclusion

We have explored a range of methods for using and constructing function space representations in machine learning models. In Chapter 2 we use the function space distribution of Gaussian processes to imply distributions over the kernel functions within GPs themselves. We introduced Functional Kernel Learning (FKL) and Volt, two methods for forming distributions over covariance functions in Gaussian process models. FKL is focused on general applications, and provides support for all stationary covariance functions, making it an appealing choice for many standard time series problems. Volt, on the other hand, is much more specialized and is designed around cases where we can safely assume certain types of stochastic volatility models will accurately describe our data. By relieving us from the need to learn and rely on a single kernel function, both FKL and Volt provide powerful tools in Gaussian process models that improve forecast accuracy and uncertainty.

In Chapter 3 we move from focusing on the functional properties in GPs to considering functional constraints in neural networks. We introduced Residual Pathway Priors (RPP) and Augerino for building equivariant and invariant functions in an imperfect world. In an ideal setting we would be able to prescribe models that are perfectly equivariant to full ranges of transformations, but in reality we are often forced to work with only approximate equivariance or only a limited range of transformations. The ability of these models to handle cases where we may not know what equivariances are present a priori gives us a framework for deploying equivariant models in real world setting where perfect assumptions will not always hold. RPP and Augerino

both provide principled ways of building equivariance-inspired models where we can learn the appropriate transformations directly from the data.

Finally, in Chapter 4, we connect the parameter space view point to a functional perspective in neural networks. By exploring neural network loss surfaces and the types of solutions found through training, as well as the posterior contraction of these solutions we are able to draw stronger conclusions about how the functional properties of our models change through training. Finally, using these insights about loss surfaces we introduced SPRO, a method for quickly and efficiently finding diverse ensembles of models. While the preceding chapters focused directly on functional properties either through the function space distribution provided by Gaussian processes or via equivariance constraints, SPRO focuses on using parameter space understanding to help build more accurate functions. By forming low loss simplexes of models in parameter space, SPRO is able to sample collections of accurate models, and by constructing simplexes that are as large as possible in parameter space we are able to find diverse ensembles of models in function space.

Collectively these works provide a set of tools for engaging with functional properties of models, be they function space distributions or specific qualities like equivariance. While in many cases it is mathemetically easier to engage with the parameters our models take on, such as assigning priors in neural networks or training hyperparameters in Gaussian processes, ultimately we should only be concerned with these parameters insofar as they affect the functions our models produce. In this thesis it is through a functional lens that we are not only able to better understand the properties of our models, but also to take a prescriptive approach and describe methods for building better models.

We have shown empirically and theoretically that such functional constructions can lead to a range of desirable outcomes. These outcomes include models with some predetermined traits, such as correspondence to a prior known volatility model or soft equivariance constraints. They can also include models that are more robust to the presence of data corruption, or form more

accurate ensembles as is the case with SPRO. We hope that these methods and discoveries inspire future work, especially as researchers continue to improve our understanding of the connection between neural network parameters and the functions they produce.

A | Appendix

A.1 Appendix For Function-Space Distributions over Kernels

A.1.1 Computational Complexity

Note that when sampling at N data points and I frequencies, the storage costs for this model are naively $O(N^2+I^2)$ with the computational cost for prediction of $O(N^3+I^3)$. Using pre-conditioned conjugate gradients for inverses and stochastic Lanczos quadrature (SLQ) or the log determinants [Dong et al. 2017] as implemented in GPyTorch [Gardner et al. 2018a] for the data and likelihood calls can immediately reduce the computational cost to $O(N^2 + I^2)$. However, the randomness in the log determinant calculations proved to be problematic for ESS and we only used SLQ for the gradient-based updates, keeping the overall time complexity cubic in N. Given that the latent Gaussian processes are on a pre-defined grid, we can utilize fast Toeplitz matrix multiplications [Wilson et al. 2014] to reduce the time complexity to $O(N^3 + I \log I)$ and the memory complexity to $O(N^3 + I)$.

Extending the model to multi-dimensional inputs and multiple outputs adds on a linear term for both dimensionality D and tasks T independently, so for a multi-task model with T tasks predictions are done in $O(T(N^3 + I))$. Note that this is significant improvement over the $O(T^3N^3)$

Algorithm 2: Alternating Sampler

Input: Data (x, y), Initial hyper-parameters ϕ_0, Sampling frequencies ω, Initial Latent GP $g(\omega)$, Number of gradient steps to take per iteration N_{optim}, Number of ESS samples per update per iteration N_{ESS},
repeat
  for $i = 1$ **to** N_{optim} **do**
    Update ϕ using gradient descent given $g(\omega)$ and Eqn. 2.7
  end for
  for $i = 1$ **to** N_{ESS} **do**
    Update $g(\omega)$ using elliptical slice sampling given ϕ and Eqn. 2.9
  end for
until convergence

needed to do exact inference in previous multi-task work such as Bonilla et al. [2008].

For enhanced scalability, we can approximate the kernel matrices in single (and low) dimensions by utilizing scalable kernel interpolation (SKI) as introduced by Wilson and Nickisch [2015]. Using m inducing points we can achieve an inference cost of $O(N + m \log m + I \log I)$ or $O(T(N + m \log m + I \log I))$ for the multi-task setting.

A.1.2 Latent Model Specification

A.1.2.1 Initialization

FKL proves to be robust to initialization, thus for simplicity we initialize the spectral density to be constant, $S(\omega) = 1$, for a large range of frequencies. An experiment detailing the models robustness is given in the Appendix.

A.1.2.2 Specification of the Latent GP

We fix the mean and covariance of the latent process $g(\omega)$ to take the following forms:

$$\begin{aligned} &\{\text{log of RBF spectral density}\} && \mu(\omega;\theta) = \theta_0 - \frac{\omega^2}{2\tilde{\theta}_1^{\,2}} \\ &\{\text{Matérn kernel}\} && k_g(\omega,\omega';\theta) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\sqrt{2\nu}\frac{|\omega-\omega'|}{\tilde{\theta}_2}\right)K_\nu\left(\sqrt{2\nu}\frac{|\omega-\omega'|}{\tilde{\theta}_2}\right) + \tilde{\theta}_3\delta_{\tau=0} \end{aligned} \tag{A.1}$$

The $\tilde{\theta}_i$'s are non-negative variables, so are computed with $\tilde{\theta}_i = \log(e^{\theta_i}+1)$, the softplus of the raw value. The mean parametrization coupled with the constraints fixes the latent mean to be negative quadratic, like the logarithm of an RBF spectral density.

A.1.2.3 Prior Specification

For the noise terms, we place smoothed box priors[1] on the range (1e-8, 1e-3) to control both numerical instability and the noise terms. For the constant mean terms in both the data and latent means, we place uninformative $\mathcal{N}(0, 100)$ priors. For the length-scale in the spectral density mean along with the length-scale and output-scale of the covariance of the spectral density GP, we place standard log-normal priors.

A.1.3 Density and Error Bounds of FKL

A.1.3.1 Error Rate of Trapezoidal Rule Approximation

Given a sample path from a Gaussian process with a Matérn kernel as is used in our implementation, we can get explicit $O(1/I)$ error bounds on the error of trapezoid rule integration of the warped GP instead by checking Holder continuity of sample draws from the latent GP [Belyaev

[1] A smooth approximation to uniform priors, where $B(x) = \{a \le x \le b\}$ then $d(x,B) := \min_{x'\in B}|x-x'|$ and finally the density is given by $f(x) := \exp\{-d(x,B)^2/\sqrt{2\sigma^2}\}$. See https://gpytorch.readthedocs.io/en/latest/priors.html for further implementation details.

Algorithm 3: Multi-Task Alternating Sampler

Input: Data (x, Y), Initial hyper-parameters ϕ_0, Sampling frequencies ω, Initial Latent GPs $g_i(\omega)$ for $i = 1, \ldots, T$, Number of gradient steps to take per iteration N_{optim}, Number of ESS samples per update per iteration N_{ESS},
repeat
  for $i = 1$ **to** N_{optim} **do**
    Update ϕ using gradient descent given $g(\omega)$ and Eqn. 2.8
  end for
  for $t = 1$ **to** T **do**
    for $i = 1$ **to** N_{ESS} **do**
      Update $g_t(\omega)$ using elliptical slice sampling given ϕ and Eqn. 2.9 with respect to $f_t(x)$
    end for
  end for
until convergence

1961], and using results on the error of trapezoid rule for Holder continuous functions [Cruz-Uribe and Neugebauer 2002]. Note that we could use standard error bounds if we use a GP with twice differentiable sample paths.

A.1.3.2 Density Amongst Stationary Kernels

We next note that the trapezoidal rule is just a finite sample version of both Riemann and Darboux integrals. Thus, functional kernel learning can also be written as a linear combination of the trigonemetric basis expansions and the spectral density (e.g. in sparse spectrum form like Lázaro-Gredilla et al. [2010]). Thus, FKL can model discontinuous but finite measures because mixtures of Gaussians are dense approximations of Riemann integrable densities (see Theorem 5 of Shen et al. [2019]). Thus, the trapezoid rule will be an approximator of the true kernel on the compact set $[0, \omega_{max}]$, converging as $\omega_{max} \to \infty$ (e.g. as the number of basis functions goes to infinity).

Finally, we note that in the multi-dimensional case, FKL does not provide support over all stationary covariances (like other spectral approaches [Shen et al. 2019; Wilson and Adams 2013]), but we find in practice that the domain of support is great enough for accurate performance on most tasks. We would need to at least model the ω's for each dimension on a grid to provide full

support, at a cost of af the number of grid points exponentially increasing. Future work will help to alleviate this issue.

A.1.4 Sensitivity to Initialization

Part of the strength of FKL, particularly over competing methods like spectral mixture (SM) kernels, is robustness to initialization. We compare the performance of FKL and SM kernels on interpolating data generated from a GP with a quasi-periodic kernel.

In GPyTorch spectral mixture (SM) kernels are initialized to,

$$\mu = \log(\exp(0) + 1)$$
$$\sigma = \log(\exp(0) + 1)$$
$$w = \log(\exp(0) + 1),$$

i.e. the means, variances, and weights of each mixture component is the softplus of 0 prior to calling the data initialization routine [Gardner et al. 2018a]. The data-based initialization routine uses statistics of the data to randomly initialize the parameters of the mixture components, and performance is highly dependent on this initialization.

In the current implementation FKL is initialized with a spectral density that is constant,

$$S_0(\omega) = 1 \quad \forall \omega$$
$$g_0(\omega) = 0 \quad \forall \omega,$$

where $g(\omega)$ is the log-spectral density, which is modeled using a latent GP. The surprising fact, and what makes FKL such an appealing model for complex problems, is robustness to initialization. In practice we see no gains in predictive performance when initializing in a more sophisticated fashion than is currently done. This robustness goes far enough that we don't even see perfor-

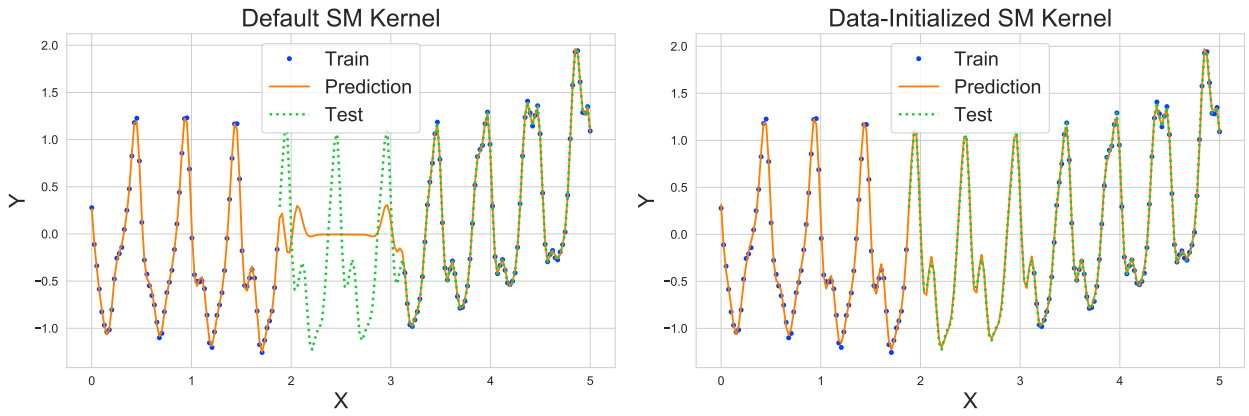


Figure A.1: Comparison of naive and data-based initialized SM kernels on interpolation tasks. **Left**: the default (naive) initialized kernel, **Right**: the data-based initialized kernel.

mance gains when we have access to ground truth data and can initialize the spectral density to be near to the spectral density of the kernel of generative model itself.

Data are generated using a GP with quasi-periodic kernel and the middle portion of the data are held out as a testing set. Using the inverse Fourier transform we can compute the spectral density of the generating quasi-periodic kernel directly, $S^*(\omega)$. First we train and predict using a SM kernel that is has parameters initialized to the constant values from above, and compare to a SM kernel using GPyTorch's built in data-based initialization. Next we repeat the procedure using a default initialized FKL model, then compare to an FKL model where the spectral density has been initialized to a corrupted version of the ground truth spectral density. Thus we compare FKL models with the initializations,

$$S_0(\omega) = 1 \quad \forall \omega$$

$$S_0(\omega) = S^*(\omega) + \mathcal{N}(0, 0.1) \quad \forall \omega.$$

The results are shown in Figures A.1 and A.2. What we see is that a naive implementation of SM kernels leads to poor performance on the testing set, while FKL performs nearly the same whether we initialize the spectral density to an arbitrary value, or to nearly the ground truth.

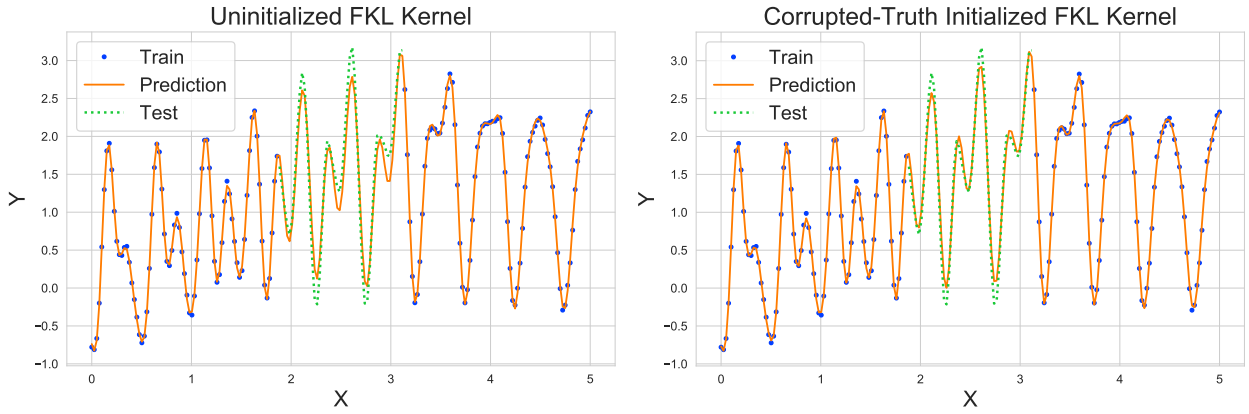


Figure A.2: Comparison of basic and ground-truth initialized FKL kernels on interpolation tasks. **Left**: the default (naive) initialized kernel, **Right**: the ground-truth initialized kernel.

A.1.5 Further Experiments

A.1.5.1 Recovery of Known Kernels

Spectral Mixture Kernel Extending from Section 2.5.1, we also display the accuracy of the kernel reconstruction given the samples drawn in the latent space. Figure A.3 shows the accurately sampled spectral density, and the kernels reconstructed from these samples.

Quasi-Periodic Kernel Synthetic data are generated from a mean zero Gaussian process with kernel,

$$k(\tau;\ell,\omega) = \exp\left(-\frac{\tau^2}{2\ell^2}\right)\exp\left(-2\sin^2(\pi\tau\omega)\right). \tag{A.2}$$

Since there is inherent periodicity in the generative model, the true spectral density has distinct modes corresponding to the period length of the sinusoidal component of the kernel. The spectral density of this kernel is not analytically computed, however using the known kernel the discrete Fourier transform allows an approximation of the ground-truth spectrum to be found, and comparison in the spectral domain can be made.

Using this latent GP model accurate reconstruction of both the spectral density and kernel are

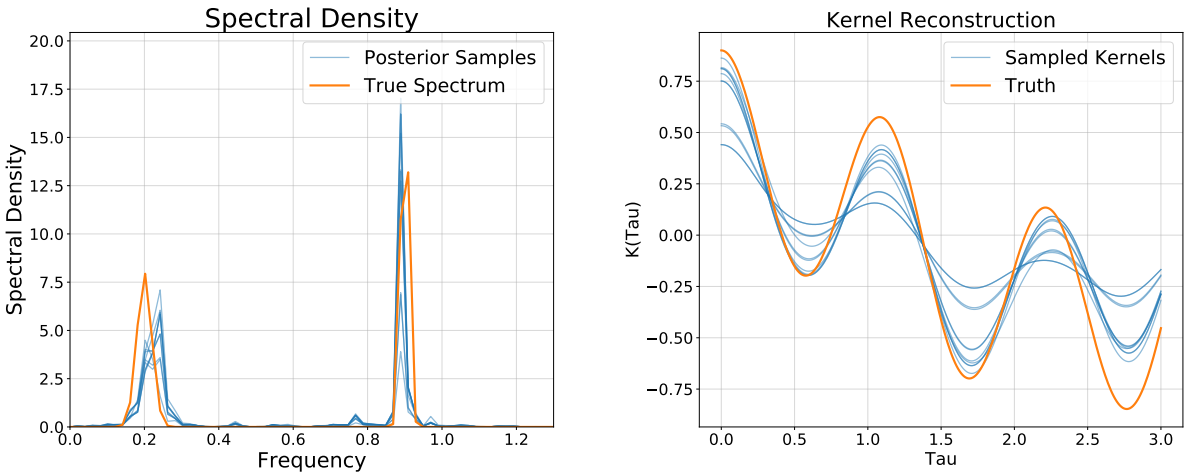


Figure A.3: Samples from the latent GP displayed in the spectral domain along with the ground truth (Left) and the reconstructed kernels generated by these samples (Right).

obtained using only training data. Further more, infilling into the testing set shows high accuracy and the confidence region encompasses the data.

A.1.5.2 Foreign Exchange Rates Dataset

We consider multi-output prediction tasks on a foreign exchange rates dataset originally developed in [Álvarez et al. 2010]. The dataset consists of the exchange rates of 10 currencies and 3 precious metals with respect to the US dollar in 2007. The task is to predict the Canadian dollar (CAD) on days 50-100, Japanese yen (JPY) on days 100-150, and Australian dollar (AUD) on days 150-200, given the exchange rate information for all other days. Due to market differences, there are occasionally also missing data. Like in [Requeima et al. 2019a], we measure performance with the standardized mean square error (SMSE). The results from this experiment are shown in Table A.1 with comparisons taken from both [Requeima et al. 2019a] and [Nguyen et al. 2014]. FKL performs considerably better than both types of collaborative Gaussian process, which constrain the outputs considerably more. By comparison, the GPAR [Requeima et al. 2019a] outperforms FKL on this task, perhaps due to its explicit ordering of tasks and its increased depth (the GPAR

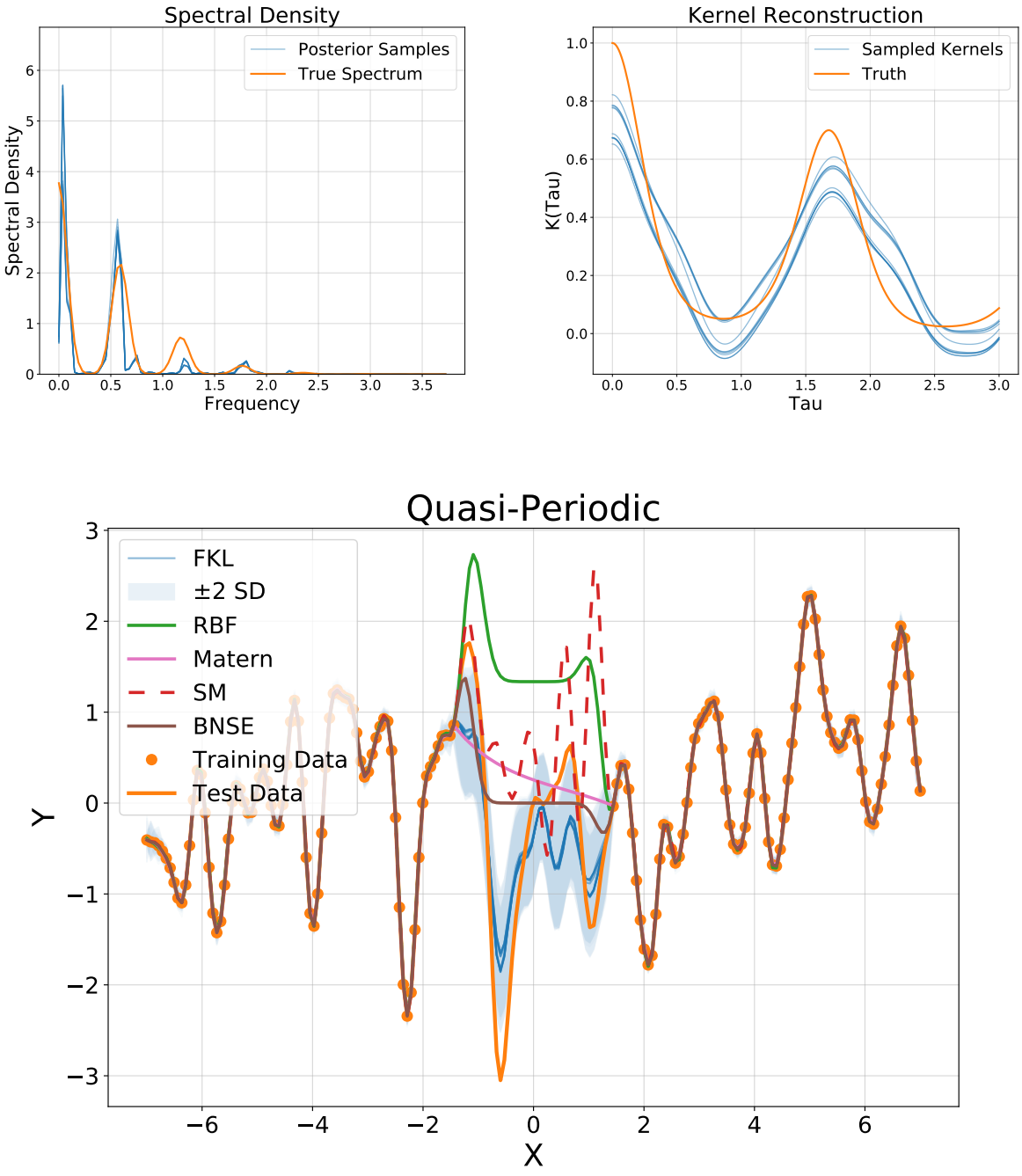


Figure A.4: Spectrum (Above Left) and kernel (Above Right) reconstruction, and resulting data prediction (Below) for data generated by a quasi-periodic kernel.

Table A.1: Standardized mean squared error on FX dataset. Comparisons are with independent Gaussian processes (IGP), convolved multi-output GP (CMOGP) [Álvarez and Lawrence 2011], collaborative GP (CGP) [Nguyen et al. 2014], and Gaussian process autoregressive model (GPAR). Note that the GPAR is perhaps best viewed as a deep Gaussian process with known inputs. Comparisons taken from [Requeima et al. 2019a]. Note that FKL multi-task outperforms the standard multi-task GP methods) averaged over 10 random trials.

Model	IGP	CMOGP	CGP	GPAR	FKL(multi-task)
SMSE	0.5996	0.2427	0.2125	**0.0302**	0.1392 ± 0.01

Table A.2: UCI Regression RMSEs, comparisons are with RBF, ARD, and ARD Matérn kernels, N points D input dimensions. We compare to separate latent GPs for each input dimension, finding that sharing a single latent GP across dimensions works better than both the standard fixed spectrum approaches and separate latent GPs. Each of the experiments were conducted 10 times with random 90/10 train/test splits and we report the average RMSE ± one standard deviation.

Dataset	N	D	RBF	ARD	ARD Matérn	FKL-PB (separate)	FKL-PB (shared)
challenger	23	4	0.713 ± 0.348	0.659 ± 0.368	0.612 ± 0.268	0.58 ± 0.225	**0.548 ± 0.174**
fertility	100	9	0.159 ± 0.036	0.177 ± 0.035	**0.148 ± 0.038**	0.19 ± 0.047	0.182 ± 0.022
concreteslump	103	7	36.302 ± 7.934	27.377 ± 7.782	26.335 ± 7.482	59.444 ± 12.879	**4.385 ± 1.332**
servo	167	4	0.305 ± 0.056	**0.23 ± 0.075**	0.256 ± 0.06	0.282 ± 0.086	0.288 ± 0.063
yacht	308	6	0.17 ± 0.07	0.187 ± 0.078	0.269 ± 0.048	0.193 ± 0.13	**0.11 ± 0.054**
autompg	392	7	2.651 ± 0.488	3.077 ± 0.544	**2.516 ± 0.332**	2.838 ± 0.374	2.69 ± 0.492
housing	506	13	3.771 ± 0.675	3.222 ± 0.846	3.261 ± 0.624	4.679 ± 0.632	**2.703 ± 0.227**
stock	536	11	**0.005 ± 0.001**	**0.005 ± 0.001**	**0.005 ± 0.001**	0.018 ± 0.002	0.016 ± 0.001
pendulum	630	9	1.297 ± 0.315	1.185 ± 0.326	**1.013 ± 0.207**	2.747 ± 0.737	1.562 ± 0.554
energy	768	8	1.839 ± 0.253	0.457 ± 0.035	0.373 ± 0.062	**0.296 ± 0.066**	0.334 ± 0.063
concrete	1030	8	7.001 ± 0.513	6.125 ± 0.456	6.058 ± 0.373	**3.781 ± 0.501**	4.047 ± 0.693
airfoil	1503	5	2.503 ± 0.202	1.696 ± 0.243	1.595 ± 0.296	**1.378 ± 0.176**	1.39 ± 0.181

is a special case of deep Gaussian processes [Damianou and Lawrence 2013]).

Here, we utilize 5 rounds of the alternating sampler with 10 optimization and 50 ESS iterations and run on a single GPU (with 10 repetitions taking about 3 minutes).

A.1.5.3 UCI Tables

Tables A.2, A.3, and A.4 show the RMSE, standardized log loss, and negative log likelihoods of FKL (both separate and shared latent models) compared to standard parametric models on UCI regression tasks.

Table A.3: UCI Regression Mean Standardized Log loss, comparisons are with RBF, ARD, and ARD Matérn kernels, N points D input dimensions. We compare to separate latent GPs for each input dimension. Each of the experiments were conducted 10 times with a random 90/10 train/test split and reported over ± a standard deviation.

Dataset	N	D	RBF	ARD	ARD Matérn	FKL-PB (separate)	FKL-PB (shared)
challenger	23	4	0.83 ± 1.085	0.91 ± 1.951	0.383 ± 0.778	**-0.053 ± 0.192**	0.216 ± 0.292
fertility	100	9	-0.049 ± 0.075	**-0.094 ± 0.137**	-0.077 ± 0.295	0.013 ± 0.06	-0.0 ± 0.017
concreteslump	103	7	30.821 ± 12.039	20.055 ± 11.079	17.247 ± 9.789	-0.125 ± 0.131	**-2.57 ± 0.23**
servo	167	4	-1.076 ± 0.216	-1.242 ± 0.386	-1.25 ± 0.121	**-1.28 ± 0.218**	-0.981 ± 0.272
yacht	308	6	5.136 ± 8.696	-2.001 ± 2.369	4.943 ± 7.521	**-2.62 ± 0.225**	-2.477 ± 0.17
autompg	392	7	-1.065 ± 0.216	-0.93 ± 0.306	**-1.085 ± 0.152**	-1.034 ± 0.149	-0.888 ± 0.482
boston	506	13	-0.912 ± 0.196	-1.077 ± 0.213	-1.031 ± 0.13	-0.86 ± 0.085	**-1.191 ± 0.109**
stock	536	11	-0.831 ± 0.082	-0.82 ± 0.088	**-0.868 ± 0.105**	0.014 ± 0.04	-0.001 ± 0.017
pendulum	630	9	-1.12 ± 0.084	-1.358 ± 0.147	-1.586 ± 0.227	-0.323 ± 0.181	**-1.685 ± 0.263**
energy	768	8	-1.684 ± 0.127	-3.062 ± 0.093	-3.11 ± 0.05	**-3.49 ± 0.133**	-3.302 ± 0.081
concrete	1030	8	-0.417 ± 0.232	-0.717 ± 0.171	**-0.745 ± 0.154**	-0.489 ± 1.37	-0.311 ± 1.345
airfoil	1503	5	-0.994 ± 0.064	-1.177 ± 0.078	-1.31 ± 0.048	-1.448 ± 0.336	**-1.586 ± 0.198**

Table A.4: UCI Regression Negative Log-likelihoods, comparisons are with RBF, ARD, and ARD Matérn kernels, N points D input dimensions. We compare to separate latent GPs for each input dimension. Each of the experiments were conducted 10 times with a random 90/10 train/test split and reported over ± a standard deviation.

Dataset	N	D	RBF	ARD	ARD Matérn	FKL-PB (separate)	FKL-PB (shared)
challenger	23	4	5.74 ± 4.547	6.064 ± 7.283	3.753 ± 3.05	**2.82 ± 0.809**	2.966 ± 0.854
fertility	100	9	-3.901 ± 1.76	-2.861 ± 2.187	**-4.408 ± 2.582**	-1.83 ± 3.336	-2.738 ± 1.252
concreteslump	103	7	400.451 ± 134.157	282.544 ± 124.796	250.299 ± 108.762	60.248 ± 2.542	**33.016 ± 1.965**
servo	167	4	5.144 ± 3.995	1.101 ± 5.871	1.374 ± 3.14	**0.93 ± 3.867**	4.686 ± 5.271
yacht	308	6	221.42 ± 271.437	1.65 ± 76.479	**-19.949 ± 14.092**	-15.703 ± 8.233	-14.52 ± 4.7
autompg	392	7	96.189 ± 8.025	104.563 ± 13.36	**94.012 ± 5.033**	98.942 ± 6.135	101.757 ± 19.333
housing	506	13	139.617 ± 11.546	131.22 ± 15.034	130.841 ± 10.506	143.75 ± 5.714	**122.618 ± 3.91**
stock	536	11	**-191.624 ± 1.626**	-191.515 ± 1.472	-191.154 ± 1.318	-140.055 ± 6.679	-147.805 ± 2.577
pendulum	630	9	84.964 ± 3.402	69.371 ± 7.299	62.64 ± 5.692	141.121 ± 20.914	**53.86 ± 16.301**
energy	768	8	157.1 ± 8.894	52.118 ± 5.835	47.776 ± 3.591	**17.808 ± 9.927**	30.222 ± 6.881
concrete	1030	8	395.596 ± 21.02	361.792 ± 20.077	**357.248 ± 14.532**	384.242 ± 140.779	405.779 ± 137.561
airfoil	1503	5	358.932 ± 8.932	325.059 ± 6.605	305.588 ± 7.462	284.895 ± 48.796	**270.073 ± 28.424**

A.1.6 Large-Scale Precipitation Extrapolation

We demonstrate the scalability and practicality of FKL by extending this to a much larger dataset; modeling 108 different stations in seven American states across the northeast (ME, MA, VT, NH, RI, CT, NY) with a single latent Gaussian process, training on the first 300 days of the year, and attempting to extrapolate on the final 65 days. Despite not including any geographic information (e.g. longitude and latitude), FKL fits the trends across this climatologically diverse region We show extrapolation on 120 stations in Figure 15 in the Appendix. Note that this corresponds to a dataset size of greater than 30,000 data points, and that we were able to fit this dataset on a single Nvidia 1080 Ti GPU in roughly 30 minutes.

A.2 Appendix For Volatility Based Kernels and Moving Average Means for Accurate Forecasting with Gaussian Processes

A.2.1 Tutorial

This section should serve as a useful reference on much of the more domain-specific language and methodology used throughout the paper.

In the context of a time series S_t, we use *volatility*, denoted V_t, to refer to the standard deviation of the variability in price over some time period. In financial applications we consider stock prices on the daily time scale, and as is standard report volatility as *annualized volatility*, which corresponds to the volatility of a stock over the course of a year.

More specifically, we assume that the *log returns* in observations, $\log\left(\frac{S_{t+1}}{S_t}\right)$, are normally distributed with standard deviation V_t. In this paper we make the common assumption that the volatility itself is a time varying stochastic process, meaning we expect the magnitude of the daily

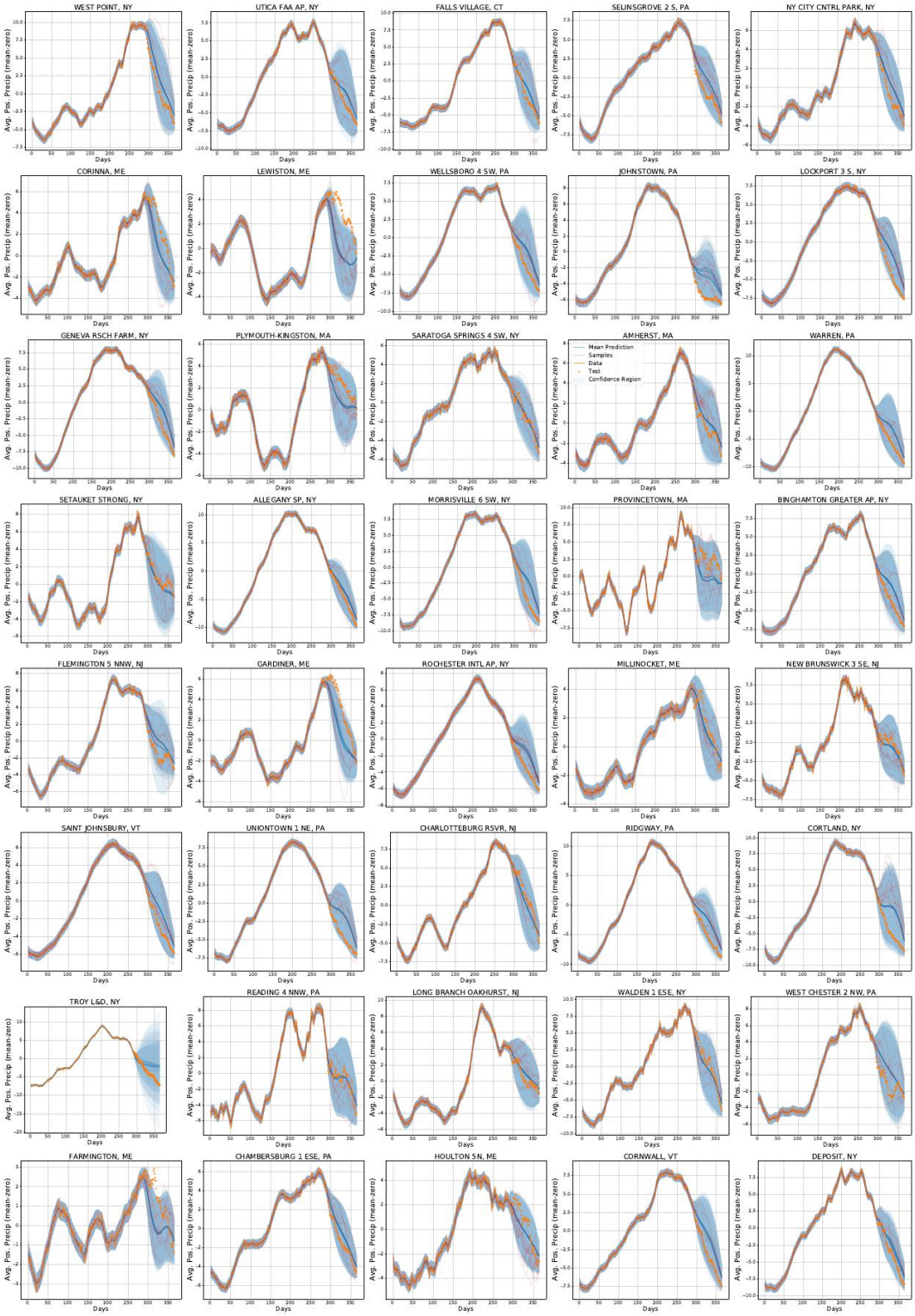


Figure A.5: 40 stations modelled in the multi-task extrapolation test. The multi-task FKL both interpolates and extrapolates well even for relatively geographically diverse datasets.

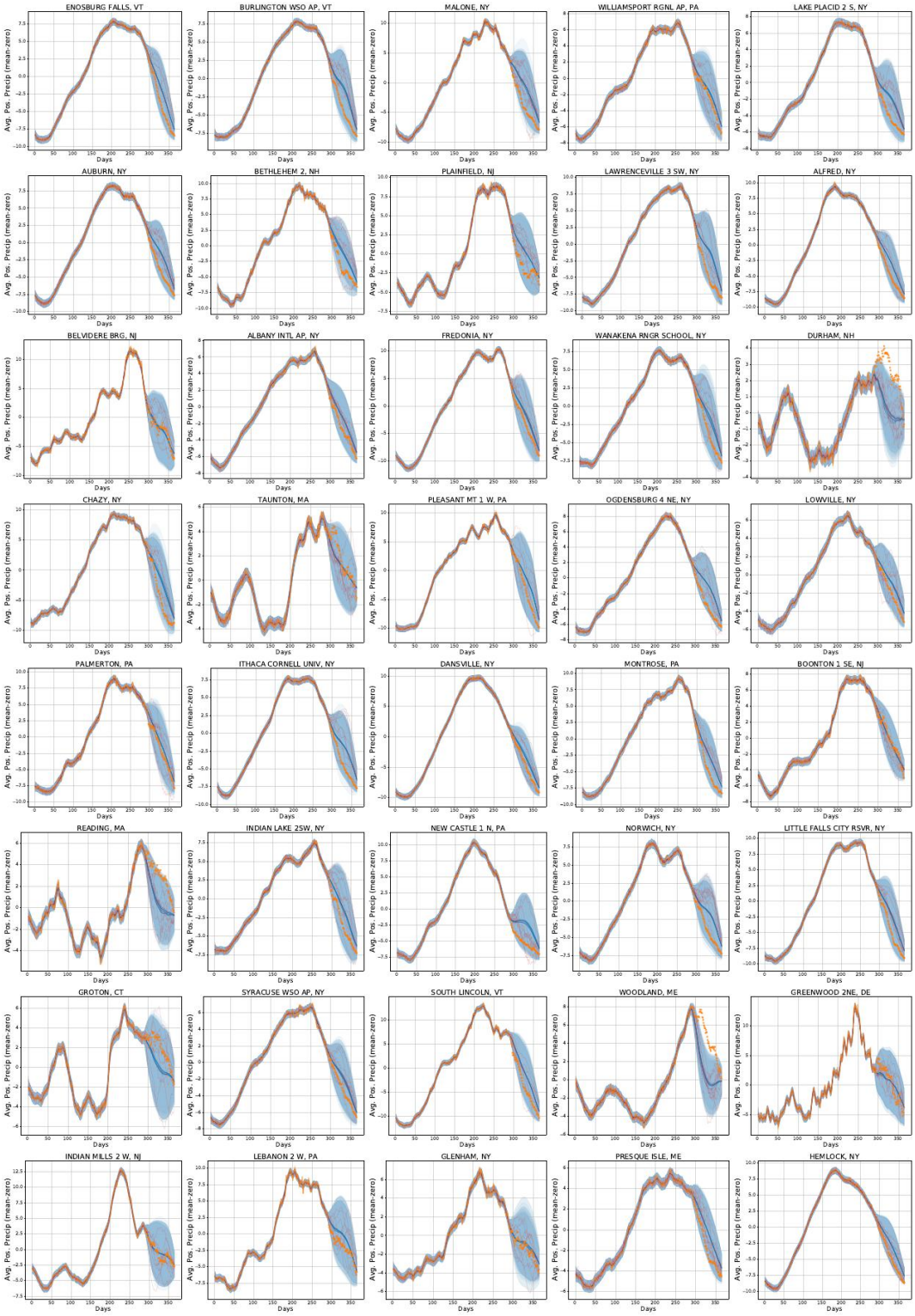


Figure A.6: 40 stations modelled in the multi-task extrapolation test. The multi-task FKL both interpolates and extrapolates well even for relatively geographically diverse datasets.

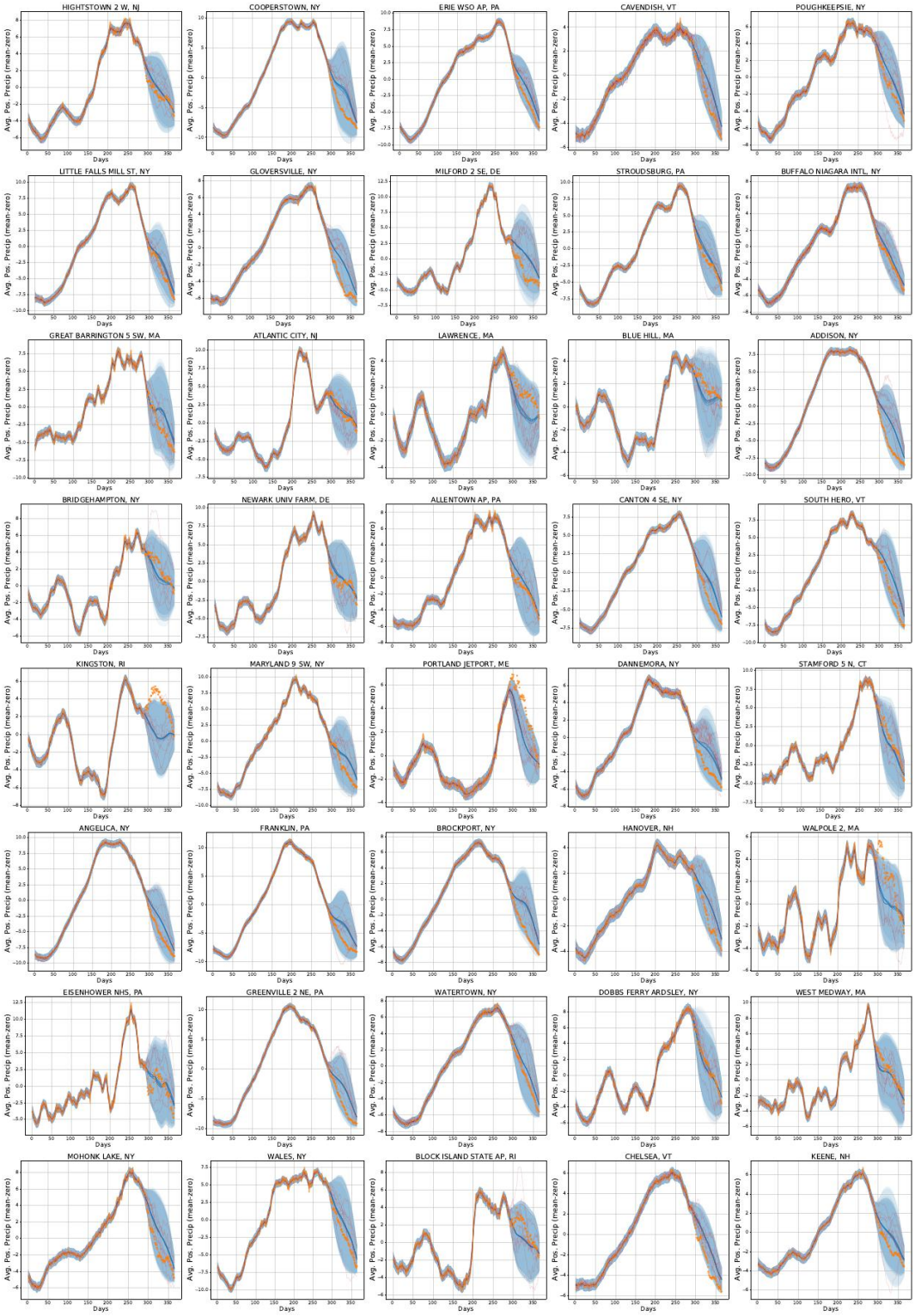


Figure A.7: 40 stations modeled in the multi-task extrapolation test. The multi-task FKL both interpolates and extrapolates well even for relatively geographically diverse datasets.

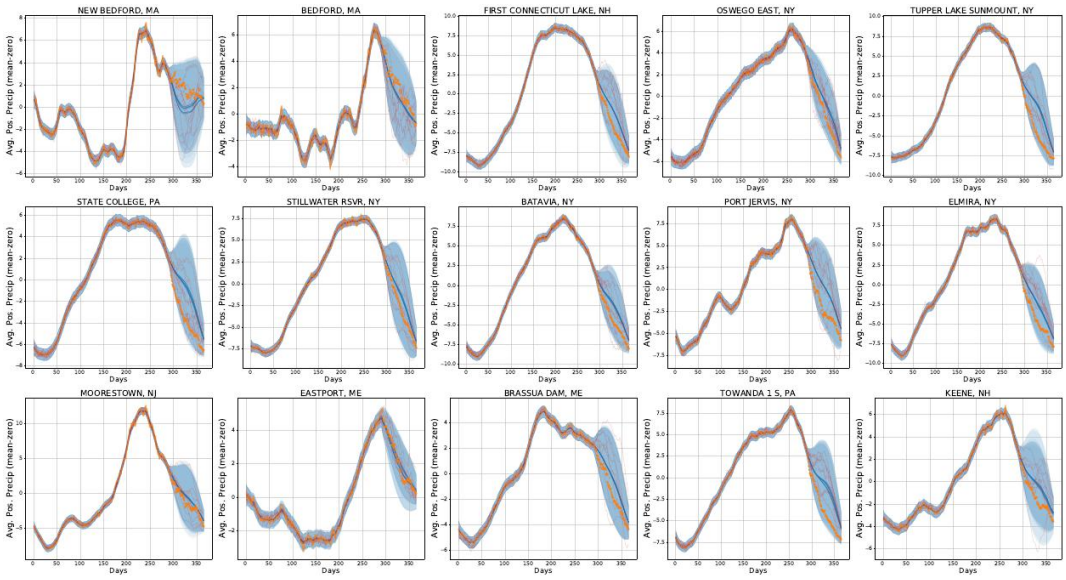


Figure A.8: 15 stations modeled in the multi-task extrapolation test. The multi-task FKL both interpolates and extrapolates well even for relatively geographically diverse datasets.

returns to vary over time.

Figure A.10 provides an example of the connection between price, log returns, and volatility. On the left we have a simulated set of price observations over one year, and in the center we have the associated log returns. Finally, on the right, we have the volatility path overlaid on the returns. We can see that where volatility is high we have larger returns (both positive and negative), and where volatility is low the returns tend to be small. Naturally if we wish to understand how the price will evolve in the future we need to also understand how volatility will evolve.

A.2.2 Extended Methods

A.2.2.1 Moving Average Gaussian Processes

Figure A.11 gives an example of how various moving averages (or Magpie prior means) appear given a series of price observations for a stock. On the left, the standard EMA formulation displays a clear lag effect, that is ameliorated by using either Double or Triple moving averages (DEMA and TEMA). On the right, we see how the DEMA moving average varies for different smoothing

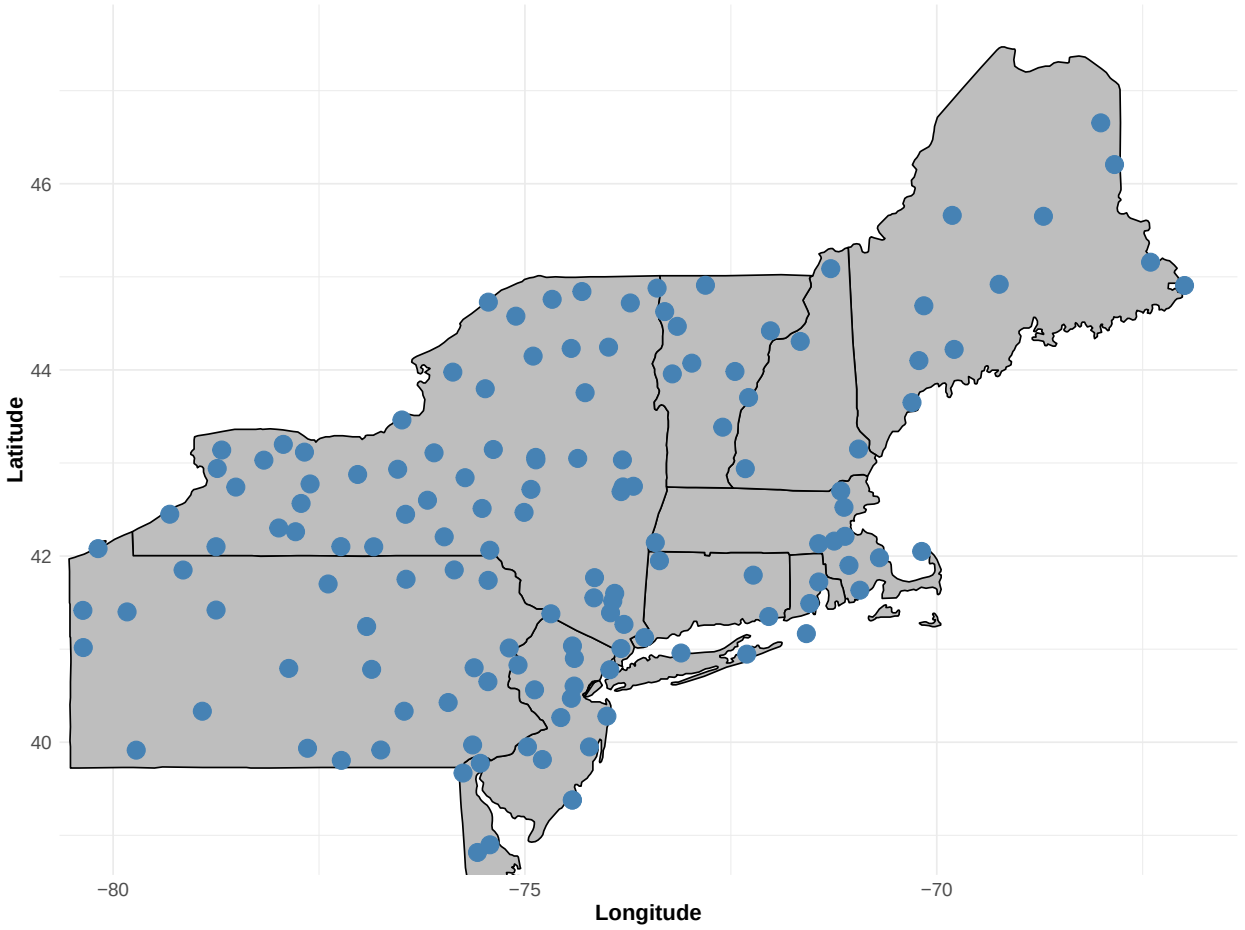


Figure A.9: Map of locations used for large scale multi task experiment.

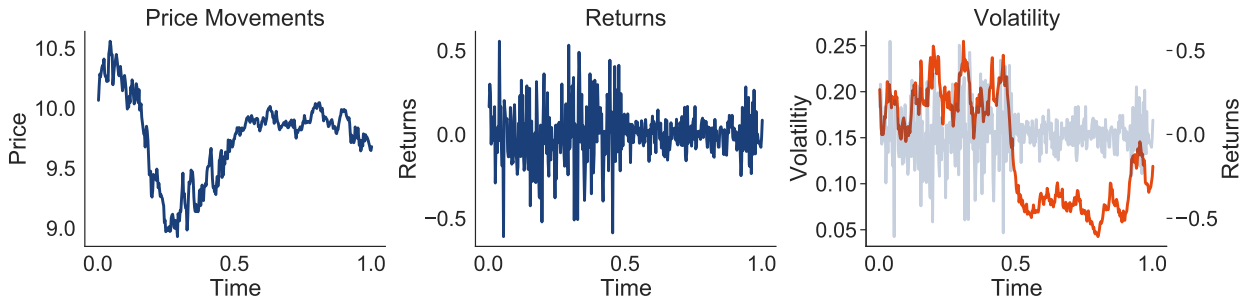


Figure A.10: Left: Price movements over time. Movements tend to be larger up to time $t = 0.5$ **Center:** Returns over time, as calculated by S_{t+1}/S_t. Here, returns are clearly larger in the first half of the time series. **Right:** Volatility overlaid with returns for the same price. Volatility is clearly higher when the returns have a larger absolute magnitude, whether positive or negative.

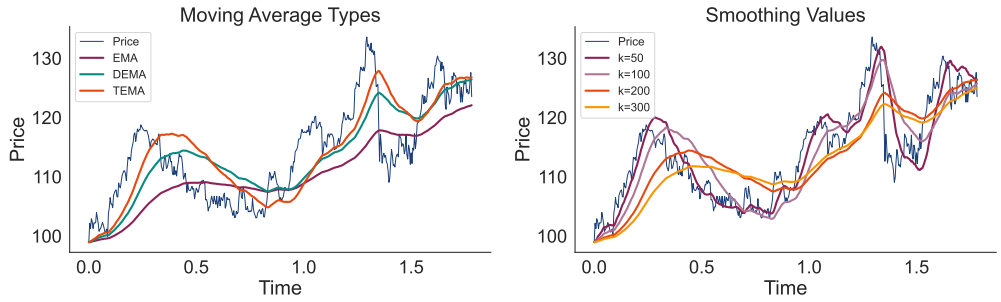


Figure A.11: Left: a comparison of EMA, DEMA, and TEMA methods for producing moving averages for $k = 200$. Note that for a fixed value of k the DEMA and TEMA curves resolve a portion of the lag issue seen in the EMA curve. **Right:** DEMA curves for various values of k. Increasing k averages over more historical data.

parameters k; for larger values the moving average is less sensitive to fluctuations in the data, but exhibits more bias, similarly smaller values of k produce moving averages that more closely match the data, but are susceptible to outliers.

A.2.2.2 Proofs from Derivations

Log-Volatility Kernel Function Recall the SDE governing movements in the log-volatility:

$$dv(t) = -\frac{\sigma^2}{2}dt + \sigma dZ(t). \tag{A.3}$$

We now derive the covariance function $\text{Cov}(v(t), v(t'))$, assuming without loss of generality, that $t < t'$. For ease of notation, and as the mean does not affect the covariance structure, let $\tilde{v}(t)$ be the same process as $v(t)$ with the mean trend removed.

Using independence of increments of the SDE we can determine the covariance as follows:

$$\begin{aligned}
\text{Cov}(v(t), v(t')) &= \text{Cov}(v(t) - E[v(t)], v(t') - E[v(t')]) \\
&= \text{Cov}(\tilde{v}(t), \tilde{v}(t')) \\
&= E[\tilde{v}(t)\tilde{v}(t')] - E[\tilde{v}(t)]E[\tilde{v}(t')] \\
&= E[\tilde{v}(t)\tilde{v}(t')] \\
&= E[\tilde{v}(t)(\tilde{v}(t') - \tilde{v}(t))] + E[\tilde{v}(t')^2] \\
&= E[\tilde{v}(t')^2] = t'\sigma^2 = \min\{t, t'\}\sigma^2.
\end{aligned}$$

So finally we have $\text{Cov}(v(t), v(t')) = K_v(t, t') = \min\{t, t'\}\sigma^2$.

Log-Price Kernel Function

$$ds(t) = \mu_s dt + V(t)dW(t) \tag{A.4}$$

The covariance function of $s(t)$ can be derived using the fact that the $s(t)$ diffusion has independent increments; first assume that $t < t'$ and that $\tilde{s}(t)$ is the same process as $s(t)$ with the

mean trend removed. Therefore,

$$\begin{aligned}
\text{Cov}(s(t), s(t')) &= \text{Cov}(s(t) - E[s(t)], s(t') - E[s(t')]) \\
&= \text{Cov}(\tilde{s}(t), \tilde{s}(t')) \\
&= E[\tilde{s}(t)\tilde{s}(t')] - E[\tilde{s}(t)]E[\tilde{s}(t')] \\
&= E[\tilde{s}(t)\tilde{s}(t')] \\
&= E[\tilde{s}(t)(\tilde{s}(t') - \tilde{s}(t))] + E[\tilde{s}(t')^2] \\
&= E[\tilde{s}(t')^2],
\end{aligned}$$

now since $E[\tilde{s}(t)] = 0$, $E[\tilde{s}(t)^2] = \text{Var}(\tilde{s}(t)) = \text{Var}(s(t))$ which is just the integral of the variance of the diffusion in Equation (2.13), leaving us with

$$Cov(s(t), s(t')) = \int_0^{\min\{t,t'\}} V(t)^2 dt = K_s(t, t'; V(t)).$$

A.2.2.3 GPCV Training

GPCV Likelihood As described in the main text, we model the log returns, $w(t)$, at time t as independently distributed following the construction of Wilson and Ghahramani [2010]. That is, $w(t) \sim \mathcal{N}(0, \gamma^2(t))$, where $\gamma(t)$ is the latent standard deviation. We choose $\gamma(t) = \exp\{f(t)\}$, which is equivalent to the parameterization used in Lázaro-Gredilla and Titsias [2011]. The exponential parameterization has the nice property that we are also modelling the log prices in the SDE formulation described in the rest of the paper, unlike Wilson and Ghahramani [2010]'s softplus transformation of the latent process. Wilson and Ghahramani [2010] also study the exponential parameterization for a few experiments.

We note that $\gamma(t)$ is a daily volatility and to convert to an annualized volatilty like in the rest of the paper, we need to rescale it by a factor of $1/\sqrt{t}$, so that $\hat{\gamma}(t) = \gamma(t)/\sqrt{t}$.

Inference Scheme Following Hensman et al. [2013, 2015], we want to compute the ELBO as

$$\log p(y) \geq \mathbb{E}_{q(f)}(\log p(y|f)) - \mathrm{KL}(q(u)||p(u)), \tag{A.5}$$

where $p(y|f)$ is the GPCV volatility likelihood and $\mathrm{KL}(q(u)||p(u))$ is the Kullback-Leibler divergence between the the variational distribution $q(u) = \mathcal{N}(m, S)$ and the prior $p(u)$. We need to optimize $q(u)$, our free form variational distribution and estimate $\mathbb{E}_{q(f)}(\log p(y|f))$ using Bayesian quadrature as in Hensman et al. [2015].

As T is generally pretty small, we set the inducing points, u, to be the training data points, e.g. $\{t_i\}_{i=1}^{T}$. We initialize the variational mean m to be the logarithm of the running standard deviation of the log returns, and the variational covariance to be $K_{uu}(K_{uu} + K_{uu}\Sigma_y K_{uu})^{-1}K_{uu}$ where Σ_y is the negative Hessian at the initial value of m.

Computational and memory costs then run at about $O(T^3)$ time. In the future, we hope to use sliding windows for the inducing points, enabling mini-batching, reducing the cost to $O(T^3_{\text{window}})$ time [Hensman et al. 2015]. Finally, our inference scheme is simply a more flexible version of the fixed-form heteroscedastic scheme used in Lázaro-Gredilla and Titsias [2011], which we found to be too inflexible to fit rougher volatility paths well.

Multi-task Parameterization We follow the ICM-like model parameterization of Dai et al. [2017] by parameterizing $q(u) = \mathcal{N}(m, S_x \otimes S_T)$ and assume that $p(u) = \mathcal{N}(\mu(u), K_{uu} \otimes K_{TT})$. Then we need to compute $q(f)$ which can be done for single-task models as $q(f) = \mathcal{N}(K_{fu}K_{uu}^{-1}m, K_{ff} +$

$K_{fu}K_{uu}^{-1}(S - K_{uu})K_{uu}^{-1}K_{uf}$). In the multi-task setting, this is algebraically written as:

$$
\begin{aligned}
q(f) = \mathcal{N}&((K_{fu} \otimes K_{TT})(K_{uu} \otimes K_{TT})^{-1}m, \\
&(K_{ff} \otimes K_{TT}) + (K_{fu} \otimes K_{TT})(K_{uu} \otimes K_{TT})^{-1}(S_x \otimes S_t - (K_{uu} \otimes K_{TT}))(K_{uu} \otimes K_{TT})^{-1}(K_{fu} \otimes K_{TT})^\top) \\
&= \mathcal{N}\left((K_{fu}K_{uu}^{-1} \otimes I)m, (K_{ff} \otimes K_{TT}) + (K_{fu}K_{uu}^{-1} \otimes I)(S_x \otimes S_t - (K_{uu} \otimes K_{TT}))(K_{fu}K_{uu}^{-1} \otimes I)^\top\right) \\
&= \mathcal{N}\left((K_{fu}K_{uu}^{-1} \otimes I)m, (K_{ff} - K_{fu}K_{uu}^{-1}K_{uf} \otimes K_{TT}) + (K_{fu}K_{uu}^{-1}S_xK_{uu}^{-1}K_{uf} \otimes S_t)\right) \qquad \text{(A.6)}
\end{aligned}
$$

Note that the variational mean term is a batch matrix vector multiplication, while the variational covariance form is a sum of two Kronecker products. Together we can sample from the posterior distribution in $\mathcal{O}(T^3 + P^3)$ time by using Kronecker identities as described in Rakitsch et al. [2013].

In the multi-task setting, we also initialize the variational covariance to be the average initial covariance across tasks and the variational intertask covariance to be the covariance of m across tasks. The intertask covariance is a $P \times P$ matrix parameterized as rank one plus diagonal; we regularize it with a LKJ prior with $\eta = 5.0$ [Lewandowski et al. 2009].

Additionally, we exploit Kronecker identities to efficiently compute the KL divergence in the variational distribution so that training stays at $\mathcal{O}(T^3 + P^3)$ time by broadly following the approach of Dai et al. [2017].

A.2.2.4 Model Training

All models were trained in GPyTorch [Gardner et al. 2018b] and PyTorch [Paszke et al. 2019] on either a single 24GB GPU or a single 12GB GPU; the multi-task wind experiment used a 48GB Titan RTX GPU. Training time was negligible, with models typically taking less than 1 minute to train. For training, we use 500 steps of Adam with learning rate 0.1 and optimize through the log marginal likelihood.

Multitask GPs We use the ICM model of Bonilla et al. [2007]. Like in the GPCV setting, we use a rank one plus diagonal intertask covariance, regularized with a LKJ prior [Lewandowski et al. 2009]. By structure exploitation, these models cost $O(P^3 + T^3)$ for fitting and $O(P^3 + T^3)$ for posterior sampling when using Matheron's rule [Maddox et al. 2021a].

Data Space GPs We use a standard Gaussian likelihood for these responses on the log transformed data and optimize both the scale of the volatility as well as the noise term, initializing the noise to be 10^{-4}. As these models reduce to a standard exact GP conditional on volatility, computational and memory costs then run at $O(T^3)$ time.

A.2.3 Experimental Details

A.2.3.1 Details from Section 2.9.1

We source daily closing prices for stocks in the Nasdaq 100 for 2 years prior to January 2022. Volt models are trained according to the outline in Section 2.8.2, and standard GPs are implemented and trained via GPyTorch and BoTorch [Gardner et al. 2018b; Balandat et al. 2020]. The LSTM model is implemented with 2 hidden layers each with 128 units and takes the form

$$f(s_t, s_{t-1}, s_{t-2}, s_{t-3}, s_{t-4}) = \{\hat{\mu}_{t+1}\hat{\sigma}_{t+1}\}$$

where $\hat{\mu}_{t+1}$ is the predicted mean at time $t+1$, and $\hat{\sigma}_{t+1}$ is the predicted standard deviation at time $t+1$.

For each stock in our universe we select 25 cutoff times at which we generate forecasts, using the preceding 400 observations as training data. At each cutoff time we forecast the log closing price 100 days into the future, and compute the calibration and negative log likelihood of the forecasts 75 to 100 days out. We specifically focus on longer horizon forecasts, as it is generally a harder task for which out of the box methods are ill-suited.

	Stock Prices	Wind Speeds	FX
Volt + Magpie	5.88 ± 0.02	4.28 ± 0.16	−1.69 ± 0.02
Volt + Con.	**4.69 ± 0.03**	**3.38 ± 0.05**	−1.60 ± 0.02
Matérn + Magpie	9.80 ± 0.27	12.13 ± 0.81	4.23 ± 0.30
Matérn + Con.	7.74 ± 0.21	18.03 ± 1.90	−0.36 ± 0.04
SM + Magpie	147.84 ± 1.84	110.07 ± 7.81	562.67 ± 15.71
SM + Con.	80.43 ± 0.57	70.14 ± 5.03	356.55 ± 11.98
LSTM	49.95 ± 0.59	45.13 ± 1.82	10.66 ± 0.44
Volt-VHGP + Con.	4.76 ± 3.05	5.75 ± 0.44	−1.58 ± 0.03
Volt-VHGP + Magpie	6.97 ± 1.24	5.91 ± 0.34	−1.66 ± 0.02
GPCV	5.45 ± 1.51	4.89 ± 0.04	**−1.79 ± 0.02**

Table A.5: Negative log likelihoods (NLLs) per test point for the methods compared on both the stock forecasting and wind speed tasks, averaged of tens of thousands of forecasts. While there is a slight improvement in NLL from using a constant mean, the inclusion of Magpie is central to achieving high calibration.

A.2.3.2 Details from Section 2.9.2

We source data from Diamond et al. [2013] for the 2021 calendar year. Wind measurements are taken at 15 minute intervals for all 154 stations in the observation network. In order to treat the observed wind speed as log-normally distributed we add 1 to each observation (to shift the 0 m/s observations to a value of 1), and then model the log of the resulting time series.

Figure A.12 compares the performance of Volt alone and Volt with Magpie mean functions with various smoothing parameters. Magpie means aid in calibration, although the effect is less pronounced as we see with stock forecasting in Figure 2.12.

A key distinction between wind speed forecasting and stock price forecasting is that wind speeds tend to revert to a consistent level, whereas stock prices may increase by thousands then stabilize at a new level. For this reason we explore the use of mean reversion in our rollout forecasts. To add mean reversion to the rollouts we simply adjust the posterior mean of the GP towards the mean of the training data by a factor of θ. That is, rather than sampling from the GP posterior $s_{t^*} \sim \mathcal{N}(\mu^*_{f|\mathcal{D}}|\Sigma^*_{f|\mathcal{D}})$ we sample from $s_{t^*} \sim \mathcal{N}(\mu^*_{f|\mathcal{D}} - \theta(\mu^*_{f|\mathcal{D}} - \frac{1}{N}\sum_i s_i)|\Sigma^*_{f|\mathcal{D}})$.

In this mean reversion setting, θ controls the speed at which rollouts tend to revert towards

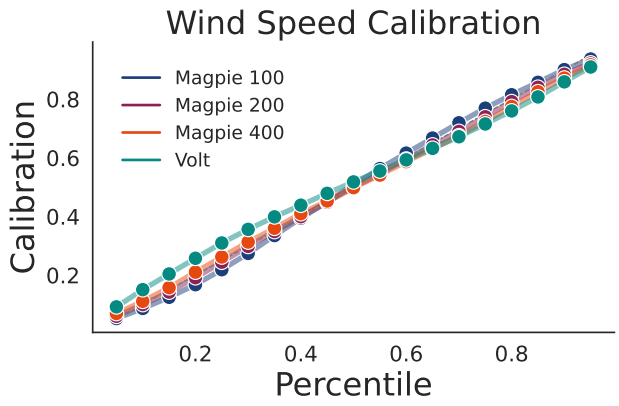


Figure A.12: A comparison of Volt with a constant mean, and Volt with various Magpie means in terms of calibration in wind forecasts. While Volt with a constant mean is well calibrated, it is aided by the inclusion of a Magpie mean with large smoothing parameter.

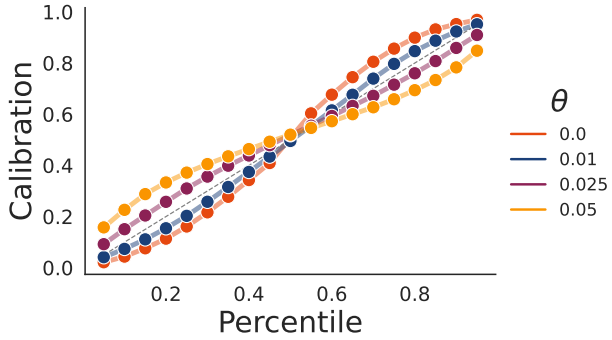


Figure A.13: Calibration of different mean reversion θ values across stock prices.

the mean. At $\theta = 0$ we are in the standard GP prediction case, at $\theta = 1$, we only ever sample from a distribution centered around the mean of the training observations. Figure A.13 provides a comparison of the calibration under differing levels of mean reversion for Volt. The standard Volt rollouts are in general well calibrated for this problem, but we see that just a small amount of mean reversion can increase the overall calibration notably.

A.2.4 Details from Section 2.10

In Figure A.14, we construct a multi-task SABR volatility model with correlations given by the farthest right panel and volatility processes given as the blue lines in the left three panels. We

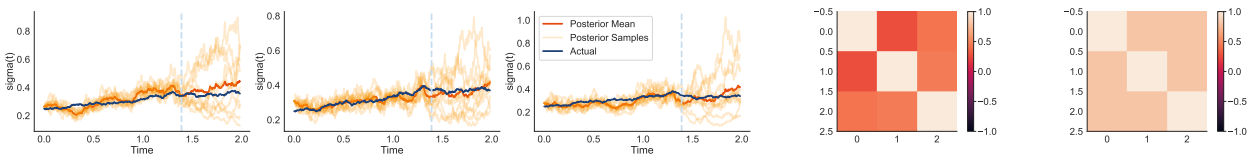


Figure A.14: Left three panels: Predicted correlated volatility models. **Fourth panel:** Estimated correlation matrix between volatility models. **Fifth panel:** True volatility correlation. Multitask GPCVs are able to both predict volatility while also estimating the true correlation between volatility.

then use our multi-task GPCV model to estimate and predict the true volatilities for each task in the left three panels, while also estimating the true relationships between each volatility. The estimated relationships are shown in the fourth panel from left, which is pleasingly similar to the true correlation shown at far right.

For Figure 2.15left and Figures A.15, A.17(a), A.17(b), we fit stocks comprising of five different exchange traded fund SPDRs[2] collected over 5 years of daily data from 09/2016 to 09/2021. These SPDRS are XLE, XLF, XLK, XLRE, XLY; each had six stocks in it except for XLF which had 30. We fit on 300 days and evaluated 100 days into the future, with 5 rolling testing sets for each prediction.

For Figure 2.15 center, we used the same training data except used only five stocks from the XLE SPDR and five from the XLF SPDR.

For Figure 2.15 right and Figures A.16, A.18 we fit about 100 different wind stations (depending on amount of missing data) at 5 minute intervals across 2021 with 25 indepdendent rolling splits. We fit on 252 increments and tested on 100 increments. Here, on the multi-task ones, we used a larger RTX 8000 GPU.

[2] https://en.wikipedia.org/wiki/SPDR

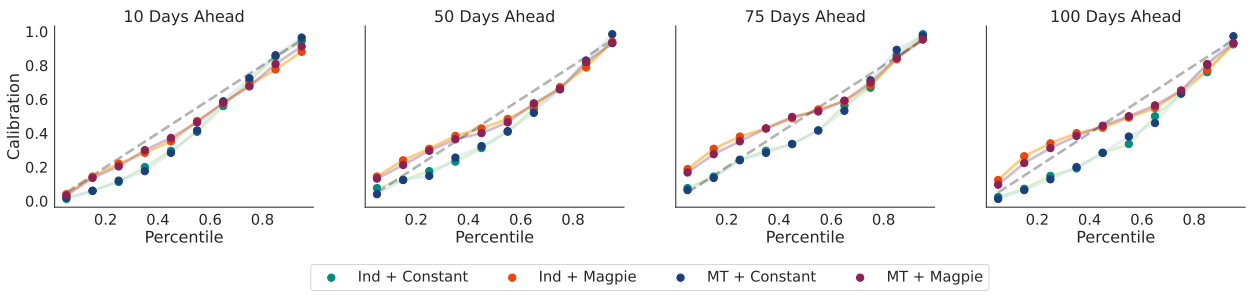


Figure A.15: Calibration of multi-task Volt and independent models across time step lookaheads for 5 different SPDRS.

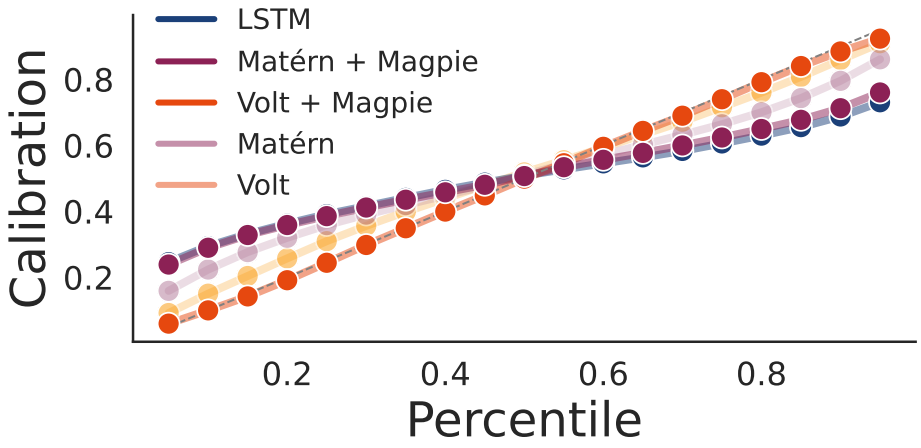
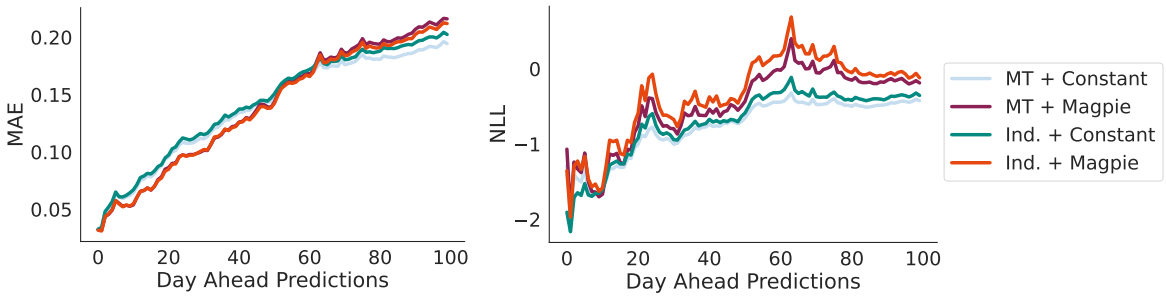


Figure A.16: Calibration of multi-task Volt and independent models across time step lookaheads for the wind forecasting datasets.

A.3 Appendix for Residual Pathway Priors for Soft Equivariance Constraints

Appendix Outline In Section 3.7 discuss potential for negative impact. In Section A.3.2 we investigate the utility of using RPP-EMLP for the policy function only on the Mujoco tasks. In Section A.3.3 we detail the datasets and experimental methodology used in the paper. Finally in Sections A.3.4 and A.3.5 we break down the components of the Mujoco environment state and action spaces, and the representations that we use for them.



(a) MAE **(b)** NLL

Figure A.17: Left panel: Mean absolute error of rollouts. **Right panel:** Negative log likelihood of rollouts.

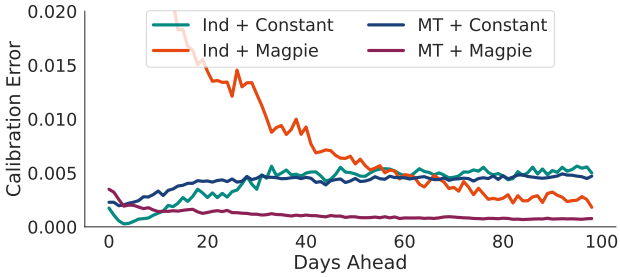


Figure A.18: Calibration error of the models across different time step lookaheads for the wind forecasting task.

A.3.1 Potential Negative Impacts

As one of our primary application areas is reinforcement learning, and specifically exploiting approximate symmetries in reinforcement learning, we must address the potential negative impacts of the deployment of RPPs in RL systems. In general model free RL algorithms tend to be brittle, and often policies and behavior learned in a simulated environment like Mujoco don't transfer easily to real world robots. This point is acknowledged by most RL researchers, and a large effort is being made to improve the situation. Applying neural networks to the control of real robots can be dangerous if the functions are important or failure can cause injury to the robot or humans. We believe that RL will ultimately be impactful for robot control, however practitioners need to be responsible and exercise caution.

A.3.2 Benefit of Equivariant Value Functions

In principle both the policy and the value or critic function can benefit from equivariance. However, the policy learns from the value function in the policy update which is approximately equivalent to minimizing the KL divergence

$$\mathbb{E}_{s\sim\mathcal{D}}[\mathrm{KL}(\pi_\phi(\cdot|s)|\exp(Q_\theta(\cdot,s))/Z_\theta(s))]$$

as derived in Haarnoja et al. [2018b]. If the value function Q is a standard MLP yielding a non equivariant distribution and the policy function π is an RPP that merely has a bias towards equivariance, then the RPP policy will learn to fit the non equivariant parts of Q as if it were a ground truth dataset that is not equivariant. This likely explains why we find in practice that using an RPP for the value function has a stronger impact on performance as shown in Figure 3.5.

A.3.3 Experimental Details

Here we present the training details of the models used in the paper. Experiments were run on private servers with NVIDIA Titan RTX and RTX 2080 Ti GPUs. We estimate that all runs performed in the initial experimentation and final evaluation on the RL tasks used approximately 500 GPU hours. The experiments on dynamical systems, CIFAR-10, and UCI data required an additional 200 GPU hours.

A.3.3.1 Synthetic Dataset Experiments (3.5.1 and 3.5.3)

The windy pendulum dataset is a variant of the double spring pendulum Hamiltonian system from Finzi et al. [2021]. In addition to the Hamiltonian of the base system

$$H_0(x_1,x_2,p_1,p_2) = V(x_1,x_2) + T(p_1,p_2)$$

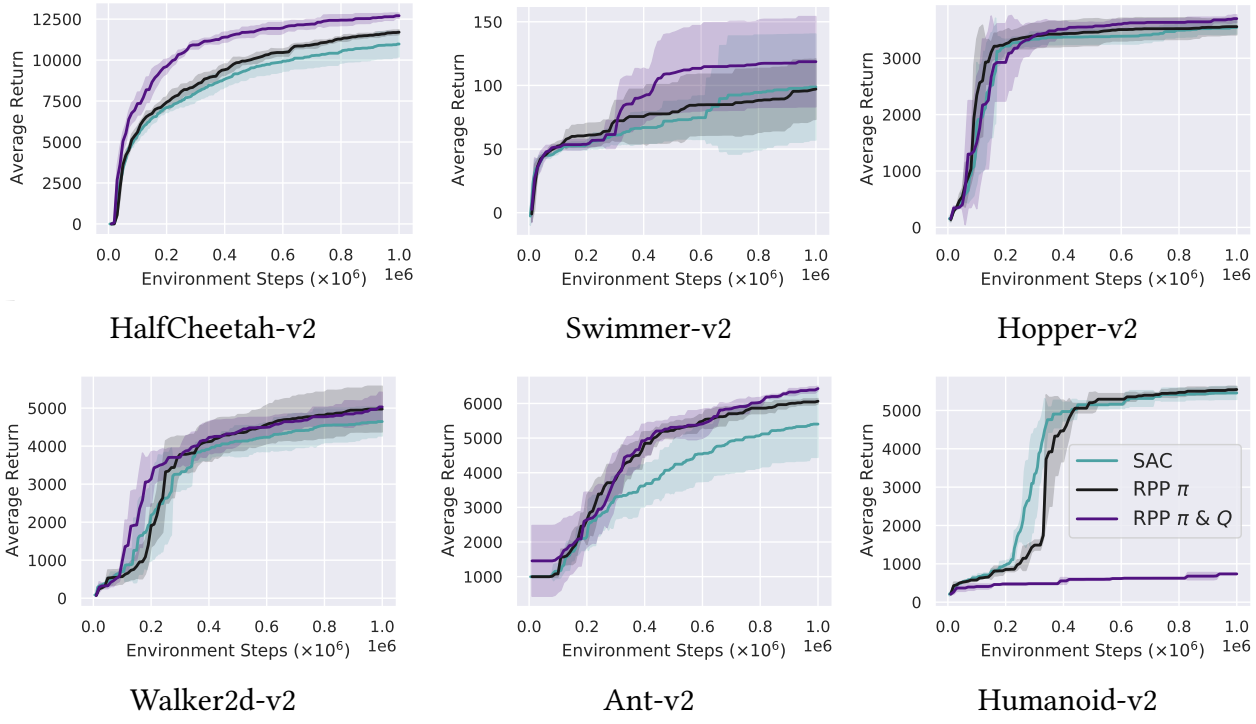


Figure A.19: Average reward curves (max over steps) for an RPP-EMLP applied to the policy π only, as well as an RPP-EMLP for both the policy π and the critic Q. Mean and standard deviation taken over 4 trials shown in the shaded region. Only minor performance gains are achieved if using RPP for the policy only, however this variant is more stable and can to train on Humanoid-v2 without diverging.

where $T(p_1, p_2) = \|p_1\|^2/2m_1 + \|p_2\|^2/2m_2$ and $V(x_1, x_2) =$

$$\tfrac{1}{2}k_1(\|x_1\| - \ell_1)^2 + \tfrac{1}{2}k_2(\|x_1 - x_2\| - \ell_2)^2 + m_1 g^\top x_1 + m_2 g^\top x_2,$$

we add a perturbation $H_1(x_1, x_2, p_1, p_2) = -w^\top x_1 - w^\top x_2$ that is the energy of the wind acting as a constant force pushing in the $w = [-8, -5, 0]$ direction. Setting $H = H_0 + \epsilon H_1$, we can control the strength of the wind and we choose $\epsilon = 0.01$. This perturbation breaks the SO(2) symmetry about the z axis.

For the MLP, EMLP, and RPP we use 3 layer deep 128 hidden unit Hamiltonian neural networks [Greydanus et al. 2019] to fit the data using the rollouts of an ODE integrator [Chen et al. 2018] with an MSE loss on rollouts of length 5 timesteps with $\Delta t = 0.2$. For training we use 500 trajectory chunks and use another 500 for testing. We train all models in section 3.5.1 for 1000 epochs, sufficient for convergence. The input and output representation for EMLP and RPP-EMLP is $V_{\mathrm{O}(3)}^4 \to \mathbb{R}$, where $V_{\mathrm{O}(3)}$ is the restricted representation from the standard representation of a 3D rotation matrix to the given group in question, like SO(2) for rotations about the z axis. The input is $V_{\mathrm{O}(3)}^4$ because there are two point masses each of which has a 3D vectors for position and for momentum. The scalar $\mathbb{R}$ output is the Hamiltonian function.

The Modified Inertia dataset is a small regression dataset off of the task also from Finzi et al. [2021] for learning the moment of inertia matrix in 3D of a collection of 5 point masses. For the base Inertia dataset, the targets are $\mathcal{I} = \sum_{i=1}^5 m_i(x_i^\top x_i I - x_i x_i^\top)$ from the input tuples $(m_i, x_i)_{i=1}^5$. In order to break the equivariance of the dataset, we add an additional term so that the target is $y = \mathrm{vec}(\mathcal{I} + 0.3\mathcal{I}^2 \hat{z}\hat{z}^\top \mathcal{I})$ where $\hat{z}$ is the unit vector along the z axis. The input and output representations for EMLP and RPP-EMLP on this problem are $(\mathbb{R} \oplus V)^5 \to V \otimes V$, representing the 5 point masses and vectors mapping to matrices $V \otimes V$.

We use 1000 train and test examples for the inertia datasets and we train for 500 epochs. In both cases we use an Adam optimizer [Kingma and Ba 2014] with a learning rate of 0.003.

A.3.3.2 Image and UCI experiments (3.5.4)

We use the CIFAR-10 and UCI datasets, taken from Krizhevsky et al. [2009] and Dua and Graff [2017] respectively. In Section 3.5 we train models on dynamical systems and CIFAR-10 and UCI regression data. For the CIFAR-10 experiments we use a convolutional neural network (and the equivalent MLP) with 9 convolutional layers and 1 fully connected layer, and max-pooling layers after the third and sixth convolutional layers. The channel sizes of the 9 layers are, in order: 16, 16, 16, 32, 32, 32, 32, 32, 32. We train for 200 epochs using a cosine learning rate schedule with an initial learning rate of 0.05 and the Adam optimizer.

For the UCI tasks we use a small convolutional neural network, and the equivalent MLP, with 3 convolutional layers and 1 fully connected layer, with each convolutional layer having 32 channels. Models are trained for 1000 epochs using an Adam optimizer with a learning rate of 0.01 and a cosine learning rate schedule.

A.3.3.3 Model Free RL

We train on the Mujoco locomotion tasks in the OpenAI gym environments [Brockman et al. 2016]. We follow the implementation details and hyperparameters from Haarnoja et al. [2018c], with a learned temperature function, stochastic policies, and double critics. Additionally we use the recommendation from Andrychowicz et al. [2020] to initialize the last layer of the policy network with 100x smaller weights, which we find slightly improves the performance of both RPP and the baseline. Additionally for RPP which can be less stable than standard SAC, we use the Adam betas $\beta_1 = 0.5$ and $\beta_2 = 0.999$ that are used in he GAN community [Miyato et al. 2018] rather than the defaults. Training with the RPP π and Q functions on the Mujoco locomotion tasks takes about 8 hours for 1 million steps.

We found it necessary to reduce the speed τ of the critic moving average to keep SAC stable on some of the environments, with values shown in Table A.6. In general, higher τ's are favorable

for learning quickly. Unfortunately we were not able to get SAC with an RPP Q function to train reliably on Humanoid, even after trying multiple values of τ.

	Walker2d	Hopper	HalfCheetah	Swimmer	Ant	Humanoid
Baseline τ	.005	.005	.005	.005	.005	.005
RPP τ	.004	.005	.005	.004	.005	✗

Table A.6: Critic moving average speed τ.

A.3.3.4 Transition Models for Mujoco

We train the transition models on a dataset of 50000 transitions which are composed of 5000 trajectory chunks of length 10. These trajectory chunks are sampled uniformly from the replay buffer collected over the course of training a standard SAC agent for 10^6 steps on each of the environments. We train by minimizing the $\ell 1$ norm of the rollout error over a 10 step trajectory, and we evaluate on a holdout set of 50 trajectories of length 100.

The models are simple MLPs or RPPs mapping from the state and control actions to the state space, predicting the change in state,

$$x_{t+1} = x_t + \text{NN}(x_t, u_t).$$

For the MLPs and RPPs we use 2 hidden layers of size 256 as well as swish activations [Ramachandran et al. 2017]. We use a prior variance of 10^6 in the equivariant subspace and 3 in the non equivariant subspace. The RPP is a standard RPP-EMLP with the input representation $\rho_X \oplus \rho_U$ (concatenation of the representation of the state space and the action space), output representation ρ_X, and symmetry group described in subsection A.3.4 the same as for the model free experiments. We train the transition models for 500 epochs which takes about 45 minutes for RPP compared to 15 minutes for the standard MLPs.

A.3.4 Mujoco State and Action Representations

Based on the state and action spaces of the Mujoco environments we describe in subsection A.3.5, we define appropriate group representations on these spaces. Let V be the base representation of the group acted upon by permutations for $\mathbb{Z}_n$ and by rotation matrices for SO(2), let $\mathbb{R}$ denote a scalar representation (of dimension 1) that is unaffected by the transformations, and let P be a pseudoscalar representation (of dimension 1) that transforms by the sign of the permutation. For $\mathbb{Z}_2$, P takes the values 1 and -1 and acts by negating the values when a flip or L/R reflection is applied.

Table A.7: Mujoco Locomotion State and Action Representations used for RPP-EMLP

Env	State Representation	Action Rep	Group
Hopper	$\mathbb{R} \oplus P^5 \oplus \mathbb{R} \oplus P^4$	P^3	$\mathbb{Z}_2$
Swimmer	$\mathbb{R} \oplus P_{\leftrightarrow} \oplus (P_{\leftrightarrow} \otimes V_{\updownarrow}) \oplus (\mathbb{R} \oplus P)^2 \oplus (P_{\leftrightarrow} \otimes V_{\updownarrow})$	$P_{\leftrightarrow} \otimes V_{\updownarrow}$	$\mathbb{Z}_2^{\leftrightarrow} \times \mathbb{Z}_2^{\updownarrow}$
HalfCheetah	$\mathbb{R} \oplus P^8 \oplus \mathbb{R} \oplus P^7$	P^6	$\mathbb{Z}_2$
Walker2d	$\mathbb{R}^2 \oplus V^3 \oplus \mathbb{R}^3 \oplus V^3$	V^3	$\mathbb{Z}_2$
Ant	$\mathbb{R}^5 \oplus V^2 \oplus \mathbb{R}^6 \oplus V^2$	V^2	$\mathbb{Z}_4$
Humanoid	$\mathbb{R} \oplus V_{\mathrm{SO}(3)}^{\otimes 2} \oplus \mathbb{R}^{17} \oplus V_{\mathrm{SO}(3)}^2 \oplus \mathbb{R}^{17}$	$\mathbb{R}^{17}$	SO(2)

From the raw state and action spaces listed in subsection A.3.5, we convert quaternions to 3D rotation matrices for Humanoid and Ant, and we reorder elements to group together left/right pairs for Walker2d and Swimmer. The representations of these transformed state and action vectors are shown in Table A.7. Note that V^3 denotes $V \oplus V \oplus V = V^{\oplus 3}$, and is simply the concatenation of 3 copies of V as $\mathbb{R}^3$ would be 3 copies of $\mathbb{R}$. This is not to be confused with powers of the tensor product, $V^{\otimes 3} = V \otimes V \otimes V$. For Humanoid, we denote the restricted representation of 3D rotation matrices restricted to the SO(2) rotations about the z axis as $V_{\mathrm{SO}(3)}$.

A.3.5 Mujoco State and Action Spaces

In order to build symmetries into the state and action representations for Mujoco environments, we need to have a detailed understanding of what the state and action spaces for these environments represent. As these spaces are not well documented, for each of the Mujoco environments we experimented in the simulator and identified the meanings of the state vectors in Tables A.12, A.14, A.13, A.9, A.11, A.8, and A.10. We hope that these detailed descriptions can be useful to other researchers.

Table A.8: Hopper-v2 State and Action Spaces

State Space	X (Unobserved)
	Y
	Orientation Angle
	Hip Angle
	Knee Angle
	Ankle Angle
	X Velocity
	Y Velocity
	Orientation Angular Velocity
	Hip Angular Velocity
	Knee Angular Velocity
	Ankle Angular Velocity
Action Space	Hip
	Knee
	Ankle

Table A.9: Swimmer-v2 State and Action Spaces

State Space	X (Unobserved)
	Y (Unobserved)
	Orientation Angle
	Head Joint Angle
	Tail Joint Angle
	X Velocity
	Y Velocity
	Orientation Angular Velocity
	Head Joint Angular Velocity
	Tail Joint Angular Velocity
Action Space	Head Joint
	Tail Joint

Table A.10: HalfCheetah-v2 State and Action Spaces

State Space	X (Unobserved)
	Y
	Orientation Angle
	Rear Hip Angle
	Rear Knee Angle
	Rear Ankle Angle
	Front Hip Angle
	Front Knee Angle
	Front Ankle Angle
	X Velocity
	Y Velocity
	Orientation Angular Velocity
	Rear Hip Angular Velocity
	Rear Knee Angular Velocity
	Rear Ankle Angular Velocity
	Front Hip Angular Velocity
	Front Knee Angular Velocity
	Front Ankle Angular Velocity
Action Space	Rear Hip
	Rear Knee
	Rear Ankle
	Front Hip
	Front Knee
	Front Ankle

Table A.11: Walker2d-v2 State and Action Spaces

State Space	X (Unobserved)
	Y
	Orientation Angle
	Right Hip Angle
	Right Knee Angle
	Right Ankle Angle
	Left Hip Angle
	Left Knee Angle
	Left Ankle Angle
	X Velocity
	Y Velocity
	Orientation Angular Velocity
	Right Hip Angular Velocity
	Right Knee Angular Velocity
	Right Ankle Angular Velocity
	Left Hip Angular Velocity
	Left Knee Angular Velocity
	Left Ankle Angular Velocity
Action Space	Right Hip
	Right Knee
	Right Ankle
	Left Hip
	Left Knee
	Left Ankle

A.4 Appendix for Learning Invariances in Neural Networks

A.4.1 Forming The Invariant Model

We form a model that is approximately invariant to transformations in $\text{supp}(\mu_\theta) = \mathcal{S}$ by taking the expectation over transformations $g \sim \mu_\theta$:

$$\bar{f}(x) = \mathrm{E}_{g\sim\mu_\theta} f(gx). \tag{A.7}$$

If μ_θ is uniform over the full span of a transformation, such as rotations in $[-\pi, \pi]$, then $\bar{f}(x)$ will be exactly invariant with respect to that transformation. In cases where $\mathcal{S}$ has only partial support over transformations, Equation (A.7) alone does not imply invariance. For example, let μ_θ be a uniform distribution over rotations in $[-\pi/2, \pi/2]$. Then for an input image x and and an input $x' = r_{\pi/2}x$, i.e. the image x rotated by $\pi/2$ radians, we have

$$\bar{f}(x) = \int_{-\pi/2}^{\pi/2} f(r_\phi x) d\phi$$
$$\bar{f}(x') = \int_{-\pi/2}^{\pi/2} f(r_\phi x') d\phi = \int_{0}^{\pi} f(r_\phi x) d\phi \,.$$

Therefore without additional properties on f, we cannot guarantee that $\bar{f}(x) = \bar{f}(x')$. This behaviour is in contrast to the case of having a complete invariance where the support of μ_θ is closed over transformations.

However, even in these cases of partial support over invariances, the training procedure still leads to invariant or nearly invariant models (also referred to as *insensitivity* in van der Wilk et al. [2018]). This empirical fact can be naturally understood from the perspective of data augmentation. Once we iterate through the training set many times, then for each input x the network $\bar{f}$ will have been trained on inputs gx for many $g \sim \mu_\theta$. If our network achieves near 0 training loss, as is typical for image problems, then we will have a network which predicts the same correct label for each input gx with $g \sim \mu_\theta$, giving a network $\bar{f}$ that is approximately invariant to the correct augmentations. In practice, the network will generalize this insensitivity to transformations on unseen test data.

In particular, Augerino learns the maximal possible augmentations that do not hurt training performance. For example, suppose we observe rotations of the digit '6' in the range $[-\pi/4, \pi/4]$ from the vertical. Augerino will learn rotation invariance up to $\pi/4$, as rotating further will move some of the observations below the upper half plane, where they may be more correctly labelled as '9'. Once μ_θ has converged to $[-\pi/4, \pi/4]$, $\bar{f}$ will be trained to correctly classify observations

of the digit '6' rotated over the upper half plane, giving approximate invariance to any rotation in $[-\pi/4, \pi/4]$.

A.4.2 Lie Group Generators

The six Lie group generating matrices for affine transformations in 2D are,

$$G_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad G_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$
$$G_4 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad G_5 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad G_6 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{A.8}$$

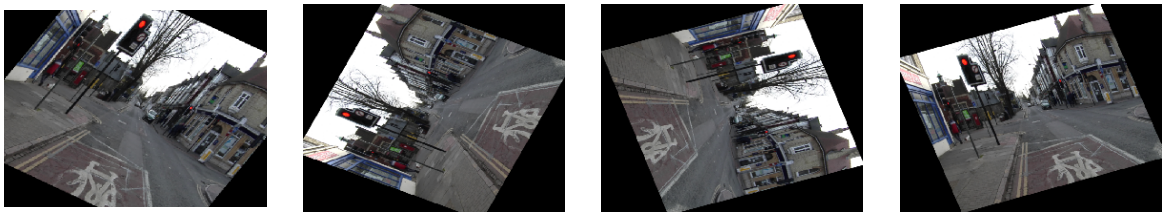
Applying the exponential map to these matrices produces affine matrices that can be used to transform images. In order, these matrices correspond to translations in x, translations in y, rotations, scaling in x, scaling in y, and shearing.

A.4.3 Semantic Segmentation: Details

In Section 3.15, we apply Augerino to semantic segmentation on the rotCamVid dataset (see Figure A.20).

To generate the rotCamVid dataset, we rotate all images in the CamVid by a random angle, analogously to the rotMNIST dataset [Larochelle et al. 2007]. We note that rotCamVid only contains a single rotated copy of each image, which is not the same as applying rotational augmentation during training. When computing the training loss and test acccuracy, we ignore the padding pixels which appear due to rotating the image.

For the segmentation experiment we used the simpler augmentation distribution covering



(a) Original Data **(b)** Augerino Sample **(c)** Augerino Sample **(d)** Augerino Sample

Figure A.20: Augmentations learned by Augerino on the rotCamVid dataset. **(a)**: original data from rotCamVid; **(b)-(d)**: three random samples of augmentations from the learned augerino distribution. Augerino learns to be invariant to rotations but not translations.

rotations and translations instead of the affine transformations (Section 3.10.2). We use a Gaussian parameterization of the distribution:

$$t = (t_1, t_2, t_3) \sim \mathcal{N}(\mu, \Sigma), \quad A(t) = \begin{bmatrix} \cos(t_1) & -\sin(t_1) & 2 \cdot t_2/(w+h) \\ \sin(t_1) & \cos(t_1) & 2 \cdot t_3/(w+h) \end{bmatrix}, \tag{A.9}$$

where μ, Σ are trainable parameters, and $A(t)$ is the affine transformation matrix for the random sample t; w and h are the width and height of the image.

Augerino achieves pixel-wise segmentation accuracy of 69.8% while the baseline model with standard augmentation achieves 68.7%.

A.4.4 Training Details

Network Training Hyperparameters We train the networks in Sections 3.11 and 3.13 for 200 epochs, using an initial learning rate of 0.01 with a cosine learning rate schedule and a batch size of 128. We use the cross entropy loss function for all classification tasks, and mean squared error for all regression tasks except for QM9 where we use mean absolute error.

Train- and Test-Time Augmentations In Algorithm 1 we include a term *ncopies* that denotes the number of sampled augmentations during training. We find that we can achieve strong

(a) Original Data **(b)** Augerino Sample **(c)** Augerino Sample **(d)** Augerino Sample

Figure A.21: Color-space augmentation distribution learned by Augerino. **(a)**: original data from STL-10; **(b)-(d)**: three random samples of augmentations from the learned augerino distribution. Augerino learns to be invariant to a broad range of color and contrast adjustments while matching the performance of the baseline.

performance with Augerino, with minimally increased training time, by setting *ncopies* to 1 at train-time and then applying multiple augmentations by increasing *ncopies* at *test-time*. Thus we train using a single augmentation for each input, and then apply multiple augmentations at test-time to increase accuracy, as seen in Table 3.4.

A.4.5 Color-Space Augmentations: Details

In Section 3.16, we apply Augerino to learning color-space invariances on the STL-10 dataset. We consider two transformations:

- Brightness adjustment by a value t transforms the intensity c in each channel additively:

$$c' = \max(\min(c + t, 255), 0). \tag{A.10}$$

  Positive t increases, and negative t decreases brightness.

- Contrast adjustment by a value t transforms the intensity c in each channel as follows[3]:

$$c' = \max\left(\min\left(\frac{259 \cdot (t + 255)}{255 \cdot (259 - t)} \cdot (c - 128) + 128, \quad 255\right), 0\right) \tag{A.11}$$

We apply brightness and contrast adjustments sequentially and independently from each other. We learn the range of a uniform distribution over the values t in (A.10), (A.11). The learned data augmentation strategy is visualized in Figure A.21.

A.4.6 QM9 Experiment

We reproduce the training details from Finzi et al. [2020]. Affine transformations in 3d, there are 9 generators, 3 for translation, 3 for rotation, 2 for squeezing and 1 for scaling, a straightforward extension of those listed in equation A.8 to 3 dimensions. Like before, we parametrize the bounds on the uniform distribution for each of these generators. We use a regularization strength of 10^{-3}.

A.4.7 Width of Augerino Solutions

To help explain the increased generalization seen in using Augerino, we train 10 models on CIFAR-10 both with and without Augerino. In Figure A.22 we present the test error of both types of models for along with the corresponding effective dimensionalities and sensitivity to parameter perturbations of the networks as a measure of the *flatness* of the optimum found through training. Maddox et al. [2020] shows that effective dimensionality can capture the flatness of optima in parameter space and is strongly correlated to generalization, with lower effective dimensionality implying flatter optima and better generalization. Overall we see that Augerino enables networks to find much flatter solutions in the loss surface, corresponding to better compressions of the data and better generalization.

[3] https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/

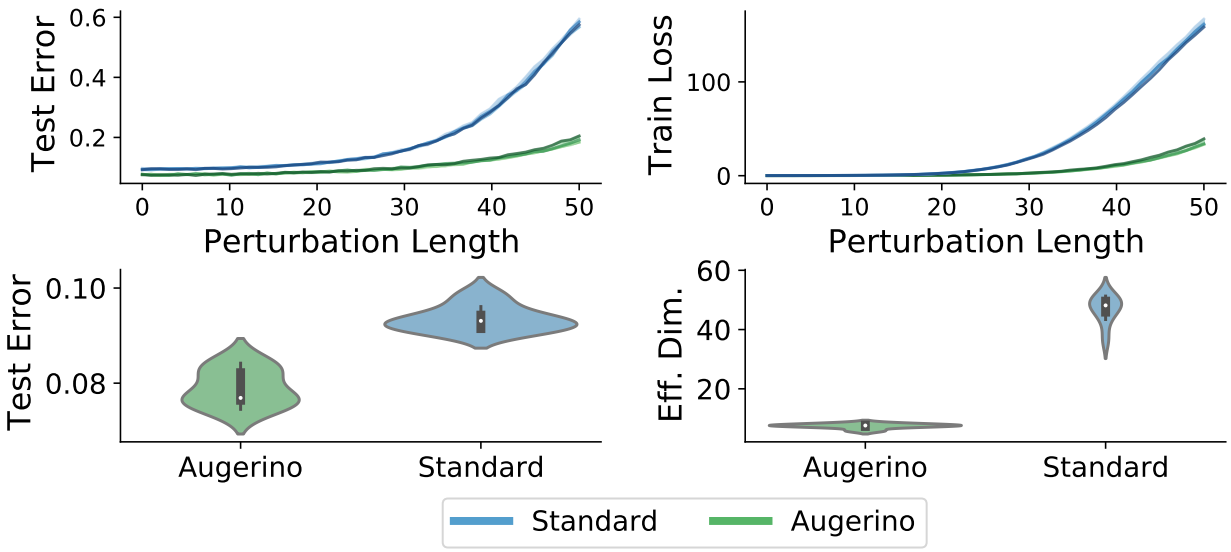


Figure A.22: Top: Test error and train loss as a function of perturbation lengths along random rays from the SGD found training solution for models. Each curve represents a different ray. **Bottom:** Test error and effective dimensionality for models trained on CIFAR-10. Results from 8 random initializations are presented violin-plot style where width represents the kernel density estimate at the corresponding y-value.

A.5 Appendix for Effective Dimensionality Revisited

A.5.1 The Hessian and Effective Dimensionality over the Course of Training

One possible limitation of using the Hessian as a measurement for posterior contraction for (Bayesian) deep learning would be if the Hessian was constant through the training procedure, or if the eigenvalues of the Hessian remained constant. Jacot et al. [2018] showed that in the limit of infinite width neural networks, the Hessian matrix converges to a constant, in a similar manner to how the Fisher information matrix and Jacobian matrices converge to a constant limit, producing the neural tangent kernel (NTK) [Jacot et al. 2018]. However, Lee et al. [2019] recently showed that while the infinite width NTK is a good descriptor of finitely wide neural networks, the corresponding finite width NTK is not constant throughout training. Similarly, the empirical

observations of Papyan [2018], Sagun et al. [2017], and Ghorbani et al. [2019] demonstrate that even for extremely wide neural networks, the Hessian is not constant through training.

Preliminary experiments with both the Fisher information matrix (using fast Fisher vector products as described in Maddox et al. [2019b]) and the NTK demonstrated similar empirical results in terms of double descent and effective dimensionality as the Hessian matrix.

A.5.2 Further Statements on Effective Dimensionality

In this subsection, we provide further results the effective dimensionality, including its connection to both the bias-variance decomposition of predictive risk [Geman et al. 1992; Dobriban and Wager 2018] as well as the Hilbert space norm of the induced kernel [Rasmussen and Williams 2008].

A.5.2.1 Effective Dimensionality of the Inverse of A

We show that

$$rank(A) - N_{eff}(A, \alpha) = N_{eff}(A^{+}, 1/\alpha), \tag{A.12}$$

formalizing the idea that as the effective dimensionality of the covariance increases, the effective dimensionality of the inverse covariance decreases. This statement is alluded to in the analysis of MacKay [1992a] but is not explicitly shown.

We assume that A has rank r and that $\alpha \neq 0$; we also assume that A^{+} is formed by inverting the non-zero eigenvalues of A and leaving the zero eigenvalues fixed in the eigendecomposition

of A (i.e. the Moore-Penrose pseudo-inverse). With λ_i as the eigenvalues of A, we can see that

$$\begin{aligned} r - N_{eff}(A, \alpha) &= \sum_{i=1}^{r} \frac{\lambda_i + \alpha - \lambda_i}{\lambda_i + \alpha} = \alpha \sum_{i=1}^{r} \frac{1}{\lambda_i + \alpha} \\ &= \sum_{i=1}^{r} \frac{1}{1/\alpha} \frac{1}{\lambda_i + \alpha} = \sum_{i=1}^{r} \frac{1}{\lambda_i/\alpha + 1} \\ &= \sum_{i=1}^{r} \frac{1/\lambda_i}{1/\lambda_i(\lambda_i/\alpha + 1)} = \sum_{i=1}^{r} \frac{1/\lambda_i}{1/\alpha + 1/\lambda_i} \\ &= N_{eff}(A^+, 1/\alpha). \end{aligned}$$

When A is invertible, the result reduces to $k - N_{eff}(A, \alpha) = N_{eff}(A^{-1}, 1/\alpha)$ for $A \in \mathbb{R}^{k \times k}$.

A.5.2.2 Predictive Risk for Bayesian Linear Models

Dobriban and Wager [2018] and Hastie et al. [2019] have extensively studied over-parameterized ridge regression. In particular, Theorem 2.1 of Dobriban and Wager [2018] gives the predictive risk (e.g. the bias-variance decomposition of Geman et al. [1992]) as a function of effective dimensionality and intrinsic noise. The critical aspect of their proof is to decompose the variance of the estimate into the effective dimensionality and a second term which then cancels with the limiting bias estimate. For completeness, we restate Theorem 2.1 of Dobriban and Wager [2018] theorem for fixed feature matrices, Φ, and an explicit prior on the parameters, $\beta \sim \mathcal{N}(0, \alpha^2 I)$, leaving the proof to the original work.

Theorem A.1 (Predictive Risk of Predictive Mean for Ridge Regression)**.** *Under the assumption of model correct specification, $y = \Phi\beta + \epsilon$, with β drawn from the prior and $\epsilon \sim \mathcal{N}(0, I_n)$, and defining $\hat{f} = \Phi\hat{\beta}$, with $\hat{\beta} = (\Phi^\top\Phi + \alpha^{-2}I)^{-1}\Phi^\top y$ (the predictive mean under the prior specification), then*

$$R(\Phi) = \mathbb{E}(||Y - \hat{f}||_2^2) = 1 + \frac{1}{n} N_{eff}(\Phi\Phi^\top, \alpha^{-2}). \tag{A.13}$$

A.5.2.3 Expected RKHS Norm

Finally, we show another unexpected connection of the effective dimensionality — that the reproducing kernel Hilbert space (RKHS) norm is in expectation, under model correct specification, the effective dimensionality. We follow the definition of Gaussian processes of Rasmussen and Williams [2008] and focus on the definition of the RKHS given in Rasmussen and Williams [2008, Chapter 6], which is defined as $||f||^2_{\mathcal{H}} = \langle f, f \rangle_{\mathcal{H}} = \sum_{i=1}^{N} f_i^2 / \lambda_i$, where λ_i are the eigenvalues associated with the kernel operator, K, of the RKHS, $\mathcal{H}$.[4] The kernel is the covariance matrix of the Gaussian process, and assuming that the response is drawn from the same model, then $y \sim \mathcal{N}(0, K + \sigma^2 I)$, then $a = (K + \sigma^2 I)^{-1} y$, where a is the optimal weights of the function with respect to the kernel, e.g. $f = \sum_{i=1}^{N} a_i K(x, .)$. To compute the Hilbert space norm, we only need to compute the optimal weights and the eigenvalues of the operator. For finite (degenerate) Hilbert spaces this computation is straightforward:

$$
\begin{aligned}
\mathbb{E}_{p(y)}(||f||^2_{\mathcal{H}}) &= \mathbb{E}_{p(y)}(a^\top K a) \\
&= \mathbb{E}_{p(y)}(y^\top (K + \sigma^2 I)^{-1} K (K + \sigma^2 I)^{-1} y) \\
&= \mathbb{E}_{p(y)} tr(y^\top (K + \sigma^2 I)^{-1} K (K + \sigma^2 I)^{-1} y) \\
&= \mathbb{E}_{p(y)} tr((K + \sigma^2 I)^{-1} K (K + \sigma^2 I)^{-1} y y^\top) \\
&= tr((K + \sigma^2 I)^{-1} K (K + \sigma^2 I)^{-1} (K + \sigma^2 I)) \\
&= N_{eff}(K, \sigma^2)
\end{aligned}
$$

with the second equality coming by plugging in the optimal a (see Rasmussen and Williams [2008, Chapter 6] and Belkin et al. [2019a] as an example). As linear models with Gaussian priors are Gaussian processes with a degenerate feature expansion, the expected RKHS norm becomes $N_{eff}(\Phi^\top \Phi, \sigma^2/\alpha^2)$, which is the same value as our definition of posterior contraction. Further

[4] Note that the expectation we take in the following is somewhat separate than the expectation taken in Rasmussen and Williams [2008] which is directly over f_i.

research connecting these two ideas is needed.

A.5.3 Measuring Posterior Contraction in Bayesian Generalized Linear Models

We first consider the over-parametrized case, $k > n$:

$$\begin{aligned}
\Delta_{post}(\theta) &= tr(Cov_{p(\theta)}(\theta)) - tr(Cov_{p(\theta|\mathcal{D})}(\theta)) \\
&= \sum_{i=1}^{k} \alpha^2 - \sum_{i=1}^{n} (\lambda_i + \alpha^{-2})^{-1} + \sum_{i=n+1}^{k} \alpha^2 \\
&= k\alpha^2 - (k-n)\alpha^2 - \sum_{i=1}^{n} (\lambda_i + \alpha^{-2})^{-1} \\
&= \sum_{i=1}^{n} \frac{1 - \alpha^2(\lambda_i + \alpha^{-2})}{\lambda_i + \alpha^{-2}} \\
&= \alpha^2 \sum_{i=1}^{n} \frac{\lambda_i}{\lambda_i + \alpha^{-2}}; \qquad \text{(A.14)}
\end{aligned}$$

where we have used Theorem 4.3 to assess the eigenvalues of the posterior covariance. When $n > k$, we have the simplified setting where the summation becomes to k instead of n, giving us that all of the eigenvalues are shifted from their original values to become $\lambda_i + \alpha^{-2}$, and so

$$\Delta_{post.}(\theta) = \alpha^{-2} \sum_{i=1}^{k} \frac{\lambda_i}{\lambda_i + \alpha^{-2}}, \qquad \text{(A.15)}$$

where λ_i is the ith eigenvalue of $\Phi^\top \Phi / \sigma^2$.

A.5.3.1 Contraction in Function Space

We can additionally consider the posterior contraction in function space. For linear models, the posterior covariance on the training data in function space becomes

$$\Phi\Sigma_{\beta|\mathcal{D}}\Phi^\top = \sigma^2\Phi(\Phi^\top\Phi + \frac{\sigma^2}{\alpha^2}I_p)^{-1}\Phi^\top, \tag{A.16}$$

while the prior covariance in function space is given by $\alpha^2\Phi\Phi^\top$. We will make the simplifying assumption that the features are normalized such that $tr(\Phi\Phi^\top) = rank(\Phi\Phi^\top) = r$. Now, we can simplify

$$\begin{aligned}
\Delta_{post}(f) &= tr(Cov_{p(f)}(f) - tr(Cov_{p(f|\mathcal{D})}(f)) \\
&= \alpha^2 r - \sigma^2 \sum_{i=1}^{r} \frac{\lambda_i}{\lambda_i + \sigma^2/\alpha^2} \\
&= \alpha^2 \sum_{i=1}^{r} \frac{\lambda_i + \sigma^2/\alpha^2}{\lambda_i + \sigma^2/\alpha^2} - \sigma^2 \sum_{i=1}^{r} \frac{\lambda_i}{\lambda_i + \sigma^2/\alpha^2} \\
= (\alpha^2 - \sigma^2) \sum_{i=1}^{r} \frac{\lambda_i}{\lambda_i + \sigma^2/\alpha^2} &+ \sigma^2 \sum_{i=1}^{r} \frac{1}{\lambda_i + \sigma^2/\alpha^2}\,.
\end{aligned}$$

Simplifying and recognizing these summations as the effective dimensionalities of $\Phi^\top\Phi$ and $(\Phi^\top\Phi)^+$, we get that

$$\begin{aligned}
\Delta_{post}(f) &= (\alpha^2 - \sigma^2)N_{eff}(\Phi^\top\Phi, \sigma^2/\alpha^2) \\
&\quad + \sigma^2 N_{eff}((\Phi^\top\Phi)^+, \alpha^2/\sigma^2) \\
&= \sigma^2 r + (\alpha^2 - 2\sigma^2)N_{eff}(\Phi^\top\Phi, \sigma^2/\alpha^2),
\end{aligned} \tag{A.17}$$

thereby showing that the posterior contraction in function space is explicitly tied to the effective dimensionality of the Gram matrix.

A.5.4 Posterior Contraction and Function-Space Homogeneity Proofs and Additional Theorems

In this subsection we complete the proofs to Theorems 4.3 and 4.4 and extend the results from linear models to generalized linear models.

A.5.4.1 Proof and Extensions to Theorem 4.3

Theorem (Posterior Contraction in Bayesian Linear Models)**.** *Let $\Phi = \Phi(x) \in \mathbb{R}^{n \times k}$ be a feature map of n data observations, x, with $n < k$ and assign isotropic prior $\beta \sim \mathcal{N}(0_k, S_0 = \alpha^2 I_k)$ for parameters $\beta \in \mathbb{R}^k$. Assuming a model of the form $y \sim \mathcal{N}(\Phi\beta, \sigma^2 I_n)$ the posterior distribution of β has an $p - k$ directional subspace in which the variance is identical to the prior variance.*

Proof. The posterior distribution of β in this case is known and given as

$$\begin{aligned} \beta|\mathcal{D} &\sim \mathcal{N}\left((\mu|\mathcal{D}), (\Sigma|\mathcal{D})\right) \\ \mu|\mathcal{D} &= (\Phi^\top\Phi/\sigma^2 + S_0^{-1})^{-1}\Phi^\top y/\sigma^2 \\ \Sigma|\mathcal{D} &= (\Phi^\top\Phi/\sigma^2 + S_0^{-1})^{-1} \end{aligned} \tag{A.18}$$

We want to examine the distribution of the eigenvalues of the posterior variance. Let $\Phi^\top\Phi/\sigma^2 = V\lambda_n V^\top$ be the eigendecomposition with eigenvalues $\Lambda = \text{diag}(\gamma_1, \ldots, \gamma_n, 0_{n+1}, \ldots, 0_k)$; $k - n$ of the eigenvalues are 0 since the gram matrix $\Phi^\top\Phi$ is at most rank n by construction.

Substitution into the posterior variance of β yields,

$$\begin{aligned}(\Phi^\top\Phi/\sigma^2 + S_0^{-1})^{-1} &= (V\Lambda V^\top + \alpha^{-2}I_k)^{-1} \\ &= V(\Lambda + \alpha^{-2}I_k)^{-1}V^\top \\ &= V\Gamma V^\top.\end{aligned} \tag{A.19}$$

The eigenvalues of the posterior covariance matrix are given by the entries of

$$\Gamma = \left((\gamma_1 + \alpha^{-2})^{-1}, \ldots, (\gamma_n + \alpha^{-2})^{-1}, \alpha^2, \ldots, \alpha^2\right),$$

where there are $k - n$ eigenvalues that retain a value of α^2.

Therefore the posterior covariance has $p - n$ directions in which the posterior variance is unchanged and n directions in which it has contracted as scaled by the eigenvalues of the gram matrix $\Phi^\top\Phi$. □

Generalized linear models (GLMs) do not necessarily have a closed form posterior distribution. However, Neal and Zhang [2006] give a straightforward argument using the invariance of the likelihood of GLMs to orthogonal linear transformation in order to justify the usage of PCA as a feature selection step. We can adapt their result to show that overparameterized GLMs have a $k - n$ dimensional subspace in which the posterior variance is identical to the prior variance.

Theorem A.2 (Posterior Contraction in Generalized Linear Models)**.** *We specify a generalized linear model, $E[Y] = g^{-1}(\Phi\beta)$ and $Var(Y) = V(g^{-1}(\Phi\beta))$, where $\Phi \in \mathbb{R}^{n\times k}$ is a feature matrix of n observations and k features and $\beta \in \mathbb{R}^k$ are the model parameters. In the overparameterized setting with isotropic prior on β, there exists a $k - n$ dimensional subspace in which the posterior variance is identical to the prior variance.*

Proof. First note that the likelihood of a GLM takes as argument $\Phi\beta$, thus transformations that leave $\Phi\beta$ unaffected leave the likelihood, and therefore the posterior distribution, unaffected.

Let R be an orthogonal matrix, $R^\top R = RR^\top = I_p$, and $\tilde{\beta} = R\beta \sim N(0, \sigma^2 I)$. If we assign a standard isotropic prior, to β then $\tilde{\beta} = R\beta \sim \mathcal{N}(0, \sigma^2 R I_k R^\top = \sigma^2 I_k)$. If we also rotate the feature matrix, $\tilde{\Phi} = \Phi R^\top \in \mathbb{R}^{n \times k}$ so that $\tilde{\Phi}\tilde{\beta} = \Phi R^\top R\beta = \Phi\beta$, showing that the likelihood and posterior remain unchanged under such transformations.

In the overparameterized regime, $k > n$, with linearly independent features we have that Φ has rank at most k, and we can therefore choose R to be a rotation such that ΦR has exactly $k - n$ columns that are all 0. This defines a $k - n$ dimensional subspace of $\beta \in \mathbb{R}^k$ in which the the likelihood is unchanged. Therefore the posterior remains no different from the prior distribution in this subspace, or in other words, the posterior distribution has not contracted in $k - n$ dimensions. □

A.5.4.2 Function-Space Homogeneity

Theorem (Function-Space Homogeneity in Linear Models)**.** *Let $\Phi = \Phi(x) \in \mathbb{R}^{n \times k}$ be a feature map of n data observations, x, with $n < k$ and assign isotropic prior $\beta \sim \mathcal{N}(0_k, S_0 = \alpha^2 I_k)$ for parameters $\beta \in \mathbb{R}^k$. The minimal eigenvectors of the Hessian define a $k - n$ dimensional subspace in which parameters can be perturbed without changing the training predictions in function-space.*

Proof. The posterior covariance matrix for the parameters is given by

$$\Sigma_{\beta|\mathcal{D}} = \left(\frac{\Phi^\top \Phi}{\sigma^2} + \alpha^{-2} I_k \right)^{-1},$$

and therefore the Hessian of the log-likelihood is $\left(\frac{\Phi^\top \Phi}{\sigma^2} + \alpha^{-2} I_k \right)$. By the result in Theorem 4.3 there are $k - n$ eigenvectors of the Hessian all with eigenvalue α^{-2}. If we have some perturbation to the parameter vector u that resides in the span of these eigenvectors we have

$$\left(\frac{\Phi^\top \Phi}{\sigma^2} + \alpha^{-2} I_k \right) u = \alpha^{-2} u,$$

which implies u is in the nullspace of $\Phi^\top\Phi$. By the properties of gram matrices we have that the nullspace of $\Phi^\top\Phi$ is the same as that of Φ, thus u is also in the nullspace of Φ Therefore any prediction using perturbed parameters takes the form $\hat{y} = \Phi(\beta + u) = \Phi\beta$, meaning the function-space predictions on training data under such perturbations are unchanged. □

Theorem A.3 (Function-Space Homogeneity in Generalized Linear Models)**.** *We specify a generalized linear model, $E[Y] = g^{-1}(\Phi\beta)$, where $\Phi \in \mathbb{R}^{n\times k}$ is a feature matrix of n observations and k features and $\beta \in \mathbb{R}^k$ are the model parameters. In the overparameterized setting with isotropic prior on β, there exists a $k - n$ dimensional subspace in which parameters can be perturbed without changing the training predictions in function-space or the value of the Hessian.*

Proof. The Hessian of the log-likelihood for GLMs can be written as a function of the feature map, Φ, and the product of the feature map and the parameters, $\Phi\beta$, i.e. β only appears multiplied by the feature map [Nelder and Wedderburn 1972]. We can then write $\mathcal{H}_\beta = f(\Phi\beta, \Phi)$ Additionally predictions are generated by $y = g^{-1}(\Phi\beta)$. Since $\Phi \in \mathbb{R}^{n\times p}$ with $n < p$ there is a nullspace of Φ with dimension at least $n - p$. Thus for any $u \in \text{null}(\Phi)$ we have $g^{-1}(\Phi(\beta + u)) = g^{-1}(\Phi\beta) = y$ and $f(\Phi(\beta + u), \Phi) = f(\Phi\beta, \Phi) = \mathcal{H}_\beta$, which shows that the training predictions and the Hessian remain unchanged. □

A.5.5 Perturbations on CIFAR-10

To demonstrate that the results presented in subsection 4.5 apply to larger architectures similar to those seen in practice we train a convolutional classifier provided by Pytorch on the CIFAR-10 dataset.[5] The network has approximately 62000 parameters and is trained on 50000 images.

Figure A.23 shows the presence of degenerate directions in parameter space. We compute the top 200 eigenvectors of the Hessian of the loss and consider perturbations in the directions of the top 2 eigenvectors, as well as in all parameter directions *except* the top 200 eigenvectors of the

[5]The architecture is provided here: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

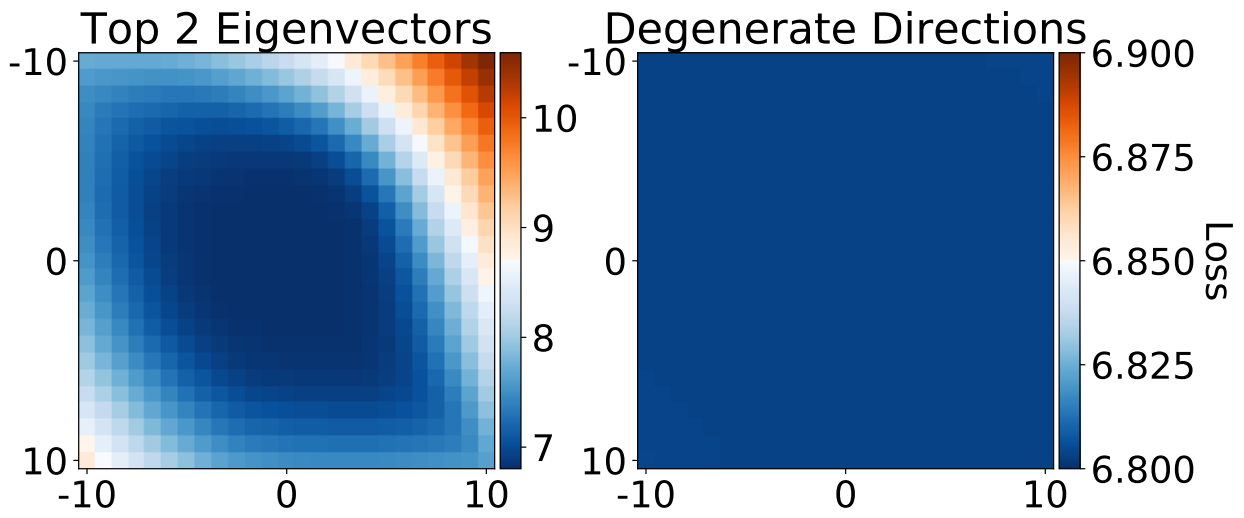


Figure A.23: Left: A visualization of the log-loss surface taken in the direction of the top two eigenvectors of the Hessian of the loss. **Right:** A visualization of a random projection of the log-loss surface in all parameter directions *except* the top 200 eigenvectors of the Hessian. We can see that in nearly all directions the loss is constant even as we move far from the optimal parameters. **Note** the scale difference, even as we increase the resolution of the degenerate loss surface we still see no structure.

Hessian. We see that even for larger networks and more complex datasets degenerate directions in parameter space are still present and comprise most possible directions.

Figure A.24 demonstrates that the degenerate directions in parameter space lead to models that are homogeneous in function space on both training and testing data. As increasingly large perturbations are made in degenerate parameter directions, we still classify more than 99% of both training and testing points the same as the unperturbed classifier.

A.5.6 More Classifiers

Figures A.25 and A.26 provide more examples of perturbations in high and low curvature directions and the effect of the scale of the perturbation on function-space predictions for the two-spirals experiment in subsection 4.5.

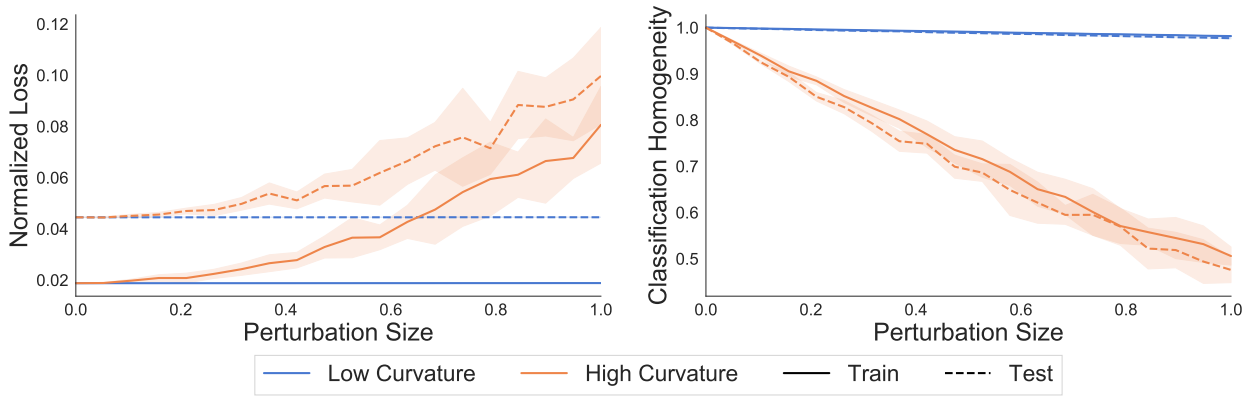


Figure A.24: Left: Loss, normalized by dataset size, on both train and test sets as perturbations are made in high curvature directions and degenerate directions. **Right:** Classification homogeneity, the fraction of data points classified the same as the unperturbed model, as perturbations are made in both high curvature and degenerate directions.

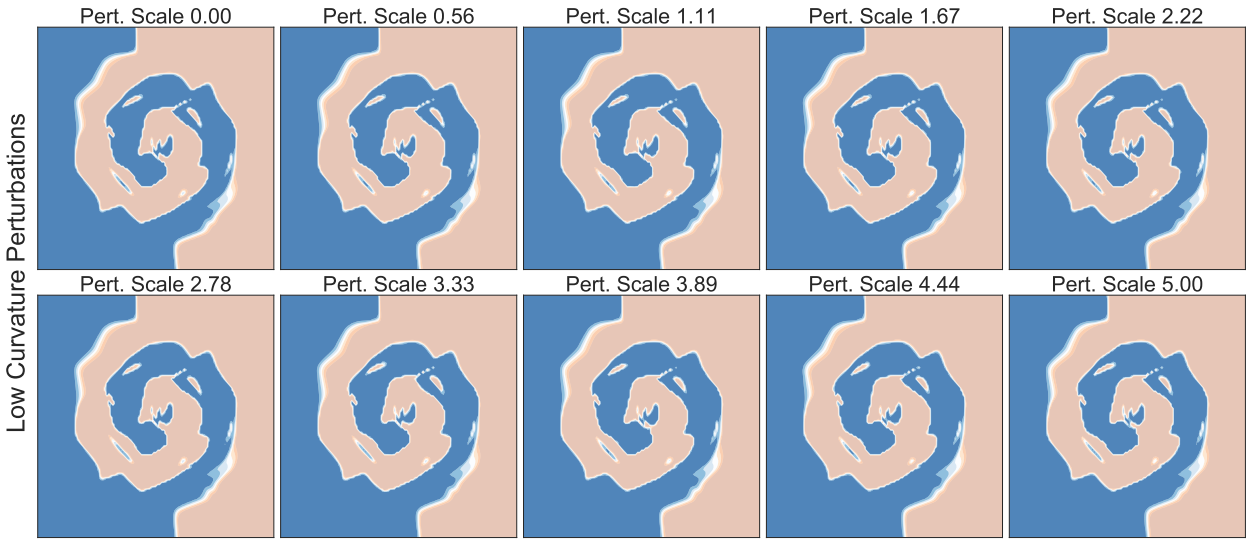


Figure A.25: Classifiers as the parameters are shifted in random directions within the span of the bottom 1500 eigenvectors of the Hessian of the loss. Scales of the perturbation range from 0 (upper left) to 2 (lower right).

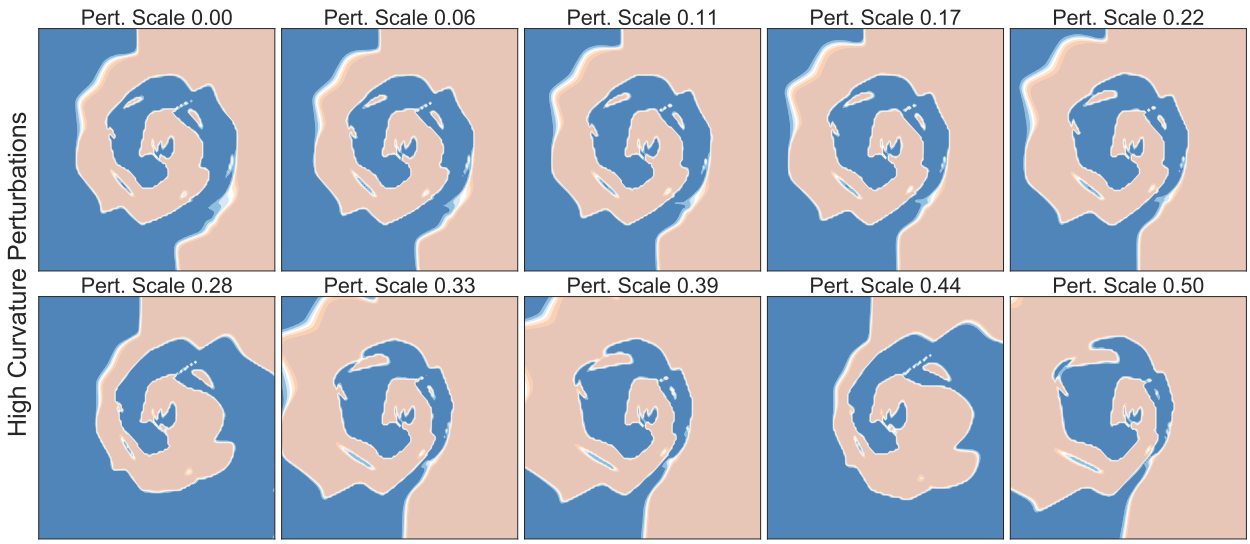


Figure A.26: Classifiers as the parameters are shifted in random directions within the span of the top 3 eigenvectors of the Hessian of the loss. Scales of the perturbation range from 0 (upper left) to 0.5 (lower right).

A.6 Appendix for Loss Surface Simplexes for Mode Connecting Volumes and Fast Ensembling

Outline

The Appendix is outlined as follows:

In Appendix A.6.2, we give a more detailed description of our methods, focusing first on computing the simplex volume and sampling from the simplexes, then describe vertex initialization and regularization, giving training details, and finally describing the training procedure for multi-dimensional mode connectors.

In Appendix A.6.2.1, we describe several more results on volume and ensembling, particularly on the number of samples required for good performance with SPRO and ESPRO.

Finally, in Appendix A.6.5, we plot the results of a larger suite of corruptions on CIFAR-10 for ESPRO, deep ensembles, and MultiSWAG.

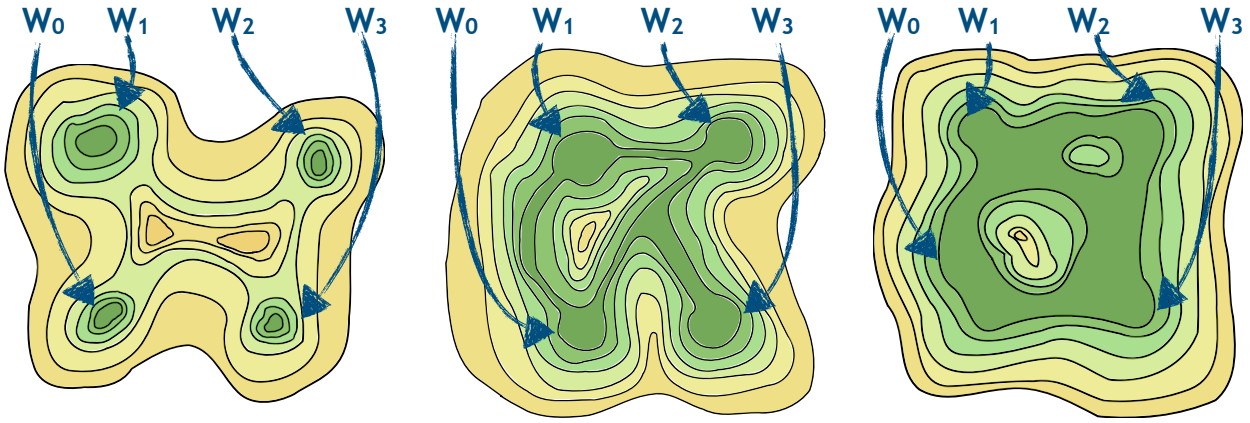


Figure A.27: A simplified version of the progressive understanding of the loss landscape of neural networks. **Left:** The traditional view in which low loss modes are disconnected in parameter space. **Center:** The updated understanding provided by works such as Draxler et al. [2018], Fort and Jastrzebski [2019], and Garipov et al. [2018], in which modes are connected along thin paths or tunnels. **Right:** The view we present in this work: independently trained models converge to points on the same *volume* of low loss.

A.6.1 Extended Methodology

First, we present a two dimensional version of the schematic in Figure 4.6 in A.27, which explains the same progressive illustration, but in two dimensions.

A.6.2 Simplex Volume and Sampling

We employ simplexes in the loss surface for two reasons primarily:

- sampling uniformly from within a simplex is straightforward, meaning we can estimate the expected loss within any found simplexes easily,
- computing the Volume of a simplex is efficient, allowing for regularization encouraging high-Volume simplexes.

Sampling from Simplexes: Sampling from the standard simplex is just a specific case of sampling from a Dirichlet distribution with concentration parameters all equal to 1. The standard n-simplex is a simplex is a simplex formed by the vectors $\mathbf{v}_0, \ldots, \mathbf{v}_n$ such that the $\mathbf{v}_i$'s are the

standard unit vectors. Therefore, to draw samples from a standard n-simplex in a d dimensional space with vertices $\mathbf{v}_0, \ldots, \mathbf{v}_n$, we follow the same procedure to sample from a Dirichlet distribution.

To sample vector $\mathbf{x} = [x_0, \ldots, x_d]^T$ we first draw $y_0, \ldots, y_n \overset{\text{i.i.d.}}{\sim} \text{Exp}(1)$, then set $\tilde{y}_i = \frac{y_i}{\sum_{j=1}^{d} y_j}$. Finally, $\mathbf{x} = \sum_{i=1}^{n} \tilde{y}_i \mathbf{v}_i$.

While this method is sufficient for simulating vectors uniformly at random from the *standard* simplex, there is no guarantee that such a sampling method produces uniform samples from an arbitrary simplex, and thus samples of the loss over the simplex that we use in Equation 4.6 may not be an unbiased estimate of the expected loss over the simplex. Practically, we do not find this to be an issue, and are still able to recover low loss simplexes with this approach.

Furthermore, Figure A.28 shows that the distribution of samples in a unit simplex is visually similar to the samples from an elongated simplex where we multiply one of the basis vectors by a factor of 100. This figure serves to show that although there may be some bias in our estimate of the loss over the simplex in Equation 4.6, it should not be (and is not in practice) limiting to our optimization routine. Note too, this may appear like a simplistic case, but typically the simplexes found by SPRO contain only a small number of vertices, so a 2-simplex whose edge lengths vary by a factor of nearly 100 is a reasonable comparison to a scenario we may find in practice.

Computing Simplex Volume: Simplex Volumes can be easily computed using Cayley-Menger determinants [Colins 2021]. If we have an n-simplex defined by the parameter vectors $w_0, \ldots, w_n$

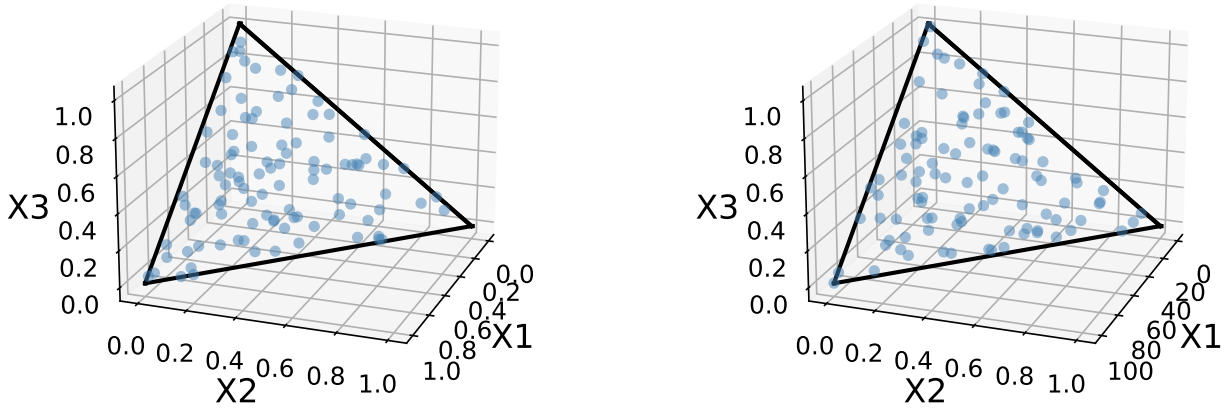


Figure A.28: Left: 100 samples drawn uniformly from within the unit simplex. **Right:** 100 samples drawn from a non-unit simplex (note the scale of the $X1$ axis). The distribution of points in both simplexes is visually indistinguishable — evidence that the method for sampling from a unit simplex is sufficient to draw samples from arbitrary simplexes.

the Cayley-Menger determinant is defined as

$$CM(w_0, \ldots, w_n) = \begin{vmatrix} 0 & d_{01}^2 & \cdots & d_{0n}^2 & 1 \\ d_{01}^2 & 0 & \cdots & d_{0n}^2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ d_{n0}^2 & d_{n1}^2 & \cdots & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{vmatrix}. \tag{A.20}$$

The Volume of the simplex $S_{(w_0,\ldots,w_n)}$ is then given as

$$\mathrm{V}(S_{w_0,\ldots,w_n)})^2 = \frac{(-1)^{n+1}}{(n!)^2 2^n} CM(w_0, \ldots, w_n). \tag{A.21}$$

While in general we may be adverse to computing determinants or factorial terms the simplexes we work with in this paper are generally low order (all are under 10 vertices total) meaning that computing the Cayley-Menger determinants is generally a quite fast and stable computation.

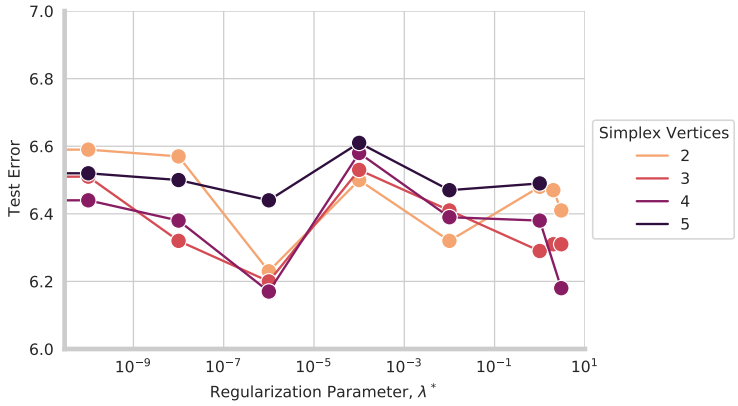


Figure A.29: CIFAR-10 test accuracy as a function of regularization parameter λ^* and colored by the number of vertices. Accuracy is essentially unchanged for the various regularization parameters.

A.6.2.1 Initialization and Regularization

Vertex Initialization: We initialize the j^{th} parameter vector corresponding to a vertex in the simplex as the mean of the previously found vertices, $w_j = \frac{1}{j}\sum_{i=0}^{j-1} w_i$ and train using the regularized loss in Eq. 4.6.

Regularization Parameter: As the order of the simplex increases, the Volume of the simplex increases exponentially. Thus, we define a distinct regularization parameter, λ_j, in training each θ_j to provide consistent regularization for all vertices. To choose the λ_k's we define a λ^* and compute

$$\lambda_k = \frac{\lambda^*}{\log \mathrm{V}(\mathcal{K})}, \tag{A.22}$$

where $\mathcal{K}$ is randomly initialized simplicial complex of the same structure that the simplicial complex will have while training θ_j. Eq. A.22 normalizes the λ_k's such that they are similar when accounting for the exponential growth in volume as the order of the simplex grows. In practice we need only small amounts of regularization, and choose $\lambda^* = 10^{-8}$. As we are spanning a space of near constant loss any level of regularization will encourage finding simplexes with non-trivial Volume.

Finally, when dealing with models that use batch normalization, we follow the procedure of Garipov et al. [2018] and compute several forwards passes on the training data for a given sample from the simplex to update the batch normalization statistics. For layer normalization, we do not need to use this procedure as layer norm updates at test time.

A.6.2.2 Training Details

We used VGG-16 like networks originally introduced in Simonyan and Zisserman [2015] from https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py. For training, we used standard normalization, random horizontal flips, and crops and a batch size of 128. We used SGD with momentum = 0.9, and a cosine annealing learning rate with a single cycle, a learning rate of 0.05, and weight decay $5e-4$ training for 300 epochs for the pre-trained VGG models. For SPRO, we used a learning rate of 0.01 and trained for 20 epochs for each connector.

In our experiments with transformers, we used the *ViT-B_16* image transformer model [Dosovitskiy et al. 2021] pre-trained on ImageNet from https://github.com/jeonsworld/ViT-pytorch and trained on CIFAR100 with upsampled image size of 224 with a batch size of 512 for 50000 steps (the default fine-tuning on CIFAR-100). Again, we used random flips and crops for data augmentation. To train these SPRO models, we used a learning rate of 0.001 and trained with SGD for 30 epochs for each connector, using 20 samples from the simplex at test time.

A.6.3 Multi-Dimensional Mode Connectors

To train the multi-dimensional SWAG connectors, we connected two pre-trained networks following Garipov et al. [2018] using a piece-wise linear curve, trained for 75 epochs with an initial learning rate of 0.01, decaying the learning rate to $1e-4$ by epoch 40. At epoch 40, we reset the learning rate to be constant at $5e-3$. The final individual sample accuracy (not SWA) was 91.76%, which is similar to the final individual sample accuracies for standard training of VGG networks with SWAG. We used random crops and flips for data augmentation.

A.6.4 Extended Volume and Ensembling Results

A.6.4.1 Volumes on MNIST

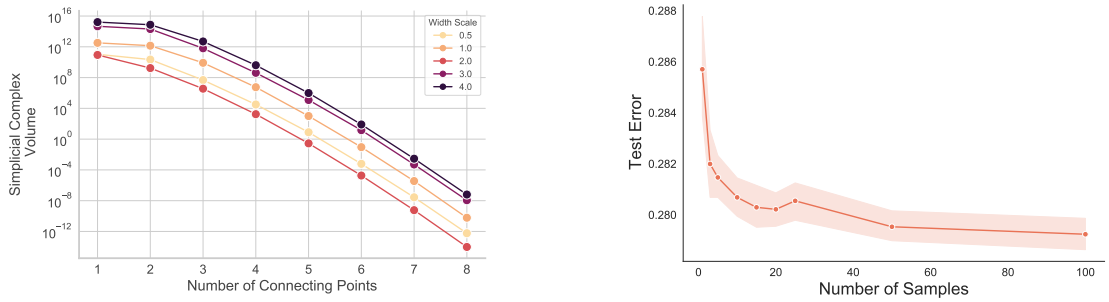
In a similar construction to the dimensionality experiment in Figure 4.10, we next consider lower bounding the dimensionality of the connecting space that SPRO can find for LeNet-5s on MNIST [LeCun et al. 1998b][6], varying the width of the convolutional networks from a baseline of 1 (standard parameterization), either halving the width or consecutively widening the layers by a constant factor. We find in Figure A.30(a) that the volumes of the simplicial complex can vary by several powers of 10 for the as we increase the widths. However, all width networks generally follow the same patter of decaying volume as we increase the number of connecting points (e.g. increasing the dimensionality of the simplicial complex).

A.6.4.2 Test Error vs. Simplex Samples

SPRO gives us access to a whole space of model parameters to sample from rather than just a discrete number of models to use as in deep ensembles. Therefore a natural question to ask is how many models and forwards passes need to be sampled from the simplex to achieve the highest accuracy possible without incurring too high of a cost at test time.

Figure A.30(b) shows that for a VGG-16 network trained on CIFAR-100 we achieve near constant accuracy for any number of ensemble components greater than approximately 25. Therefore, for the ensembling experiments in Section 4.10.4 we use 25 samples from each simplex to generate the SPRO ensembles. In this work we are not focused on the issue of test time compute cost, and if that were a consideration for deployment of a SPRO model we could evaluate the trade-off in terms of test time compute vs accuracy, or employ more sophisticated methods such as ensemble distillation.

[6]Implementation from https://github.com/activatedgeek/LeNet-5/blob/master/lenet.py



(a) Log volumes, MNIST.

(b) Test error vs. number of samples, CIFAR-100

Figure A.30: (a) Log volumes as a function of LeNet-5 layer width. Volumes are generally highest for wider models, and the volume of the simplicial complex tends to decrease as the dimension of the space increases. **(b)**Test error vs. number of samples, J, in the ensemble on CIFAR-100 using a VGG-16 network and a 3-simplex trained with SPRO. For any number of components in the SPRO ensemble greater than approximately 25 we achieve near constant test error.

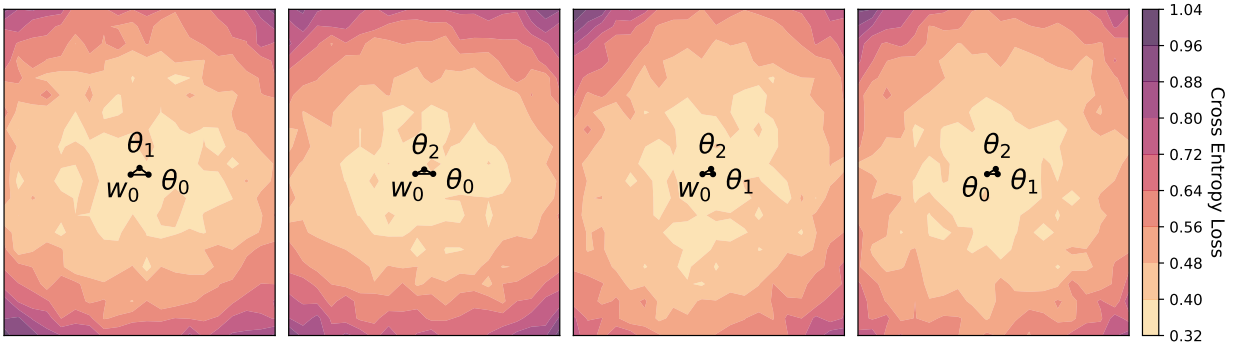


Figure A.31: Loss surface visualizations of the faces of a sample ESPRO 3-simplex for a Transformer architecture [Dosovitskiy et al. 2021] fine-tuned on CIFAR-100. Here, the volume is considerably smaller, but a low loss region is found.

A.6.4.3 Loss Surfaces of Transformers

Next, we show the results of training a SPRO 3−simplex with an image transformer on CIFAR-100 [Dosovitskiy et al. 2021] in Figure A.31. Due to computational requirements, the transformer was pre-trained on ImageNet before being fine-tuned on CIFAR-100 for 50, 000 minibatches. We then trained each vertex for an additional 10 epochs. Due to the inflexibility of the architecture, we observed training instability, which ultimately produced a small volume of the simplex found (approximately 10^{-21}). Furthermore, the small volume of the simplex produced less diverse solu-

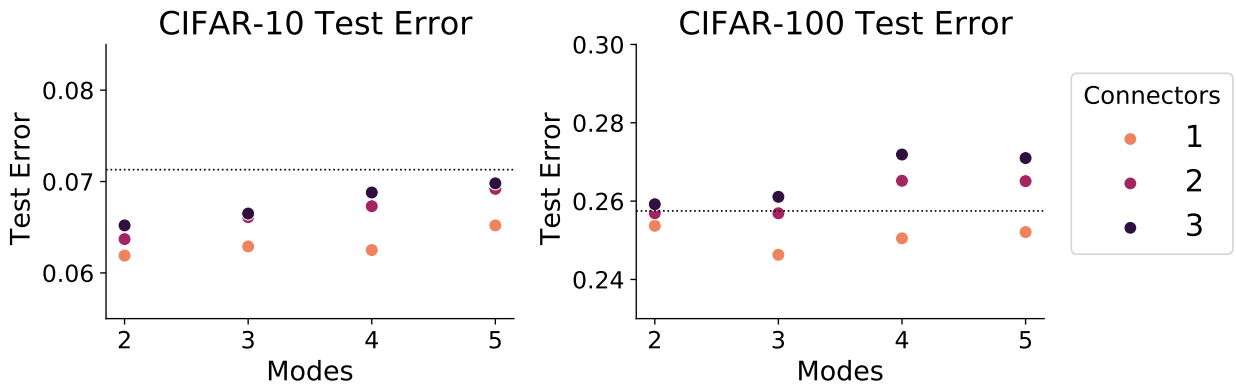


Figure A.32: Test error for mode connecting simplexes that connect various numbers of modes through various numbers of connecting points in the parameter space of VGG-16 networks trained on CIFAR-10 and CIFAR-100. The error rates of baseline models are shown as horizontal dotted lines. In general the highest performing models are those with the fewest modes and the fewest connecting points, but the performance gaps between configurations are small.

tions, limiting the benefits of ensembling transformer models as shown in Figure A.34. However, these results demonstrate that a region of low loss can be found in subspaces of transformer models, and further work will be necessary to efficiently exploit these regions of low loss, much like has been done with CNNs and ResNets.

A.6.4.4 Ensembling Mode Connecting Simplexes

We can average predictions over these mode connecting volumes, generating predictions as ensembles, $\hat{y} = \frac{1}{H}\sum_{\phi_h \sim \mathcal{K}} f(x, \phi_h)$, where $\phi_h \sim \mathcal{K}$ indicates we are sampling models uniformly at random from the simplicial complex $\mathcal{K}(S_{(w_0,\theta_0,\dots)}, S_{(w_1,\theta_0,\dots)}, \dots)$. Test error for such ensembles for volumes in the parameter space of VGG-16 style networks on both CIFAR-10 and CIFAR-100 are given in Figure A.32. We see that while some improvements over baselines can be made, mode connecting simplexes do not lead to highly performant ensembles.

A.6.4.5 Ensembling Modes of SPRO

Figure A.33 presents the results of Figure 4.13 in the main text, but against the total training budget rather than the number of ensemble components. We see from the plot that on either

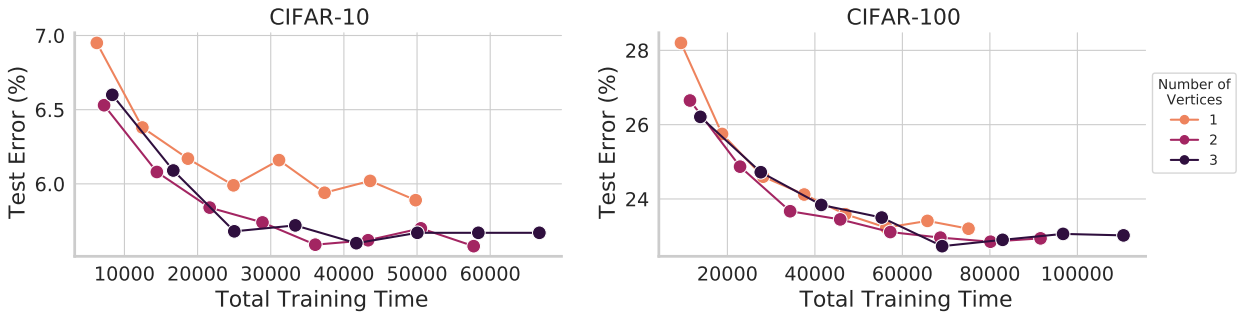


Figure A.33: Test error of ESPRO models on CIFAR-10 (left) and CIFAR-100 (right) as a function of total training time (training the original models and the ESPRO simplexes). The color of the curves indicate the number of the vertices in the simplex, and the points corresponding to increasing numbers of ensemble components moving left to right (ranging from 1 to 8). We see that on either dataset for nearly any fixed training budget, we are better off training fewer models overall and using ESPRO to construct simplexes to sample from.

dataset, for nearly any fixed training budget the most accurate model is an ESPRO model, even if that means using our budget to train ESPRO simplexes but fewer models overall.

Times correspond to training models sequentially on an NVIDIA Titan RTX with $24GB$ of memory.

Finally, Figure A.34 presents the results of ensembling with SPRO using state of the art transformers architectures on CIFAR-100 [Dosovitskiy et al. 2021]. We find, counterintuitively that there is only a very small performance difference from ensembling with SPRO compared to the base architecture. We suspect that this is because it is currently quite difficult to train transformers without using significant amounts of unlabelled data.

A.6.5 Extended Uncertainty Results

A.6.5.1 Further NLL and Calibration Results

Finally, we include the results across 18 different corruptions for the ensemble components. In order, these are *jpeg*, fog, snow, brightness, pixelate, zoom blur, saturate, contrast, motion blur, defocus blur, speckle noise, gaussian blur, glass blur, shot noise, frost, spatter, impulse noise and,

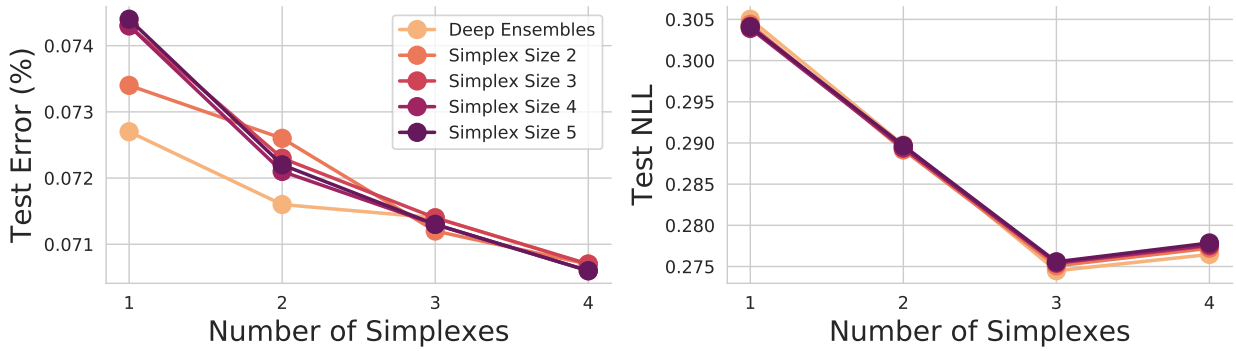


Figure A.34: Test Error and NLL for the number of components in SPRO ensembles using image transformers on CIFAR-100. ESPRO with four dimensional simplexes is slightly better in test accuracy and slightly worse in test NLL than deep ensembles.

elastic transform.

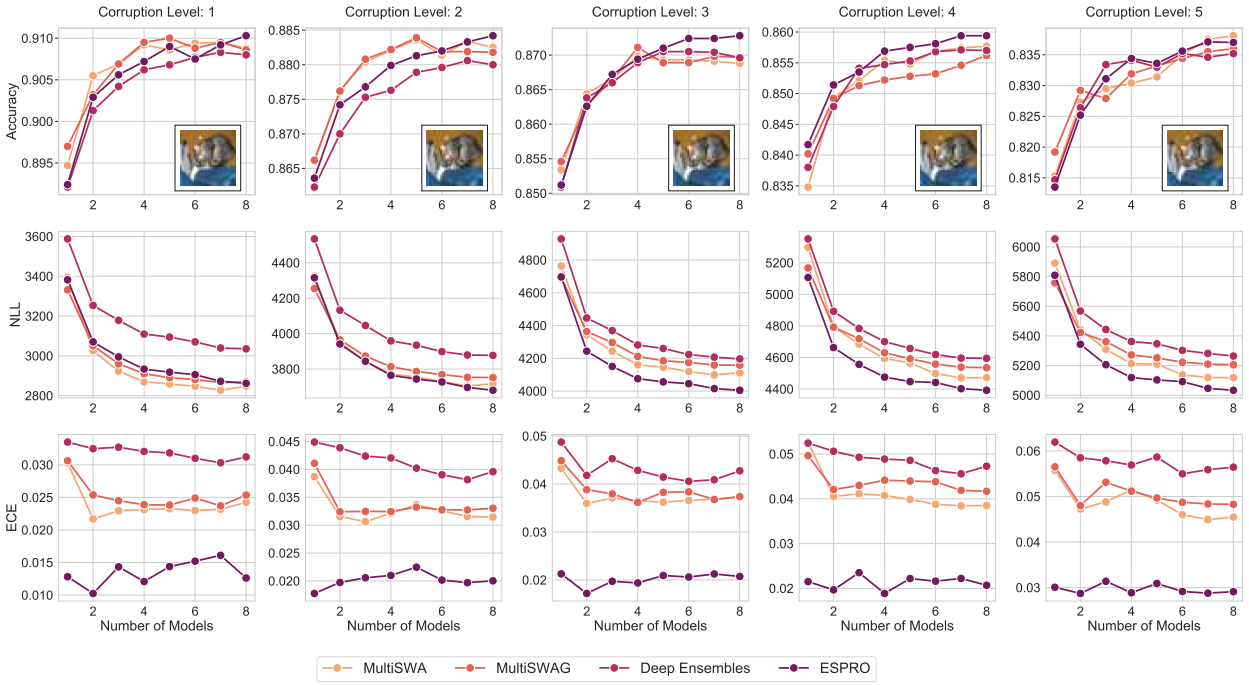


Figure A.35: Accuracy, NLL and ECE with increasing intensity of the *jpeg compression* corruption (from left to right).

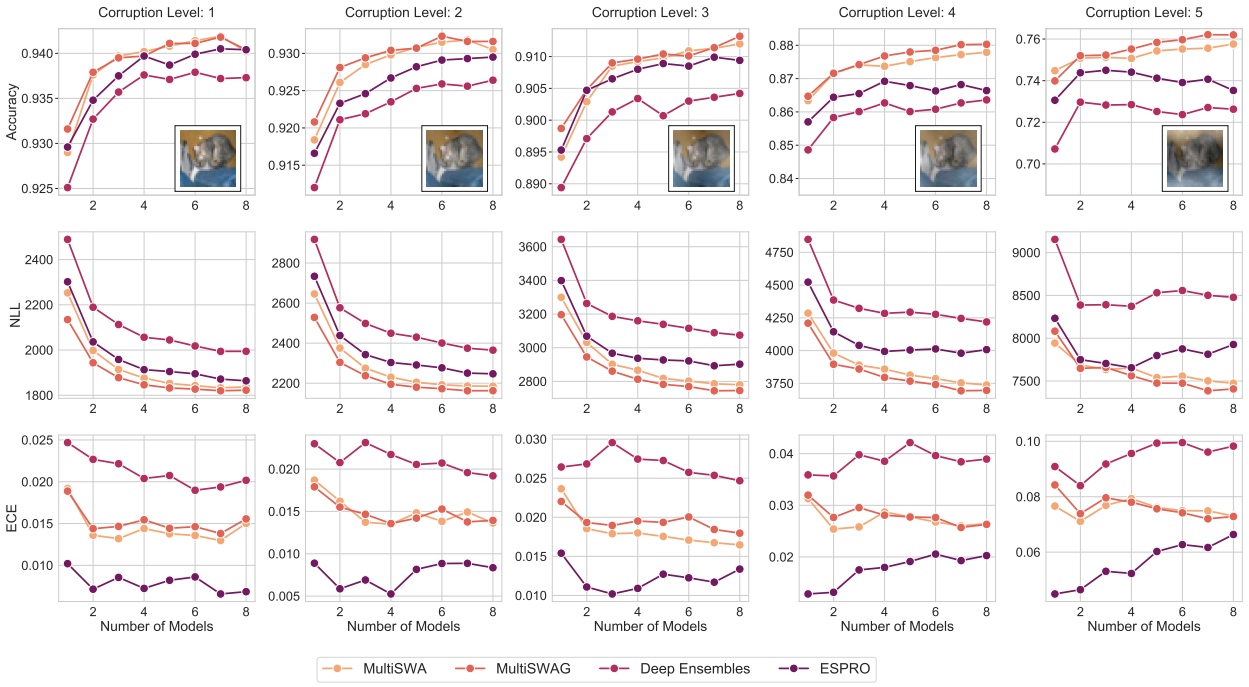


Figure A.36: Accuracy, NLL and ECE with increasing intensity of the *fog* corruption (from left to right).

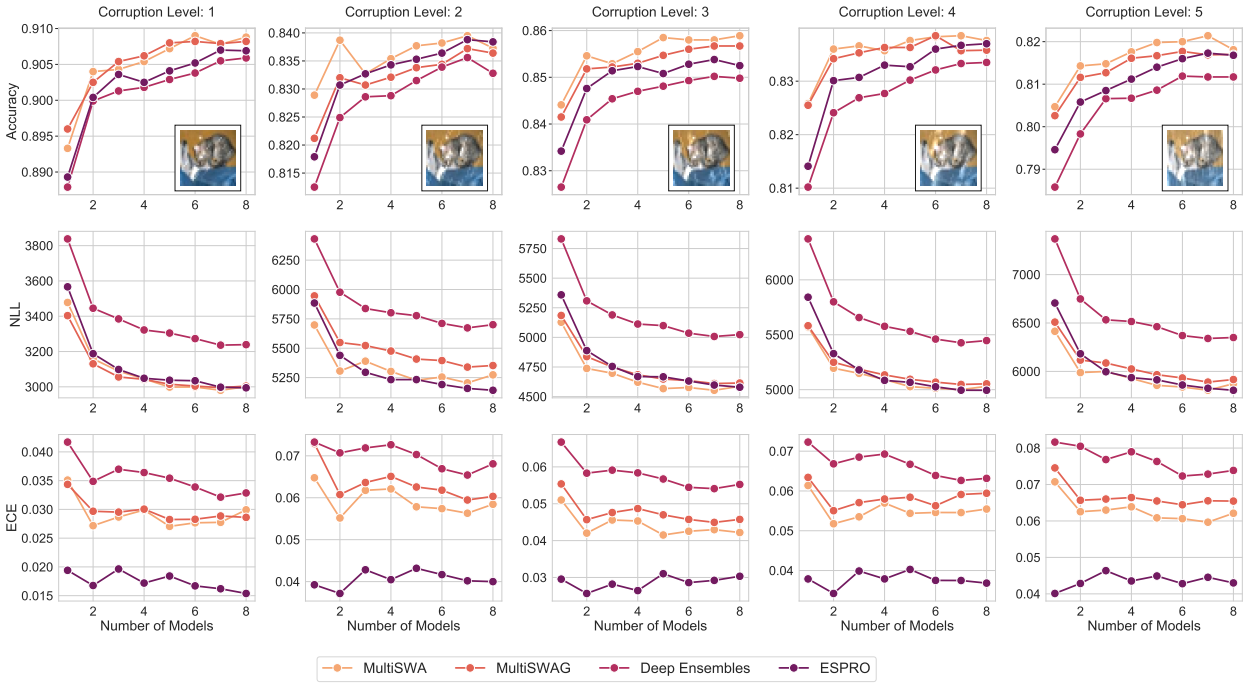


Figure A.37: Accuracy, NLL and ECE with increasing intensity of the *snow* corruption (from left to right).

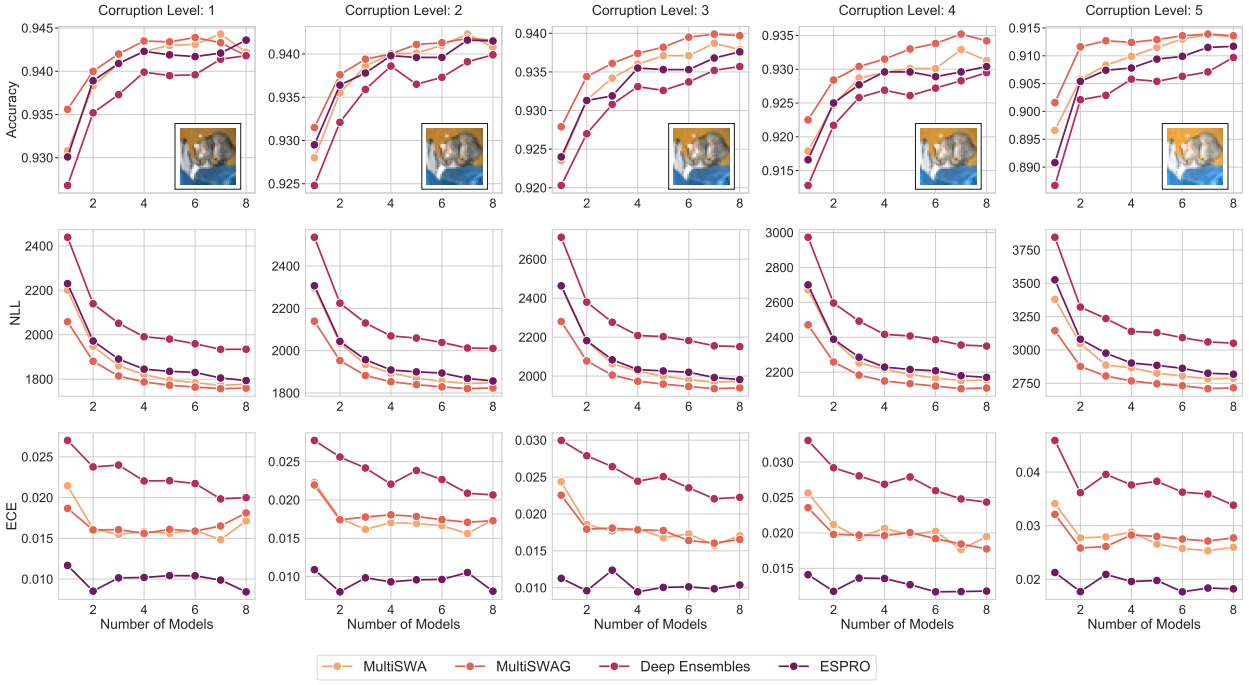


Figure A.38: Accuracy, NLL and ECE with increasing intensity of the *brightness* corruption (from left to right).

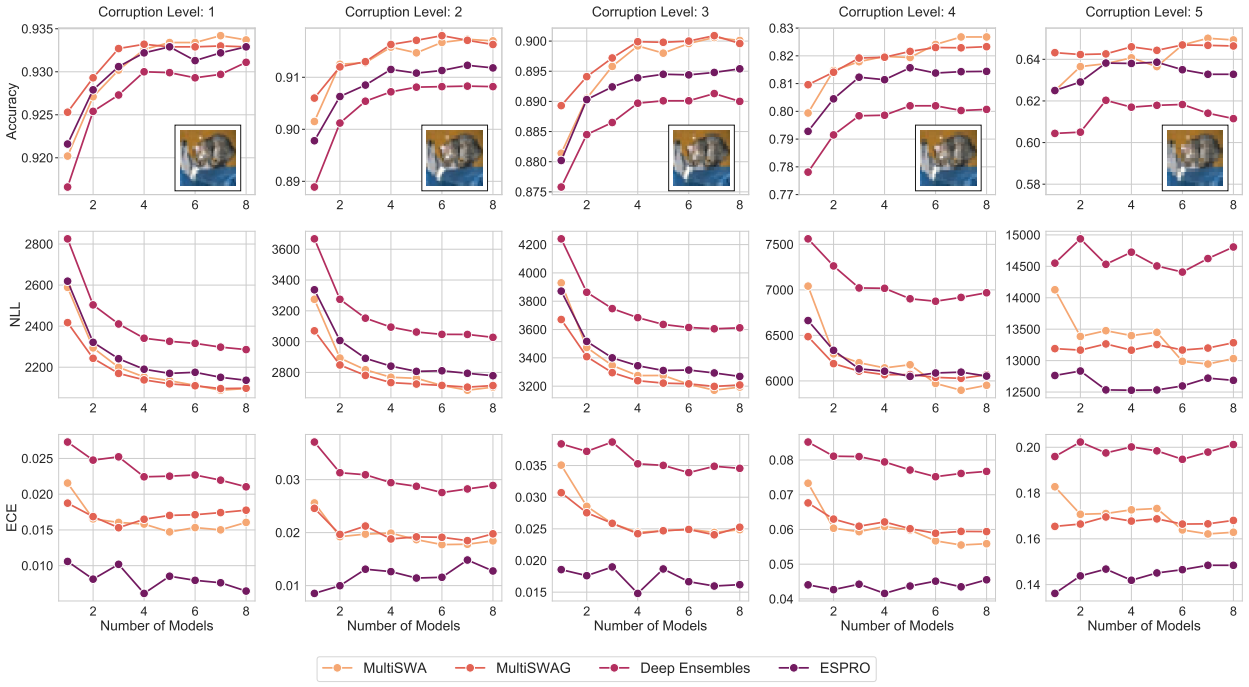


Figure A.39: Accuracy, NLL and ECE with increasing intensity of the *pixelate* corruption (from left to right).

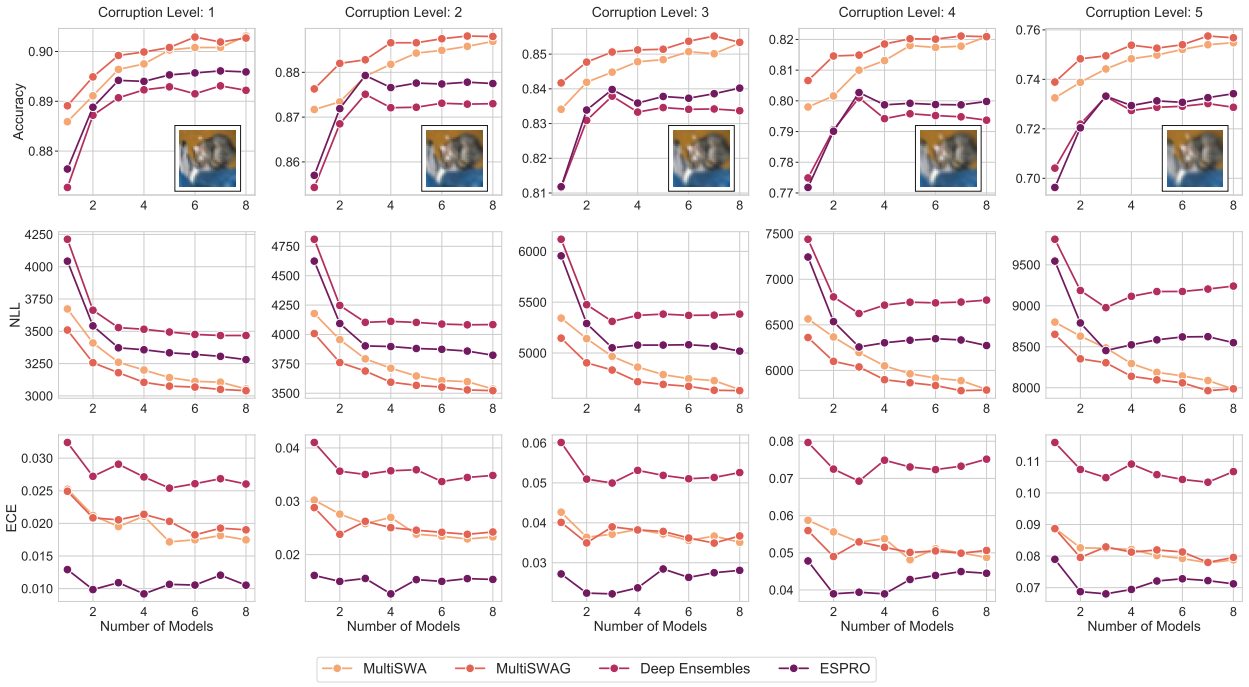


Figure A.40: Accuracy, NLL and ECE with increasing intensity of the *zoom blur* corruption (from left to right).

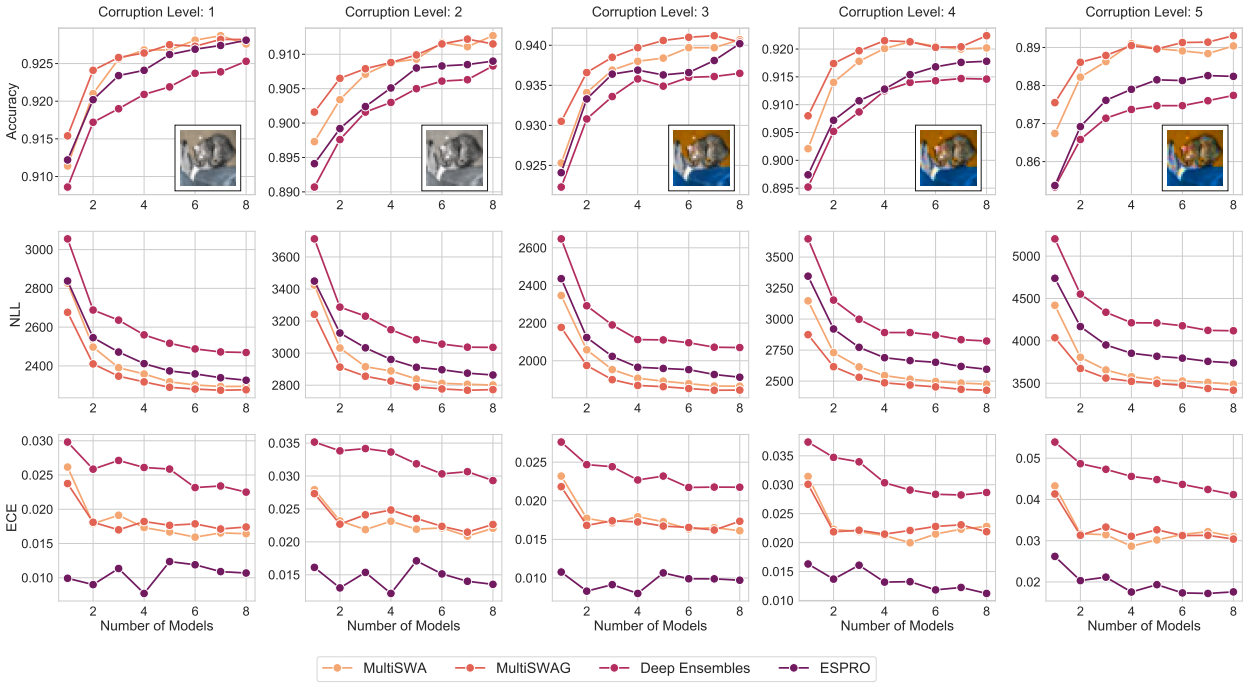


Figure A.41: Accuracy, NLL and ECE with increasing intensity of the *saturate* corruption (from left to right).

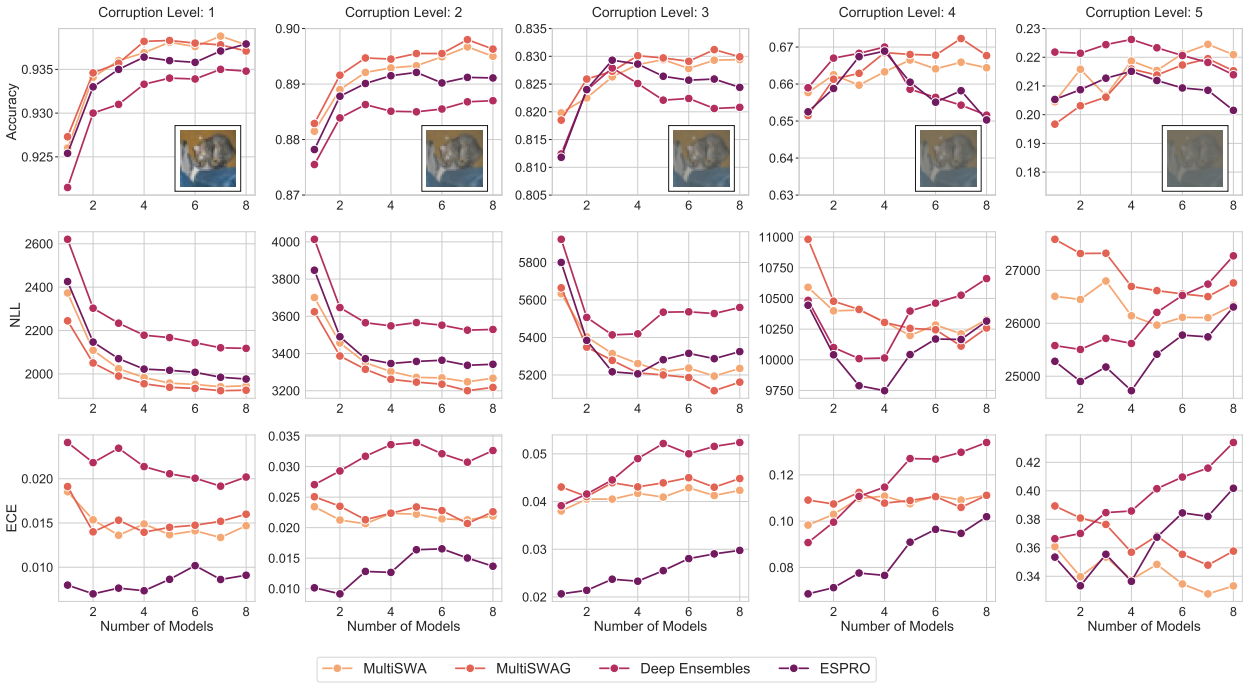


Figure A.42: Accuracy, NLL and ECE with increasing intensity of the *contrast* corruption (from left to right).

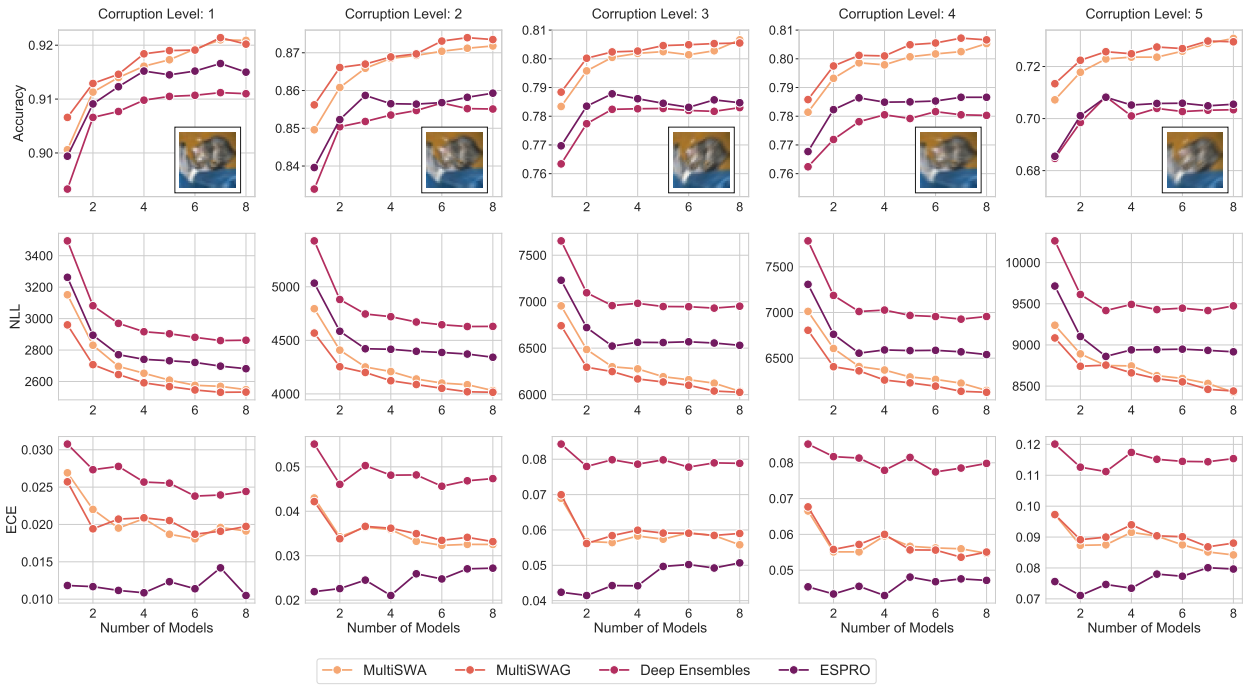


Figure A.43: Accuracy, NLL and ECE with increasing intensity of the *motion blur* corruption (from left to right).

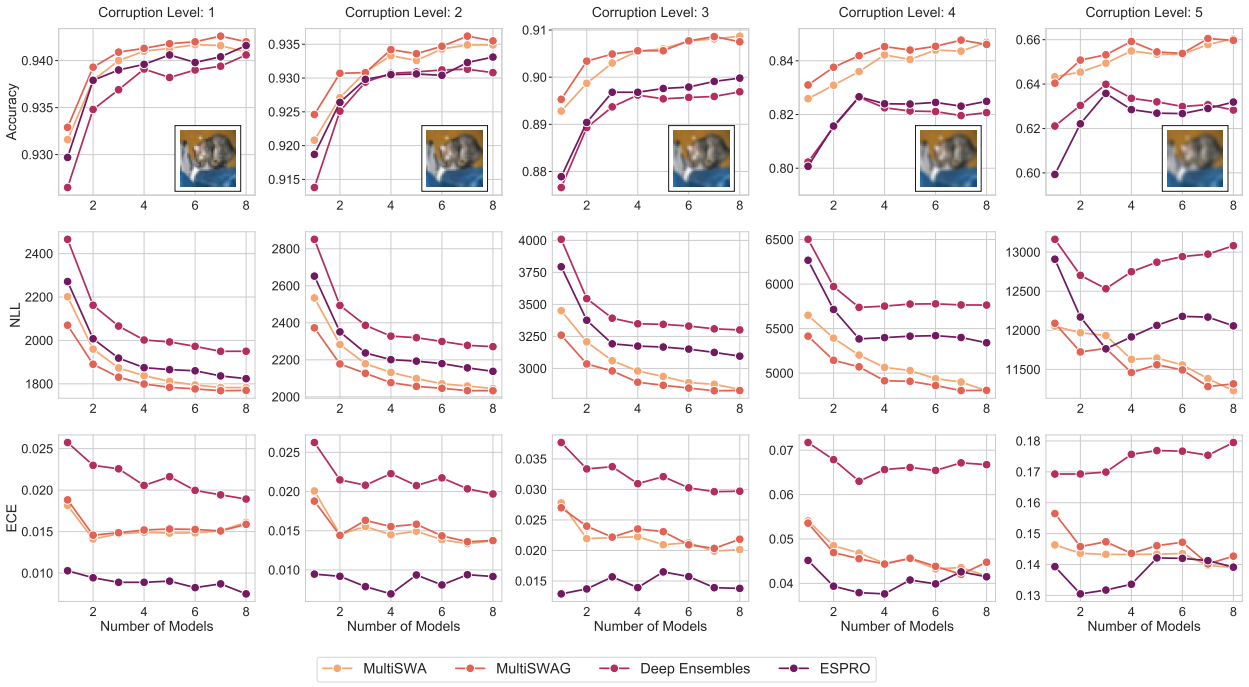


Figure A.44: Accuracy, NLL and ECE with increasing intensity of the *defocus blur* corruption (from left to right).

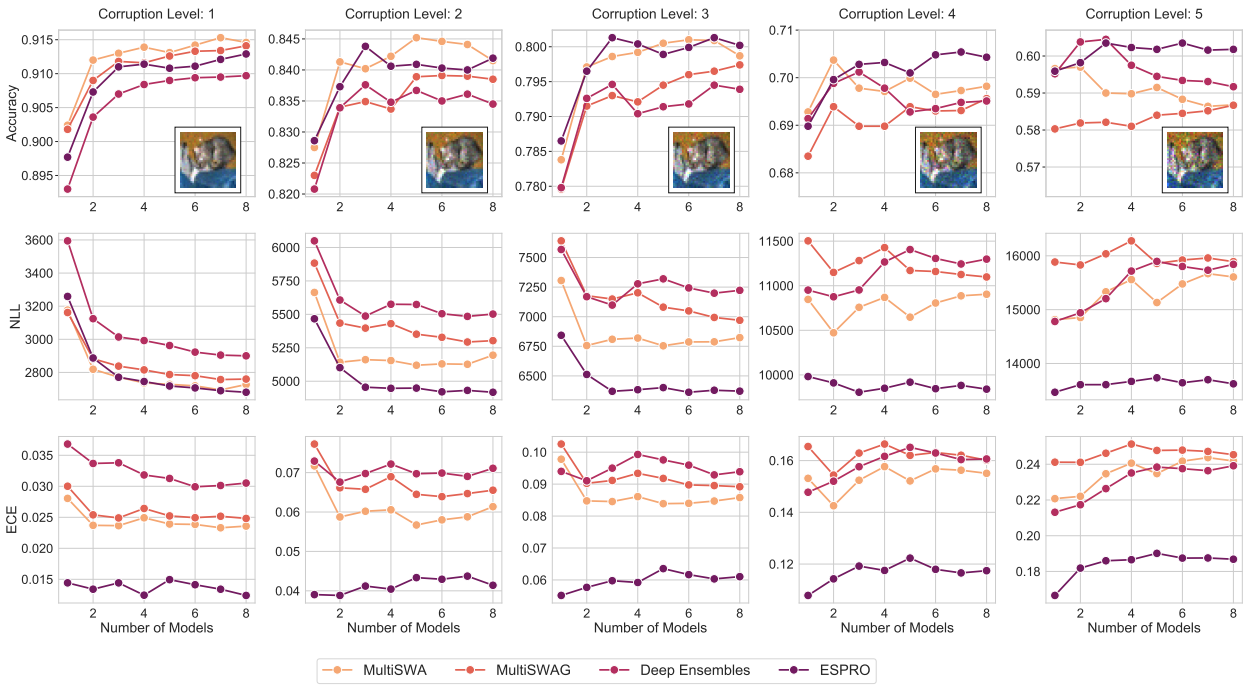


Figure A.45: Accuracy, NLL and ECE with increasing intensity of the *speckle noise* corruption (from left to right).

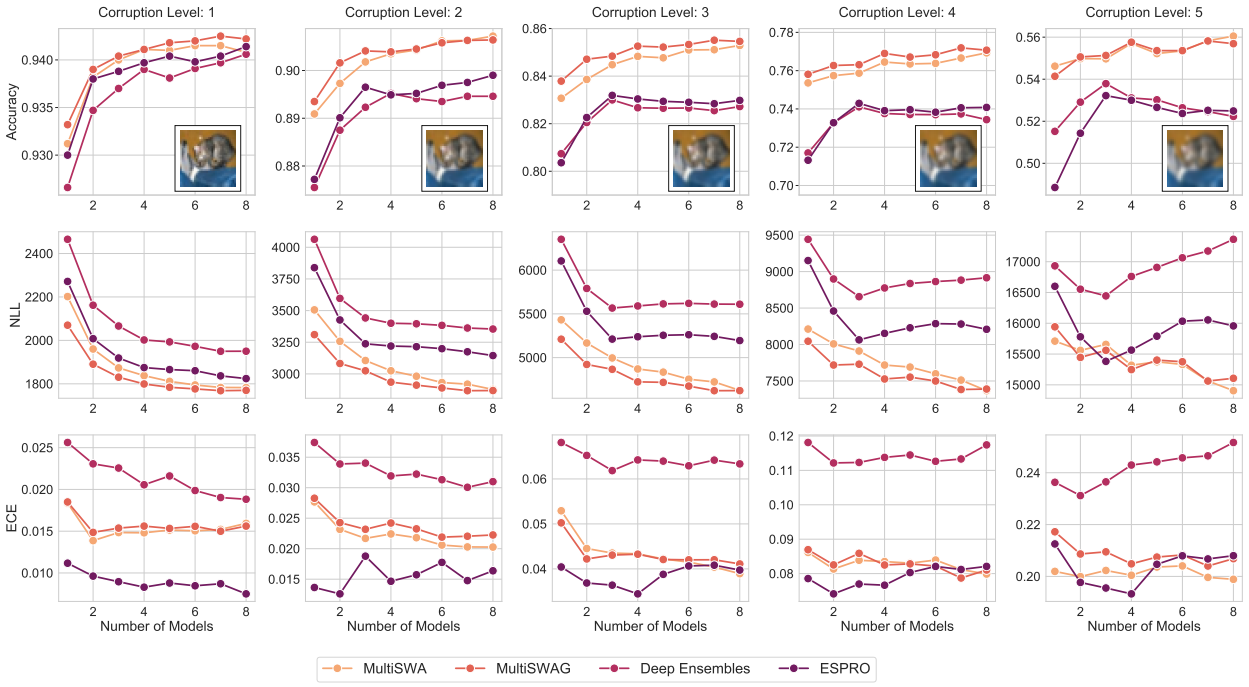


Figure A.46: Accuracy, NLL and ECE with increasing intensity of the *Gaussian blur* corruption (from left to right).

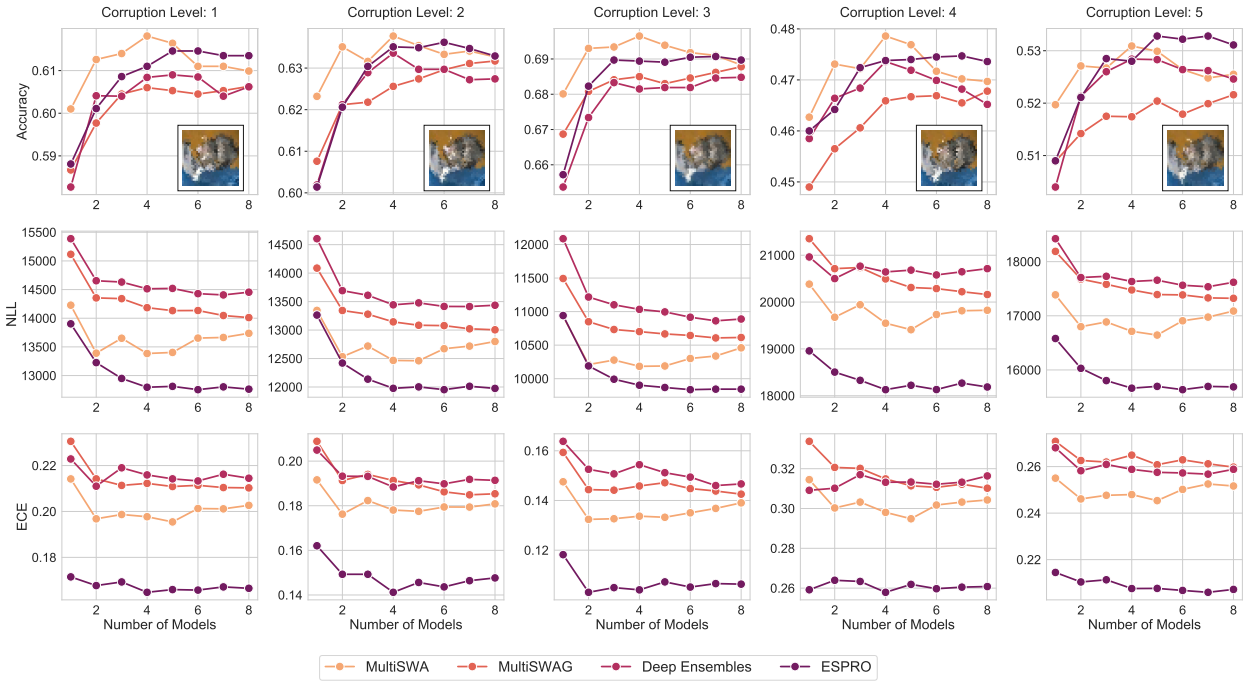


Figure A.47: Accuracy, NLL and ECE with increasing intensity of the *glass blur* corruption (from left to right).

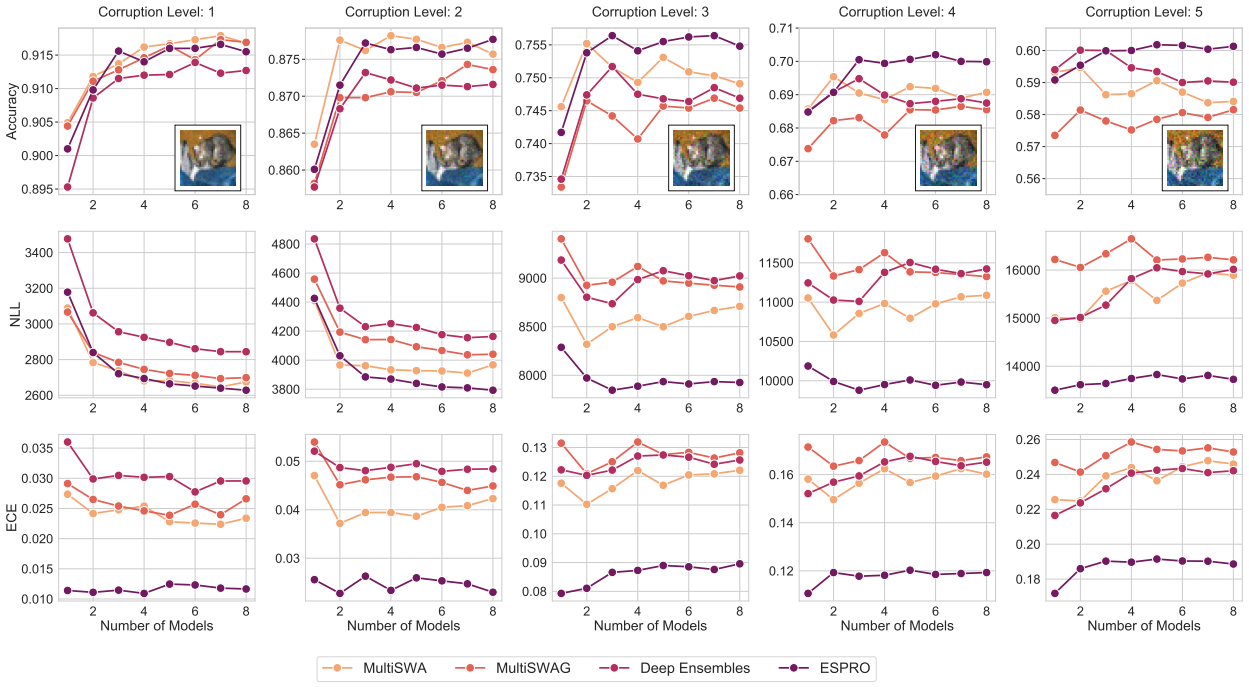


Figure A.48: Accuracy, NLL and ECE with increasing intensity of the *shot noise* corruption (from left to right).

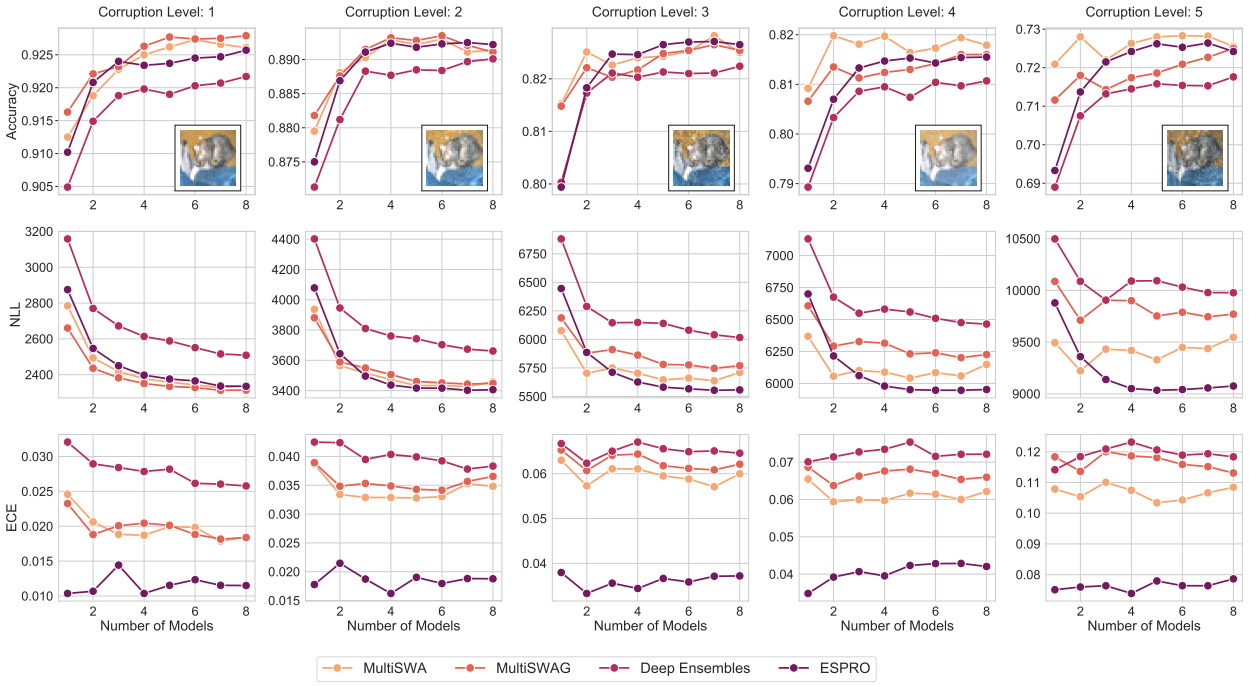


Figure A.49: Accuracy, NLL and ECE with increasing intensity of the *frost* corruption (from left to right).

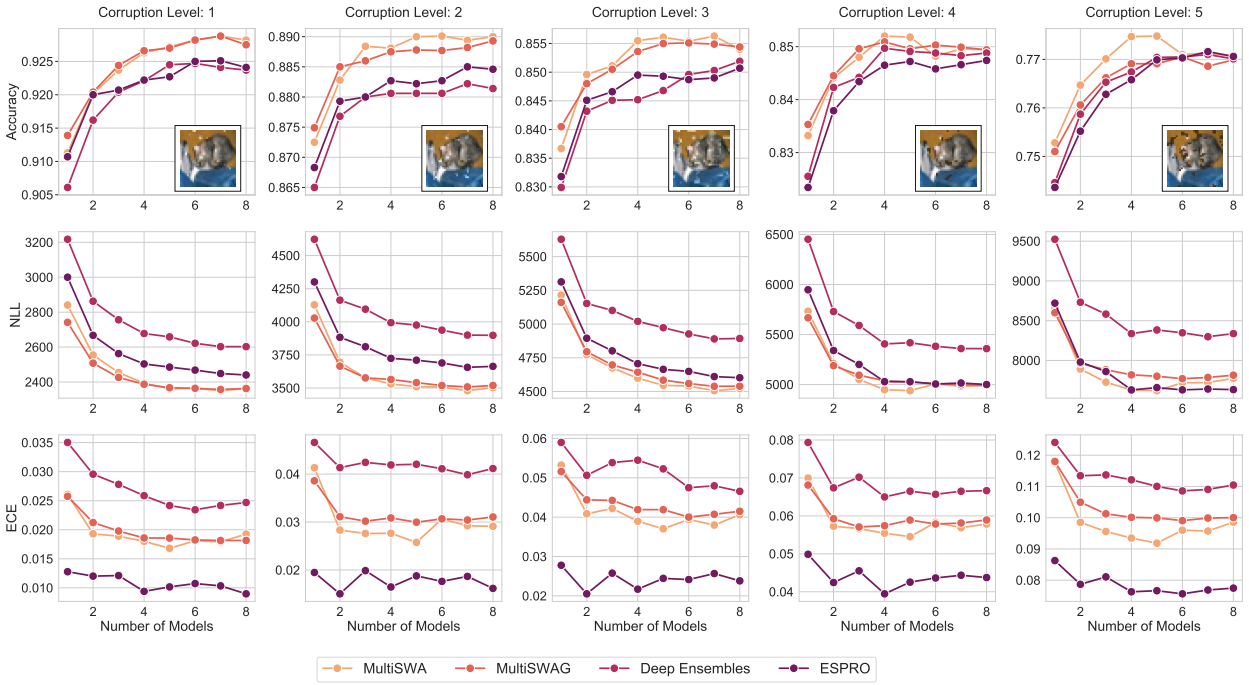


Figure A.50: Accuracy, NLL and ECE with increasing intensity of the *spatter* corruption (from left to right).

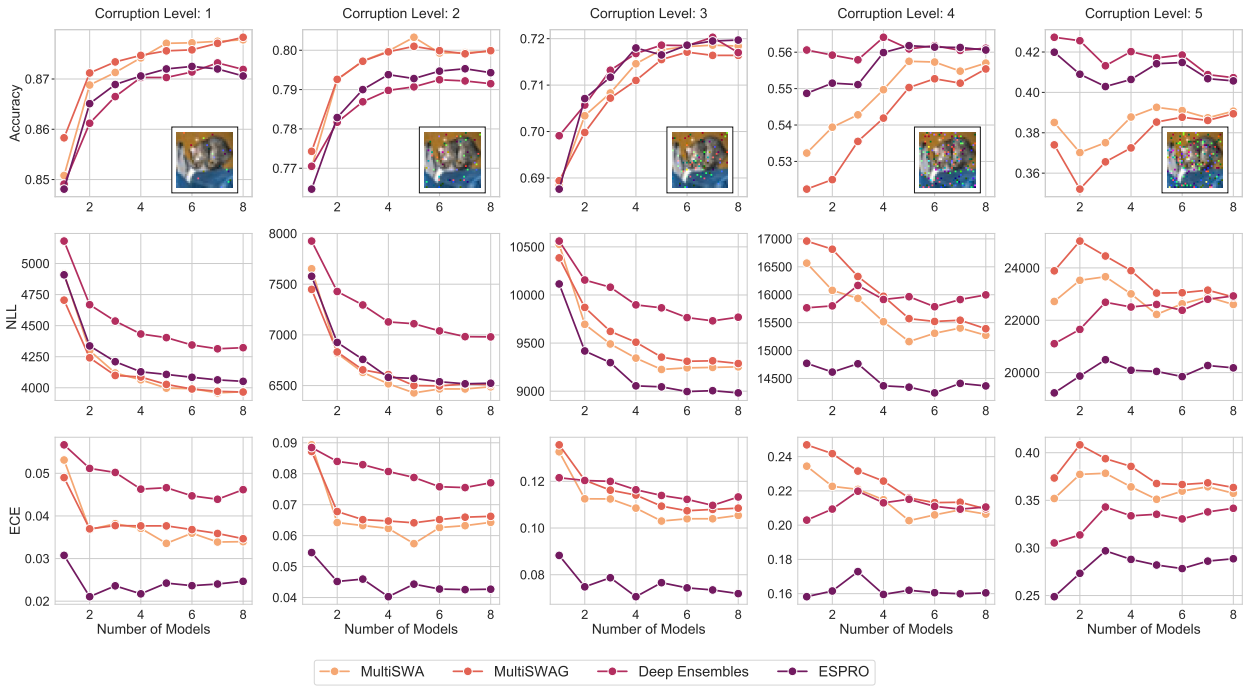


Figure A.51: Accuracy, NLL and ECE with increasing intensity of the *impulse noise* corruption (from left to right).

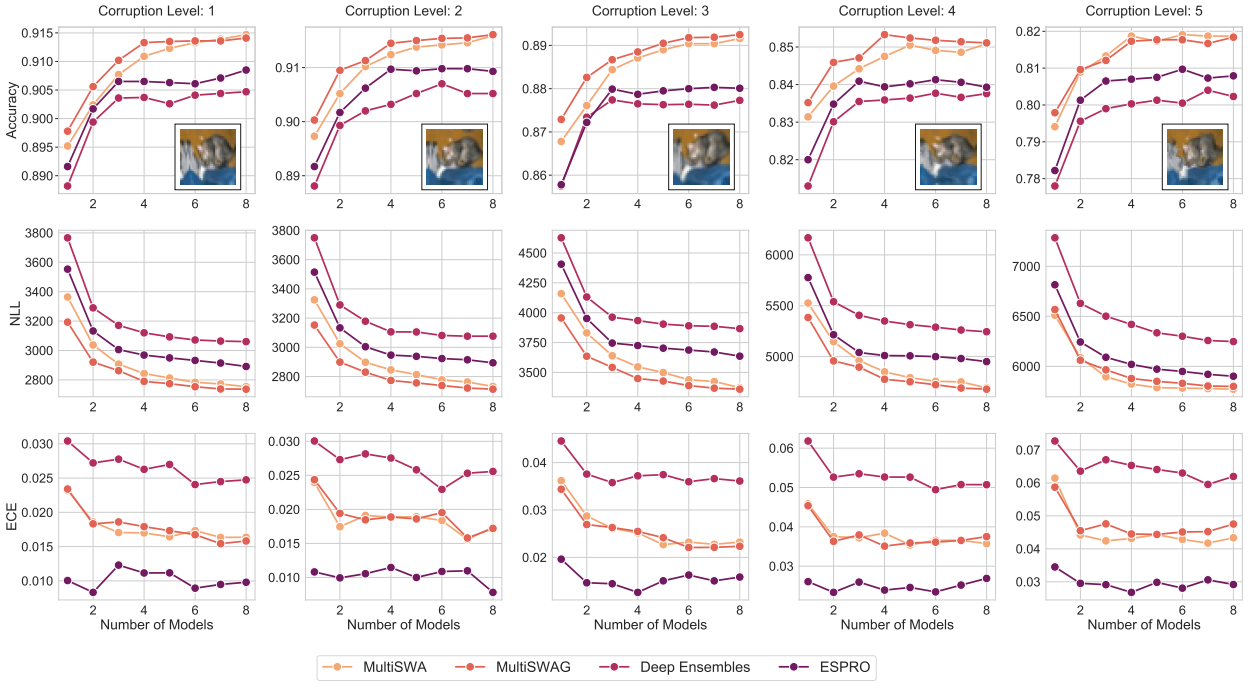


Figure A.52: Accuracy, NLL and ECE with increasing intensity of the *elastic transform* corruption (from left to right).

Table A.12: Ant-v2 State and Action Spaces

State Space	X (Unobserved)
	Y (Unobserved)
	Z
	Orientation Quaternion ($4D$)
	Limb 2 Left/Right
	Limb 2 Up/Down
	Limb 3 Left/Right
	Limb 3 Up/Down
	Limb 4 Left/Right
	Limb 4 Up/Down
	Limb 1 Left/Right
	Limb 1 Up/Down
Action Space	Limb 1 Left/Right
	Limb 1 Up/Down
	Limb 2 Left/Right
	Limb 2 Up/Down
	Limb 3 Left/Right
	Limb 3 Up/Down
	Limb 4 Left/Right
	Limb 4 Up/Down

Table A.13: Humanoid-v2 Action Space

Action Space	Torso Forward/Backward
	Torso Z
	Torso Left/Right
	Right Hip Left/Right
	Right Hip Up/Down
	Right Hip Front/Back
	Right Knee Front/Back
	Left Hip Left/Right
	Left Hip Up/Down
	Left Hip Front/Back
	Left Knee Front/Back
	Right Shoulder Left/Right
	Right Shoulder Front/Back
	Right Elbow Front/Back
	Left Shoulder Left/Right
	Left Shoulder Front/Back
	Left Elbow Front/Back

Table A.14: Humanoid-v2 State Space

State Space (Position)	X (Unobserved)
	Y (Unobserved)
	Z
	Orientation Quaternion ($4D$)
	Torso Z
	Torso Forward/Backward
	Torso Left/Right
	Right Hip Left/Right
	Right Knee Left/Right
	Right Hip Up/Down
	Right Knee Up/Down
	Left Hip Left/Right
	Left Knee Left/Right
	Left Hip Up/Down
	Left Knee Up/Down
	Right Shoulder Left/Right
	Right Shoulder Up/Down
	Right Elbow Left/Right
	Left Shoulder Left/Right
	Left Shoulder Up/Down
	Left Elbow Left/Right
State Space (Velocity)	Body Linear Velocity ($3D$)
	Body Angular Velocity ($3D$)
	Torso Z
	Torso Forward/Backward
	Torso Left/Right
	Right Hip Left/Right
	Right Knee Left/Right
	Right Hip Up/Down
	Right Knee Up/Down
	Left Hip Left/Right
	Left Knee Left/Right
	Left Hip Up/Down
	Left Knee Up/Down
	Right Shoulder Left/Right
	Right Shoulder Up/Down
	Right Elbow Left/Right
	Left Shoulder Left/Right
	Left Shoulder Up/Down
	Left Elbow Left/Right

Bibliography

Abdolhosseini, F., Ling, H. Y., Xie, Z., Peng, X. B., and van de Panne, M. (2019). On learning symmetric locomotion. In *Motion, Interaction and Games*, pages 1–10.

Achille, A. and Soatto, S. (2018). Emergence of invariance and disentanglement in deep representations. *The Journal of Machine Learning Research*, 19(1):1947–1980.

Adams, R. P., Murray, I., and MacKay, D. J. C. (2009). Nonparametric Bayesian density modeling with Gaussian processes. *arXiv:0912.4896 [math, stat]*. arXiv: 0912.4896.

Alvarez, M., Luengo, D., and Lawrence, N. D. (2009). Latent force models. In *Artificial Intelligence and Statistics*, pages 9–16. PMLR.

Álvarez, M., Luengo, D., Titsias, M., and Lawrence, N. D. (2010). Efficient multioutput gaussian processes through variational inducing kernels. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 25–32.

Alvarez, M., Rosasco, L., and Lawrence, N. (2012). Kernels for vector-valued functions: a review. *Foundations and Trends® in Machine Learning*, 4(3):195–266.

Álvarez, M. A. and Lawrence, N. D. (2011). Computationally efficient convolved multiple output gaussian processes. *Journal of Machine Learning Research*, 12(May):1459–1500.

Amos, B., Stanton, S., Yarats, D., and Wilson, A. G. (2020). On the model-based stochastic value gradient for continuous reinforcement learning. *arXiv preprint arXiv:2008.12775*.

Anderson, B., Hy, T. S., and Kondor, R. (2019). Cormorant: Covariant molecular neural networks. In *Advances in Neural Information Processing Systems*, pages 14510–14519.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al. (2020). What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*.

Balandat, M., Karrer, B., Jiang, D., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. (2020). Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33.

Bartlett, P. L., Long, P. M., Lugosi, G., and Tsigler, A. (2019). Benign overfitting in linear regression. *arXiv preprint arXiv:1906.11300*.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019a). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.

Belkin, M., Hsu, D., and Xu, J. (2019b). Two models of double descent for weak features. *arXiv preprint arXiv:1903.07571*.

Belyaev, Y. K. (1961). Continuity and hölder's conditions for sample functions of stationary gaussian processes. In *Proc. Fourth Berkeley Symp. Math. Statist. Prob*, volume 2, pages 23–33.

Benton, G., Finzi, M., Izmailov, P., and Wilson, A. G. (2020). Learning invariances in neural networks. *arXiv preprint arXiv:2010.11882*.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. (2019). Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978.

Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 4. Springer.

Blum, L. C. and Reymond, J.-L. (2009). 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *J. Am. Chem. Soc.*, 131:8732.

Bochner, S. (1959). *Lectures on Fourier integrals*. Princeton University Press.

Bogatskiy, A., Anderson, B., Offermann, J., Roussi, M., Miller, D., and Kondor, R. (2020). Lorentz group equivariant neural network for particle physics. In *International Conference on Machine Learning*, pages 992–1002. PMLR.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, 31(3):307–327.

Bonilla, E. V., Chai, K. M., and Williams, C. (2007). Multi-task gaussian process prediction. *Advances in neural information processing systems*, pages 153–160.

Bonilla, E. V., Chai, K. M., and Williams, C. (2008). Multi-task gaussian process prediction. In *Advances in Neural Information Processing Systems*, pages 153–160.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Brostow, G. J., Fauqueur, J., and Cipolla, R. (2008a). Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*.

Brostow, G. J., Shotton, J., Fauqueur, J., and Cipolla, R. (2008b). Segmentation and recognition using structure from motion point clouds. In *ECCV (1)*, pages 44–57.

Bui, T. D., Nguyen, C. V., and Turner, R. E. (2017). Streaming sparse gaussian process approximations. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3301–3309.

Cao, J., Chen, J., and Hull, J. (2020). A neural network approach to understanding implied volatility movements. *Quantitative Finance*, 20(9):1405–1413.

Caponnetto, A. and Vito, E. D. (2007). Optimal rates for the regularized least-squares algorithm. *Foundations of Computational Mathematics*, 7(3):331–368.

Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. (2019). Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018.

Chauhan, L., Alberg, J., and Lipton, Z. (2020). Uncertainty-aware lookahead factor models for quantitative investing. In *International Conference on Machine Learning*, pages 1489–1499. PMLR.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583.

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. PMLR.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*.

Cleveland, W. S. (1979). Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, 74(368):829–836.

Coates, A., Ng, A., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 215–223, Fort Lauderdale, FL, USA. PMLR.

Cohen, T. and Welling, M. (2016a). Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999.

Cohen, T. S. and Welling, M. (2016b). Steerable cnns. *arXiv preprint arXiv:1612.08498*.

Colins, K. D. (2021). Cayley-menger determinant. From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein, https://mathworld.wolfram.com/Cayley-MengerDeterminant.html.

Cruz-Uribe, D. and Neugebauer, C. (2002). Sharp error bounds for the trapezoidal rule and simpson's rule. *J. Inequal. Pure Appl. Math*, 3.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2019). Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 113–123.

Czarnecki, W. M., Osindero, S., Pascanu, R., and Jaderberg, M. (2019). A deep neural network's loss surface contains every low-dimensional pattern. *arXiv preprint arXiv:1912.07559*.

Dai, Z., Álvarez, M. A., and Lawrence, N. D. (2017). Efficient modeling of latent information in supervised learning using gaussian processes. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5137–5145.

Dai, Z., Liu, H., Le, Q. V., and Tan, M. (2021). Coatnet: Marrying convolution and attention for all data sizes. *arXiv preprint arXiv:2106.04803*.

Damianou, A. and Lawrence, N. (2013). Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215.

d'Ascoli, S., Touvron, H., Leavitt, M., Morcos, A., Biroli, G., and Sagun, L. (2021). Convit: Improving vision transformers with soft convolutional inductive biases. *arXiv preprint arXiv:2103.10697*.

DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout.

Diamond, H. J., Karl, T. R., Palecki, M. A., Baker, C. B., Bell, J. E., Leeper, R. D., Easterling, D. R., Lawrimore, J. H., Meyers, T. P., Helfert, M. R., et al. (2013). Us climate reference network after one decade of operations: Status and assessment. *Bulletin of the American Meteorological Society*, 94(4):485–498.

Dobriban, E. and Wager, S. (2018). High-dimensional asymptotics of prediction: Ridge regression and classification. *Ann. Statist.*, 46(1):247–279.

Dong, K., Eriksson, D., Nickisch, H., Bindel, D., and Wilson, A. G. (2017). Scalable log determinants for Gaussian process kernel learning. In *Advances in Neural Information Processing Systems*. arXiv: 1711.03481.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, volume 9.

Draxler, F., Veschgini, K., Salmhofer, M., and Hamprecht, F. (2018). Essentially no barriers in neural network energy landscape. In *International Conference on Machine Learning*, pages 1309–1318.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Dziugaite, G. K. and Roy, D. M. (2017). Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *Uncertainty in Artificial Intelligence (UAI)*.

Elesedy, B. and Zaidi, S. (2021). Provably strict generalisation benefit for equivariant models. *arXiv preprint arXiv:2102.10333*.

Falorsi, L., de Haan, P., Davidson, T. R., and Forré, P. (2019). Reparameterizing distributions on lie groups. *arXiv preprint arXiv:1903.02958*.

Finzi, M., Stanton, S., Izmailov, P., and Wilson, A. G. (2020). Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. *arXiv preprint arXiv:2002.12880*.

Finzi, M., Welling, M., and Wilson, A. G. (2021). A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups. *arXiv preprint arXiv:2104.09459*.

Fort, S., Hu, H., and Lakshminarayanan, B. (2019). Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*.

Fort, S. and Jastrzebski, S. (2019). Large scale structure of neural network loss landscapes. In *Advances in Neural Information Processing Systems*, volume 32, pages 6709–6717.

Freeman, C. D. and Bruna, J. (2017). Topology and geometry of half-rectified network optimization. In *International Conference on Learning Representations*.

Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*. Springer series in statistics New York.

Fuchs, F. B., Worrall, D. E., Fischer, V., and Welling, M. (2020). Se (3)-transformers: 3d roto-translation equivariant attention networks. *arXiv preprint arXiv:2006.10503*.

Fukumizu, K., Yamaguch, S., Mototake, Y.-i., and Tanaka, M. (2019). Semi-flat minima and saddle points by embedding neural networks to overparameterization. In *Advances in neural information processing systems*.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. (2018a). Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, pages 7576–7586.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. (2018b). Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *Advances in Neural Information Processing Systems*, 31:7576–7586.

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*, 31:8789–8798.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58.

Genton, M. G. (2001). Classes of kernels for machine learning: a statistics perspective. *Journal of machine learning research*, 2(Dec):299–312.

Ghorbani, B., Krishnan, S., and Xiao, Y. (2019). An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*.

Gönen, M. and Alpaydın, E. (2011). Multiple kernel learning algorithms. *Journal of machine learning research*, 12(Jul):2211–2268.

Gonzalvez, J., Lezmi, E., Roncalli, T., and Xu, J. (2019). Financial applications of gaussian processes and bayesian optimization. *arXiv preprint arXiv:1903.04841*.

Goodfellow, I. J., Vinyals, O., and Saxe, A. M. (2015). Qualitatively characterizing neural network optimization problems. In *International Conference on Learning Representations*, volume 3.

Greydanus, S., Dzamba, M., and Yosinski, J. (2019). Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pages 15379–15389.

Gull, S. F. (1989). Developments in maximum entropy data analysis. In *Maximum entropy and Bayesian methods*, pages 53–71. Springer.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *International Conference on Machine Learning*, volume 70, pages 1321–1330. PMLR.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018a). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018b). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018c). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hagan, P. S., Kumar, D., Lesniewski, A. S., and Woodward, D. E. (2002). Managing smile risk. *The Best of Wilmott*, 1:249–296.

Hastie, T., Montanari, A., Rosset, S., and Tibshirani, R. J. (2019). Surprises in High-Dimensional Ridgeless Least Squares Interpolation. *arXiv:1903.08560*.

Hastie, T. and Tibshirani, R. (1990). *Generalized additive models*. Chapman and Hall, London.

Hataya, R., Zdenek, J., Yoshizoe, K., and Nakayama, H. (2019). Faster autoaugment: Learning augmentation strategies using backpropagation. *arXiv preprint arXiv:1911.06987*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, pages 282–290.

Hensman, J., Matthews, A., and Ghahramani, Z. (2015). Scalable variational gaussian process classification. In *Artificial Intelligence and Statistics*, pages 351–360. PMLR.

Herlands, W., Neill, D. B., Nickisch, H., and Wilson, A. G. (2018). Change surfaces for expressive multidimensional changepoints and counterfactual prediction. *arXiv preprint arXiv:1810.11861*.

Heston, S. L. (1993). A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The review of financial studies*, 6(2):327–343.

Hinton, G. E. and Salakhutdinov, R. R. (2008). Using deep belief nets to learn covariance kernels for gaussian processes. In *Advances in neural information processing systems*, pages 1249–1256.

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.

Hochreiter, S. and Schmidhuber, J. (1997a). Flat Minima. *Neural Computation*, 9(1):1–42.

Hochreiter, S. and Schmidhuber, J. (1997b). Flat minima. *Neural Computation*, 9(1):1–42.

Hoffman, M. D. and Gelman, A. (2014). The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623.

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*.

Hyndman, R. J. (2005). Time series data library.

Izmailov, P., Maddox, W. J., Kirichenko, P., Garipov, T., Vetrov, D., and Wilson, A. G. (2019a). Subspace inference for bayesian deep learning. *Uncertainty in Artificial Intelligence (UAI)*.

Izmailov, P., Maddox, W. J., Kirichenko, P., Garipov, T., Vetrov, D., and Wilson, A. G. (2019b). Subspace inference for bayesian deep learning. In *Uncertainty in Artificial Intelligence*, pages 1169–1179. PMLR.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. (2018). Averaging weights leads to wider optima and better generalization. In *Uncertainty in Artificial Intelligence*.

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.

Jaderberg, M., Simonyan, K., Zisserman, A., et al. (2015). Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025.

Jang, P. A., Loeb, A., Davidow, M., and Wilson, A. G. (2017). Scalable levy process priors for spectral kernel learning. In *Advances in Neural Information Processing Systems*, pages 3940–3949.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. *arXiv preprint arXiv:1906.08253*.

Jayasumana, S., Hartley, R., Salzmann, M., Li, H., and Harandi, M. (2013). Combining multiple manifold-valued descriptors for improved object recognition. In *2013 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–6. IEEE.

Jégou, S., Drozdzal, M., Vazquez, D., Romero, A., and Bengio, Y. (2017). The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 11–19.

Jiang, J., Dun, C., Huang, T., and Lu, Z. (2018). Graph convolutional reinforcement learning. *arXiv preprint arXiv:1810.09202*.

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*.

Johannink, T., Bahl, S., Nair, A., Luo, J., Kumar, A., Loskyll, M., Ojea, J. A., Solowjow, E., and Levine, S. (2019). Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE.

Kashinath, K., Mustafa, M., Albert, A., Wu, J., Jiang, C., Esmaeilzadeh, S., Azizzadenesheli, K., Wang, R., Chattopadhyay, A., Singh, A., et al. (2021). Physics-informed machine learning: case studies for weather and climate modelling. *Philosophical Transactions of the Royal Society A*, 379(2194):20200093.

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. In *International Conference on Learning Representations*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kostrikov, I., Yarats, D., and Fergus, R. (2020). Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Kuleshov, V., Fenner, N., and Ermon, S. (2018). Accurate uncertainties for deep learning using calibrated regression. In *International Conference on Machine Learning*, pages 2796–2804. PMLR.

Lai, G., Chang, W.-C., Yang, Y., and Liu, H. (2018). Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 95–104.

Laine, S. and Aila, T. (2016). Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30:6402–6413.

Lanckriet, G. R., Cristianini, N., Bartlett, P., Ghaoui, L. E., and Jordan, M. I. (2004). Learning the kernel matrix with semidefinite programming. *Journal of Machine learning research*, 5(Jan):27–72.

Laptev, D., Savinov, N., Buhmann, J. M., and Pollefeys, M. (2016). Ti-pooling: transformation-invariant pooling for feature learning in convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 289–297.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 473–480, New York, NY, USA. Association for Computing Machinery.

Lázaro-Gredilla, M. and Titsias, M. K. (2011). Variational heteroscedastic gaussian process regression. In *International Conference on Machine Learning*.

LeCun, Y., Bengio, Y., et al. (1998a). Convolutional networks for images, speech, and time series, the handbook of brain theory and neural networks.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998b). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Sohl-Dickstein, J., and Pennington, J. (2019). Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in Neural Information Processing Systems*.

Lewandowski, D., Kurowicka, D., and Joe, H. (2009). Generating random correlation matrices based on vines and extended onion method. *Journal of multivariate analysis*, 100(9):1989–2001.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the loss landscape of neural nets. In *Advances in neural information processing systems*, pages 6389–6399.

Li, Y., Hu, G., Wang, Y., Hospedales, T., Robertson, N. M., and Yang, Y. (2020). Dada: Differentiable automatic data augmentation. *arXiv preprint arXiv:2003.03780*.

Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., and Li, Z. (2019). Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*.

Lim, S., Kim, I., Kim, T., Kim, C., and Kim, S. (2019). Fast autoaugment. In *Advances in Neural Information Processing Systems*, pages 6662–6672.

Lin, Y., Huang, J., Zimmer, M., Guan, Y., Rojas, J., and Weng, P. (2020). Invariant transform experience replay: Data augmentation for deep reinforcement learning. *IEEE Robotics and Automation Letters*, 5(4):6615–6622.

Liu, B., Kiskin, I., and Roberts, S. (2020a). An overview of gaussian process regression for volatility forecasting. In *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 681–686. IEEE.

Liu, H., Erdem, E., and Shi, J. (2011). Comprehensive evaluation of arma–garch (-m) approaches for modeling the mean and volatility of wind speed. *Applied Energy*, 88(3):724–732.

Liu, H., Simonyan, K., and Yang, Y. (2018a). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Liu, I.-J., Yeh, R. A., and Schwing, A. G. (2020b). Pic: permutation invariant critic for multi-agent deep reinforcement learning. In *Conference on Robot Learning*, pages 590–602. PMLR.

Liu, R., Lehman, J., Molino, P., Such, F. P., Frank, E., Sergeev, A., and Yosinski, J. (2018b). An intriguing failing of convolutional neural networks and the coordconv solution. *arXiv preprint arXiv:1807.03247*.

Liu, Z., Chen, Y., Du, Y., and Tegmark, M. (2021). Physics-augmented learning: A new paradigm beyond physics-informed learning. *arXiv preprint arXiv:2109.13901*.

Lloyd, J., Duvenaud, D., Grosse, R., Tenenbaum, J., and Ghahramani, Z. (2014). Automatic construction and natural-language description of nonparametric regression models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

Luo, R., Zhang, W., Xu, X., and Wang, J. (2018). A neural stochastic volatility model. In *Thirty-second AAAI conference on artificial intelligence*.

Lázaro-Gredilla, M., Quiñonero-Candela, J., Rasmussen, C. E., and Figueiras-Vidal, A. R. (2010). Sparse spectrum Gaussian process regression. *Journal of Machine Learning Research*, page 17.

MacKay, D. J. (1992a). Bayesian interpolation. *Neural computation*, 4(3):415–447.

MacKay, D. J. (1992b). Bayesian model comparison and backprop nets. In *Advances in neural information processing systems*, pages 839–846.

MacKay, D. J. (1998). Introduction to gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Maddox, W. J., Balandat, M., Wilson, A. G., and Bakshy, E. (2021a). Bayesian optimization with high-dimensional outputs. *Advances in neural information processing systems*.

Maddox, W. J., Benton, G., and Wilson, A. G. (2020). Rethinking parameter counting in deep models: Effective dimensionality revisited. *arXiv preprint arXiv:2003.02139*.

Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. (2019a). A simple baseline for bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems*, 32:13153–13164.

Maddox, W. J., Stanton, S., and Wilson, A. G. (2021b). Conditioning sparse variational gaussian processes for online decision-making. *Advances in Neural Information Processing Systems*, 34.

Maddox, W. J., Tang, S., Moreno, P. G., Wilson, A. G., and Damianou, A. (2019b). On Linearizing Neural Networks for Transfer Learning. *NeurIPS MetaLearn Workshop*.

Madras, D., Atwood, J., and D'Amour, A. (2019). Detecting extrapolation with local ensembles. *arXiv preprint arXiv:1910.09573*.

Marcos, D., Volpi, M., Komodakis, N., and Tuia, D. (2017). Rotation equivariant vector field networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5048–5057.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. (2018). Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*.

Maron, H., Litany, O., Chechik, G., and Fetaya, E. (2020). On learning sets of symmetric elements. *arXiv preprint arXiv:2002.08599*.

Mavalankar, A. (2020). Goal-conditioned batch reinforcement learning for rotation invariant locomotion. *arXiv preprint arXiv:2004.08356*.

Mehdizadeh, S., Kozekalani Sales, A., and Safari, M. J. S. (2020). Estimating the short-term and long-term wind speeds: implementing hybrid models through coupling machine learning and linear time series models. *SN Applied Sciences*, 2:1–15.

Mei, S. and Montanari, A. (2019). The generalization error of random features regression: Precise asymptotics and double descent curve. *arXiv preprint arXiv:1908.05355*.

Menne, M. J., Williams, C. N., and Vose, R. S. (2015). United states historical climatology network daily temperature, precipitation, and snow data. *Carbon Dioxide Information Analysis Center, Oak Ridge National Laboratory, Oak Ridge, Tennessee*.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*.

Moody, J. E. (1991). Note on generalization, regularization and architecture selection in nonlinear learning systems. In *Neural Networks for Signal Processing Proceedings of the 1991 IEEE Workshop*, pages 1–10. IEEE.

Moody, J. E. (1992). The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In *Advances in neural information processing systems*, pages 847–854.

Murray, I. and Adams, R. P. (2010). Slice sampling covariance hyperparameters of latent gaussian models. In *Advances in Neural Information Processing Systems*, pages 1732–1740.

Murray, I., Prescott Adams, R., and MacKay, D. J. (2010). Elliptical slice sampling. In *Artificial Intelligence and Statistics*.

Muthukumar, V., Vodrahalli, K., and Sahai, A. (2019). Harmless interpolation of noisy data in regression. *arXiv preprint arXiv:1903.09139*.

Nau, R. (2014). Forecasting with moving averages. *Fuqua School of Business, Duke University*, pages 1–3.

Neal, R. M. and Zhang, J. (2006). High Dimensional Classification with Bayesian Neural Networks and Dirichlet Diffusion Trees. In Guyon, I., Nikravesh, M., Gunn, S., and Zadeh, L. A., editors, *Feature Extraction: Foundations and Applications*, Studies in Fuzziness and Soft Computing, pages 265–296. Springer, Berlin, Heidelberg.

Nelder, J. A. and Wedderburn, R. W. (1972). Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384.

Neyshabur, B. (2020). Towards learning convolutions from scratch. *arXiv preprint arXiv:2007.13657*.

Neyshabur, B., Tomioka, R., and Srebro, N. (2014). In search of the real inductive bias: On the role of implicit regularization in deep learning. *arXiv preprint arXiv:1412.6614*.

Nguyen, T. V., Bonilla, E. V., et al. (2014). Collaborative multi-output gaussian processes. In *Uncertainty in Artificial Intelligence*, pages 643–652.

Nintendo (1990). Super mario world.

Oliva, J. B., Dubey, A., Wilson, A. G., Póczos, B., Schneider, J., and Xing, E. P. (2016). Bayesian nonparametric kernel-learning. In *Artificial Intelligence and Statistics*, pages 1078–1086.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J., Lakshminarayanan, B., and Snoek, J. (2019). Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, volume 32, pages 13991–14002.

Papyan, V. (2018). The full spectrum of deepnet hessians at scale: Dynamics with sgd training and sample size. *arXiv preprint arXiv:1811.07062*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037.

Phan, D., Pradhan, N., and Jankowiak, M. (2019). Composable effects for flexible and accelerated probabilistic programming in numpyro. *arXiv preprint arXiv:1912.11554*.

Raj, A., Kumar, A., Mroueh, Y., Fletcher, T., and Schölkopf, B. (2017). Local group invariant representations via orbit embeddings. In *Artificial Intelligence and Statistics*, pages 1225–1235.

Rakitsch, B., Lippert, C., Borgwardt, K., and Stegle, O. (2013). It is all in the noise: Efficient multi-task gaussian process inference with structured residuals. *Advances in neural information processing systems*, 26:1466–1474.

Rakotomamonjy, A., Bach, F., Canu, S., and Grandvalet, Y. (2007). More efficiency in multiple kernel learning. In *Proceedings of the 24th international conference on Machine learning*, pages 775–782. ACM.

Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*, volume 2. MIT Press Cambridge, MA.

Rasmussen, C. E. and Williams, C. K. I. (2008). *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass., 3. print edition.

Ravindran, B. and Barto, A. G. (2004). Approximate homomorphisms: A framework for non-exact minimization in markov decision processes.

Reddi, S. J., Kale, S., and Kumar, S. (2019). On the convergence of adam and beyond. In *International Conference on Learning Representations*.

Requeima, J., Tebbutt, W., Bruinsma, W., and Turner, R. E. (2019a). The gaussian process autoregressive regression model (gpar). In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1860–1869.

Requeima, J., Tebbutt, W., Bruinsma, W., and Turner, R. E. (2019b). The gaussian process autoregressive regression model (gpar). In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1860–1869. PMLR.

Rupp, M., Tkatchenko, A., Müller, K.-R., and von Lilienfeld, O. A. (2012). Fast and accurate modeling of molecular atomization energies with machine learning. *Physical Review Letters*, 108:058301.

Saatçi, Y. (2012). *Scalable inference for structured Gaussian process models*. PhD thesis, Citeseer.

Sagun, L., Bottou, L., and LeCun, Y. (2017). Eigenvalues of the hessian in deep learning: Singularity and beyond. In *ICLR Workshop track, arXiv preprint arXiv:1611.07476*.

Särkkä, S. (2011). Linear operators and stochastic partial differential equations in gaussian process regression. In *International Conference on Artificial Neural Networks*, pages 151–158. Springer.

Särkkä, S. and Solin, A. (2019). *Applied stochastic differential equations*, volume 10. Cambridge University Press.

Satorras, V. G., Hoogeboom, E., and Welling, M. (2021). E (n) equivariant graph neural networks. *arXiv preprint arXiv:2102.09844*.

Sauer, T. (2012). Numerical solution of stochastic differential equations in finance. In *Handbook of computational finance*, pages 529–550. Springer.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.

Shen, Z., Heinonen, M., and Kaski, S. (2019). Harmonizable mixture kernels with variational fourier features. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3273–3282.

Silver, T., Allen, K., Tenenbaum, J., and Kaelbling, L. (2018). Residual policy learning. *arXiv preprint arXiv:1812.06298*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, volume 3.

Skorokhodov, I. and Burtsev, M. (2019). Loss landscape sightseeing with multi-point optimization. *arXiv preprint arXiv:1910.03867*.

Smith, S. L. and Le, Q. V. (2017). A bayesian perspective on generalization and stochastic gradient descent. *arXiv preprint arXiv:1710.06451*.

Sosnovik, I., Szmaja, M., and Smeulders, A. (2019). Scale-equivariant steerable networks. *arXiv preprint arXiv:1910.11093*.

Sukhbaatar, S., Szlam, A., and Fergus, R. (2016). Learning multiagent communication with backpropagation. *arXiv preprint arXiv:1605.07736*.

Sun, S., Zhang, G., Shi, J., and Grosse, R. (2019). Functional variational bayesian neural networks. In *International Conference on Learning Representations*.

Sun, S., Zhang, G., Wang, C., Zeng, W., Li, J., and Grosse, R. (2018). Differentiable compositional kernel learning for gaussian processes. In *International Conference on Machine Learning*, pages 4828–4837. PMLR.

Suzuki, T. (2018). Fast generalization error bound of deep learning from a kernel perspective. In Storkey, A. and Perez-Cruz, F., editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1397–1406, Playa Blanca, Lanzarote, Canary Islands. PMLR.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

Taylor, J., Precup, D., and Panagaden, P. (2008). Bounding performance loss in approximate mdp homomorphisms. *Advances in Neural Information Processing Systems*, 21:1649–1656.

Tian, S., Fu, Y., Ling, P., Wei, S., Liu, S., and Li, K. (2018). Wind power forecasting based on arima-lgarch model. In *2018 International Conference on Power System Technology (POWERCON)*, pages 1285–1289. IEEE.

Tobar, F. (2018). Bayesian nonparametric spectral estimation. In *Advances in Neural Information Processing Systems*, pages 10127–10137.

Tobar, F., Bui, T. D., and Turner, R. E. (2015). Learning Stationary Time Series using Gaussian

Processes with Nonparametric Kernels. In *Advances in Neural Information Processing Systems*, page 9.

Tokdar, S. T. and Ghosh, J. K. (2007). Posterior consistency of logistic gaussian process priors in density estimation. *Journal of statistical planning and inference*, 137(1):34–42.

Tran, T., Pham, T., Carneiro, G., Palmer, L., and Reid, I. (2017). A bayesian data augmentation approach for learning deep models. In *Advances in neural information processing systems*, pages 2797–2806.

van der Pol, E., Kipf, T., Oliehoek, F. A., and Welling, M. (2020a). Plannable approximations to mdp homomorphisms: Equivariance under actions. *arXiv preprint arXiv:2002.11963*.

van der Pol, E., Worrall, D., van Hoof, H., Oliehoek, F., and Welling, M. (2020b). Mdp homomorphic networks: Group symmetries in reinforcement learning. *Advances in Neural Information Processing Systems*, 33.

van der Vaart, A. W. (1998). *Asymptotic Statistics*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge.

van der Wilk, M., Bauer, M., John, S., and Hensman, J. (2018). Learning invariances using the marginal likelihood. In *Advances in Neural Information Processing Systems*, pages 9938–9948.

Wang, R., Albooyeh, M., and Ravanbakhsh, S. (2020). Equivariant maps for hierarchical structures. *arXiv preprint arXiv:2006.03627*.

Wang, T. and Ba, J. (2019). Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*.

Wei, G. C. G. and Tanner, M. A. (1990). A Monte Carlo Implementation of the EM Algorithm and the Poor Man's Data Augmentation Algorithms. *Journal of the American Statistical Association*, 85(411):699–704.

Weiler, M. and Cesa, G. (2019). General e (2)-equivariant steerable cnns. In *Advances in Neural Information Processing Systems*, pages 14334–14345.

Whilliams, C. (2010). Probabilistic modelling and reasoning time series modelling: Ar, ma, arma and all that. *School of Informatics, University of Edinburgh*.

Wilson, A. and Adams, R. (2013). Gaussian process kernels for pattern discovery and extrapolation. In *International conference on machine learning*, pages 1067–1075. PMLR.

Wilson, A. and Ghahramani, Z. (2010). Copula Processes. In *Advances in Neural Information Processing Systems*, volume 23, page 9.

Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International Conference on Machine Learning*, pages 1775–1784. PMLR.

Wilson, A. G. (2014a). *Covariance kernels for fast automatic pattern discovery and extrapolation with Gaussian processes*. PhD thesis, University of Cambridge.

Wilson, A. G. (2014b). *Covariance kernels for fast automatic pattern discovery and extrapolation with Gaussian processes*. PhD thesis, University of Cambridge.

Wilson, A. G., Dann, C., Lucas, C., and Xing, E. P. (2015). The human kernel. In *Advances in neural information processing systems*, pages 2854–2862.

Wilson, A. G., Gilboa, E., Nehorai, A., and Cunningham, J. P. (2014). Fast kernel learning for multidimensional pattern extrapolation. In *Advances in Neural Information Processing Systems*, pages 3626–3634.

Wilson, A. G. and Izmailov, P. (2020). Bayesian deep learning and a probabilistic perspective of generalization. In *Advances in Neural Information Processing Systems*, volume 33.

Worrall, D. and Welling, M. (2019). Deep scale-spaces: Equivariance over scale. In *Advances in Neural Information Processing Systems*, pages 7364–7376.

Worrall, D. E., Garbin, S. J., Turmukhambetov, D., and Brostow, G. J. (2017). Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5028–5037.

Wortsman, M., Horton, M., Guestrin, C., Farhadi, A., and Rastegari, M. (2021). Learning neural network subspaces. In *Internatinal Conference on Machine Learning*, volume 139. PMLR.

Wu, Y., Hernández-Lobato, J. M., and Ghahramani, Z. (2014). Gaussian process volatility model. *Advances in Neural Information Processing Systems*, 27:1044–1052.

Xiao, T., Dollar, P., Singh, M., Mintun, E., Darrell, T., and Girshick, R. (2021). Early convolutions help transformers see better. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., and Xiong, H. (2019). Pc-darts: Partial channel connections for memory-efficient differentiable architecture search. *arXiv preprint arXiv:1907.05737*.

Yang, Z., Wilson, A., Smola, A., and Song, L. (2015). A la carte–learning fast kernels. In *Artificial Intelligence and Statistics*, pages 1098–1106.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. In *Advances in neural information processing systems*, pages 3391–3401.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations (ICLR) arXiv:1611.03530*.

Zhang, T. (2005). Learning bounds for kernel regression using effective data dimensionality. *Neural Computation*, 17(9):2077–2098.

Zhou, W., Veitch, V., Austern, M., Adams, R. P., and Orbanz, P. (2018). Non-vacuous generalization bounds at the imagenet scale: a pac-bayesian compression approach. In *International Conference on Learning Representations*.

Zhou, Y., Ye, Q., Qiu, Q., and Jiao, J. (2017). Oriented response networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 519–528.

Zhu, B. and Dunson, D. B. (2013). Locally adaptive bayes nonparametric regression via nested gaussian processes. *Journal of the American Statistical Association*, 108(504):1445–1456.