Exploiting diversity for efficient machine learning

Krzysztof J. Geras

Doctor of Philosophy
Institute for Adaptive and Neural Computation
School of Informatics
University of Edinburgh
2017

Abstract

A common practice for solving machine learning problems is currently to consider each problem in isolation, starting from scratch every time a new learning problem is encountered or a new model is proposed. This is a perfectly feasible solution when the problems are sufficiently easy or, if the problem is hard when a large amount of resources, both in terms of the training data and computation, are available. Although this naïve approach has been the main focus of research in machine learning for a few decades and had a lot of success, it becomes infeasible if the problem is too hard in proportion to the available resources. When using a complex model in this naïve approach, it is necessary to collect large data sets (if possible at all) to avoid overfitting and hence it is also necessary to use large computational resources to handle the increased amount of data, first during training to process a large data set and then also at test time to execute a complex model.

An alternative to this strategy of treating each learning problem independently is to leverage related data sets and computation encapsulated in previously trained models. By doing that we can decrease the amount of data necessary to reach a satisfactory level of performance and, consequently, improve the accuracy achievable and decrease training time. Our attack on this problem is to exploit diversity – in the structure of the data set, in the features learnt and in the inductive biases of different neural network architectures.

In the setting of learning from multiple sources we introduce multiple-source cross-validation, which gives an unbiased estimator of the test error when the data set is composed of data coming from multiple sources and the data at test time are coming from a new unseen source. We also propose new estimators of variance of the standard k-fold cross-validation and multiple-source cross-validation, which have lower bias than previously known ones.

To improve unsupervised learning we introduce scheduled denoising autoencoders, which learn a more diverse set of features than the standard denoising auto-encoder. This is thanks to their training procedure, which starts with a high level of noise, when the network is learning coarse features and then the noise is lowered gradually, which allows the network to learn some more local features. A connection between this training procedure and curriculum learning is also drawn. We develop further the idea of learning a diverse representation by explicitly incorporating the goal of obtaining a diverse representation into the training objective. The proposed model, the composite denoising autoencoder, learns multiple subsets of features focused on modelling variations in the data set at different levels of granularity.

Finally, we introduce the idea of model blending, a variant of model compression, in which the two models, the teacher and the student, are both strong models, but different in their inductive biases. As an example, we train convolutional networks using the guidance of bidirectional long short-term memory (LSTM) networks. This allows to train the convolutional neural network to be more accurate than the LSTM network at no extra cost at test time.

Lay summary

A common practice for solving machine learning problems is currently to consider each problem in isolation, starting from scratch every time a new learning problem is encountered or a new model is proposed. This is a feasible solution when the problems are easy or, if the problem is hard, large amount of resources, both in terms of the training data and computation, are available. Although this approach was the main focus of research in machine learning for a few decades and had a lot of success, it becomes infeasible if the problem is too hard in proportion to the available resources. When using a complex model in this naïve approach, it is necessary to collect large data sets to avoid the model simply memorising the training data and hence it is also necessary to use large computational resources to handle the increased amount of data, first during training to process a large data set and then also at test time to execute a complex model.

An alternative to this strategy of treating each learning problem independently is to leverage related data sets and computation encapsulated in previously trained models. By doing that we can decrease the amount of data necessary to reach a satisfactory level of performance and, consequently, improve the accuracy achievable and decrease training time. Our attack on this problem is to exploit diversity – in the structure of the data set, in the features learnt and in the design of different neural network architectures.

In the setting of learning from multiple sources we introduce an error estimation procedure which gives accurate estimates of the test error when the data set is composed of data coming from multiple sources and the data at test time are coming from a new unseen source. We also propose new ways of estimating variability of the test error estimation which are more accurate than previously known ones. To improve learning without labels we introduce scheduled denoising autoencoders and composite denoising autoencoder, which learn a more diverse set of features than the denoising auto-encoders. Finally, we introduce the idea of model blending, a way to transfer knowledge between two machine learning models, in which the two models, the teacher and the student, are both strong models, but different in their design. As an example, we train convolutional networks using the guidance of a recurrent network. This allows to train the convolutional neural network to be more accurate than the recurrent network at no extra cost at test time.

Acknowledgements

I would like to thank a number of the members of the School of Informatics. Primarily, my supervisor, **Charles Sutton**, whose enthusiasm and encouragement was a great inspiration to me. My second supervisor, **Amos Storkey**, who was not only a source of advice on various topics but also a great companion organising Edinburgh Deep Learning Workshops. I am also indebted to other faculty members, especially **Chris Williams**, **Iain Murray**, **Guido Sanguinetti** and **Vittorio Ferrari**. I also owe a lot to **Rich Caruana** who was my mentor during the internship at Microsoft Research. I have learnt a lot from all of you.

I would also like to thank my parents, **Elżbieta Geras** and **Jerzy Geras**, and my sisters, **Antonina Geras** and **Agnieszka Geras**, for their continuous support. Finally, I would like to thank **Catriona Marwick** for correcting drafts of this thesis and for always being my point **0**.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Krzysztof J. Geras*)

To my heroes.

Table of Contents

Chapter 1

Introduction

1.1 Data, models and algorithms in the wild

A common practice for solving machine learning problems is currently to consider each problem in isolation, starting from scratch every time a new learning problem is encountered or a new model is proposed. This typical pipeline involves collecting a new data set and training the model starting from random initialisation of its parameters (or from generic priors if it is a Bayesian model). This is a perfectly feasible solution when the problems are sufficiently easy or, if the problem is hard, when a large amount of resources, both in terms of the amount of training data and computation, are available. We will consider a problem to be simple if a simple model, trained with little data, can solve the problem with high accuracy in a very short period of time. For example, in the case of classification, we will consider a learning problem to be easy if the data is low-dimensional and different classes in the data are linearly separable in the input space. On the other hand, we will consider a problem to be hard, if the model necessary to achieve a desirable level of performance has to be very complex to be sufficiently expressive to capture relevant patterns in the data or, less often, if a simple model is capable of achieving high performance at a huge computational cost. We would argue that any interesting machine learning problem is, indeed, hard according to the above definition. Artificial intelligence, the long-term goal of machine learning, even limited just to perceptual tasks like large-scale computer vision and speech recognition, certainly falls into this category.

Although this naïve approach was the main focus of research in machine learning for a few decades and had a lot of success, it becomes infeasible if the problem is too hard in proportion to the available resources. When using a complex model in this naïve approach, it is necessary to collect large data sets (if possible at all) to avoid overfitting and hence it is also necessary to use large computational resources to handle the increased amount of data. First during training to process a large data set and then also at test time to execute a complex model. An alternative to this strategy of treating each learning problem independently is to leverage related data sets and computation encapsulated in previously trained models. By doing that we can decrease the amount of data necessary to reach a satisfactory level of performance and, consequently, improve the accuracy achievable and decrease training time. Our attack on this problem is to exploit diversity – in the structure of the data set, in the features learnt and in the inductive biases of different neural network architectures. Leveraging this concept allows us to make progress in the direction of more efficient machine learning in following three aspects of machine learning.

- **Learning from multiple sources of data (diversity in the structure of the data set).** In many applications of machine learning, such as sentiment classification, machine translation or speech recognition, data come from multiple sources both at training time and at test time. These sources can be, for example, different types of products being reviewed for sentiment classification or different speakers for speech recognition. Training data are then sampled from the same underlying distribution within each source but the distributions might differ between sources. At test time, the data might also come from a different distribution associated with a new unseen source. The structure of sources of the data is often known. As demonstrated in Chapter 3, this knowledge can be leveraged not only for making more accurate predictions but also for more accurate evaluation of machine learning models. We show that when working with such a diverse data set, while the standard k-fold cross-validation provides a biased estimate of the test error, multiple-source cross-validation which we propose is unbiased. We also give new estimators of the variance for both of these cross-validation procedures.

- **Unsupervised learning (diversity in the features learnt).** The majority of data that we, humans, see have no explicit labels. Yet, we are capable of learning to recognise new object classes based on very few examples. That is an existence proof that building strong representations from unsupervised data is possible. Unsupervised learning, especially within the framework of neural networks, is an attempt to achieve this kind of label-efficient learning. For example, if the final task is classification, unsupervised pre-training regularises the network by discovering important factors of variation in the data before approaching the final task. In Chapter 4 and Chapter 5 we present two new methods for unsupervised learning which can learn features at multiple levels of granularity. The representations learnt this way are more robust, containing features which could have been learned by a diverse set of models.

- **Transferring knowledge from previously trained models (diversity in the inductive biases).** As the data sets grow bigger, so too do the models that capture relevant patterns in the data. An obvious consequence of this growth is the complexity of execution of such large highly accurate predictive models. A possible question to ask in this situation in the context of neural networks is what is the relation between the class of functions learnable within a given architecture and the class of functions representable within the same architecture. For example, previous work [Ba and Caruana, 2014] suggests that, in some cases, shallow networks have enough capacity to represent accurate functions learnt by deep networks, even though standard training methods do not allow them to learn these functions. In Chapter 6, we present experimental evidence suggesting that, fast to execute, convolutional neural networks, at least for speech recognition, can represent as complicated functions as, slow to execute, long short-term memory (LSTM) networks, if the convolutional neural network (CNN) is provided with guidance of an LSTM during training. We train the CNN with model blending through model compression, which leverages the fact that the two models, the CNN and the LSTM, are strong, yet different in their inductive bias. A resulting model is a form of an implicit ensemble.

Exploiting diversity has a long history in machine learning in a number of contexts. One of the classic machine learning techniques embracing diversity in the data is transfer learning (e.g. Thrun [1996], Pan and Yang [2010]). Its aim is to leverage data from one task (*source* task) to help solve another, related, task (*target* task). Within transfer learning, different problems arise depending on the assumptions made on how the source and targets tasks differ in the input, $p(\mathbf{x})$, and the output, $p(y|\mathbf{x})$. Pan and Yang [2010] differentiate between *inductive* transfer learning, where the source and target tasks differ in the output (potentially the support of the distribution, as well as the distribution itself), and *transductive* transfer learning, where the source and target task differ in the the input (again, potentially over the support of the distribution as well as the distribution itself). One important special case of the latter is domain adaptation (e.g. Daumé [2007]), where the source and target data differ in the distribution over the input but the distribution over the output is the same for both tasks. Importantly, the two tasks are assumed to have the same support of the corresponding input and output distributions. Another related technique is multi-task learning [Caruana, 1997], where multiple prediction problems are solved simultaneously, sharing at least some of the input features.

A different family of methods leverage diversity inside the learning model. The most important examples from this group include: ensembling (e.g. Dietterich [2000]), where predictions of multiple diverse models are averaged and boosting [Schapire, 1990], where a sequence of models is built specifically in such a way that the models later in the sequence learn patterns different than the models that precede them.

The approaches to exploiting diversity we explore in this thesis are related. In learning from multiple sources, similarly to the setting of domain adaptation, we merge data from multiple sources to use a larger training set which improves generalisation. In our work, we show that we can leverage the knowledge of the diverse structure of the data set to improve the quality of the estimate of the test error. In the context of applications of neural networks we consider in this thesis, the benefits of diversity are similar to what we observe when using a classic technique of ensembling. Features learnt at different scales and different neural network architectures, which have different inductive biases, capture different patterns in the data. In both cases, by combining the two elements we get a more robust model, as shown experimentally in this thesis. Our findings are consistent with earlier work on enforcing and exploiting diversity in neural networks. For example, Ciresan et al. [2012] construct a multi-column deep convolutional network, where each column of the network learns a different representation due to the training being data pre-processed in a different way. Similarly to our work on unsupervised learning, Tang and Mohamed [2012] were building feature representations at different scales, although they enforced diversity in the representation by directly feeding the neural network copies of images at different resolutions.

1.2 Outline of the thesis

The rest of this thesis has the following structure.

- **Chapter 2** provides the background for the rest of the thesis.

- **Chapter 3** introduces multiple-source cross-validation, which gives an unbiased estimator of the test error when the data set is composed of data coming from multiple sources and the data at test time are coming from a new unseen source. We also propose new estimators of variance of the standard k-fold cross-validation and multiple-source cross-validation, which have lower bias than previously known ones.

  The content of this chapter was published as the following.

  Krzysztof J. Geras and Charles Sutton. Multiple-source cross-validation. In *International Conference on Machine Learning*, 2013

- **Chapter 4** introduces scheduled denoising autoencoders, which learn a more diverse set of features than the standard denoising autoencoder. This is thanks to their training procedure, which starts with a high level of noise, when the network is learning coarse features and then the noise is lowered gradually, which allows the network to learn some more local features. A connection between this training procedure and curriculum learning is also drawn.

  The content of this chapter was published as the following[1].

  Krzysztof J. Geras and Charles Sutton. Scheduled denoising autoencoders. In *International Conference on Learning Representations*, 2015

- **Chapter 5** develops ideas from Chapter 4 to explicitly incorporate the goal of obtaining a diverse representation into the training objective. The proposed model, the composite denoising autoencoder, learns multiple subsets of features focused on modelling variations in the data set at different levels of granularity.

[1]Minor typographical corrections were made to the original papers

The content of this chapter was published as the following[1].

Krzysztof J. Geras and Charles Sutton. Composite denoising autoencoders. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2016

- **Chapter 6** introduces the idea of model blending, a variant of model compression, in which the two models, the teacher and the student, are both strong models, but different in their inductive biases. As an example, we train convolutional networks using the guidance of bidirectional LSTM networks. This allows to train the convolutional neural network to be more accurate than the LSTM network at no extra cost at test time.

  The content of this chapter was published as the following[1].

  Krzysztof J. Geras, Abdel-rahman Mohamed, Rich Caruana, Gregor Urban, Shengjie Wang, Ozlem Aslan, Matthai Philipose, Matthew Richardson, and Charles Sutton. Blending lstms into cnns. In *International Conference on Learning Representations (workshop track)*, 2016

- **Chapter 7** summarises the thesis and suggests future research directions.

Chapter 2

Background

In this chapter we provide the background necessary for the rest of the thesis. We introduce the topics in the order they appear in the machine learning pipeline. We begin with motivation for learning representations with neural networks instead of using hard-coded feature transformations (Section 2.1). Then we explain the basics of classification and regression with neural networks and discuss different neural network architectures (Section 2.2). In that context, we introduce the idea of curriculum learning (Section 2.3). Finally, we give a short introduction to evaluating machine learning models (Section 2.4).

2.1 Why learn representations

The main goal of representation learning is to find a vector space where semantically similar data are close to each other according to some natural metric, such as L^1 or L^2, while dissimilar ones are far apart. To understand why this is important, consider the following example within the domain of computer vision. A horizontal reflection of a natural image will yield an image which is semantically very similar, yet very far from the original one according to the L^2 distance. The same applies to small translations or rotations of the image. Hence, the pixel space is not a good representation for natural images. An accurate predictor working directly in that space would have to simultaneously have an enormous capacity to memorise all possible variations of training images and also be very well regularised to be able to generalise accurately on test data. It is very hard to imagine what ought to be the hard-coded internal transformations of the data such a predictor would perform if they were supposed to be so general that they could be efficient independently of the data set and the exact task, even within only one domain, such as computer vision. Similar difficulties arise for speech recognition or natural language processing.

Despite the task being challenging, several attempts have been made in that direction, especially in computer vision. Some notable examples include Histograms of Oriented Gradients [Dalal and Triggs, 2005] and Scale-Invariant Feature Transform [Lowe, 1999]. Until the methods for learning representations for very large data sets were made practical, mostly thanks to the increase in computational power of modern computers, these approaches of creating hand-crafted features were considered state-of-the-art for many years.

2.1.1 Unsupervised learning vs. supervised learning

For the purposes of this thesis we will consider unsupervised learning any learning procedure that does not make use of class labels. That definition includes unsupervised learning methods using neural networks but also other methods such as clustering or principal component analysis. The reason why unsupervised learning is important is in wide availability of high quality unlabelled data in comparison to labelled data. For example, while on average 300 million [fac] images are uploaded to Facebook every day, the biggest available data set for image classification, the ImageNet data set [Deng et al., 2009], contains only 14,197,122 labelled images[1].

Another important motivation to study unsupervised learning is that it is largely the way that humans learn. For example, the supervision we get for our visual system throughout our lives is very sparse. We are capable of learning new object classes from single examples due to the efficiency of representations in our brain, which must be formed largely using a form of unsupervised learning. Obviously, purely unsupervised learning is a very abstract problem. In practice, we are always given some supervision and we only need to use unsupervised learning as a means to an end, for example classification, rather than a goal on its own. Therefore, learning paradigms between unsupervised and supervised learning, such as semi-supervised learning (e.g. [Weston et al., 2008, Rasmus et al., 2015, Kingma et al., 2014]) in which some small number of labelled data are available, are more practically applicable.

[1]As of the 16th of May 2016.

2.1.2 Shallow learning vs. deep learning

Another important distinction between machine learning methods is whether the model used is "deep" or "shallow". We will consider a model to be shallow if only a few (typically not more than two) stages of non-linear data transformations precede a classifier or a regressor. Some important examples of shallow learning models are: linear regression, logistic regression, decision trees and SVMs. On the other hand, we will consider models which involve multiple stages of non-linear processing to be deep. The most important, and probably the only commonly used currently, class of deep models are deep neural networks with deterministic parameters trained with cross-entropy loss or square loss as an objective[2].

2.2 Neural networks

In full generality, all neural networks are simply function approximators (with learnable parameters) of functions from $\mathbf{x}$ to $\mathbf{y}$ of a very general form

$$\mathbf{y} = s(\mathbf{h}(\mathbf{x})),$$

where s is, typically, either a linear function of $\mathbf{h}(\mathbf{x})$ in the case of regression or softmax function in the case of classification. The complexity and rich variety of forms of neural networks is hidden in how the function $\mathbf{h}$ is defined. In the simplest case, $\mathbf{h}$ can be a linear function of $\mathbf{x}$ followed by a non-linearity, such as a sigmoid function, hyperbolic tangent or rectified linear function. Such a network would be a shallow single-layer network. Alternatively, $\mathbf{h}$ can be a composition of several nested non-linear functions of $\mathbf{x}$. Typically, these linear functions are simply of the form $\mathbf{Wz}+\mathbf{b}$, where $\mathbf{z}$ is a vector, which is an intermediate result of the computation in the network, $\mathbf{W}$ is a dense matrix and $\mathbf{b}$ is a vector. We will call a network of this structure a deep fully-connected network. More interesting architectures can be defined within this framework by imposing constraints on the network in the form of tying subsets of its parameters or by requiring special sparsity structures of its parameters. Details of specific neural networks architectures considered in this thesis are discussed in Section 2.2.1.

[2]Some interesting alternatives which might become more widely applied in the future include, for example, Variational Autoencoder [Kingma and Welling, 2014] and deep Gaussian Processes [Damianou and Lawrence, 2013].

As long as s and $\mathbf{h}$ are differentiable with respect to their parameters, neural networks can be trained in an iterative manner simply updating their parameters by taking a step in the direction opposite to the gradient of the loss between the value computed by the network $\mathbf{y}$ and the true value in the training data set $\mathbf{y}^*$, denoted by $L(\mathbf{y}, \mathbf{y}^*)$, averaged over the training data. An efficient way of computing the gradient of the loss with respect to the parameters of a neural network is known as backpropagation. In practice, averaging over the entire data set is often too slow. An approximation to the gradient can then be computed using only a random subset of the data. This method is known as stochastic gradient descent (SGD).

It is important to note that even though neural networks are a very hot topic in modern machine learning, ironically, most of the basic ideas used in modern neural networks are very old[3] and come from the period before decision trees, boosting, kernel methods and Bayesian methods dominated machine learning for a few decades. Just to name a few: convolutional neural networks (e.g. Fukushima [1980], LeCun et al. [1989, 1998]), recurrent neural networks (e.g. Hochreiter and Schmidhuber [1997]), auto-encoders (e.g. Ballard [1987], Bourlard and Kamp [1988]), denoising auto-encoders (Seung [1998]), sparsity in neural networks (e.g. LeCun et al. [1990], Zemel [1993]). Until recently, it was not possible to realise the full potential of these ideas due to a lack of sufficiently large data sets and a lack of sufficiently fast computers.

2.2.1 Neural network architectures

Autoencoders

Autoencoders are a broad class of neural network models, which serve the purpose of unsupervised learning. In full generality, an autoencoder consists of two neural networks tied together, an encoder network and a decoder network, in such a way that the input of the encoder is also the output of the decoder.

More precisely, let $\mathbf{x} \in \mathbb{R}^d$ be the input to the encoder. The output of the encoder is a hidden representation $\mathbf{y} \in \mathbb{R}^{d'}$. If the encoder is a single-layer network it is simply computed as $f_\theta(\mathbf{x}) = s(\mathbf{W}\mathbf{x} + \mathbf{b})$, where the matrix $\mathbf{W} \in \mathbb{R}^{d' \times d}$ and the vector $\mathbf{b} \in \mathbb{R}^{d'}$ are the parameters of the network, and s is a typically

[3]For a very comprehensive review of the history of neural networks see the review by Schmidhuber [2014] or the book by Goodfellow et al. [2016].

nonlinear transfer function, such as a sigmoid. We write $\theta = (\mathbf{W}, \mathbf{b})$. The other half of an autoencoder, a *decoder*, "reconstructs" the input vector from the hidden representation, which is used when training the network. The decoder has a similar form to the encoder, namely, $g_{\theta'}(\mathbf{y}) = t(\mathbf{W}'\mathbf{y} + \mathbf{b}')$, except that here $\mathbf{W}' \in \mathbb{R}^{d \times d'}$ and $\mathbf{b}' \in \mathbb{R}^{d}$. It can be useful to allow the transfer function t for the decoder to be different from that for the encoder. Typically, $\mathbf{W}$ and $\mathbf{W}'$ are constrained by $\mathbf{W}' = \mathbf{W}^T$ by analogy to the interpretation of principal components analysis as a linear encoder and decoder.

During training, our objective is to learn the encoder parameters $\mathbf{W}$ and $\mathbf{b}$. As a byproduct, we will need to learn the decoder parameters $\mathbf{b}'$ as well. We train the autoencoder weights to be able to reconstruct a random input from the training distribution $\mathbf{x}$ by running the encoder and the decoder in sequence. Formally, this process is described by the optimisation problem

$$\theta^*, \theta'^* = \arg\min_{\theta, \theta'} \mathbb{E}_{\mathbf{x}} \left[L \left(\mathbf{x}, g_{\theta'}(f_\theta(\mathbf{x})) \right) \right], \tag{2.1}$$

where L is a loss function over the input space, such as squared error. Typically we minimize this objective function using stochastic gradient descent with mini-batches.

A classic result [Baldi and Hornik, 1989] states that when $d' < d$, under certain conditions, an autoencoder learns the same subspace as PCA. If the dimensionality of the hidden representation is too large, i.e., if $d' > d$, then the autoencoder can obtain zero reconstruction error simply by learning the identity map. To push the autoencoder to learn more interesting features capturing important patterns in the data it is necessary to regularise it further. There are multiple ways of implementing this regularisation by modifying the training objective in Equation 2.1. The most common are the following.

- Adding a penalty enforcing sparsity of the activations in the hidden representation. This is implemented simply by placing an L^1 penalty on $f(\mathbf{x})$, that is, $\|f(\mathbf{x})\|_1 = \sum_j |f_j(\mathbf{x})|$. Adding this penalty to the training objective leads to what is known as sparse autoencoder (e.g. Ng [2011], Makhzani and Frey [2015a], Zemel [1993]).

- Adding a penalty on the sensitivity of the function f mapping the input to the hidden representation with respect to the input. This is implemented by penalising the Frobenius norm of the Jacobian of $f(\mathbf{x})$, that is, $\|J_f(\mathbf{x})\|_F^2 =$

$\sum_{i,j}\left(\frac{\partial f_j(\mathbf{x})}{\partial \mathbf{x}_i}\right)^2$. An autoencoder with that penalty added to the training objective is known as the contractive autoencoder [Rifai et al., 2011].

- Adding noise to the input while still trying to reconstruct the uncorrupted version of it. Such a model is known as denoising autoencoder [Vincent et al., 2008, 2010] and will be the basis for the models developed in Chapter 4 and Chapter 5. The objective functions in Equation 2.1 is modified for the denoising autoencoder to

$$\theta^*, \theta'^* = \underset{\theta,\theta'}{\arg\min}\, \mathbb{E}_{\mathbf{x},\tilde{\mathbf{x}}}\left[L\left(\mathbf{x}, g_{\theta'}(f_\theta(\tilde{\mathbf{x}}))\right)\right], \tag{2.2}$$

  where $\tilde{\mathbf{x}}$ is a corrupted version of $\mathbf{x}$.

The denoising autoencoder (DA) is based on the same intuition as in the work of Seung [1998] that a good representation should contain enough information to reconstruct corrupted versions of an original input. In a denoising autoencoder the noise forces the model to learn interesting structure even when there are a large number of hidden units. Indeed, in practical denoising autoencoders, often the best results are found with *overcomplete* representations for which $d' > d$.

There are several hyperparameters that need to be decided, including the noise distribution, the transformations s and t and the loss function L. For the loss function L, for continuous $\mathbf{x}$, squared error can be used. For binary $\mathbf{x}$ or $\mathbf{x} \in [0,1]$, it is common to use the *cross entropy* loss,

$$L(\mathbf{x},\mathbf{z}) = -\sum_{i=1}^{D}\left(x_i \log z_i + (1-x_i)\log(1-z_i)\right).$$

For the transfer functions, common choices include the sigmoid $s(v) = \frac{1}{1+e^{-v}}$ for both the encoder and decoder, or to use a rectifier $s(v) = \max(0,v)$ in the encoder paired with the sigmoid decoder. Another important parameters in a denoising autoencoder is the noise distribution p. For continuous $\mathbf{x}$, Gaussian noise $p(\tilde{\mathbf{x}}|\mathbf{x},\nu) = N(\tilde{\mathbf{x}};\mathbf{x},\nu)$ can be used. For binary $\mathbf{x}$ or $\mathbf{x} \in [0,1]$, it is most common to use *masking noise*, that is, for each $i \in 1, 2, \ldots d$, we sample $\tilde{x}_i$ independently as

$$p(\tilde{x}_i|x_i,\nu) = \begin{cases} 0 & \text{with probability } \nu, \\ x_i & \text{otherwise.} \end{cases} \tag{2.3}$$

It is also possible to train multiple layers of representations with denoising autoencoders by training a denoising autoencoder whose input is a representation learnt by another denoising autoencoder. This model is known as the stacked denoising autoencoder [Vincent et al., 2008, 2010].

Representations learnt in an unsupervised manner as explained above can be used very effectively as initialisations for supervised neural networks [Erhan et al., 2010]. It can be done simply by copying the parameters of the encoder, θ, and adding a softmax regression layer. Using unsupervised layer-wise learning of representations as a form of pre-training for classification purposes with various building blocks appeared also, for example, in the work of Bengio et al. [2007] and Hinton et al. [2006].

Convolutional networks

Convolutional networks are a class of neural networks with restricted connectivity inspired by the animal visual cortex. Convolutional networks are very similar to the standard fully-connected multi-layer neural networks except that some of the layers are replaced with "convolutional layers" and "pooling layers". The two main principles behind the design of convolutional networks are the following.

- **Weight sharing**. Groups of hidden units in the convolutional layer share parameters between themselves. This reduces the number of parameters.

- **Local connectivity**. Each hidden unit in the convolutional layer is only connected to a small localised region of the input. That reduces both the number of parameters and the amount of computation.

Thanks to these design principles convolutional networks can deal efficiently with high-dimensional inputs of regular multi-dimensional topology, allowing them to be partially invariant to translation and small rotations of the input.

An input to a convolutional layer is a multi-dimensional tensor. For example, if the input to the first layer in a convolutional network is an image, we will consider it a three-dimentional tensor, where the three dimensions represent respectively, one of the RGB channels, position on the x-axis and position on the y-axis. The output of a convolutional layer is also a multidimensional tensor, composed of slices called "feature maps". Mathematically, $\mathbf{h}_j^k$, the j-th feature

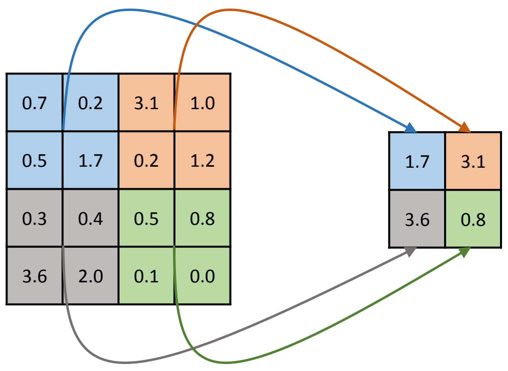


Figure 2.1: The operation of non-overlapping max pooling on one channel with 2×2 pooling regions.

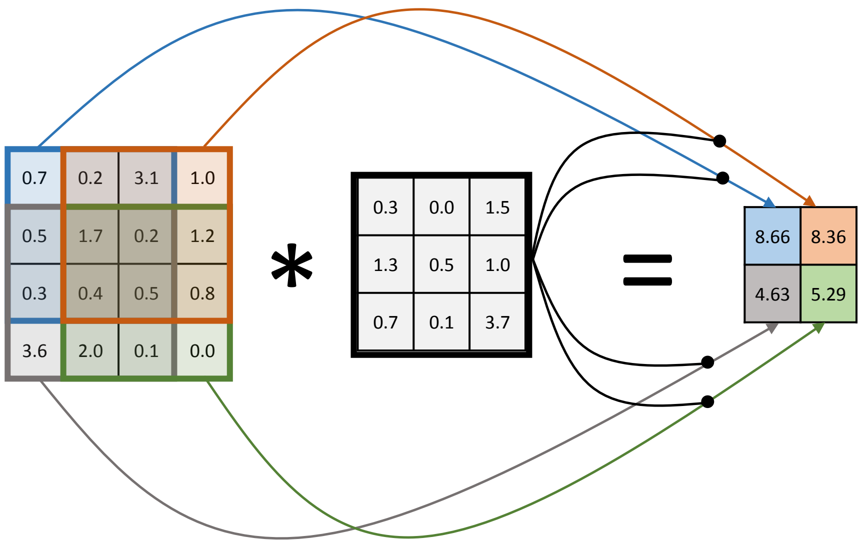


Figure 2.2: Convolution with a 3×3 filter and a stride of one pixel. For example the top left element of the matrix on the right-hand side is computed as $0.7 \cdot 0.3 + 0.2 \cdot 0.0 + 3.1 \cdot 1.5 + 0.5 \cdot 1.3 + 1.7 \cdot 0.5 + 0.2 \cdot 1.0 + 0.3 \cdot 0.7 + 0.4 \cdot 0.1 + 0.5 \cdot 3.7 = 8.66$

map in the k-th hidden layer is computed as

$$h_j^k = f\left(b_j + \left(\sum_i \mathbf{h}_i^{k-1} * \mathbf{W}_{(i,j)}^k\right)\right), \tag{2.4}$$

where f is a nonlinear function, $\mathbf{W}_{(i,j)}^k$ are matrices of parameters (also called “filters” in the context of convolutional networks) and $*$ is a discrete convolutional operation[4]. The operation of convolution is illustrated in Figure 2.2. The number

[4]A discrete convolution operation is defined in the following manner. For a feature map $\mathbf{h}$ and a filter $\mathbf{W} \in \mathrm{R}^{r,r}$, the (i,j) element of the matrix $\mathbf{h} * \mathbf{W}$ is computed as $(\mathbf{h} * \mathbf{W})_{(i,j)} = \sum_{p,q} \mathbf{h}_{(i+p,j+q)} \mathbf{W}_{(r+p,r+q)}$. Please also note that although this operation is

of feature maps in a hidden layer is a design choice analogous to the number of hidden units in a fully-connected layer.

The pooling layers act on each feature map independently. Mathematically, an element (l, m) of the j-th channel in the k-th layer is computed by the pooling layer as $\left(\mathbf{h}_j^k\right)_{(l,m)} = \max_{p,q} \left\{ \left(\mathbf{h}_j^{k-1}\right)_{(l+p,m+q)} \right\}$ for max pooling, which retains only the maximum value or $\left(\mathbf{h}_j^k\right)_{(l,m)} = \frac{1}{PQ} \sum_{p,q} \left(\mathbf{h}_j^{k-1}\right)_{(l+p,m+q)}$ for average pooling. Pooling regions may be overlapping. The pooling operation is illustrated in Figure 2.1. Although the description of convolution and pooling above is for two-dimensional inputs, convolutional layers and pooling layers can be easily generalised to more dimensions.

Typically convolutional networks are built by interleaving convolutional layers and pooling layers with optional one or two fully-connected layers at the top before a linear layer (in case of regression) or a softmax layer (in case of classification. Designs of convolutional networks have evolved a lot over time with respect to the size of convolutional filters, the size of pooling regions and the number of hidden layers. For modern convolutional networks it is common to use very small convolutional filters (3×3) and very small pooling regions (2×2) with a stride of two pixels (cf. [Simonyan and Zisserman, 2014]). Most accurate models of this kind used in computer vision use hundreds of hidden layers [He et al., 2015].

So far, convolutional neural networks had the greatest impact in computer vision. One of their first practical applications was in character recognition [LeCun et al., 1998]. Later, scaling them to larger images and data sets became feasible (e.g. Krizhevsky et al. [2012], Simonyan and Zisserman [2014]). They have also been successfully applied to speech recognition (e.g. [Abdel-Hamid et al., 2012, Sainath et al., 2013, Abdel-Hamid et al., 2014, Sainath et al., 2015, Geras et al., 2016]) and natural language processing (e.g. Kalchbrenner et al. [2014], Zhang et al. [2015]).

Recurrent networks

Recurrent neural networks (RNNs) are a family of networks specialised for processing sequential data. Given a sequence of input vectors $\boldsymbol{x} = (x_1, \ldots, x_T)$, an RNN computes the hidden vector sequence $\boldsymbol{h} = (h_1, \ldots, h_T)$ by iterating the

usually called "convolution" in the machine learning community, the correct term in signal processing for this operation is "cross-correlation" (c.f. Smith [1997]).

following from $t = 1$ to T:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right).$$

The output sequence is computed as $\boldsymbol{y} = (y_1, \ldots, y_T)$, where $y_t = W_{hy}h_t + b_y$. The W terms denote weight matrices (e.g. W_{xh} is the input-hidden weight matrix), the b terms denote bias vectors (e.g. b_h is hidden bias vector) and $\mathcal{H}$ is the hidden layer function.

While there are multiple possible choices for $\mathcal{H}$, prior work [Graves, 2012, Graves et al., 2013a, Sak et al., 2014] has shown that the LSTM architecture [Hochreiter and Schmidhuber, 1997], which uses purpose-built *memory cells* to store information, is better at finding and exploiting longer context. Figure 2.3 illustrates a single LSTM memory cell. Probably the most common version of the LSTM cell [Gers et al., 2003] implements $\mathcal{H}$ as the following composite function, which we will denote by $\mathcal{H}_{\text{LSTM}}$:

$$\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right), \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right), \\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right), \\
h_t &= o_t \tanh(c_t),
\end{aligned}$$

where $\odot$ indicates element-wise multiplication, σ is the logistic sigmoid function, and i, f, o and c are respectively the *input gate*, *forget gate*, *output gate* and *cell* activation vectors, all of which are the same size as the hidden vector h. The weight matrices from the cell to gate vectors (e.g. W_{ci}) are diagonal, so element m in each gate vector only receives input from element m of the cell vector.

One shortcoming of conventional RNNs is that they are only able to make use of previous context. Bidirectional RNNs (BRNNs) [Schuster and Paliwal, 1997] exploit past and future contexts by processing the data in both directions with two separate hidden layers, which are then fed forwards to the same output layer. As illustrated in Figure 2.3, a BRNN computes the *forward* hidden sequence $\overrightarrow{\boldsymbol{h}} = (\overrightarrow{h}_1, \ldots, \overrightarrow{h}_T)$ and the *backward* hidden sequence $\overleftarrow{\boldsymbol{h}} = (\overleftarrow{h}_1, \ldots, \overleftarrow{h}_T)$ by iterating from $t = 1$ to T:

$$\begin{aligned}
\overrightarrow{h}_t &= \mathcal{H}\left(W_{x\overrightarrow{h}}x_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right), \\
\overleftarrow{h}_t &= \mathcal{H}\left(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right).
\end{aligned}$$

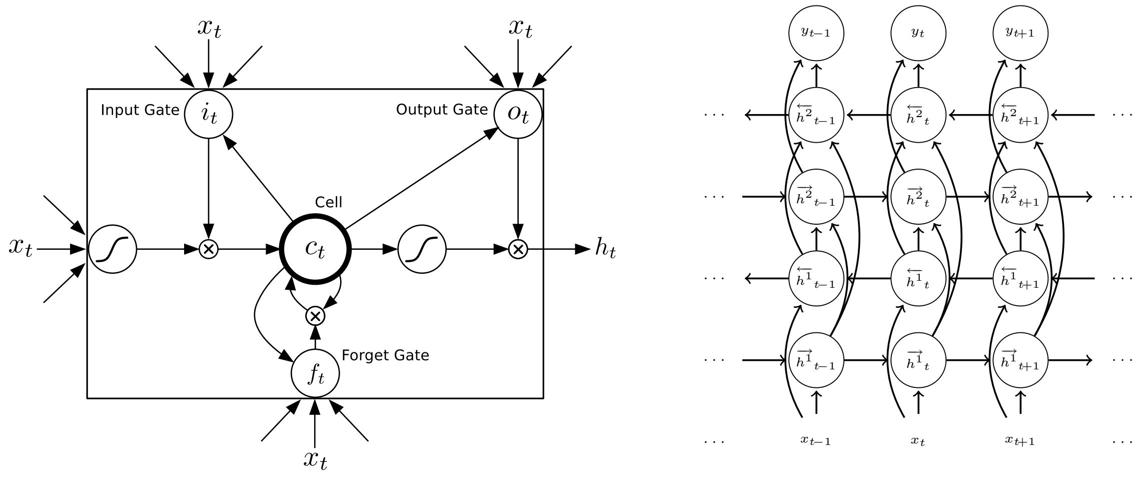


Figure 2.3: Left panel: the LSTM cell. Figure from Graves et al. [2013a]. Right panel: bidirectional RNN with two layers.

The output is computed as $\boldsymbol{y} = (y_1, \ldots, y_T)$, where $y_t = W_{\overrightarrow{h}y}\overrightarrow{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y$. Combining BRNNs with LSTMs by setting $\mathcal{H}$ to $\mathcal{H}_{\text{LSTM}}$ gives the bidirectional LSTM, which can access the context in both directions.

Finally, deep RNNs can be created by stacking multiple RNN hidden layers on top of each other, with the output sequence of one layer forming the input sequence for the next. Assuming the same hidden layer function is used for all N layers in the stack, the hidden vector sequences $\boldsymbol{h}^n$ are iteratively computed from $n = 1$ to N and $t = 1$ to T:

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n\right),$$

where we define $\boldsymbol{h}^0 = \boldsymbol{x}$. The network output sequence is computed as as $\boldsymbol{y} = (y_1, \ldots, y_T)$, where $y_t = W_{h^N y} h_t^N + b_y$.

Deep bidirectional RNNs can be implemented by replacing each hidden sequence $\boldsymbol{h}^n$ with the forward and backward sequences $\overrightarrow{\boldsymbol{h}}^n$ and $\overleftarrow{\boldsymbol{h}}^n$, and ensuring that every hidden layer receives input from both the forward and backward layers at the level below. If bidirectional LSTMs are used for the hidden layers we get deep bidirectional LSTMs, the architecture we use as a teacher network in this paper.

RNNs, especially LSTMs, exhibit great performance in a range of tasks, including speech recognition (e.g. [Graves et al., 2013b]), handwriting recognition and generation [Graves and Schmidhuber, 2009, Graves, 2014], machine translation (e.g. [Sutskever et al., 2014, Bahdanau et al., 2014]) and parsing [Vinyals et al., 2015].

2.3 Curriculum learning

The classical way of training machine learning models assumes that the same training objective is optimised throughout the entire training procedure. Inspired by research in optimisation [Allgower and Georg, 1980] and cognitive science and psychology [Skinner, 1958, Krueger and Dayan, 2009], curriculum learning [Bengio et al., 2009] is a technique that takes a different approach by training the model to solve a sequence of problems of increasing difficulty. In this sequence, a solution to the k-th optimisation problem is used as an initialisation for the $(k+1)$-th optimisation problem. Such curricula can often be designed using prior knowledge (e.g. Gulcehre and Bengio [2016]). Alternatively, they can be designed artificially as we do in Chapter 4 and Chapter 5.

In the context of machine learning, curriculum learning is motivated as a way to avoid local optima of the objective function by making the objective less multimodal. An example of such a series of objectives is shown in Figure 2.4.

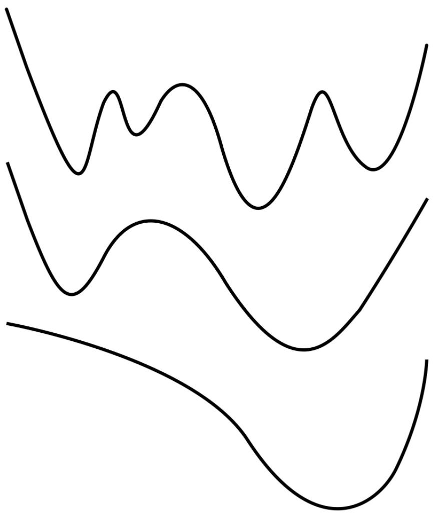


Figure 2.4: A series of objective functions with different degrees of smoothing applied. The original objective at the top has many more local minima. The minimum of the smoothest curve at the bottom is not the global minimum of the original objective at the top, but when using this minimum as a starting point for the next optimisation problem we can avoid getting stuck in a local minimum far from the global minimum.

2.4 Evaluation of machine learning models

When evaluating machine learning models, it is important to distinguish between evaluating performance of a specific model already trained with a particular data set and evaluating performance of a learning algorithm producing a different model depending on a training data set given. We will call the former the "prediction error" and will call the latter the "expected prediction error". Given a loss function $L(y, \hat{y})$ and a learning algorithm A that maps a data set $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ to a predictor $A^D : \mathbf{x} \mapsto y$, we define the prediction error as

$$\mathrm{PE}(A) = \mathbb{E}[L(A^D(\mathbf{x}), y)],$$

where the expectation is taken only over test instances $(\mathbf{x}, y)$. The expected prediction error is defined as

$$\mu = \mathrm{EPE}(A) = \mathbb{E}[L(A^D(\mathbf{x}), y)],$$

where the expectation is taken both over training sets D and test instances $(\mathbf{x}, y)$. Obviously, in practice, we are never given infinite data sets to be able to compute these quantities and we need to estimate them using finite data sets at hand. If the size of the training data is small, the two might differ significantly. In machine learning, we are usually interested in comparing learning algorithms, so we will focus our discussion on estimating the expected prediction error. In specific applications of machine learning estimating the prediction error is more meaningful.

The most common way of evaluating learning algorithms is the k-fold cross-validation[5] [Stone, 1974]. We introduce some notation to compactly describe this procedure. Partition the data as $D = D_1 \cup \ldots \cup D_K$ with $|D_k| = M$ for all k. This means that $N = KM$. Let $\mathrm{HO}(A, D_1, D_2)$ denote the holdout estimate, i.e. an average loss incurred when training on set D_1, and testing on set D_2. In k-fold cross-validation, we first compute the average loss,

$$\mathrm{HO}(A, D \backslash D_k, D_k) = \frac{1}{|D_k|} \sum_{(\mathbf{x}_i, y_i) \in D_k} L(A^{D \backslash D_k}(\mathbf{x}_i), y), \tag{2.5}$$

[5]Other closely related to k-fold cross-validation estimators are also possible. For a detailed review of various cross-validation type procedures please see the survey by Arlot and Celisse [2010].

when training A on $D \backslash D_k$ and testing on D_k, for each $k \in \{1, \ldots, K\}$ and then we average these to get the final error estimate

$$\hat{\mu} = \mathrm{CV}(A, D_{1:K}) = \frac{1}{K} \sum_{k=1}^{K} \mathrm{HO}(A, D \backslash D_k, D_k). \tag{2.6}$$

Note that even though the holdout estimate requires k times less computation, it is not necessarily a good estimator of μ since it ignores the variability of D, which is important if the training data set is small. If we care about estimating μ, the holdout estimator should be used only if cross-validation is too expensive to be practical.

For the k-fold cross-validation as defined above, every instance in D is a test instance exactly once. For $(\mathbf{x}_m, y_m) \in D_k$, denote this test error by the random variable $e_{km} = L(y_m, A^{D \backslash D_k}(\mathbf{x}_m))$. Using this notation, $\hat{\mu} = \frac{1}{KM} \sum_k \sum_m e_{km}$, so the CV estimate is a mean of correlated random variables. If the training data in D are iid the e_{km}'s have a special exchangeability[6] structure: For each k, the sequence $\mathbf{e}_k = (e_{k1}, \ldots, e_{kM})$ is exchangeable, and the sequence of vectors $\mathbf{e} = (\mathbf{e}_1, \ldots, \mathbf{e}_K)$ is also exchangeable.

Although k-fold cross-validation provides an unbiased estimator of μ[7] if test data comes from the same distribution as training data, in practice, we only have one data set D, so we would also like to estimate the variance $\theta = \mathbb{V}[\hat{\mu}]$ of this and related CV estimators. If the examples in D are iid, Bengio and Grandvalet [2003] show that the variance θ can be decomposed into a sum of three terms. The exchangeability structure of the e_{km} implies that there are only three distinct entries in their covariance matrix: σ^2, ω and γ, where

$$\begin{aligned} \mathbb{V}\left[e_{ki}\right] &= \sigma^2, & \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, K\}, \\ \mathrm{Cov}\left[e_{ki}, e_{kj}\right] &= \omega, & \forall i, j \in \{1, \ldots, M\}, i \neq j, \forall k \in \{1, \ldots, K\}, \\ \mathrm{Cov}\left[e_{ki}, e_{\ell j}\right] &= \gamma, & \forall i, j \in \{1, \ldots, M\}, \forall k, \ell \in \{1, \ldots, K\}, k \neq \ell. \end{aligned} \tag{2.7}$$

Applying the formula for the variance of the sum of correlated variables yields

$$\theta = \mathbb{V}[\hat{\mu}] = \frac{1}{KM}\sigma^2 + \frac{M-1}{KM}\omega + \frac{K-1}{K}\gamma. \tag{2.8}$$

This decomposition can be used to show that an unbiased estimator of θ does not exist. The reasoning behind this is the following. As θ has only second order

[6]We call a sequence of random variables $X_1, \ldots, X_K$ exchangeable if for any permutation σ of indices $1, \ldots, K$, the joint probability distribution of the permuted sequence $X_{\sigma(1)}, \ldots, X_{\sigma(K)}$ is the same as the joint probability of the original sequence (cf. Chow and Teicher [2012]).

[7]For a training data set of size $\frac{K-1}{K}|D|$

terms in e_{ki}'s, so would an unbiased estimator. Thus, if there existed such an estimator, it would be of the form $\hat{\theta} = \sum_{k,\ell} \sum_{ij} w_{k\ell} e_{ki} e_{\ell j}$. Coefficients $w_{k\ell}$ would be found by equating coefficients of σ^2, ω and γ in $\mathbb{E}\left[\hat{\theta}\right]$ and in θ. Unfortunately, the resulting system of equations has no solution, unless some assumptions about σ^2, ω or γ are made. In later work, Grandvalet and Bengio [2006] suggested several biased estimators of θ based on this variance decomposition and simplifying assumptions on σ^2, ω and γ.

Cross-validation, as explained above, is relying on the assumption that training data and test data come from the same distribution. Although this is mathematically a very convenient assumption, it is not always true. One interesting case of when the training and test distribution are different is when we know that the training data are coming from multiple sources and we also know that it will come from a new unseen source at test time. A cross-validation procedure we suggest using in such scenarios, the multiple-source cross-validation, is the focus of Chapter 3.

Chapter 3

Multiple-source cross-validation

3.1 Introduction to the paper

In some applications of machine learning, the training data set is composed of data coming from multiple sources and the data at test time is coming from a new unseen source. An example domain where this setting is common is sentiment classification of product reviews [Blitzer et al., 2007]. In this case, reviews of products of various categories are considered different sources. In this setting using the standard k-fold cross validation leads to a biased estimate of the test error. We demonstrate how knowledge about the diversity structure of the training data can be used for unbiased estimation of the test error. We do this with a new type of cross-validation procedure, called multiple-source cross-validation, in which the data set is partitioned with respect to the sources from which the data came instead of randomly.

Another contribution of this work is the characterisation of the form of the bias in all estimators of the variance of the standard k-fold cross-validation and multiple-source cross-validation. Based on that insight, we propose new estimators of variance of cross-validation for both the standard k-fold cross-validation and multiple-source cross-validation empirically yielding lower bias.

Multiple-source cross-validation

Krzysztof J. Geras K.J.GERAS@SMS.ED.AC.UK
Charles Sutton CSUTTON@INF.ED.AC.UK
School of Informatics, University of Edinburgh

Abstract

Cross-validation is an essential tool in machine learning and statistics. The typical procedure, in which data points are randomly assigned to one of the test sets, makes an implicit assumption that the data are exchangeable. A common case in which this does not hold is when the data come from multiple *sources*, in the sense used in transfer learning. In this case it is common to arrange the cross-validation procedure in a way that takes the source structure into account. Although common in practice, this procedure does not appear to have been theoretically analysed. We present new estimators of the variance of the cross-validation, both in the multiple-source setting and in the standard iid setting. These new estimators allow for much more accurate confidence intervals and hypothesis tests to compare algorithms.

1. Introduction

Cross-validation is an essential tool in machine learning and statistics. The procedure estimates the expected error of a learning algorithm by running a training and testing procedure repeatedly on different partitions of the data. In the most common setting, data items are assigned to a test partition uniformly at random. This scheme is appropriate when the data are independent and identically distributed, but in modern applications this iid assumption often does not hold.

One common situation is when the data arise from multiple *sources*, each of which has a characteristic generating process. For example, in document classification, text that is produced by different authors, different organisations or of different genres will have different characteristics that affect classification (Blitzer et al., 2007; Craven et al., 1998). As another example, in biomedical data, such as EEG data (Mitchell et al., 2004), data items are associated with particular subjects, with large variation across subjects.

For data of this nature, a common procedure is to arrange the cross-validation procedure by source, rather than assigning the data points to test blocks randomly. The idea behind this procedure is to estimate the performance of the learning algorithm when it is faced with data that arise from a new source that has not occurred in the training data. We call this procedure *multiple-source cross-validation*. Although it is commonly used in applications, we are unaware of theoretical analysis of this cross-validation procedure.

This paper focuses on the estimate of the prediction error that arises from the multiple-source cross-validation procedure. We show that this procedure provides an unbiased estimate of the performance of the learning algorithm on a new source that was unseen in the training data, which is in contrast to the standard cross-validation procedure. We also analyse the variance of the estimate of the prediction error, inspired by the work of Bengio & Grandvalet (2003). Estimating the variance enables the construction of approximate confidence intervals on the prediction error, and hypothesis tests to compare learning algorithms.

We find that estimators of the variance based on the standard cross-validation setting (Grandvalet & Bengio, 2006) perform extremely poorly in the multiple-source setting, in some cases, even failing to be consistent if allowed infinite data. Instead, we propose a new family of estimators of the variance based on a simple characterisation of the space of possible biases that can be achieved by a class of reasonable estimators. This viewpoint yields a new estimator not only for the variance of the multiple-source cross-validation but for that of standard cross-validation as well.

On a real-world text data set that is commonly used for studies in domain adaptation (Blitzer et al., 2007), we

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

demonstrate that the new estimators are much more accurate than previous estimators.

2. Background

Cross-validation (CV) is a common means of assessing the performance of learning algorithms (Stone, 1974). Given a loss function $L(y, \hat{y})$ and a learning algorithm A that maps a data set D to a predictor $A^D : x \mapsto y$, CV can be interpreted as a procedure to estimate the expected prediction error $\mu = \mathrm{EPE}(A) = \mathbb{E}[L(A^D(x), y)]$. The expectation is taken over training sets D and test instances (x, y).

In this paper we focus on k-fold CV. We introduce some notation to compactly describe the procedure. Denote the training set by $D = \{(x_i, y_i)\}_{i=1}^N$ and partition the data as $D = D_1 \cup \ldots \cup D_K$ with $|D_k| = M$ for all k. This means that $N = KM$. Let $\mathrm{HO}(A, D_1, D_2)$ denote the average loss incurred when training on set D_1, and testing on set D_2.

In k-fold cross-validation, we first compute the average loss, $\mathrm{HO}(A, D \backslash D_k, D_k)$, when training A on $D \backslash D_k$ and testing on D_k, for each $k \in \{1, \ldots, K\}$. We average these to get the final error estimate

$$\hat{\mu} = \mathrm{CV}(A, D_{1:K}) = \frac{1}{K} \sum_{k=1}^{K} \mathrm{HO}(A, D \backslash D_k, D_k). \quad (1)$$

If $(D_1, \ldots, D_K)$ is a random partition of D, we will refer to this procedure as *random cross-validation* (CVR) to differentiate it from the *multiple-source cross-validation* (CVS) that is the principal focus of this paper.

Notice that every instance in D is a test instance exactly once. For $(x_m, y_m) \in D_k$, denote this test error by the random variable $e_{km} = L(y_m, A^{D \backslash D_k}(x_m))$. Using this notation, $\hat{\mu} = \frac{1}{KM} \sum_k \sum_m e_{km}$, so the CV estimate is a mean of correlated random variables. For CVR, the e_{km}'s have a special exchangeability structure: For each k, the sequence $\mathbf{e}_k = (e_{k1}, \ldots, e_{kM})$ is exchangeable, and the sequence of vectors $\mathbf{e} = (\mathbf{e}_1, \ldots, \mathbf{e}_K)$ is also exchangeable.

Our goal will be to estimate the variance $\theta_{\mathrm{CVR}} = \mathbb{V}[\hat{\mu}_{\mathrm{CVR}}]$ of this and related CV estimators. If the examples in D are iid, Bengio & Grandvalet (2003) show that the variance θ_{CVR} can be decomposed into a sum of three terms. The exchangeability structure of the e_{km} implies that there are only three distinct entries in their covariance matrix: σ^2, ω and γ, where

$$\begin{aligned}
\mathbb{V}[e_{ki}] &= \sigma^2, && \forall i \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, K\}, \\
\mathrm{Cov}[e_{ki}, e_{kj}] &= \omega, && \forall i, j \in \{1, \ldots, M\}, i \neq j, \forall k \in \{1, \ldots, K\}, \\
\mathrm{Cov}[e_{ki}, e_{\ell j}] &= \gamma, && \forall i, j \in \{1, \ldots, M\}, \forall k, \ell \in \{1, \ldots, K\}, k \neq \ell.
\end{aligned}$$

Applying the formula for the variance of the sum of correlated variables yields

$$\theta_{\mathrm{CVR}} = \mathbb{V}[\hat{\mu}_{\mathrm{CVR}}] = \frac{1}{KM}\sigma^2 + \frac{M-1}{KM}\omega + \frac{K-1}{K}\gamma. \quad (2)$$

This decomposition can be used to show that an unbiased estimator of θ_{CVR} does not exist. The reasoning behind this is the following. Because θ_{CVR} has only second order terms in e_{ki}'s, so would an unbiased estimator. Thus, if there existed such an estimator, it would be of the form $\hat{\theta} = \sum_{k,\ell} \sum_{ij} w_{k\ell} e_{ki} e_{\ell j}$. Coefficients $w_{k\ell}$ would be found by equating coefficients of σ^2, ω and γ in $\mathbb{E}\left[\hat{\theta}\right]$ and in θ_{CVR}. Unfortunately, the resulting system of equations has no solution, unless some assumptions about σ^2, ω or γ are made. In later work, Grandvalet & Bengio (2006) suggested several biased estimators of θ_{CVR} based on this variance decomposition and simplifying assumptions on σ^2, ω and γ. These are described in Table 2.

3. Multiple-source cross-validation

In many practical problems, the data arise from a number of different *sources* that have different generating processes. Examples of sources include different genres in document classification, or different patients in a biomedical domain. In these cases, often the primary interest is in the performance of a classifier on a new source, rather than over only the sources in the training data. This is essentially the same setting used in domain adaptation and transfer learning, except that we are interested in estimating the error of a prediction procedure rather than in developing the prediction procedure itself.

This situation can be modelled by a simple hierarchical generative process. We assume that each source $k \in \{1, \ldots, K\}$ has a set of parameters β_k that define its generative process and that the parameters $\beta_1, \ldots, \beta_K$ for each source are iid according to an unknown distribution. We assume that the source of each datum in the training set is known, i.e., each training example is of the form (y_m, x_m, k_m), where k_m is the index of the source of data item m. The data are then modelled as arising from a distribution $p(y_m, x_m | \beta_{k_m})$ that is different for each source.

The goal of cross-validation in this setting is to estimate the *out-of-source* error, i.e., the error on data that arises from a new source that does not occur in the training data. This error is

$$\mathrm{OSE} = \mathbb{E}[L(A^D(x), y)], \quad (3)$$

where the expectation is taken with respect to training sets D and testing examples (y, x) that are drawn from a new source, that is, $p(y, x) = \int p(y, x|\beta)p(\beta)d\beta$.

In the multiple-source setting, it is intuitively obvious that the standard cross-validation estimator of (1) is inappropriate. (We shall make this intuition precise in the next section.) Instead it is common practice to modify the CV procedure so that no source is split across test blocks, a procedure that we will call *multiple-source cross-validation*. Let S_k denote the set of all training points that were sampled from source k. Multiple-source cross-validation works exactly as standard CV, except that we partition the data as $D = S_1 \cup \ldots \cup S_K$ instead of assigning the training instances to blocks randomly. The resulting estimate of the error, which we denote by CVS, is

$$\hat{\mu}_{\text{CVS}} = \text{CVS}(A, S_{1:K}) = \frac{1}{K}\sum_{k=1}^{K} \text{HO}(A, D\backslash S_k, S_k).$$

The idea behind this procedure is that it accounts for the fact that we expect to be surprised by the new source in the test data, by using the information in the training data to simulate the effect of predicting on an unseen source. Although this estimator is commonly used in practice, we are unaware of previous work concerning its asymptotic or finite sample behaviour.

4. Analysis of multiple-source cross-validation

We give the mean (Section 4.1) and variance (Section 4.2) of the CVS estimator. The variance has an analogous decomposition to the CVR estimator, but the individual terms in the CVS variance behave differently from those in CVR, in a way that has a large impact on effective estimators of the variance. Finally, we present novel estimators of the variance of the CVS error estimate (Section 4.2).

4.1. Mean of the error estimate

First, we consider the expected value of both the CVS and the CVR error estimates, in the case where the data was in fact generated from the multiple-source process described in the previous section. The idea is to make clear what $\hat{\mu}_{\text{CVS}}$ is estimating, and to provide a more careful justification for the common use of $\hat{\mu}_{\text{CVS}}$ on multiple-source data.

Let $D = S_1 \cup \ldots \cup S_K$. Then it is easy to show using the exchangeability of $S_1, \ldots, S_K$ that the expected value of the estimate $\hat{\mu}_{\text{CVS}}$ is

$$\begin{aligned}\mathbb{E}[\hat{\mu}_{\text{CVS}}] &= \mathbb{E}_{(S_{1:K})}\left[\text{HO}(A, S_{1:(K-1)}, S_K)\right] \\ &= \mathbb{E}_{(S_{1:(K-1)},X,Y)}\left[L(A^{S_{1:(K-1)}}(X), Y)\right], \quad (4)\end{aligned}$$

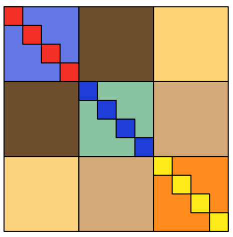


Figure 1. Diagram of the covariance matrix of the error variables for CVS. Blocks of the same colour represent blocks of variables with identical covariance.

where this notation indicates that the expectation is taken over $S_{1:(K-1)}$, X and Y. This is the expected error when the algorithm A is trained with data coming from $K-1$ sources and the test point is going to come from a newly sampled source. This is the out-of-source error (3), but on a slightly smaller training set than D.

On the other hand, consider the CVR estimate on the same data set D. Let $D_1 \cup \ldots \cup D_K = D$ be a random partition of D with $|D_k| = M$ for all k. The same symmetry argument which was used in (4) yields

$$\begin{aligned}\mathbb{E}[\hat{\mu}_{\text{CVR}}] &= \mathbb{E}_{(D_{1:K})}\left[\text{HO}(A, D_{1:(K-1)}, D_K)\right] \\ &= \mathbb{E}_{(D_{1:(K-1)},X,Y)}\left[L(A^{D_{1:(K-1)}}(X), Y)\right], \quad (5)\end{aligned}$$

which is formally similar to the above, but has the crucial difference that $D_{1:(K-1)}, X, Y$ have a different joint distribution. Here X and Y are drawn from the same family of K sources as the training data $D_{1:(K-1)}$. So $\mathbb{E}[\hat{\mu}_{\text{CVR}}]$ is the expected error of an algorithm trained with $M(K-1)$ data items from K sources, when the test point arises from one of the training sources.

Neither (4) or (5) is exactly the same as the out-of-sample error (3) that we want to estimate. Looking at the above, we see that $\hat{\mu}_{\text{CVS}}$ is biased for (3) because $\hat{\mu}_{\text{CVS}}$ uses slightly fewer training sources than OSE. On the other hand, $\hat{\mu}_{\text{CVR}}$ is biased for OSE because the training sets are slightly smaller and also because in $\hat{\mu}_{\text{CVR}}$ the training and test data are drawn from the same set of sources. However, if it is important to have a conservative estimate of the error, $\hat{\mu}_{\text{CVR}}$ has the advantage that its bias will tend to be negative, based on the expected effect of a smaller training set. (We verify this intuition experimentally in Section 6.)

4.2. Variance of the error estimate

Now we consider the variance of $\hat{\mu}_{\text{CVS}}$. The key difference between this case and the CVR case is that error variables e_{ki} and $e_{\ell j}$ are no longer identically distributed if $k \neq \ell$, as the corresponding data points arise from different sources. There is still a block structure to the covariance matrix of $\mathbf{e}$, but it is more complex. Namely, the error variables have identical covariance structure within each source but not across sources. More formally,

$$\begin{aligned} \mathbb{V}\left[e_{ki}\right] &= \sigma_k^2, && \forall i \in \{1,\dots,M\}, \\ \operatorname{Cov}\left[e_{ki}, e_{kj}\right] &= \omega_k, && \forall i,j \in \{1,\dots,M\}, i \neq j, \\ \operatorname{Cov}\left[e_{ki}, e_{\ell j}\right] &= \gamma_{k\ell}, && \forall i,j \in \{1,\dots,M\}, k \neq \ell. \end{aligned} \quad (6)$$

This covariance structure follows from the generating process described in Section 3, and is depicted graphically in Figure 1. The means of the error variables have an analogous structure, that is, $\mathbb{E}\left[e_{ki}\right] = \mathbb{E}\left[e_{kj}\right] := \mu_k$, which follows because the data points within each source are exchangeable.

This allows us to obtain a decomposition of the variance $\theta_{\mathrm{CVS}} = \mathbb{V}[\hat{\mu}_{\mathrm{CVS}}]$ of the CVS error estimate. This decomposition is

$$\theta_{\mathrm{CVS}} = \frac{1}{K^2M}\sum_k \sigma_k^2 + \frac{M-1}{K^2M}\sum_k \omega_k + \frac{1}{K^2}\sum_{k\neq l}\gamma_{kl}, \quad (7)$$

which again follows because θ_{CVS} is the variance of a sum of correlated random variables. Notice the difference to decomposition of θ_{CVR} in (2).

In the rest of this section we consider various estimates of θ_{CVS}. As θ_{CVS} depends quadratically on the variables e_{kl}, we will restrict our attention to estimators of the form $\hat{\theta} = \sum_{k,\ell}\sum_{i,j} w_{k\ell}e_{ki}e_{\ell j}$. That is, the estimator is a quadratic function of the variables $\mathbf{e}$, and as the error variables $\mathbf{e}_k$ within a single source k are exchangeable, we require such variables to be weighted equally. We refer to an estimator of this form as *quadratic*.

Every quadratic estimator can be written as a function of the empirical second moments of the error variables. If $\hat{\theta}$ is quadratic,

$$\hat{\theta} = \sum_k a_k s_k^{\sigma^2} + \sum_k b_k s_k^{\omega} + \sum_{k\neq l} c_{kl} s_{kl}^{\gamma},$$

where the $s_k^{\sigma^2}$, s_k^{ω} and s_{kl}^{γ}, are empirical moments

$$s_k^{\sigma^2} = \frac{1}{M}\sum_i e_{ki}^2, \quad s_k^{\omega} = \frac{1}{M(M-1)}\sum_{i\neq j} e_{ki}e_{kj},$$
$$s_{kl}^{\gamma} = \frac{1}{M^2}\sum_{i,j} e_{ki}e_{lj}.$$

4.2.1. No Unbiased Estimator

Using the decomposition (7), we can now follow the same reasoning as Bengio & Grandvalet (2003) to show that there is no unbiased estimator of θ_{CVS}. First, because $\theta_{\mathrm{CVS}} = \frac{1}{(KM)^2}\sum_{k,\ell}\sum_{i,j}\operatorname{Cov}\left[e_{ki}e_{\ell j}\right]$, an unbiased estimator must also be quadratic. The expectation of a quadratic estimator has the form

$$\mathbb{E}[\hat{\theta}] = \sum_k a_k(\sigma_k^2 + \mu_k^2) + \sum_k b_k(\omega_k + \mu_k^2) + \sum_{k\neq l} c_{kl}(\gamma_{kl} + \mu_k\mu_l). \quad (8)$$

To get $\mathbb{E}[\hat{\theta}] = \theta_{\mathrm{CVS}}$, we need to match the coefficients in the equations (7) and (8), including the coefficients of the μ_k^2 and $\mu_k\mu_\ell$ terms that need to equal zero, since there clearly exist distributions such that $\mu_k\mu_l > 0$. This yields the system of equations

$$a_k = \frac{1}{K^2M}, \;\; b_k = \frac{M-1}{K^2M}, \;\; a_k + b_k = 0, \;\; c_{kl} = \frac{1}{K^2}, \;\; c_{kl} = 0. \quad (9)$$

Clearly, these equations are unsatisfiable, so no unbiased estimator of θ_{CVS} exists.

4.2.2. Naive Estimators

We can derive new estimators of the variance θ_{CVS} by following the reasoning used in the standard CV setting by Grandvalet & Bengio (2006). Unfortunately, as we will see, the resulting estimators perform poorly.

The idea is rather than attempting to define an estimator that is unbiased for all data distributions, instead define an estimator that is unbiased for a restricted class of data distributions, defined by assumptions on μ_k, σ_k^2, ω_k and γ_{kl}.

First, we restrict our attention to cases in which the mean prediction error across sources is the same, i.e., $\mu_k = \mu_l := \mu$. A quadratic estimator which is unbiased for this class of data distributions must satisfy the equations

$$\begin{gathered} a_k = \frac{1}{K^2M}, \qquad b_k = \frac{M-1}{K^2M}, \qquad c_{kl} = \frac{1}{K^2}, \\ \sum_k a_k + \sum_k b_k + \sum_{k\neq l} c_{kl} = 0. \end{gathered} \quad (10)$$

These equations also have no solution. However, if we further assume (following the reasoning of Grandvalet & Bengio (2006), even though it is unlikely to be a good assumption) that one of the sources $\tilde{k}$ has $\omega_{\tilde{k}} = 0$, then we no longer have to match the value of the coefficient $b_{\tilde{k}}$, removing one of the constraints. Solving the remaining equations yields

$$\begin{gathered} a_k = \frac{1}{K^2M}, \qquad c_{kl} = \frac{1}{K^2}, \\ b_{\tilde{k}} = \frac{M-1}{K^2M} - 1, \qquad b_k = \frac{M-1}{K^2M}, \forall k \neq \tilde{k}. \end{gathered}$$

The corresponding estimator and its bias are shown in the first line of Table 1. Similarly, instead of assuming

$\omega_{\tilde{k}} = 0$, we could restrict ourselves to have a single $\gamma_{\tilde{k}\tilde{l}}$, or to have all of the ω_k's or γ_{kl}'s equal zero. This yields the remaining estimators in Table 1.

These estimators are appealingly simple, but, unfortunately, they have serious problems. The main problem is that their biases depend on the mean error μ_k of the sources. This is often an order of magnitude larger than the true variance θ_{CVS} that we are trying to predict. In particular, $\hat{\theta}^{\gamma}$, which is an analog of $\hat{\theta}_3$, the preferred estimator in the original CVR setting in the work of Grandvalet & Bengio (2006), is poor in this regard. Unlike the true variance, it does not even converge to 0 as $N \to \infty$, because the second term $\frac{1}{K^2}\left(\sum_{k=1}^{K}\mu_k^2 - \frac{\sum_{k\neq l}\mu_k\mu_l}{(K-1)}\right)$ in its bias does not converge to 0 in general.

The end result is that we can attempt to follow a similar strategy as in standard cross-validation to estimate the error variance, but doing so leads to extremely poor estimators. Instead, we will introduce new estimators that are specific to the multiple-source cross-validation setting.

4.2.3. Design of new estimators

Instead of the naive estimators from the previous section, we can derive better estimators by taking a different perspective. Instead of trying to design estimators that are unbiased for a restrictive set of scenarios, we consider the asymptotic behaviour of the bias decomposition of the quadratic variance estimator.

In the last section we saw that every quadratic estimator $\hat{\theta}$ has a bias of the form

$$\mathbb{E}[\hat{\theta} - \theta_{\mathrm{CVS}}] = \sum_k \left(a_k - \frac{1}{K^2M}\right)\sigma_k^2 + \sum_k \left(b_k - \frac{M-1}{K^2M}\right)\omega_k + \sum_{k\neq l}\left(c_{kl} - \frac{1}{K^2}\right)\gamma_{kl} + \sum_k (a_k + b_k)\,\mu_k^2 + \sum_{k\neq l} c_{kl}\mu_k\mu_l.$$

Choosing the coefficients a_k, b_k, c_{kl} uniquely determines an estimator $\hat{\theta}$. Let us consider the relative magnitude of the terms in the bias decomposition. The error means μ_k^2 are usually a few of orders of magnitude larger than $\frac{1}{K^2M}\sigma_k^2$, $\frac{M-1}{K^2M}\omega_k$ and $\frac{1}{K^2}\gamma_{kl}$ as long as M is not too small. (We check this intuition experimentally in Section 6.) Therefore it seems sensible to require that all of the μ_k terms vanish. This implies that $a_k + b_k = 0$ and $c_{kl} = 0$ for all k, l.

Next, usually σ_k^2 is larger than ω_k or γ_{kl}, which suggests choosing $a_k = (K^2M)^{-1}$ so that the σ_k^2 term vanishes as well. This yields the estimator

$$\hat{\theta}_A = \frac{1}{K^2M}\left(\sum_k s_k^{\sigma^2} - \sum_k s_k^{\omega}\right). \qquad (11)$$

The bias of this estimator is

$$\mathbb{E}[\hat{\theta}_A - \theta_{\mathrm{CVS}}] = -\frac{1}{K^2}\sum_k \omega_k - \frac{1}{K^2}\sum_{k\neq l}\gamma_{kl}.$$

However, we can do better than this. Instead of requiring that the σ_k^2 term vanish, which is a very stringent requirement, we could instead require its coefficient to be $(KN)^{-1} = (K^2M)^{-1}$, so that it becomes negligible for large N. This amounts to the requirment that $a_k = 2(K^2M)^{-1}$, which results in the estimator

$$\hat{\theta}_B = \frac{2}{K^2M}\left(\sum_k s_k^{\sigma^2} - \sum_k s_k^{\omega}\right). \qquad (12)$$

This estimator is especially appealing because of the form of its bias, which is

$$\mathbb{E}[\hat{\theta}_B - \theta_{\mathrm{CVS}}] = \frac{1}{K^2M}\left(\sum_k \sigma_k^2 - (M+1)\sum_k \omega_k - M\sum_{k\neq l}\gamma_{kl}\right).$$

Now σ_k^2's and ω_k's are positive, and in practice γ_{kl}'s are almost always positive if M is not small. So what is appealing about this estimator is that the three terms have differing signs. In many situations, the difference between the first term and the second two will be of smaller magnitude than either of the three terms alone, causing $\hat{\theta}_B$ to be significantly less biased than the estimators in Table 1. We show this experimentally in Section 6.

5. New estimators of θ for CVR

It is actually possible to apply the same viewpoint from the previous section in order to provide new estimators for the variance of the standard cross-validation procedure. Previously known estimators of the variance θ_{CVR} (Table 2) were designed to be unbiased for a subclass of generating processes. By considering the bias decomposition directly, as in the previous section, we can design better estimators.

First, we give a proposition that describes the space of possible biases for quadratic estimators of θ_{CVR}.

Proposition 1. *Let $\hat{\theta} = \sum_{k,\ell}\sum_{ij} w_{k\ell}e_{ki}e_{\ell j}$ be a quadratic estimator. Then the bias $\mathbb{E}\left[\hat{\theta} - \theta_{CVR}\right]$ has the form $\alpha_1\sigma^2 + \alpha_2\omega + \alpha_3\gamma + \alpha_4\mu$. Also $\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4 = -1$. Conversely, for every α_1, α_2, α_3, α_4 such that $\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4 = -1$, there exists a quadratic estimator $\hat{\theta}$ with bias*

$$\mathbb{E}\left[\hat{\theta} - \theta_{CVR}\right] = \alpha_1\sigma^2 + \alpha_2\omega + \alpha_3\gamma + \alpha_4\mu^2.$$

Proof. For a quadratic estimator $\hat{\theta}$, let $a = M\sum_k w_{kk}$, let $b = M(M-1)\sum_k w_{kk}$ and $c = M^2\sum_k\sum_{l\neq k} w_{kl}$.

Table 1. Naive estimators of θ_{CVS} coming from solutions of simplified system of equations (10).

Unbiased if	**Estimator**	**Bias**
$\omega_{\tilde{k}} = 0$	$\hat{\theta}^{\omega}_{\tilde{k}} = \frac{1}{K^2 M} \sum_k s_k^{\sigma^2} + \frac{M-1}{K^2 M} \sum_k s_k^{\omega} + \frac{1}{K^2} \sum_{k \neq l} s_{kl}^{\gamma} - s_{\tilde{k}}^{\omega}$	$-\omega_{\tilde{k}} + \frac{1}{K^2} \sum_{k,l} \mu_k \mu_l - \mu_{\tilde{k}}^2$
$\gamma_{\tilde{k}\tilde{l}} = 0$	$\hat{\theta}^{\gamma}_{\tilde{k},\tilde{l}} = \frac{1}{K^2 M} \sum_k s_k^{\sigma^2} + \frac{M-1}{K^2 M} \sum_k s_k^{\omega} + \frac{1}{K^2} \sum_{k \neq l} s_{kl}^{\gamma} - \left(\frac{s^{\gamma}_{\tilde{k}\tilde{l}} + s^{\gamma}_{\tilde{l}\tilde{k}}}{2} \right)$	$-\gamma_{\tilde{k}\tilde{l}} + \frac{1}{K^2} \sum_{k,l} \mu_k \mu_l - \mu_{\tilde{l}} \mu_{\tilde{k}}$
$\forall_k \ \ \omega_k = 0$	$\hat{\theta}^{\omega} = \frac{1}{K^2 M} \sum_k s_k^{\sigma^2} + \frac{M-1-KM}{K^2 M} \sum_k s_k^{\omega} + \frac{1}{K^2} \sum_{k \neq l} s_{kl}^{\gamma}$	$-\frac{\sum_k \omega_k}{K} + \frac{1-K}{K^2} \left(\sum_k \mu_k^2 - \frac{\sum_{k \neq l} \mu_k \mu_l}{(K-1)} \right)$
$\forall_{k,l} \ \ \gamma_{kl} = 0$	$\hat{\theta}^{\gamma} = \frac{1}{K^2 M} \sum_k s_k^{\sigma^2} + \frac{M-1}{K^2 M} \sum_k s_k^{\omega} - \frac{1}{K^2 (K-1)} \sum_{k \neq l} s_{kl}^{\gamma}$	$-\frac{\sum_{k \neq l} \gamma_{kl}}{K(K-1)} + \frac{1}{K^2} \left(\sum_k \mu_k^2 - \frac{\sum_{k \neq l} \mu_k \mu_l}{(K-1)} \right)$

Table 2. Estimators of θ for CVR suggested by Grandvalet & Bengio (2006).

Unbiased if	**Estimator**	**Bias**
$\mu = 0$	$\hat{\theta}_1 = \frac{1}{N} s_1 + \frac{M-1}{N} s_2 + \frac{N-M}{N} s_3$	μ^2
$\omega = 0$	$\hat{\theta}_2 = \frac{1}{N} s_1 - \frac{N+1-M}{N} s_2 + \frac{N-M}{N} s_3$	$-\omega$
$\gamma = 0$	$\hat{\theta}_3 = \frac{1}{N} s_1 + \frac{M-1}{N} s_2 - \frac{M}{N} s_3$	$-\gamma$
$\gamma = -\frac{M}{N-M} \omega$	$\hat{\theta}_4 = \frac{1}{N} s_1 - \frac{1}{N} s_2$	$-\frac{M}{N} \omega - \frac{N-M}{N} \gamma$

Taking expectations, we have

$$\mathbb{E}\left[\hat{\theta}\right] = a\sigma^2 + b\omega + c\gamma + (a+b+c)\mu^2.$$

Therefore the bias has the form

$$\mathbb{E}\left[\hat{\theta} - \theta_{\text{CVR}}\right] = \left(a - \frac{1}{KM}\right)\sigma^2 + \left(b - \frac{M-1}{KM}\right)\omega + \left(c - \frac{K-1}{K}\right)\gamma + (a+b+c)\mu^2.$$

So the bias has the required form, with $\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4 = -1$. Conversely, let $\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4 = -1$. Then we can obtain an estimator $\hat{\theta}$ by setting $a = \alpha_1 + \frac{1}{KM}$, $b = \alpha_2 + \frac{M-1}{KM}$, $c = \alpha_3 + \frac{K-1}{K}$, and $d = a+b+c+1$. $\square$

All of the existing estimators of θ_{CVR} (Table 2) have similar biases in the sense that the coefficients α_1, α_2, and α_3 are non-positive. But from the previous proposition, we know that there is a much larger set of coefficients available.

To design a new estimator, we observe that both in the results of Grandvalet & Bengio (2006) and our own results in Section 6, typically $\omega > \gamma$ and ω and γ are of similar magnitude. Therefore an estimator with bias of $\omega - 2\gamma$ will have smaller bias than the estimators from Table 2. Applying the previous result, this bias is achieved by the estimator

$$\hat{\theta}_5 = \frac{1}{N} s_1 + \frac{N+M-1}{N} s_2 - \frac{N+M}{N} s_3,$$

where s_1, s_2 and s_3 are the empirical moments

$$s_1 = \frac{1}{K} \sum_k s_k^{\sigma^2}, \quad s_2 = \frac{1}{K} \sum_k s_k^{\omega}, \quad s_3 = \frac{1}{K(K-1)} \sum_{k \neq l} s_{kl}^{\gamma}.$$

In the next section we show that its performance is superior in practice to the best previous estimator $\hat{\theta}_3$ given by Grandvalet & Bengio (2006).

6. Experiments

We evaluate the usefulness of $\hat{\mu}_{\text{CVR}}$ and $\hat{\mu}_{\text{CVS}}$ as estimators of out-of-source error and the estimators of θ_{CVR} and θ_{CVS} on a data set of product reviews from Amazon (Blitzer et al., 2007), which is frequently used as a benchmark data set in domain adaptation. The data contains reviews of products from 25 diverse domains corresponding to high-level categories on Amazon.com. The goal is to classify whether a review is positive or negative based on the review text. We take each product domain as being a separate source.

We experiment with the version of the data set which contains ten domains, each with 1000 positive and 1000 negative examples. We will use CV to estimate the prediction error of a simple naive Bayes classifier. (We have replicated these results with an SVM with a linear kernel.)

6.1. Bias and variance

First we measure the bias of $\hat{\mu}_{\text{CVR}}$ and $\hat{\mu}_{\text{CVS}}$ and compare them to the out-of-source error. To estimate this, we average over the set of training domains, the set of training instances, the test domain and the test instances. To get this, we first sample without replacement a given number of domains, keeping all of them but one as training domains and using the remaining one as a test domain. Given this, we sample 100 pairs consisting of training and test data sets, sampling data points from empirical distribution of the respective domains. Having these, we run CVR and CVS on the training domains, comparing the cross-validation estimate to the prediction on the out-of-source test set. Figure 2 shows this comparison as a function of the number of training sources K (left panel) and the

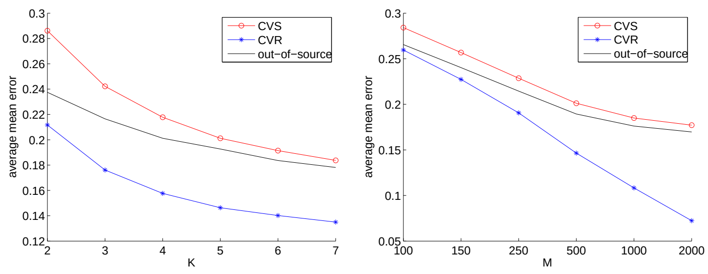


Figure 2. Values of $\hat{\mu}_{\text{CVR}}$ and $\hat{\mu}_{\text{CVS}}$ averaged over draws of training and test domains, compared to the true out-of-source error. Both plots were generated drawing domains 200 times.

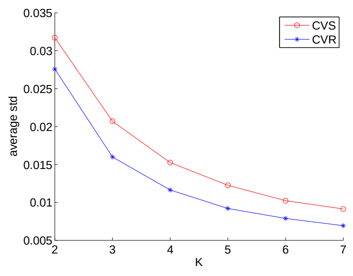


Figure 3. Standard deviation of $\hat{\mu}_{\text{CVR}}$ and $\hat{\mu}_{\text{CVS}}$ averaged draws of training and test domains. Experiment was done drawing domains 200 times.

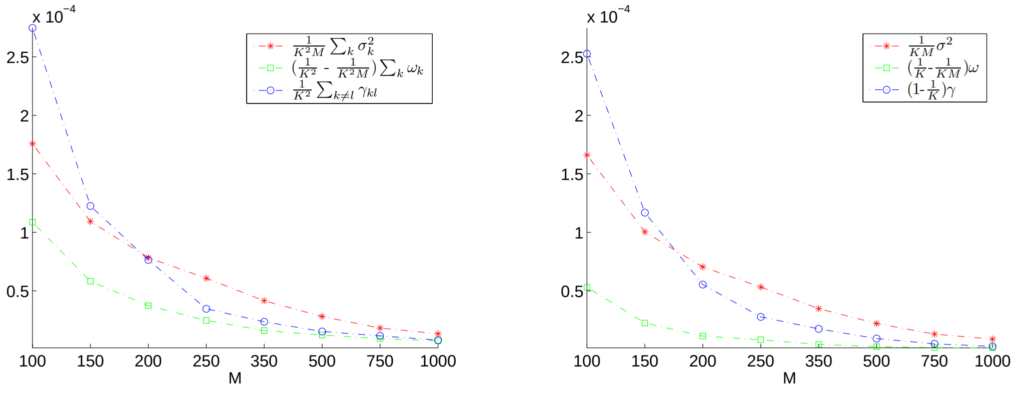


Figure 4. Decomposition of θ_{CVR} (right panel) and θ_{CVS} (left panel).

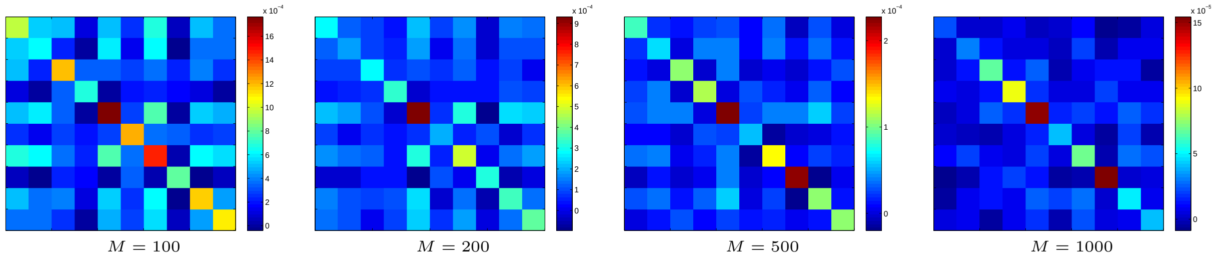


Figure 5. $\mathbb{V}[\mathbf{e}]$ for different M. σ_k^2's are typically a few orders of magnitude greater and were omitted here.

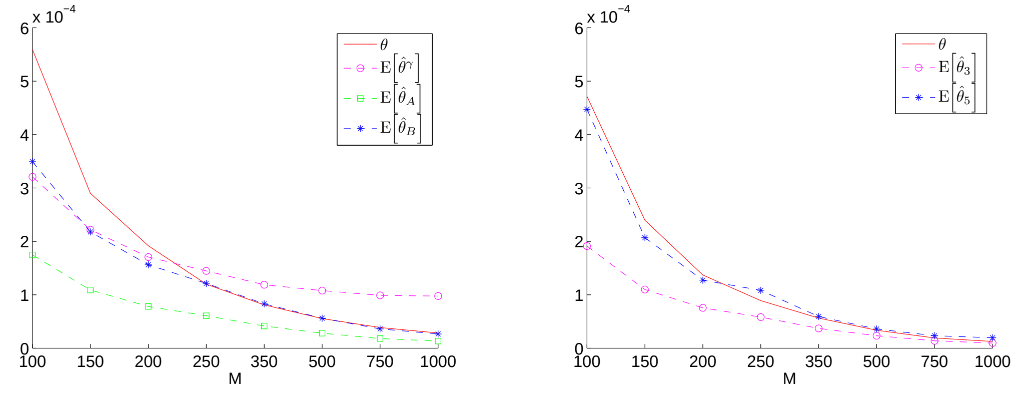


Figure 6. Comparison of estimators of θ_{CVR} (right panel) and θ_{CVS} (left panel).

number of points M per source (right panel). It can be clearly seen that CVS yields a better estimate of the out-of-source error than CVR. It is worth noting what happens when the number of training domains or the number of training data items gets larger. While CVS converges to the true out-of-source error, CVR converges to a different value. This confirms the analysis from Section 4.1. From these results, we can see that CVR yields an optimistic estimate of out-of-source error, even though the training set in each iteration of CV is smaller than the full training set. This is in contrast to CVS, which yields a more desirable pessimistic estimate.

Figure 3 shows the standard deviation of $\hat{\mu}_{\mathrm{CVS}}$ and $\hat{\mu}_{\mathrm{CVR}}$ averaged over the choice of training domains. To get this estimate, we have performed a similar procedure to Figure 2, i.e., for each draw of training domains, we sample 100 data sets, sampling data points from expirical distribution of the respective domains. Although the CVS estimate has higher variance, the variances are of the same order of magnitude and both tend to 0 for large data sets.

6.2. Variance decompositions and estimators of variance

In this section, we evaluate our new estimators of the variances θ_{CVR} and θ_{CVS}. To obtain these results, we used all domains as training domains. Estimating both θ_{CVR} and θ_{CVS} unbiasedly requires more than one independently sampled data set. To get them, we sample 1000 data sets, sampling from empirical distributions of each domain.

In the first experiment we estimated components of the decomposition of θ_{CVR} and θ_{CVS} (Figure 4). The quantities $\frac{1}{KM}\sigma^2$, $\frac{M-1}{KM}\omega$, $\frac{K-1}{K}\gamma$ and corresponding to them $\frac{1}{K^2M}\sum_k \sigma_k^2$, $\frac{M-1}{K^2M}\sum_k \omega_k$, $\frac{1}{K^2}\sum_{k\neq l}\gamma_{kl}$ have different magnitudes. Notice that $\frac{M-1}{KM}\omega$ is much larger than $\frac{M-1}{K^2M}\sum_k \omega_k$. It is not a surprise that CVS yields a strong positive correlation between errors made on points belonging to the same test block, which is not observed in CVR, where the test blocks are chosen randomly.

Secondly, we looked at $\mathbb{V}[\mathbf{e}]$ to see how much ω_k's and γ_{kl}'s vary (Figure 5). It can be seen that variation within γ_{kl}'s diminishes when M gets larger but variation within ω_k's does not change much.

Finally, we have tested estimators of θ_{CVR} and θ_{CVS}, which we have suggested in the earlier sections. The results are in Figure 6. For CVS, we compare $\hat{\theta}^\gamma$, $\hat{\theta}_A$ and $\hat{\theta}_B$. As expected, $\hat{\theta}^\gamma$ does not converge to 0 and grossly overestimates θ_{CVS} for large values of M. The new estimator $\hat{\theta}_A$ is closer to θ_{CVS} than $\hat{\theta}^\gamma$ for large values of M but its bias is consistently optimistic. On the other hand, for small M, $\hat{\theta}_B$ is not more optimistic than $\hat{\theta}^\gamma$ and for large M, while $\hat{\theta}^\gamma$ becomes very pessimistic, $\hat{\theta}_B$ has a negligible bias. Similarly, for the standard CVR setting, our new estimator $\hat{\theta}_5$ has lower bias than the previous estimators suggested by Grandvalet & Bengio (2006), being almost unbiased even for small M.

7. Related work

Various different versions of cross-validation have been analysed previously (Bengio & Grandvalet, 2003; Arlot & Celisse, 2010; Nadeau & Bengio, 2003; Markatou et al., 2005), but to our knowledge multiple-source cross-validation has not been previously analysed.

The idea of learning from a number of sources dates back to Caruana (1997) and Thrun (1996). A related issue in the context of covariate shift was suggested by Sugiyama et al. (2007), however, the resulting importance weights are difficult to estimate in practice (but see Gretton et al. (2009) for some work in this direction).

Rakotomalala et al. (2006) investigate the multiple-source cross-validation procedure empirically, but they do not perform theoretical analysis or present any estimators of θ_{CVS}. Work in domain adaptation also has considered the problem of bounding the prediction error when the training and test distribution have a different source (Ben-David et al., 2010). Unfortunately, the resulting bounds are too loose to be used for confidence intervals.

8. Conclusions

We have considered a cross-validation procedure for the multiple-source setting. We show that the bias of this procedure is better suited to this setting. We have presented several new estimates of the variance of the error estimate, both for the multiple-source cross-validation procedure and for the standard cross-validations setting, which perform well empirically.

Acknowledgments

The authors would like to thank Amos Storkey, Vittorio Ferrari, Simon Rogers and the anonymous reviewers for insightful comments on this work. This work was supported by the Scottish Informatics and Computer Science Alliance (SICSA).

References

Arlot, S. and Celisse, A. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. Wortman. A theory of learning from different domains. *Machine Learning*, 79:151–175, 2010.

Bengio, Y. and Grandvalet, Y. No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*, 5:1089–1105, 2003.

Blitzer, J., Dredze, M., and Pereira, F. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *In Proceedings of Association of Computational Linguistics*, 2007.

Caruana, R. Multi-task learning. *Machine Learning*, 28:41–75, 1997.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., and Slatteryy, S. Learning to extract symbolic knowledge from the world wide web. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, 1998.

Grandvalet, Y. and Bengio, Y. Hypothesis testing for cross-validation. Technical Report 1285, Département d'informatique et recherche opérationnelle, Université de Montréal, 2006.

Gretton, A., Smola, A., Huang, J., Schmittfull, M., Borgwardt, K., and Schölkopf, B. &. Covariate shift by kernel mean matching. In Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N.D. (eds.), *Dataset shift in machine learning*, pp. 131–160. The MIT Press, 2009.

Markatou, M., Tian, H., Biswas, S., and Hripcsak, G. Analysis of variance of cross-validation estimators of the generalization error. *Journal of Machine Learning Research*, 6:1127–1168, 2005.

Mitchell, T., Hutchinson, R., , Niculescu, R., Pereira, F., Wang, X., Justl, M., and Newman, S. Learning to decode cognitive states from brain images. *Machine Learning*, 57(1):145–175, 2004.

Nadeau, C. and Bengio, Y. Inference for the generalization error. *Machine Learning*, 52:239–281, 2003.

Rakotomalala, R., Chauchat, J.-H., and Pellegrino, F. Accuracy estimation with clustered dataset. In *In Proceedings of Australasian conference on Data mining and analytics*, 2006.

Stone, M. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147, 1974.

Sugiyama, M., Krauledat, M., and Müller, K.-R. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8:985–1005, 2007.

Thrun, S. Is learning the n-th thing any easier than learning the first? In *In Advances in Neural Information Processing Systems*, 1996.

3.2 Comments on the paper

In the paper, there were no assumptions on the learning algorithm, the loss and the data, so our results are as general as possible. There are a number of realistic assumptions that can be made. For example, we could assume that the loss is between 0 and 1. Making such assumptions could allow to extend this work by investigating what functions of hyperparameters of the CV procedure (the size of the training data, the number of random partitions) the elements of the variance decomposition are.

There are also a few general conjectures we have made on the elements of the variance decomposition, which are likely to be true but were not yet proven. If they are indeed provably true, they would be useful when deciding on which estimator to use depending on the context. The most interesting for the standard k-fold cross-validation are the following two (cf. Section 2 in the paper for definition of ω and γ), which characterise the behaviour of the elements of the variance decomposition as the function of the number of data points in the training set.

- **Conjecture 1.** *Denote ω for a training data set of size n by $\omega(n)$. The value of ω is strictly monotonically decreasing with n, i.e. for every n, $\omega(n) > \omega(n+1)$.*

- **Conjecture 2.** *Denote γ for a training data set of size n by $\gamma(n)$. The absolute value of γ is strictly monotonically decreasing with n, i.e. for every n, $|\gamma(n)| > |\gamma(n+1)|$*

Another possibility to extend this line of research would be to directly investigate estimators of the elements of the variance decomposition. For example it is possible to show that variance of $\hat{\gamma}\left(\frac{n}{2}\right) = \frac{1}{R}\sum_{r=1}^{R}\hat{\gamma}^{r}\left(\frac{n}{2}\right)$, a trivial unbiased estimator of $\gamma\left(\frac{n}{2}\right)$[1], is strictly decreasing with the number of times we resample the training set, R. The proof is as follows. Denote $\mathrm{Var}\left[\hat{\gamma}^{i}\left(\frac{n}{2}\right)\right]$ by α and $\mathrm{Cov}\left[\hat{\gamma}^{i}\left(\frac{n}{2}\right), \hat{\gamma}^{j}\left(\frac{n}{2}\right)\right]$ for $i \neq j$ by β. It is easy to verify that $\mathrm{Var}\left[\hat{\gamma}\left(\frac{\mathrm{n}}{2}\right)\right] = \beta + \frac{\alpha-\beta}{R}$. From the Cauchy-Schwartz inequality it follows that $\alpha \geq \beta$, hence $\mathrm{Var}\left[\hat{\gamma}\left(\frac{\mathrm{n}}{2}\right)\right]$ is decreasing in R.

[1]The estimator of $\gamma\left(\frac{n}{2}\right)$ is defined as $\hat{\gamma}\left(\frac{n}{2}\right) = \frac{1}{n^2\left(\frac{1}{4}-\frac{1}{K}\right)}\sum_{k=1}^{K}\sum_{l=1,l\neq k}^{K}\sum_{i\in T_k}\sum_{j\in T_l}\hat{\gamma}_{ij}$, where K is the number of partitions of the data, T_k is the kth partition of the data and $\hat{\gamma}_{ij} = \sum_{c=1}^{2}\left((e_{ci})(e_{cj}) - \left(\frac{1}{2}\sum_{c=1}^{2}e_{ci}\right)\left(\frac{1}{2}\sum_{c=1}^{2}e_{cj}\right)\right)$. The index c indicates which half of the data set is used. Analogous estimators can be created dividing the data set into more than two independent subsets.

Although the above statements are for the standard k-fold cross-validation, they can be easily translated to multiple-cross validation too.

Finally, studying properties of cross-validation for models whose predictions have an analytic form, such as Gaussian Processes [Rasmussen and Williams, 2005], would be interesting.

Chapter 4

Scheduled denoising autoencoders

4.1 Introduction to the paper

In this work we introduce a new procedure for training a denoising autoencoder, which is a commonly used model for unsupervised learning. Our method of training allows to learn a more diverse set of features than the standard training procedure. We call this model a scheduled denoising autoencoder (ScheDA), because it is using a sequence of decreasing levels of corruption. The features learnt by this model capture patterns of multiple scales, which improves accuracy when using these features in a supervised learning task. To understand this result, we first show that forming a representation by concatenating diverse sets of features is better than concatenating similar sets of features, even if learnt independently. Secondly, we verify that the representation we learnt is, indeed, more diverse by comparing the features in it to the features learnt with various levels of noise and find that elements of the representation learnt by ScheDA resemble elements of representations learnt with a broad range of noise levels.

We also draw a connection between annealing the level of noise in the denoising autoencoder and curriculum learning. This gives us another perspective on our results and also can be generalised to a broad class of unsupervised learning neural network models.

SCHEDULED DENOISING AUTOENCODERS

Krzysztof J. Geras
School of Informatics
University of Edinburgh
k.j.geras@sms.ed.ac.uk

Charles Sutton
School of Informatics
University of Edinburgh
csutton@inf.ed.ac.uk

ABSTRACT

We present a representation learning method that learns features at multiple different levels of scale. Working within the unsupervised framework of denoising autoencoders, we observe that when the input is heavily corrupted during training, the network tends to learn coarse-grained features, whereas when the input is only slightly corrupted, the network tends to learn fine-grained features. This motivates the *scheduled denoising autoencoder*, which starts with a high level of noise that lowers as training progresses. We find that the resulting representation yields a significant boost on a later supervised task compared to the original input, or to a standard denoising autoencoder trained at a single noise level. After supervised fine-tuning our best model achieves the lowest ever reported error on the CIFAR-10 data set among permutation-invariant methods.

1 INTRODUCTION

In most applications of representation learning, we wish to learn features at different levels of scale. For example, in image data, some edges will span only a few pixels, whereas others, such as a boundary between foreground and background, will span a large portion of the image. Similarly, in text data, some features in the representation might model specialized topics that use only a few words. For example a topic about electronics would often use words such as “big”, “screen” and “tv”. Other features model more general topics that use many different words. Good representations should model both of these phenomena, containing features at different levels of granularity.

Denoising autoencoders (Vincent et al., 2008; 2010; Glorot et al., 2011a) provide a particularly natural framework in which to formalise this intuition. In a denoising autoencoder, the network is trained so as to be able to reconstruct each data point from a corrupted version. The noise process used to perform the corruption is chosen by the modeller, and is an important tuning parameter that affects the final representation. On a digit recognition task, Vincent et al. (2010) noticed that using a low level of noise leads to learning blob detectors, while increasing it results in obtaining detectors of strokes or parts of digits. They also recognise that either too low or too high level of noise harms the representation learnt. The relationship between the level of noise and spatial extent of the filters was also noticed by Karklin and Simoncelli (2011) for a different feature learning model. Despite impressive practical results with denoising autoencoders, e.g. Glorot et al. (2011b), Mesnil et al. (2012), the choice of noise distribution is a tuning parameter whose effects are not fully understood.

In this paper, we introduce *scheduled denoising autoencoders* (ScheDA), which are based on the intuition that by training the same network at multiple noise levels, we can encourage it to learn features at different scales. The network is trained with a schedule of gradually decreasing noise levels. At the initial, high noise levels, the training data is highly corrupted, which forces the network to learn more global, coarse-grained features of the data. At low noise levels, the network is able to learn features for reconstructing finer details of the training data. At the end of the schedule, the network will include a combination of both coarse-grained and fine-grained features.

This idea is reminiscent of continuation methods, which have also been applied to neural networks (Bengio et al., 2009). The motivation of this work is significantly different though. Our goal is to encourage the network to learn a more diverse set of features, some which are similar to features learnt at the initial noise level, and others which are similar to features learnt at the final noise level. In Section 4.1.3, we verify quantitatively that this happens.

Experimentally, we find on both image and text data that scheduled denoising autoencoders learn better representations than standard denoising autoencoders, as measured by the features' performance on a supervised task. On both classification tasks, the representation from ScheDA yields lower test error than that from a denoising autoencoder trained at the best single noise level. After supervised fine-tuning our best ScheDA model achieves the lowest ever reported error on the CIFAR-10 data set among permutation-invariant methods.

2 Background

The core idea of learning a representation by learning to reconstruct artificially corrupted training data dates back at least to the work of Seung (1998), who suggested using a recurrent neural network for this purpose. Using unsupervised layer-wise learning of representations for classification purposes appeared later in the work of Bengio et al. (2007) and Hinton et al. (2006).

The denoising autoencoder (DA) (Vincent et al., 2008) is based on the same intuition as the work of Seung (1998) that that a good representation should contain enough information to reconstruct corrupted versions of an original input. Let $\mathbf{x} \in \mathbb{R}^d$ be the input to the network. The output of the network is a hidden representation $\mathbf{y} \in \mathbb{R}^{d'}$, which is simply computed as $f_\theta(\mathbf{x}) = s(\mathbf{W}\mathbf{x} + \mathbf{b})$, where the matrix $\mathbf{W} \in \mathbb{R}^{d' \times d}$ and the vector $\mathbf{b} \in \mathbb{R}^{d'}$ are the parameters of the network, and s is a typically nonlinear transfer function, such as a sigmoid. We write $\theta = (\mathbf{W}, \mathbf{b})$. The function f is called an *encoder* because it maps the input to a hidden representation. In an autoencoder, we have also a *decoder* that "reconstructs" the input vector from the hidden representation, which is used when training the network. The decoder has a similar form to the encoder, namely, $g_{\theta'}(\mathbf{y}) = t(\mathbf{W}'\mathbf{y} + \mathbf{b}')$, except that here $\mathbf{W}' \in \mathbb{R}^{d \times d'}$ and $\mathbf{b}' \in \mathbb{R}^d$. It can be useful to allow the transfer function t for the decoder to be different from that for the encoder. Typically, $\mathbf{W}$ and $\mathbf{W}'$ are constrained by $\mathbf{W}' = \mathbf{W}^T$ by analogy to the interpretation of principal components analysis as a linear encoder and decoder.

During training, our objective is to learn the encoder parameters $\mathbf{W}$ and $\mathbf{b}$. As a byproduct, we will need to learn the decoder parameters $\mathbf{b}'$ as well. We do this by defining a *noise distribution* $p(\tilde{\mathbf{x}}|\mathbf{x}, \nu)$. The amount of corruption is controlled by a parameter ν. We train the autoencoder weights to be able to reconstruct a random input from the training distribution $\mathbf{x}$ from its corrupted version $\tilde{\mathbf{x}}$ by running the encoder and the decoder in sequence. Formally, this process is described by the objective function

$$\theta^*, \theta'^* = \arg\min_{\theta, \theta'} \mathbb{E}_{(X, \tilde{X})} \left[L\left(X, g_{\theta'}(f_\theta(\tilde{X})) \right) \right], \tag{1}$$

where L is a loss function over the input space, such as squared error. Typically we minimize this objective function using stochastic gradient descent with mini-batches, where at each iteration we sample new values for both the uncorrupted and corrupted inputs.

In the absence of noise, this model is known simply as an autoencoder or autoassociator. A classic result (Baldi and Hornik, 1989) states that when $d' < d$, then under certain conditions, an autoencoder learns the same subspace as PCA. If the dimensionality of the hidden representation is too large, i.e., if $d' > d$, then the autoencoder can obtain zero reconstruction error simply by learning the identity map. In a denoising autoencoder, in contrast, the noise forces the model to learn interesting structure even when there are a large number of hidden units. Indeed, in practical denoising autoencoders, often the best results are found with *overcomplete* representations for which $d' > d$.

There are several tuning parameters here, including the noise distribution, the transformations s and t and the loss function L. For the loss function L, for continuous $\mathbf{x}$, squared error can be used. For binary $\mathbf{x}$ or $\mathbf{x} \in [0, 1]$, as we consider in this paper, it is common to use the *cross entropy* loss,

$$L(\mathbf{x}, \mathbf{z}) = -\sum_{i=1}^{D} \left(x_i \log z_i + (1 - x_i) \log (1 - z_i) \right).$$

For the transfer functions, common choices include the sigmoid $s(v) = \frac{1}{1+e^{-v}}$ for both the encoder and decoder, or to use a rectifier $s(v) = \max(0, v)$ in the encoder paired with sigmoid decoder.

One of the most important parameters in a denoising autoencoder is the noise distribution p. For continuous $\mathbf{x}$, Gaussian noise $p(\tilde{\mathbf{x}}|\mathbf{x}, \nu) = N(\tilde{\mathbf{x}}; \mathbf{x}, \nu)$ can be used. For binary $\mathbf{x}$ or $\mathbf{x} \in [0, 1]$, it is most common to use *masking noise*, that is, for each $i \in 1, 2, \dots d$, we sample $\tilde{x}_i$ independently as

$$p(\tilde{x}_i|x_i, \nu) = \begin{cases} 0 & \text{with probability } \nu, \\ x_i & \text{otherwise.} \end{cases} \tag{2}$$

In either case, the level of noise ν affects the degree of corruption of the input. If ν is high, the inputs are more heavily corrupted during training. The noise level has a significant effect on the representations learnt. For example, if the input data are images, masking only a few pixels will bias the process of learning the representation to deal well with local corruptions. On the other hand, masking very many pixels will push the algorithm to use information from more distant regions.

It is also possible to train multiple layers of representations with denoising autoencoders by training a denoising autoencoder with data mapped to a representation learnt by another denoising autoencoder. This model is known as the stacked denoising autoencoder (Vincent et al., 2008; 2010).

3 Scheduled denoising autoencoders

Our goal is to learn a single representation that combines the best aspects of representations learnt at different levels of noise. The scheduled denoising autoencoder (ScheDA) aims to do this by training a single DA sequentially using a *schedule* of noise levels, such that $\nu_0 > \cdots > \nu_T \geq 0$. The initial noise level ν_0 is chosen to be a high noise level that corrupts most of the input. The final noise level ν_T is chosen to be lower than the optimal noise level for a standard DA, i.e., chosen via a held-out validation set or by cross-validation. In pseudocode,

```
while $\theta$ not converged do
  Take a stochastic gradient step on (1), using noise level $\nu_0$.
end while
for $t$ in $1, \dots, T$ do
  $\nu_t := \nu_{t-1} - \Delta\nu$
  for $K$ steps do
    Take a stochastic gradient step on (1), using noise level $\nu_t$.
  end for
end for
```

This method is reminiscent of deterministic annealing (Rose, 1998), which has been applied to clustering problems, in which a sequence of clustering problems are solved at a gradually lowered noise level. However, the meaning of "noise level" is very different. In deterministic annealing, the noise is added to the mapping between inputs and cluster labels. This is to encourage data points to move between cluster centroids early in the optimization process.

ScheDA is also conceptually related to curriculum learning (Bengio et al., 2009) and continuation methods more generally (Allgower and Georg, 1980). In curriculum learning, the network is trained on a sequence of learning problems that have the property that the earlier tasks are "easier" than later tasks. In ScheDA, it is less obvious that the earlier tasks are easier since the lowest achievable reconstruction error is actually higher at the earlier high noise levels than at the later low noise levels. We observe this in practice (cf. Figure 1). On the other hand, we found that, for a given learning rate, the reconstruction error converges to a local minimum faster with large ν's (cf. the right panel of Figure 1). Thus, even though the problems that ScheDA starts with are harder in absolute terms, finding the local minima for these problems is easier. This can be understood given the insight provided by the work of Vincent (2011), who has shown that, for a DA trained with Gaussian noise and squared error, minimising reconstruction error is equivalent to matching the score (with respect to the input) of a nonparametric Parzen density estimator of the data, which depends on the level of noise. An implication of this viewpoint is that if the density learnt by the Parzen density estimator is harder to represent, it makes the DA learning problem harder too. Convolving the data with a high level of noise transforms the data generating distribution into a much smoother one, which is easier to capture. As the noise level is reduced, the density becomes more multimodal and harder to represent.

Table 1: Test errors on CIFAR-10 data set. Each ScheDA is characterised by the sequence of noise levels it was trained with and the number of epochs for which it was trained at each level of noise after the first noise level switch. Each row shows the best DA and the best ScheDA for a given number of hidden units, choosing the learning rate, the number of training epochs and the noise level using the error on the validation set.

hidden units	**best DA**	**test error**	**best ScheDA**	**test error**
1000	0.4	45.34%	0.4→0.3→0.2, K=50	43.01%
2000	0.3	41.95%	0.7→0.65→ … →0.2→0.15, K=100	40.1%
5000	0.1	38.64%	0.2→0.15→0.1→0.05, K=50	36.77%

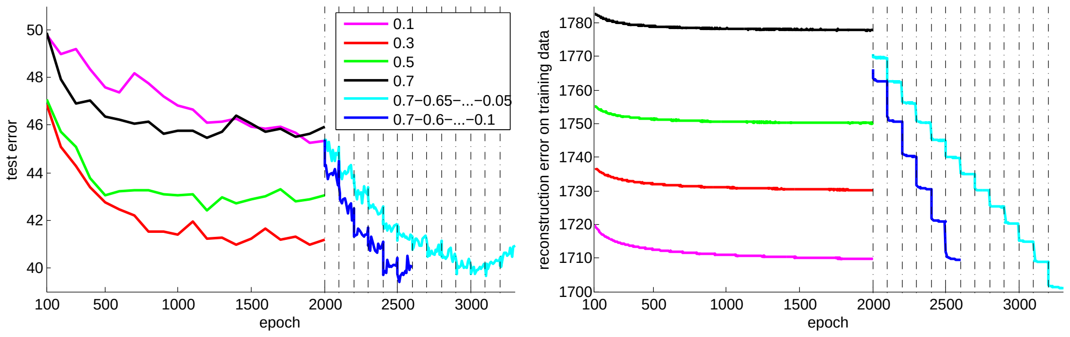


Figure 1: Experimental results with CIFAR-10 for 2000 hidden units. Test errors (left) and reconstruction errors on training set (right) as a function of the number of epochs. Dashed lines indicate a point when the level of noise was changed. Test errors were measured every 100 training epochs initially (during the first 2000 epochs). After each change of the noise level, test error was measured after the first, the third, the fifth epoch and then after every ten epochs. Reconstruction errors were measured after each training epoch for data corrupted with the noise level used for training at that epoch. For clarity, we do not show the results of DA (0.2), DA (0.4) and DA (0.6), which yield higher test error than DA (0.3).

4 Experiments

We evaluate ScheDA on two different data sets, an image classification data set (CIFAR-10), and a text classification data set (Amazon product reviews, results in the supplementary material). We use a procedure similar to one used, for example, by Coates et al. (2011)[1]. That is, in all experiments, we first learn the representation in an unsupervised fashion and then use the learnt representation within a linear classifier as a measure of its quality. In both experiments, in the unsupervised feature learning stage, we use masking noise as the corruption process, a sigmoid encoder and decoder and cross entropy loss (Equation 2)[2] following Vincent et al. (2008; 2010). All experiments with learning the representations were implemented using the Theano library (Bergstra et al., 2010). To do optimisation, we use stochastic gradient descent with mini-batches. For the classification step, we use $L2$-regularised logistic regression implemented in LIBLINEAR (Fan et al., 2008), with the regularisation parameter chosen to minimise the validation error.

4.1 Image recognition

We use the CIFAR-10 (Krizhevsky, 2009) data set for experiments with vision data. This data set consists of 60000 colour images spread evenly between ten classes. There are 50000 training and validation images and 10000 test images. Each image has a size of 32x32 pixels and each pixel has three colour channels, which are represented with a number in $\{0, \ldots, 255\}$. We divide the training and validation set into 45000 training instances and 5000 validation instances. The only preprocessing step we use is dividing the intensity of every pixel by 255 to get numbers in $[0, 1]$.

[1]We do not use any form of pooling, keeping our setup invariant to the permutation of the features.

[2]We also tried a rectified linear encoder combined with sigmoid decoder on the Amazon data set. The results were very similar, so we do not show them here.

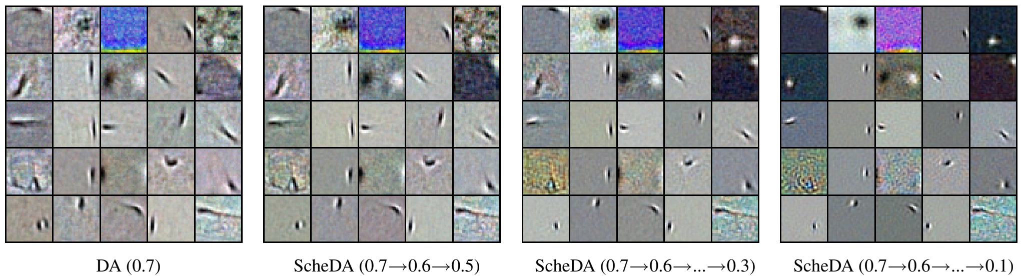


Figure 2: A sample of filters (rows of the matrix $\mathbf{W}$) learnt from CIFAR-10 with DA (0.7) and ScheDAs starting with $\nu_0 = 0.7$. All sets of filters are similar, but those that were post-trained with low level of noise are sharper. With schedules that end at a lower level of noise, the filters become more local but not as much as when only training with a low level of noise (cf. Figure 3).

To get the strongest possible DAs trained with a single noise level, we choose the noise level, learning rate and number of training epochs in order to minimise classification error on the validation set. We try all combinations of the following values of the parameters: noise level $\in \{0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.05\}$, learning rate $\in \{0.002, 0.01, 0.05\}$, number of training epochs $\in \{100, 200, \dots, 2000\}$. We choose these parameters separately for each size of the hidden layer $\in \{1000, 2000, 5000\}$.

To train ScheDA models, we first pick the best DA for each level of noise we consider, optimising the learning rate and the number of training epochs with respect to the validation error. Starting from the best DA for given ν_0, we continue the training, lowering the level of noise from ν_{t-1} to $\nu_t := \nu_{t-1} - \Delta\nu$ and training the model for K epochs. We repeat this noise reduction T times. In our experiments we consider $\Delta\nu \in \{0.05, 0.1\}$ and $K \in \{50, 100\}$. We use the learning rate of 0.01 for this stage as it turned out to always be optimal or very close to optimal for the standard DA[3]. We pick the combination of parameters $(\nu_0, \Delta\nu, K)$ and the number of noise reduction steps, T, using the validation error of a classifier after the last training epoch at each level of noise ν_t. We denote a DA trained with the level of noise ν by DA (ν) and ScheDA trained with a schedule of noise levels ν_0, ν_1, ..., ν_T by ScheDA (ν_0→ν_1→...→ν_T).

The error obtained by the classifier trained with raw pixel data equals 59.78%. A summary of the performance of DAs and ScheDAs for each number of hidden units can be found in Table 1. For each size of the hidden layer we tried, ScheDA easily outperforms DA, with a relative error reduction of about 5%. Our best model achieves the error of 36.77%. Interestingly, our method is very robust to the parameters $(\nu_0, \Delta\nu, K)$ of the schedule. See Section 4.1.1 for more details. Those results do not use supervised fine-tuning. Supervised fine-tuning of our best model yielded the error of 35.7%, which, to our knowledge, is the lowest ever reported error for permutation invariant CIFAR-10, outperforming Le et al. (2013) who achieved the error of 36.9% and Memisevic et al. (2014), who achieved the error of 36.1%. We describe the details of our supervised fine-tuning procedure in the supplementary material.

Figure 1 shows the test errors and reconstruction errors on the training data as a function of the training epoch for selected DAs and ScheDAs with 2000 hidden units. It is worth noting that, even though the schedules exhibiting the best performance go below the optimal ν for DA, training for many iterations with a level of noise that is too low hurts performance (see the final parts of the schedules shown in Figure 1). This may be due to the fact that structures learnt at low noise levels are too local to help generalisation.

The performance of our method does not appear to be solely due to better optimisation of the training objective. For example, DA (0.1) trained for 3000 epochs has a lower reconstruction error on the training data than the ScheDA (0.7→0.6→...→0.1) shown in Figure 1, while the test error it yields is higher by about 5%.

The features learnt with ScheDA are visibly noticeably different from those learnt with any single level of noise as they contain a mixture of features that could be found for various values of ν.

[3]Note that tuning this parameter could only help ScheDAs and would not affect the baselines.

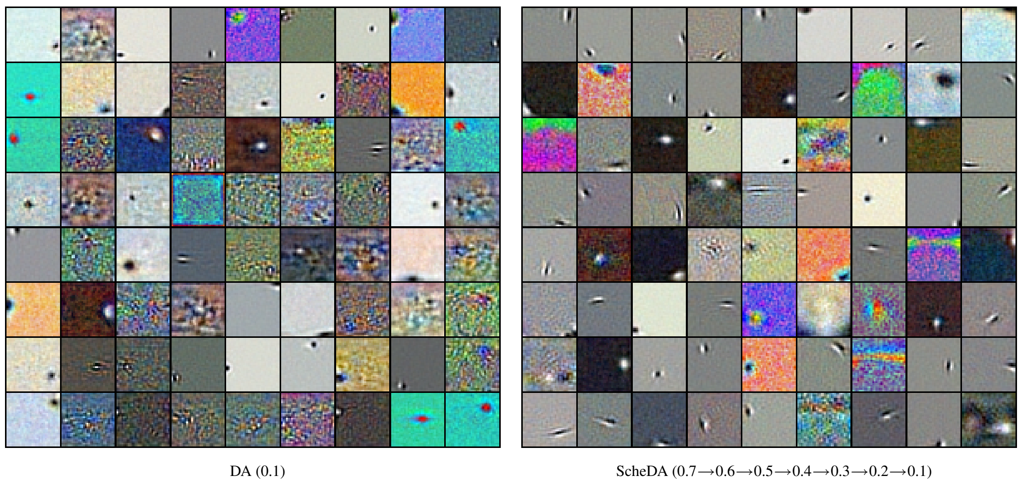


Figure 3: Samples of filters (rows of the matrix $\mathbf{W}$) learnt from CIFAR-10 with a low fixed noise level (left) and filters learnt with an initially high level of noise and post-trained with a schedule of lower levels of noise (right). These two sets of filters are visually very different. There are fewer edge detector filters among these learnt only with a low level of noise and those that are edge detectors are more local.

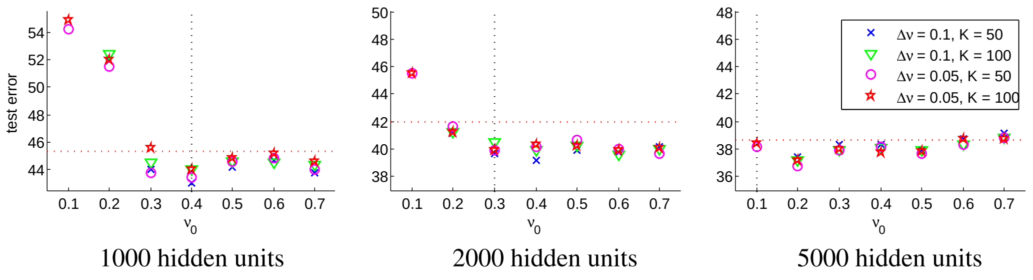


Figure 4: Comparison of different schedules for ScheDA. For each size of hidden layer, the vertical line indicates the optimal level of noise for a DA, i.e., the level of noise which allowed to train representation yielding the lowest error on the validation set, while the horizontal line indicates the test error obtained with this representation.

Figure 2 and Figure 3 display visualisations of the filters learnt by a DA and ScheDA. It can be seen in Figure 2 that when training ScheDA the features across the consecutive levels of noise are similar, which indicates that it is the initial training with a higher level of noise that puts the optimisation procedure in a basin of attraction of a different local minimum, which would not be otherwise achievable. This is shown in Figure 3, which visualises features learnt by a DA trained only with a low noise level, DA (0.1), and those learnt with ScheDA (0.7→0.6→...→0.1). The set of features learnt by DA (0.1) contains more noisy features and very few edge detectors, which are all very local. In contrast, features learnt with the schedule contain a more diverse set of edge detectors which can be learnt with high noise level (Figure 2) as well as some blob detectors which can be learnt with a low noise level (Figure 3).

4.1.1 Robustness to the choice of schedule

Our method is very robust to the choice of the parameters of the schedule, ν_0, $\Delta\nu$ and K. Figure 4 shows the performance of ScheDA for different values of those parameters. For 1000 and 2000 hidden units for all schedules ScheDA performed better than the best DA, as long as the initial level of noise ν_0 was not lower than the level of noise yielding the best DA. For 5000 hidden units, ScheDA also performed better than DA, except for the model trained with $\nu_0 = 0.7$. These results suggest than ScheDA's performance is superior to DA as long as the initial level of noise is not too large and not below the optimal level of noise for DA.

We also examined whether it is necessary for the schedule to be decreasing. To investigate this, we trained ScheDAs using the same procedure as before, except that the levels of noise were increasing. The networks had 2000 hidden units and they started with the best DA (over the learning rate and the number of training epochs) for each ν_0. We considered $\nu_0 \in \{0.1, 0.2, 0.3, 0.4\}$, $K \in \{50, 100\}$ and $\Delta\nu \in \{0.05, 0.1\}$. The largest possible final noise level ν_T was 0.7. To evaluate all combinations of these hyperparameters (ν_0, ν_T, $\Delta\nu$ and K) we used the validation set. We set the learning rate to 0.01 as it worked optimally or very close to optimally in all previous experiments and we also used this value in the experiments with decreasing schedules. The best model we obtained this way used the schedule 0.3→0.35 and $K = 100$. Its test error was 41.97%, just a little worse than a DA trained with $\nu = 0.3$ (achieving the test error of 41.95%). For comparison, ScheDA (0.1→0.2→0.3) with $K = 100$ yielded the test error of 44.99% and ScheDA (0.3→0.4→0.5→0.6→0.7) with $K = 100$ yielded the test error of 46.7%. These results provide some evidence that the initial noise levels puts the optimisation procedure in a basin of attraction of a local minimum that can be favourable, as we observe for ScheDA when starting training with higher noise levels, or detrimental, as we see here.

4.1.2 CONCATENATING SETS OF FEATURES LEARNT WITH DIFFERENT NOISE LEVELS

To explain the results above, we examine whether features learnt with different noise levels contain different information about the data. To explore this, we trained two sets of representations with 2000 hidden units independently with a standard DA. DAs in the first set were initialised with a randomly drawn set of parameters θ_1 and DAs in the second set were initialised with a different randomly drawn set of parameters θ_2. Each set contained representations learnt with $\nu = 0.1$, $\nu = 0.2$, ..., $\nu = 0.7$. Then we gathered all 49 possible pairs of representations between the two sets and concatenated representations within each pair, creating representations with 4000 features. The errors yielded by classifiers using these representations can be found in Figure 5. The important observation here is that, even though concatenating two representations learnt with the same ν but with different initialisations results in a better representation (cf. Figure 1), concatenating representations with different ν's yields even lower errors.

This is another piece of evidence strengthening our hypothesis that having both local and global features in the representation learnt with ScheDA helps classification. Note, however, that even though concatenating representations learnt with different ν helps, ScheDA is clearly a better model. For a fair comparison, we trained ScheDA with 4000 units using, which matches the number of hidden units in the concatenated architecture. While the best concatenated DA achieved 39.33% (cf. Figure 5), the best ScheDA achieved 37.77%.

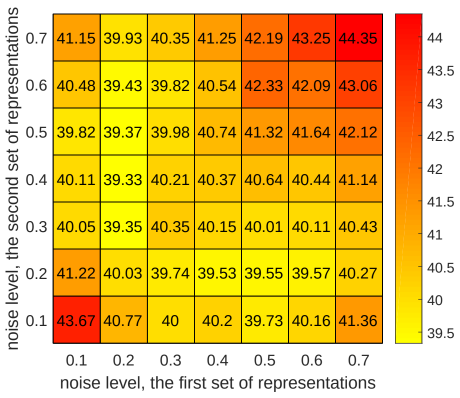


Figure 5: Test errors yielded by representations constructed by concatenating representations learnt with various levels of noise. This allows representations that are otherwise weak separately to achieve low test errors, e.g. for $\nu = 0.1$ and $\nu = 0.5$ (cf. Figure 1).

4.1.3 COMPARING SETS OF FEATURES

Having confirmed that using both features learnt with different levels of noise indeed helps classification, we experimentally verify the hypothesis that the final representation trained with ScheDA (0.7→0.6→...→0.1) contains both features similar to those learnt with low levels of noise (local features) and high levels of noise (global features).

Intuitively, two features are similar if they are active for the same set of inputs. We define the activation vector $\mathbf{a}_i$ for feature i as the vector containing the activation of the feature over all the data points. More formally, if $\mathbf{w}_i$ is the weight vector for feature i, b_i is the bias for feature i and $\mathbf{x}_n$ is a data item, the activation vector is $\mathbf{a}_i = [a_{i1}, \dots, a_{iN}]$, where $a_{in} = \mathrm{sigmoid}(\mathbf{w}_i \mathbf{x}_n + b_i)$. Here N is the total number of data items, the total number of features is I.

We compute the activation vector for all features from eight different autoencoders: DA (0.1), DA (0.2), ..., DA (0.7) and ScheDA (0.7→0.6→...→0.1). We denote the resulting activation vectors $\mathbf{a}_i^{0.1}$, ..., $\mathbf{a}_i^{0.7}$ and $\mathbf{a}_i^{S}$, respectively. Now for each feature in ScheDA (0.7→0.6→...→0.1) we can find the

Table 2: Comparison of features of ScheDA and DA. The first row shows how many ScheDA (0.7→0.6→...→0.1) features, out of 2000 in total, were closest to a feature learnt by DA (0.1), ..., DA (0.7). It demonstrates that ScheDA combines information that would be learnt from DAs at varying noise levels. The second and third row are baselines for comparison (see text for details).

	DA (0.1)	**DA (0.2)**	**DA (0.3)**	**DA (0.4)**	**DA (0.5)**	**DA (0.6)**	**DA (0.7)**
ScheDA	374	550	444	299	169	92	72
DA* (0.1)	1247	465	167	54	21	12	7
DA* (0.7)	25	30	72	165	308	587	813

closest feature among those learnt with DA (0.1), DA (0.2), ..., DA (0.7). To do this, we compute cosine similarities $\cos(\mathbf{a}_i^S, \mathbf{a}_j^{0.1})$, $\cos(\mathbf{a}_i^S, \mathbf{a}_j^{0.2})$, ..., $\cos(\mathbf{a}_i^S, \mathbf{a}_j^{0.7})$ for all pairs (i, j). Finally, we compute $\mathrm{C}_{0.1}$, the number of ScheDA features that are closest to a feature from DA (0.1) as $\mathrm{C}_{0.1} = \sum_{i=1}^{I} 1[\max_j \cos(\mathbf{a}_i^S, \mathbf{a}_j^{0.1}) > \{\max_j \cos(\mathbf{a}_i^S, \mathbf{a}_j^{0.2}), \max_j \cos(\mathbf{a}_i^S, \mathbf{a}_j^{0.3}), ..., \max_j \cos(\mathbf{a}_i^S, \mathbf{a}_j^{0.7})\}]$ and similarly for $\mathrm{C}_{0.2}$, $\mathrm{C}_{0.3}$, ..., $\mathrm{C}_{0.7}$. To see how much ScheDA differs in that respect from the standard DA trained only at the final level of noise for ScheDA, we also performed the same procedure as described above, but comparing to features learnt by DA* (0.1), which is the same as DA (0.1) but starting from a different random initialisation. We found that ScheDA contains more features similar to those learnt with higher noise levels than DA* (0.1) (see Table 2). This confirms our expectation that the ScheDA representation retains a large number of more global features from the earlier noise levels. We also put the same numbers for DA* (0.7) for comparison.

5 COMPOSITE DENOISING AUTOENCODER

The observation that more diverse representations lead to a better discriminative performance can be exploited more explicitly than in ScheDA. Instead of training all of the hidden units with a sequence of noise levels, we can partition the hidden units, training each subset of units with a different noise level. This can be done by defining the hidden representation and the reconstruction to be

$\mathbf{y} = [f(\tilde{\mathbf{x}}_{\nu_1}\mathbf{W}_1 + \mathbf{b}_1), \ldots, f(\tilde{\mathbf{x}}_{\nu_S}\mathbf{W}_S + \mathbf{b}_S)]$ and
$\mathbf{z} = g\left(\sum_{s=1}^{S} f(\tilde{\mathbf{x}}_{\nu_s}\mathbf{W}_s + \mathbf{b}_s)\mathbf{W}_s^T + \mathbf{b}'\right),$

where $\tilde{\mathbf{x}}_{\nu_s}$ denotes an input $\mathbf{x}$ corrupted with the level of noise ν_s. We call this a composite DA. Our preliminary experiments show that, even when using only two noise levels, it outperforms a standard DA and performs on par with ScheDA. Successful learning of the parameters is more complicated though. We found that standard SGD (updating all parameters at each epoch) performs much worse than a version of the SGD alternating between updating parameters associated with the two levels of noise. See Figure 6.

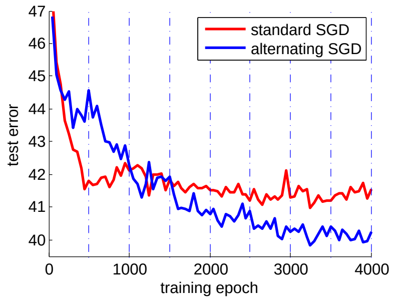


Figure 6: Test errors for a composite denoising autoencoder using two levels of noise, $\nu = 0.2$ and $\nu = 0.4$, and with 2000 hidden units divided equally between the two levels of noise. Dashed lines indicate the epochs when optimisation switched between updating different sets of parameters.

6 DISCUSSION

We have introduced a simple, yet powerful extension of an important and commonly used model for learning representations and justified its superior performance by its ability to learn a more diverse set of features than the standard denoising autoencoder. Instead of learning a denoising autoencoder with just a single level of noise, we exploit the fact that various levels of noise yield different features, which are more global for large values of ν. Starting the training with a high level of noise enables the algorithm to learn these global features first, which are partially retained when the level of noise is lower and the model is learning more local dependencies.

Erhan et al. (2010) investigated why unsupervised pretraining helps learning a deep neural network and found that the set of functions learnt by pretrained sigmoid neural networks is very different from the ones that are learnt without unsupervised pretraining. In fact, we have investigated a related question, *why does unsupervised pretraining help unsupervised pretraining?* Or, more precisely, since we are getting a large boost of performance even without supervised fine-tuning, *why does unsupervised pretraining help unsupervised training?* One of their conclusions was that, when using their architecture, unsupervised pretaining puts the optimisation procedure in a basin of attraction of a local minimum that would not otherwise be found. This is very similar to what we observe in our experiments. We often find that a DA trained with a given level of noise ν can have a lower reconstruction error than ScheDA trained with the final level of noise ν, yet ScheDA is performing better in terms of classification error. The filters at the minima for DA and ScheDA also look very different (cf. Figure 3).

This way of training a denosing autoencoder is related to walkback training (Bengio et al., 2013) in the sense that at the initial stages of training both methods attempt to correctly reconstruct corrupted examples that lie further from the data manifold. It is different though as we do not require the loss to be interpretable as log-likelihood and we do not perform any sampling from the denoising autoencoder. Additionally, Chandra and Sharma (2014) independently tried an idea similar to ScheDA, but they were unable to show consistent improvement over the results of Vincent et al. (2010).

There is a number of ways this work can be extended. Primarily, ScheDA can be stacked, which would likely improve our results. More generally, our results suggest that large improvements can be achieved by combining diverse representations, which we aim to exploit in composite denoising autoencoders.

Finally, we would like to point out that the main observation we make, namely, that it is beneficial for the feature learning algorithm to learn more global features first and then to proceed to learning more local ones, is very general and it is likely that *scheduling* is applicable to other approaches to feature learning. Indeed, in the case of dropout (Hinton et al., 2014), Rennie et al. (2014) have, independently from our work, explored the use of a schedule to decrease the dropout rate.

ACKNOWLEDGMENTS

We thank Amos Storkey, Vittorio Ferrari, Iain Murray, Chris Williams and Ruslan Salakhutdinov for insightful comments on this work.

REFERENCES

Eugene L. Allgower and Kurt Georg. *Numerical continuation methods. An introduction*. Springer-Verlag, 1980.

Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2, 1989.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, 2007.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, 2009.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *NIPS*, 2013.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. *SciPy*, 2010.

John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, Bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*, 2007.

B. Chandra and Rajesh Kumar Sharma. Adaptive noise schedule for denoising autoencoder. In *Neural Information Processing*, volume 8834 of *Lecture Notes in Computer Science*. 2014.

Adam Coates, Andrew Y. Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *JMLR*, 11, 2010.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9, 2008.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier networks. In *AISTATS*, 2011a.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML*, 2011b.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7), 2006.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *JMLR*, 15, 2014.

Yan Karklin and Eero P. Simoncelli. Efficient coding of natural images with a population of noisy linear-nonlinear neurons. In *NIPS*, 2011.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Quoc Le, Tamás Sarlós, and Alexander Smola. Fastfood-computing Hilbert space expansions in loglinear time. In *ICML*, 2013.

Roland Memisevic, Kishore Konda, and David Krueger. Zero-bias autoencoders and the benefits of co-adapting features. *arXiv:1402.3337v2*, 2014.

Grégoire Mesnil, Yann Dauphin, Xavier Glorot, Salah Rifai, Yoshua Bengio, Ian J. Goodfellow, Erick Lavoie, Xavier Muller, Guillaume Desjardins, David Warde-Farley, Pascal Vincent, Aaron C. Courville, and James Bergstra. Unsupervised and transfer learning challenge: a deep learning approach. In *ICML Unsupervised and Transfer Learning Workshop*, 2012.

Steven Rennie, Vaibhava Goel, and Samuel Thomas. Annealed dropout training of deep networks. *IEEE Workshop on Spoken Language Technology*, 2014.

Kenneth Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 86:2210–2239, 1998.

H. Sebastian Seung. Learning continuous attractors in recurrent networks. In *NIPS*, 1998.

Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23, 2011.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *JMLR*, 2010.

SUPPLEMENTARY MATERIAL

SAMPLING THE LEVEL OF NOISE

As an alternative to a schedule, which sequentially changes the level of noise, we tried to sample a different ν for each mini-batch. We tried two variants of this idea: sampling ν uniformly from a continuous interval $[0.1, 0.7]$, and sampling ν from a discrete distribution over the values in the set $\{0.1, 0.2, \ldots, 0.7\}$. Replicating the setup described in Section 4.1 for a DA with 2000 hidden units, the first method obtained test error of 44.85% and the second one obtained the test error 46.83%. Thus, both have performed much worse than DA (0.3). The result of this experiment provides evidence that training a denoising autoencoder with a sequence of noise levels is important for the success of our method.

SUPERVISED FINE-TUNING

For completeness, we also tried training a supervised single-layer neural network using parameters of the encoder as the initialisation of the parameters of the hidden layer of the network. We did that for all models in Table 1. That is, for each size of the hidden layer, we take the best DA and the best ScheDA trained in an unsupervised manner and perform supervised fine-tuning of their parameters. The learning rate, the same for all parameters, was chosen from the set $\{0.00125, 0.00125 \cdot 2^{-1}, \ldots, 0.00125 \cdot 2^{-4}\}$ and the maximum number of training epochs was 2000 (we computed the validation error after each epoch). We report the test error for the combination of the learning rate and the number of epochs yielding the lowest validation error. The numbers are shown in Table 3. Fine-tuning makes the performance of DA and ScheDA much more similar, but the advantage of ScheDA is consistent and its magnitude grows with the size of the hidden layer.

Table 3: Test errors on CIFAR-10 data set for the best DA and ScheDA models trained without supervised fine-tuning and their fine-tuned versions.

	DA		**ScheDA**	
hidden units	**no fine-tuning**	**fine-tuning**	**no fine-tuning**	**fine-tuning**
1000	45.34%	39.55%	43.01%	39.44%
2000	41.95%	36.85%	40.1%	36.22%
5000	38.64%	36.47%	36.77%	35.7%

SENTIMENT CLASSIFICATION

We also evaluate our idea on a data set of product reviews from Amazon (Blitzer et al., 2007), adapting the experimental setting used with the CIFAR-10 data set. The version of the data set we are using contains reviews of products from six domains[4] corresponding to high-level categories on Amazon.com. The goal is to classify whether a review is positive or negative based on the review text. For computational reasons, we keep only 3000 most popular words in the entire data set, transforming each example into a binary vector indicating presence or absence of a word. We divide the data set into a training set of 10000 labelled examples and 35000 unlabelled examples, a validation set of 10000 labelled examples and a test set of 10000 labelled examples, each of them consisting equal fractions of positive and negative labelled examples. The six domains are mixed among training, validation and test examples. We set the number of hidden units to 2000.

The baseline, logistic regression trained with raw data obtains the test error of 14.79%, while the best DA (0.6) yields 13.61% and the best ScheDA (0.7$\rightarrow$0.6) yields 13.41% error. The relative error reduction is smaller than on the image data, which is not surprising since the raw features are here a much stronger baseline and the improvement obtained by the standard DA is relatively smaller too. Smaller relative error reduction can be explained by the fact that the DA performance varies less with the level of noise for this data set. While the test error for the best set of features learnt by DA (0.6) was 13.61%, the worst, DA (0.1), yielded the error of 13.9%. This result suggests a simple diagnostic for whether ScheDA is likely to be effective, namely, to check whether the DA validation error is sensitive to the noise level.

[4]books, dvd, electronics, kitchen & housewares, music, video

4.2 Comments on the paper

Retrospectively, we consider interpreting scheduling the noise as a way of doing curriculum learning as one of the most interesting findings of this paper. After this work was published other researchers had similar insights. Gulcehre et al. [2016] found that adding noise to the activation functions in deep neural networks and then annealing the level of noise improves empirical performance and also interpreted annealing the level of noise as a curriculum learning method. Similarly, Neelakantan et al. [2015] showed empirically on a range of tasks that adding noise to the gradient of backpropagated error and annealing it improves performance of deep neural networks. We expect similar observations to be made in many other contexts in the future.

Another interesting practical observation from this work is that it appears that training denoising autoencoders to the limits of their performance requires considerably more training epochs than training supervised networks. Preliminary experiments we performed with CIFAR-10 indicate that, for the best hyperparameters found, a performance of a single-layer fully-connected network does not improve after less than a hundred epochs of training. On the other hand, as shown in Figure 1 in the paper, the performance of the denoising autoencoder keeps getting better for much longer. The reasons of this phenomenon remain to be understood.

In addition to the work mentioned in the paper, a few others also considered the problem of representation learning with models similar to the denoising autoencoder. Motivated by modelling learning of real neurons, Doi et al. [2006] considered an autoencoder with a linear encoder and a linear decoder, with noise added to the hidden representation. They showed that when the input has a dimensionality not greater than two the reconstruction error (mean squared error) can be minimised analytically based only on the statistics of the data and the noise. Doi and Lewicki [2014] extended that earlier model by adding deterministic blur and stochastic noise to the input. Interestingly, they found that their model fails to learn any interesting features without any additional penalty on the learnt representation. However, when penalising sparsity of the weights, sparsity of the hidden activations or spatial locality of the filters, the model managed to learn a variety of edge and blob detectors. They also found that spatial extent of the learnt filters grows with the amount of noise, which is consistent with our findings.

Furthermore, Poole et al. [2014] showed that, when using mean squared error to measure reconstruction error and a linear decoder, adding noise to the input, the hidden representation before or after the activation function are approximately equivalent.

Chapter 5

Composite denoising autoencoders

5.1 Introduction to the paper

The scheduled denoising autoencoder introduced in Chapter 4 learns a representation composed of features of multiple levels of granularity thanks to a sequence of decreasing noise levels it is trained with. While this method yields very good results and has an interesting connection to curriculum learning, it does not allow us to specify what fraction of the features we would like to learn at a given scale. This chapter extends the work in Chapter 4, introducing a different variant of the standard denoising autoencoder, which we call the composite denoising autoencoder (CDA), which is similar in its goal to the scheduled denoising autoencoder, but different in the way it achieves it. The main advantage of CDA over ScheDA is that the CDA directly encourages the network to build representations with features of multiple scales by having subsets of units which explicitly learn features using different levels of noise. This is an improvement, because it allows the experimenter to control the number of features of various scales explicitly.

It is worth noting that, the results achieved with the CIFAR-10 data set reported in this paper are extremely strong. In fact, the only work, which surpasses CDA on this data set without using convolutional networks is the paper by Urban et al. [2016], however they use a more complicated setup, which involves tuning a larger number of hyperparameters and they also use data augmentation.

Composite denoising autoencoders

Krzysztof J. Geras and Charles Sutton

Institute of Adaptive Neural Computation, School of Informatics,
The University of Edinburgh, Edinburgh, EH8 9AB, UK
k.j.geras@sms.ed.ac.uk, csutton@inf.ed.ac.uk

Abstract. In representation learning, it is often desirable to learn features at different levels of scale. For example, in image data, some edges will span only a few pixels, whereas others will span a large portion of the image. We introduce an unsupervised representation learning method called a composite denoising autoencoder (CDA) to address this. We exploit the observation from previous work that in a denoising autoencoder, training with lower levels of noise results in more specific, fine-grained features. In a CDA, different parts of the network are trained with different versions of the same input, corrupted at different noise levels. We introduce a novel cascaded training procedure which is designed to avoid types of bad solutions that are specific to CDAs. We show that CDAs learn effective representations on two different image data sets.

Keywords: denoising autoencoders, unsupervised learning, neural networks.

1 Introduction

In most applications of representation learning, we wish to learn features at different levels of scale. For example, in image data, some edges will span only a few pixels, whereas others, such as a boundary between foreground and background, will span a large portion of the image. Similarly, in speech data, different phonemes and different words vary a lot in their duration. In text data, some features in the representation might model specialized topics that use only a few words. For example a topic about electronics would often use words such as "big", "screen" and "tv". Other features model more general topics that use many different words. Good representations should model both of these phenomena, containing features at different levels of granularity.

Denoising autoencoders [28, 29, 12] provide a particularly natural framework to formalise this intuition. In a denoising autoencoder, the network is trained to be able to reconstruct each data point from a corrupted version. The noise process used to perform the corruption is chosen by the modeller, and is an important aspect of the learning process that affects the final representation. On a digit recognition task, Vincent et al. [29] noticed that using a low level of noise leads to learning blob detectors, while increasing it results in obtaining detectors of strokes or parts of digits. They also recognise that either too low or too high level of noise harms the representation learnt. The relationship between the level of noise and spatial extent of the filters was also noticed by Karklin and Simoncelli [18] for a different feature learning model. Despite impressive practical results with denoising autoencoders (e.g. [13, 23]), how to choose the noise distribution is not fully understood.

In this paper, we introduce *composite denoising autoencoders* (CDA), in which different parts of the network receive versions of the input that are corrupted with different levels of noise. This encourages different hidden units of the network to learn features at different scales. A key challenge is that finding good parameters in a CDA requires some care, because naive training methods will cause the network to rely mostly on the low-noise corruptions, without fully training the features for the high-noise corruptions, because after all the low noise corruptions provide more information about the original input. We introduce a training method specifically for CDA that sidesteps this problem.

On two different data sets of images, we show that CDAs learn significantly better representations that standard DAs. In particular, we achieve to our knowledge the best accuracy on the CIFAR-10 data set with a permutation invariant model, outperforming scheduled denoising autoencoders [10].

2 Background

The core idea of learning a representation by learning to reconstruct artificially corrupted training data dates back at least to the work of Seung [24], who suggested using a recurrent neural network for this purpose. Using unsupervised layer-wise learning of representations for classification purposes appeared later in the work of Bengio et al. [3] and Hinton et al. [16].

The denoising autoencoder (DA) [28] is based on the same intuition as the work of Seung [24] that a good representation should contain enough information to reconstruct corrupted versions of an original input. In its simplest form, it is a single-layer feed-forward neural network. Let $\mathbf{x} \in \mathbb{R}^d$ be the input to the network. The output of the network is a hidden representation $\mathbf{y} \in \mathbb{R}^{d'}$, which is simply computed as $f_\theta(\mathbf{x}) = h(\mathbf{W}\mathbf{x} + \mathbf{b})$, where the matrix $\mathbf{W} \in \mathbb{R}^{d' \times d}$ and the vector $\mathbf{b} \in \mathbb{R}^{d'}$ are the parameters of the network, and h is a typically nonlinear transfer function, such as a sigmoid. We write $\theta = (\mathbf{W}, \mathbf{b})$. The function f is called an *encoder* because it maps the input to a hidden representation. In an autoencoder, we have also a *decoder* that "reconstructs" the input vector from the hidden representation. The decoder has a similar form to the encoder, namely, $g_{\theta'}(\mathbf{y}) = h'(\mathbf{W}'\mathbf{y} + \mathbf{b}')$, except that here $\mathbf{W}' \in \mathbb{R}^{d \times d'}$ and $\mathbf{b}' \in \mathbb{R}^d$. It can be useful to allow the transfer function h' for the decoder to be different from that for the encoder. Typically, $\mathbf{W}$ and $\mathbf{W}'$ are constrained by $\mathbf{W}' = \mathbf{W}^T$, which has been justified theoretically by Vincent [27].

During training, our objective is to learn the encoder parameters $\mathbf{W}$ and $\mathbf{b}$. As a byproduct, we will need to learn the decoder parameters $\mathbf{b}'$ as well. We do this by defining a *noise distribution* $p(\tilde{\mathbf{x}}|\mathbf{x}, \nu)$. The amount of corruption is controlled by a parameter ν. We train the autoencoder weights to be able to reconstruct a random input from the training distribution $\mathbf{x}$ from its corrupted version $\tilde{\mathbf{x}}$ by running the encoder and the decoder in sequence. Formally, this process is described by minimising the autoencoder reconstruction error with respect to the parameters θ^* and θ'^*, i.e.,

$$\theta^*, \theta'^* = \arg\min_{\theta,\theta'} \mathbb{E}_{(X,\tilde{X})} \left[L\left(X, g_{\theta'}(f_\theta(\tilde{X})) \right) \right], \tag{1}$$

where L is a loss function over the input space, such as squared error. Typically we minimize this objective function using SGD with mini-batches, where at each iteration we sample new values for both the uncorrupted and corrupted inputs.

In the absence of noise, this model is known simply as an autoencoder or autoassociator. A classic result [2] states that when $d' < d$, then under certain conditions, an autoencoder learns the same subspace as PCA. If the dimensionality of the hidden representation is too large, i.e., if $d' > d$, then the autoencoder can obtain zero reconstruction error simply by learning the identity map. In a denoising autoencoder, in contrast, the noise forces the model to learn interesting structure even when there are a large number of hidden units. Indeed, in practical denoising autoencoders, the best results are found with *overcomplete* representations for which $d' > d$.

There are several choices to be made here, including the noise distribution, the transformations h and h' and the loss function L. For the loss function L, for continuous $\mathbf{x}$, squared error can be used. For binary $\mathbf{x}$ or $\mathbf{x} \in [0, 1]$, as we consider in this paper, it is common to use the *cross entropy* loss,

$$L(\mathbf{x}, \mathbf{z}) = -\sum_{i=1}^{D} \left(x_i \log z_i + (1 - x_i) \log (1 - z_i) \right).$$

For the transfer functions, common choices include the sigmoid $h(v) = \frac{1}{1+e^{-v}}$ for both the encoder and decoder, or to use a rectifier $h(v) = \max(0, v)$ in the encoder paired with sigmoid decoder.

One of the most important parameters in a denoising autoencoder is the noise distribution p. For continuous $\mathbf{x}$, Gaussian noise $p(\tilde{\mathbf{x}}|\mathbf{x}, \nu) = N(\tilde{\mathbf{x}}; \mathbf{x}, \nu)$ can be used. For binary $\mathbf{x}$ or $\mathbf{x} \in [0, 1]$, it is most common to use *masking noise*, that is, for each $i \in 1, 2, \ldots d$, we sample $\tilde{x}_i$ independently as

$$p(\tilde{x}_i | x_i, \nu) = \begin{cases} 0 & \text{with probability } \nu, \\ x_i & \text{otherwise.} \end{cases} \tag{2}$$

In either case, the level of noise ν affects the degree of corruption of the input. If ν is high, the inputs are more heavily corrupted during training. The noise level has a significant effect on the representations learnt. For example, if the input data are images, masking only a few pixels will bias the process of learning the representation to deal well with local corruptions. On the other hand, masking many pixels will push the algorithm to use information from more distant regions.

It is possible to train multiple layers of representations with denoising autoencoders by training a denoising autoencoder with data mapped to a representation learnt by the encoder of another denoising autoencoder. This model is known as the stacked denoising autoencoder [28, 29]. As an alternative to stacking, constructing deep autoencoders with denoising autoencoders was explored by Xie et al. [30].

Although the standard denoising autoencoders are not, by construction, generative models, Bengio et al. [5] proved that, under mild regularity conditions, denoising autoencoders can be used to sample from a distribution which consistently estimates the data generating distribution. This method, which consists of alternately adding noise to a sample and denoising it, yields competitive performance in terms of estimated log-likelihood of the samples. An important connection was also made by Vincent [27], who showed that optimising the training objective of a denoising autoencoder is equivalent to performing score matching [17] between the Parzen density estimator of the training data and a particular energy-based model.

3 Composite denoising autoencoders

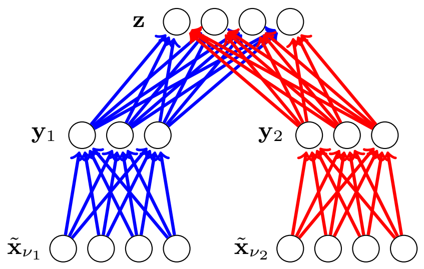


Fig. 1. A composite denoising autoencoder using two levels of noise.

Composite denoising autoencoders learn a diverse representation by leveraging the observation that the types of features learnt by the standard denoising autoencoders differ depending on the level of noise. Instead of training all of the hidden units to extract features from data corrupted with the same level of noise, we can partition the hidden units, training each subset of model's parameters with a different noise level.

More formally, let $\boldsymbol{\nu} = (\nu_1, \nu_2, \ldots, \nu_S)$ denote the set of noise levels that is to be used in the model. For each noise level ν_s the network includes a vector $\mathbf{y}_s \in \mathbb{R}^{D_s}$ of hidden units and a weight matrix $\mathbf{W}_s \in \mathbb{R}^{D_s \times d}$. Note that different noise levels may have different numbers of hidden units. We use $\mathbf{D} = (D_1, D_2, \ldots D_S)$ to denote a vector containing the number of hidden units for each noise level.

When assigning a representation to a new input $\mathbf{x}$, the CDA is very similar to the DA. In particular, the hidden representation is computed as

$$\mathbf{y}_s = h(\mathbf{W}_s \mathbf{x} + \mathbf{b}_s) \qquad \forall s \in 1, \ldots, S, \tag{3}$$

where as before h is a nonlinear transfer function such as the sigmoid. The full representation $\mathbf{y}$ for $\mathbf{x}$ is constructed by concatenating the individual representations as $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_S)$.

Where the CDA differs from the DA is in the training procedure. Given a training input $\mathbf{x}$, we corrupt it S times, once for each level of noise, yielding corrupted vectors

$$\tilde{\mathbf{x}}_s \sim p(\tilde{\mathbf{x}}_s|\mathbf{x}, \nu_s) \qquad \forall s. \tag{4}$$

Then each of the corrupted vectors are fed into the corresponding encoders, yielding the representation

$$\mathbf{y}_s = h(\mathbf{W}_s \tilde{\mathbf{x}}_s + \mathbf{b}_s) \qquad \forall s \in 1, \ldots, S. \tag{5}$$

The reconstruction $\mathbf{z}$ is computed by taking all of the hidden layers as input

$$\mathbf{z} = h' \left(\sum_{s=1}^{S} \mathbf{W}_s^\top \mathbf{y}_s + \mathbf{b}' \right), \tag{6}$$

where as before h' is a nonlinear transfer function, potentially different from h. Finally given a loss function L, such as squared error, an update to the parameters can be made by taking a gradient step on $L(\mathbf{z}, \mathbf{x})$.

This procedure can be seen as a stochastic gradient on an objective function that takes the expectation over the corruptions:

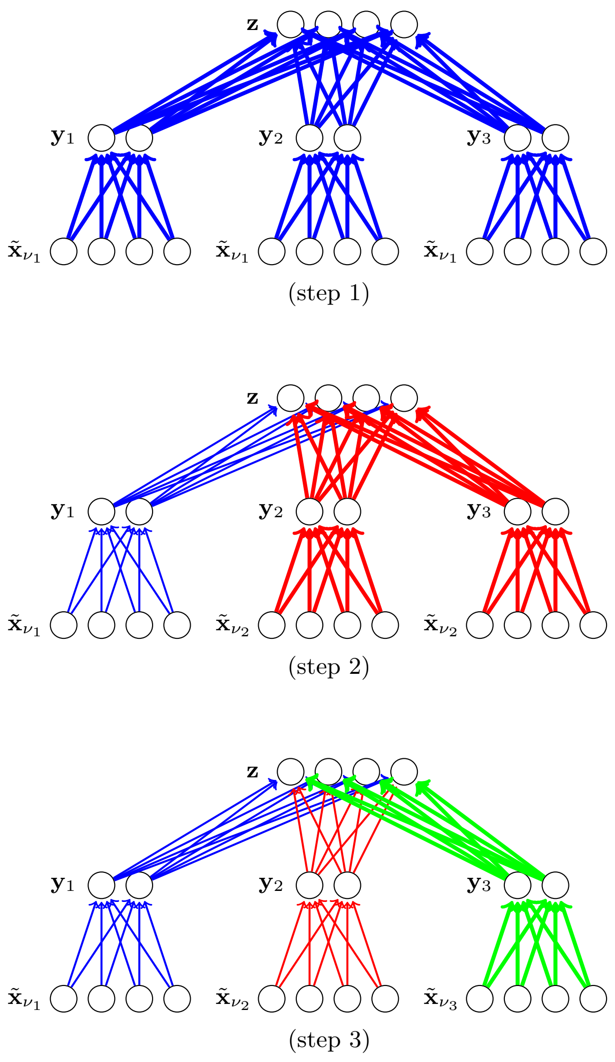
$$\mathbb{E}_{(X, \tilde{X}_{\nu_1}, \ldots, \tilde{X}_{\nu_S})} \left[L \left(X, h' \left(\sum_{s=1}^{S} \mathbf{W}_s^\top h\left(\mathbf{W}_s \tilde{\mathbf{x}}_{\nu_s} + \mathbf{b}_s\right) + \mathbf{b}' \right) \right) \right], \tag{7}$$

This architecture is illustrated in Figure 1 for two levels of noise, where we use the different colours to indicate the weights in the network that are specific to a single noise level.

3.1 Learning

A CDA could be trained by standard optimization methods, such as stochastic gradient descent on the objective (7). As we will show, however, it is difficult to achieve good performance with these methods (4.1). Instead, we propose a new cascaded training procedure for CDAs, which we describe in this section.

Cascaded training is based on two ideas. First, previous work [10] found that pretraining at high noise levels helps learning the parameters for the low noise levels. Second, and more interesting, the problem with taking a joint gradient step on (7) is that low noise levels provide more information about the original input $\mathbf{x}$ than high noise levels, which can cause a training procedure to get stuck in a local optimum in which it relies on the low noise features without using the high noise features. Cascaded training first trains the weights that correspond to high noise levels, and then freezes them before moving on to low noise levels. This way the hidden units trained with lower levels of noise are trained to correct what the hidden units associated with higher noise levels missed.



$\mathbf{z}$
$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_3$
$\tilde{\mathbf{x}}_{\nu_1}$
$\tilde{\mathbf{x}}_{\nu_1}$
$\tilde{\mathbf{x}}_{\nu_1}$
(step 1)
$\mathbf{z}$
$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_3$
$\tilde{\mathbf{x}}_{\nu_1}$
$\tilde{\mathbf{x}}_{\nu_2}$
$\tilde{\mathbf{x}}_{\nu_2}$
(step 2)
$\mathbf{z}$
$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_3$
$\tilde{\mathbf{x}}_{\nu_1}$
$\tilde{\mathbf{x}}_{\nu_2}$
$\tilde{\mathbf{x}}_{\nu_3}$
(step 3)

Fig. 2. The cascaded training procedure for a composite denoising autoencoder with three noise levels. We use the notation $\mathbf{y}_{1:3} = (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$. First all parameters are trained using the level of noise ν_1. In the second step, the blue parameters remain frozen and the red parameters are trained using the noise ν_2. Finally, in the third step, only the green parameters are trained, using the noise ν_3. This is more formally described in Algorithm 1.

Algorithm 1 Training the composite denoising autoencoder

```
for $R$ in $1, \dots, S$ do
  for $K_R$ steps do
    Randomly choose a training input $\mathbf{x}$
    Sample $\tilde{\mathbf{x}}_s \sim p(\cdot|\mathbf{x}, \nu_s)$ for $s \in \{1, 2, \dots, R-1\}$
    Sample $\tilde{\mathbf{x}}_s \sim p(\cdot|\mathbf{x}, \nu_R)$ for $s \in \{R, R+1, \dots, S\}$
    Compute $\mathbf{y}_s$ for all $s$ as in (5)
    Compute reconstruction $\mathbf{z}$ as in (6)
    Take a gradient step
        $\mathbf{W}_s \leftarrow \mathbf{W}_s - \alpha \nabla_{\mathbf{W}_s} L(\mathbf{z}, \mathbf{x})$
        $\mathbf{b}_s \leftarrow \mathbf{b}_s - \alpha \nabla_{\mathbf{b}_s} L(\mathbf{z}, \mathbf{x})$
        $\mathbf{b}' \leftarrow \mathbf{b}' - \alpha \nabla_{\mathbf{b}'} L(\mathbf{z}, \mathbf{x})$
    for $s \in \{R, R+1, \dots S\}$
  end for
end for
```

Putting these ideas together, cascaded training works as follows. We assume that the noise levels are ordered so that $\nu_1 > \nu_2 > \cdots > \nu_S$. Then the first step is that we train *all* of the parameters $\mathbf{W}_1 \dots \mathbf{W}_S, \mathbf{b}_1, \dots \mathbf{b}_S, \mathbf{b}'$, but using only the noise level ν_1 to corrupt all S copies $\tilde{\mathbf{x}}_1 \dots \tilde{\mathbf{x}}_S$ of the input. Once this is done, we freeze the weights $\mathbf{W}_1, \mathbf{b}_1$ and we do not alter them again during training. Then we train the weights $\mathbf{W}_2 \dots \mathbf{W}_S, \mathbf{b}_2 \dots \mathbf{b}_S, \mathbf{b}'$, where the corrupted input $\tilde{\mathbf{x}}_1$ is as before corrupted with noise ν_1, and the $S-1$ corrupted copies $\tilde{\mathbf{x}}_2 \dots \tilde{\mathbf{x}}_S$ are all corrupted with noise ν_2. We repeat this process until at the end we are training the weights $\mathbf{W}_S, \mathbf{b}_S, \mathbf{b}'$ using the noise level ν_S. This process is illustrated graphically in Figure 2 and in pseudocode in Algorithm 1. To keep the exposition simple, this algorithm assumes that we employ SGD with only one training example per update, although in practice we use mini-batches.

The composite denoising autoencoder builds on several ideas and intuitions. Firstly, our training procedure can be considered an application of the idea of curriculum learning [4, 14]. That is, we start by training all units with high noise level, which serves as a form of *unsupervised pretraining* for the units that will be trained later with lower levels of noise, giving them a good starting point for further optimisation. We experimentally show that the training of a denoising autoencoder learning with data corrupted with high noise levels needs less training epochs to converge, therefore, it can be considered an *easier* problem. This is shown in Figure 3. Secondly, we are inspired by multi-column neural networks (e.g. Ciresan et al. [7]), which achieve excellent performance for supervised problems. Finally, our work is similar in motivation to scheduled denoising autoencoders [10], which learn a diverse set of features thanks to the training procedure which involves using a sequence of levels of noise. Composite denoising autoencoders achieve this goal more explicitly thanks to their training objective.

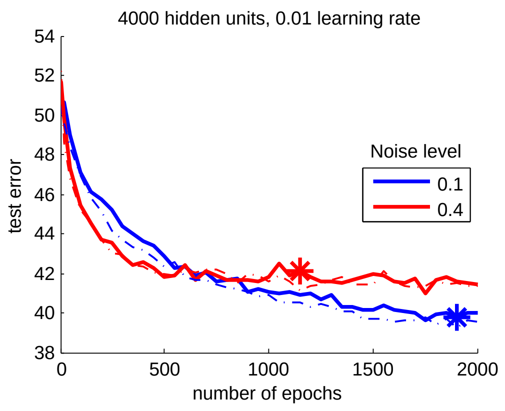


Fig. 3. Classification results with the CIFAR-10 data set yielded by representations learnt with standard denoising autoencoders and data corrupted with two different noise levels. Dashed lines indicate the errors on the validation set. The stars indicate the test errors for the epochs at which the validation errors had its lowest value. The DA trained with high noise level learns faster at the beginning but stops to improve earlier. See section 4 for the details of the experimental setup.

3.2 Recovering the standard denoising autoencoder

If, for every training example, the corrupted inputs $\tilde{\mathbf{x}}_{\nu_i}$ were always identical, $[\mathbf{W}_1, \ldots, \mathbf{W}_S]$ were initialised randomly from the same distribution as $\mathbf{W}$ in the standard denoising autoencoder, $\mathbf{b}_i$ and $\mathbf{b}'$ were initalised to $\mathbf{0}$ and $\mathbf{V}_i$ were constrained to be $\mathbf{V}_i = \mathbf{W}_i^T$, then this model is exactly equivalent to the standard denoising autoencoder described in section 2. Therefore, it is natural to incrementally corrupt the training examples shown to the composite denoising autoencoders in such a way that when all the noise levels are the same, this equivalency holds. For example, when working with masking noise, consider two noise levels ν_i and ν_j such that $\nu_i > \nu_j$. Denote the random variables indicating the presence of corruption of a pixel in a training datum by C_{ν_i} and C_{ν_j}. Assuming $\mathrm{C}_{\nu_j} \sim \mathrm{Bernoulli}(\nu_j)$, we want $\mathrm{C}_{\nu_i} \sim \mathrm{Bernoulli}(\nu_i)$, such that when $\mathrm{C}_{\nu_j} = 1$ then also $\mathrm{C}_{\nu_i} = 1$. It can be easily shown that this is satisfied when $\mathrm{C}_{\nu_i} = \max(\mathrm{C}_{\nu_j} + \mathrm{C}_{\nu_j \to \nu_i}, 1)$, where $\mathrm{C}_{\nu_j \to \nu_i} \sim \mathrm{Bernoulli}(\frac{\nu_i - \nu_j}{1 - \nu_j})$. We use this incremental noising procedure in all our experiments.

4 Experiments

We used two image recognition data sets to evaluate the CDA, the CIFAR-10 data set [19] and a variant of the NORB data set [21]. To evaluate the quality of the learnt representations, we employ a procedure similar to that used by Coates et al. [8] and by many other works[1]. That is, we first learn the representation in

[1] We do not use any form of pooling, keeping our setup invariant to the permutation of the features.

an unsupervised fashion and then use the learnt representation within a linear classifier as a measure of its quality. For both data sets, in the unsupervised feature learning stage, we use masking noise as the corruption process, a sigmoid encoder and decoder and cross entropy loss (Equation 2) following Vincent et al. (2008, 2010). To do optimisation, we use stochastic gradient descent with mini-batches. For the classification step, we use $L2$-regularised logistic regression with the regularisation parameter chosen to minimise the validation error. Additionally, with the CIFAR-10 data set, we also trained a single-layer supervised neural network using the parameters of the encoder we learnt in the unsupervised stage as an initialisation. When conducting our experiments, we first find the best hyperparameters using the validation set, then merge it with the training set, retrain the model with the hyperparameters found in the previous step and report the error achieved with this model.

We implemented all neural network models using Theano [6] and we used logistic regression implemented by Fan et al. [9]. We followed the advice of Glorot and Bengio [11] on random initialisation of the parameters of our networks.

4.1 CIFAR-10

This data set consists of 60000 colour images spread evenly between ten classes. There are 50000 training and validation images and 10000 test images. Each image has a size of 32×32 pixels and each pixel has three colour channels, which are represented with a number in $\{0, \ldots, 255\}$. We divide the training and validation set into 40000 training instances and 10000 validation instances. The only preprocessing step we use is dividing the intensity of every pixel by 255 to get numbers in $[0, 1]$.

In our experiments with this data set we trained autoencoders with the total number of 2000 hidden units (undercomplete representation) and 4000 hidden units (overcomplete representation).

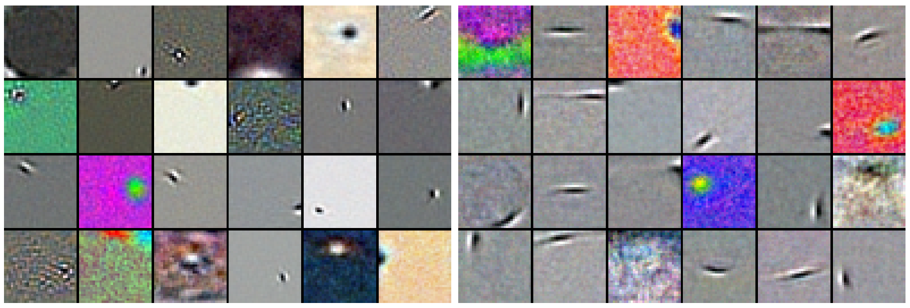


Fig. 4. Example filters (columns of the matrix $\mathbf{W}$) learnt by standard denoising autoencoders with $\nu = 0.1$ (left) and $\nu = 0.5$ (right).

Training the baselines The simplest possible baseline, logistic regression trained with raw pixel values, achieved 59.4% test error. To get the best possible baseline denoising autoencoder we explored combinations of different learning rates, noise levels and numbers of training epochs. For 2000 hidden units we considered $\nu \in \{0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$ and for 4000 hidden units we also additionally considered $\nu = 0.15$. For both sizes of the hidden layers we tried learning rates $\in \{0.01, 0.02, 0.04\}$. Each model was trained for up to 2000 training epochs and we measured the validation error every 50 epochs. The best baselines we got achieved the test errors of 40.71% (2000 hidden units) and 38.35% (4000 hidden units).

Concatenating representations learnt independently To demonstrate that diversity in noise levels improves the representation, we evaluate representations yielded by concatenating the representations from two different DAs, trained independently. We will combine DAs trained with noise levels $\nu \in \{0.1, 0.2, \ldots, 0.5\}$ for each noise level training three DAs with different random seeds. Denote parameters learnt by a DA with the noise level ν and using the random seed R by $\left(\mathbf{W}^{(R,\nu)}, \mathbf{b}^{(R,\nu)}, \mathbf{b}'^{(R,\nu)}\right)$ and denote by E_{ij}^{kl} the classification error on the test set yielded by the concatenating the representations of two independently trained DAs, the first trained with random seed R_k and noise level ν_i, and the second trained by random seed R_l and noise level ν_j. For each pair of noise levels (ν_i, ν_j), we measure the average error across random seeds, that is, $\bar{\mathrm{E}}_{ij} = \frac{1}{2\binom{K}{2}} \left(\sum_{k \neq l} \mathrm{E}_{ij}^{kl} + \mathrm{E}_{ji}^{kl} \right)$. The results of this experiment are shown in Figure 5. For every ν we used, it was optimal to concatenate the representation learnt with ν with a representation learnt with a different noise level. To understand this intuition, we visually examine features from DAs with different noise levels (Figure 4). From this figure it can be seen that features at higher noise levels depend on larger regions of the image. This demonstrates the benefit of using a more diverse representation for classification.

Comparison of CDA to DA The CDA offers freedom to choose the number of noise levels, the value ν_s for each noise level, and the number D_s of hidden units at each noise level.

For computational reasons, we limit the space of possible combinations of hyperparameters in the following manner (of course, expanding the search space would only make our results better). We considered models containing up to four different noise levels. We first consider only the models with two noise levels and hidden units divided equally between them. For 2000 total hidden units, we consider all possible pairs of noise levels drawn from the set $\{0.5, 0.4, 0.3, 0.2, 0.1, 0.05\}$. Once we have found the value of $\boldsymbol{\nu}$ that minimizes that validation error for $D_1 = D_2$, we try splitting hidden units such that the ratio $D_1 : D_2 = 1 : 3$ or $D_1 : D_2 = 3 : 1$. Similarly, for four noise levels, we consider the following sets of noise levels $\boldsymbol{\nu} \in \{(0.5, 0.4, 0.3, 0.2), (0.4, 0.3, 0.2, 0.1), (0.3, 0.2, 0.1, 0.05)\}$. We select the value of $\boldsymbol{\nu}$ that has lowest validation error for an equal split

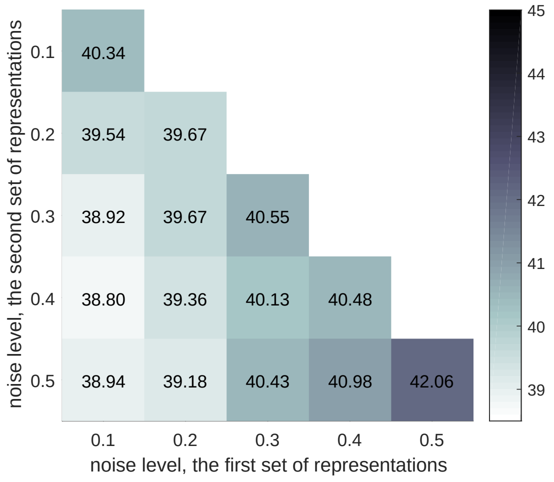


Fig. 5. Classification errors for representations constructed by concatenating representations learnt independently.

$D_1 = \cdots = D_4$, and then try splitting the hidden units with different ratios: $D_1 : D_2 : D_3 : D_4 = 3 : 1 : 1 : 1$, $D_1 : D_2 : D_3 : D_4 = 9 : 1 : 1 : 1$ and the permutations of these ratios. As for the learning rate, we train each of the cascaded DAs with the learning rate that had the best validation error for the first noise level ν_1. The models were trained for up to 500 epochs at each consecutive noise level and we computed the validation error every 50 training epochs. Note that when training with four noise levels, it is possible that the lowest validation error occurs before the training procedure has moved on to the final noise level. In this circumstance, it is possible that the final model will have only two or three noise levels instead of four.

We trained the models with 4000 hidden units the same way, except that we used different sets of noise levels for this higher number of hidden units. This is because our experience with the baseline DAs was that units with 4000 hidden units do better with lower noise levels. For the CDAs with four noise levels, we compared three difference choices for $\boldsymbol{\nu}$: $(0.4, 0.3, 0.2, 0.1)$, $(0.3, 0.2, 0.1, 0.05)$, and $(0.2, 0.15, 0.1, 0.05)$. For the models with two noise levels the values were drawn from $\{0.4, 0.3, 0.2, 0.15, 0.1, 0.05\}$.

For either number of hidden units, we find that CDAs perform better than simple DAs. The best models with 2000 hidden units and 4000 hidden units we found achieved the test errors of 38.86% and 37.53% respectively, thus yielding a significant improvement over the representations trained with a standard DA. These results are compared to the baselines in Table 1. It is also noteworthy that a CDA performs better than concatenating two indepedently trained DAs with different noise levels (cf. Figure 5).

Table 1. Classification errors of standard denoising autoencoders and composite denoising autoencoders.

hidden units	best DA	test error	best CDA	test error
2000	$\nu = 0.2$	40.71%	$\boldsymbol{\nu} = (0.3, 0.2, 0.1)$, $\mathbf{D} = (500, 500, 1000)$	38.86%
4000	$\nu = 0.1$	38.35%	$\boldsymbol{\nu} = (0.3, 0.05)$, $\mathbf{D} = (1000, 3000)$	37.53%

Comparison of Optimization Methods One could consider several simpler alternatives to the cascaded training procedure from Section 3.1. The simplest alternative, which we call *joint SGD*, is to train all of the model parameters jointly, at every iteration sampling each corrupted input $\tilde{\mathbf{x}}_s$ using its corresponding noise level ν_s. This is simply SGD on the objective (7). A second alternative, which we call *alternating SGD*, is block coordinate descent on (7), where we assign each weight matrix $\mathbf{W}_s$ to a separate block. In other words, at each iteration we choose a different parameter block $\mathbf{W}_s$, and take a gradient update only on $\mathbf{W}_s$ (note that this requires computing a corrupted input $\tilde{\mathbf{x}}_s$ for all noise levels ν_s). Neither of these simpler methods try to prevent undertraining of the parameters for the high noise levels in the way that cascaded training does.

Figure 6 shows a comparison of joint SGD, alternating SGD, and our cascaded SGD methods on a CDA with four noise levels $\boldsymbol{\nu} = (0.4, 0.3, 0.2, 0.1)$ and $\mathbf{D} = (500, 500, 500, 500)$. We ran both joint SGD and cascaded SGD until they converged in validation error, and then we ran alternating SGD until it had made the same number of parameter updates as joint SGD. This means that alternating SGD was run for four times as many iterations as joint SGD, because alternating SGD only updates one-quarter of the parameters at each iteration. Cascaded SGD was stopped early when it converged according to validation error. The vertical dashed lines in the figure indicate the epochs at which alternating SGD switched between parameter blocks.

From these results, it is clear that the cascaded training procedure is significantly more effective than either joint or alternating SGD. Joint SGD seems to have converged to much worse parameters than cascaded SGD. We hypothesize that this is because the parameters corresponding to the high noise levels are undertrained. To verify this, in Figure 7 we show the features learned by a composite CDA with joint training at two different noise levels. Note that at the higher noise level (at right) there are many filters that are mostly noise; this is not observed at the lower noise or to the same extent in an standard DA. Alternating SGD seems to converge fairly slowly. It is possible that its error would continue to decrease, but even after 8000 iterations its solution is still much worse than that found by cascading SGD after only 3500 iterations.

We have made similar comparisons for other choices of $\boldsymbol{\nu}$ and found a similar difference in performance between joint, alternating, and cascaded SGD. One exception to this is that alternating SGD seems to work much better on models with only two noise levels ($S = 2$) than those with four noise levels. In those situations, the performance of alternating SGD often equals, but usually does not exceed, that of cascaded SGD.

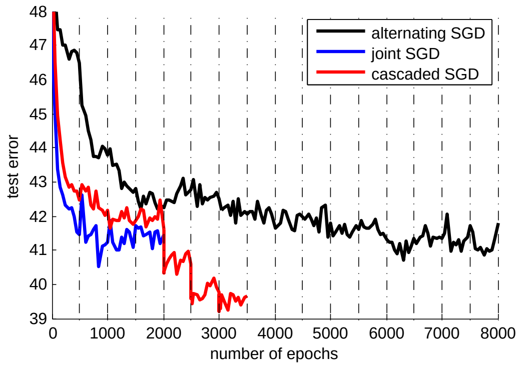


Fig. 6. Classification errors achieved by three different methods of optimising the objective in (7).

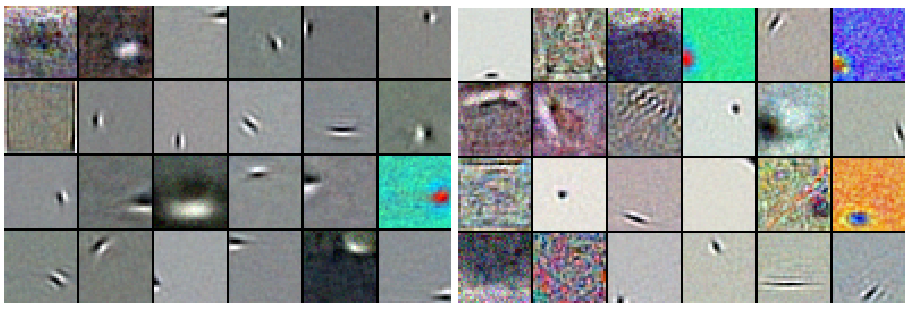


Fig. 7. Example filters (columns of the matrix $\mathbf{W}$) learnt by composite denoising autoencoders with $\nu = 0.1$ (left) and $\nu = 0.4$ (right) when all the parameters were optimise using joint SGD. While the filters associated with $\nu_2 = 0.1$ have managed to learn interesting features, many of these associated with $\nu_1 = 0.4$ remained undertrained. These hard to interpret filters are much more rare with cascaded SGD.

Fine-tuning We also trained a supervised single-layer neural network using parameters of the encoder as the initialisation of the parameters of the hidden layer of the network. This procedure is known as fine-tuning. We did that for the best standard DAs and CDAs with 4000 hidden units. The learning rate, the same for all parameters, was chosen from the set $\{0.00125, 0.00125 \cdot 2^{-1}, \ldots, 0.00125 \cdot 2^{-4}\}$ and the maximum number of training epochs was 2000 (we computed the validation error after each epoch). We report the test error for the combination of the learning rate and the number of epochs yielding the lowest validation error. The results are shown in Table 3. Fine-tuning makes the performance of DA and CDA much more similar, which is to be expected since the fine-tuning procedure is identical for both models. However, note that the result achieved with a standard denoising autoencoder and supervised fine-tuning we present here is an extremely well tuned one. In fact, its error is lower than any previous result achieved by a permutation-invariant method on the CIFAR-10 data set. Our best model, yielding the error of 35.06% is, by a considerable margin, more accurate than any previously considered permutation-invariant model for this task, outperforming a variety of methods. A summary of the best results reported in the literature is shown in Table 2.

Table 2. Summary of the results on CIFAR-10 among permutation-invariant methods.

Model	Test error
Composite Denoising Autoencoder	**35.06%**
Scheduled Denoising Autoencoder [10]	35.7%
Zero-bias Autoencoder [22]	35.9%
Fastfood FFT [20]	36.9%
Nonparametrically Guided Autoencoder [25]	43.25%
Deep Sparse Rectifier Neural Network [12]	49.52%

Table 3. Test errors on CIFAR-10 data set for the best DA and CDA models trained without supervised fine-tuning and their fine-tuned versions.

DA		**CDA**	
no fine-tuning	**fine-tuning**	**no fine-tuning**	**fine-tuning**
38.35%	35.30%	37.53%	35.06%

4.2 NORB

To show that the advantage of our model is consistent across data sets, we did the same experiment use a variant of the small NORB normalized-uniform data set [21], which contains 24300 examples for training and validation and 24300 test examples. It contains images of 50 toys belonging to five generic categories: animals, human figures, airplanes, trucks, and cars. The 50 toys are evenly divided between the training and validation set and the test set. The objects were photographed by two cameras under different lighting conditions, elevations and azimuths. Every example consists of a stereo pair of grayscale images, each of size 96×96 pixels whose intensities are represented as a number $\in \{0, \dots, 255\}$. We transform the data set by taking the middle 64×64 pixels from both images in a pair and dividing the intensity of every pixel by 255 to get numbers in $[0, 1]$. The simplest baseline, logistic regression using raw pixels, achieved the test error of 42.32%.

In the experiments with learning the representations with this data set we used the hidden layer with 1000 hidden units and adapted the set up we used for CIFAR-10. To find the best possible standard DA we considered all combinations of the noise levels $\in \{0.1, 0.2, 0.3, 0.4\}$ and the learning rates $\in \{0.005, 0.01, 0.02\}$.

The representation learnt by the best denoising autoencoder yielded 18.75% test error when used with logistic regression. By contrast, a composite denoising autoencoder with $\boldsymbol{\nu} = (0.4, 0.3, 0.2, 0.1)$ and $\mathbf{D} = (250, 250, 250, 250)$ results in a representation that yields a test error of 17.03%.

5 Discussion

We introduced a new unsupervised representation learning method, called a composite denoising autoencoder, by modifying the standard DA so that different parts of the network were exposed to corruptions of the input at different noise levels. Naive training procedures for the CDA can get stuck in bad local optima, so we designed a cascaded training procedure to avoid this. We showed that CDAs learned more effective representations than DAs on two different image data sets.

A few pieces of prior work have considered related techniques. In the context of RBMs, the benefits of learning a diverse representation was also noticed by Tang and Mohamed [26], achieving diversity by manipulating the resolution of the image. Also, ensembles of denoising autoencoders, where each member of the ensemble is trained with a different level or different type of noise, have been considered by Agostinelli et al. [1]. This work differs from ours because in their method all DAs in the ensemble are trained independently, whereas we show that training the different representations together is better than independent training. The cascaded training procedure has some similarities in spirit to the incremental training procedure of Zhou et al. [31], but that work considered only DAs with one level of noise. Usefulness of varying the level of noise during training of neural nets was also noticed by Gulcehre et al. [15], who add noise to the activation functions. Our training procedure also resembles the walkback training suggested by Bengio et al. [5], however, we do not require our training loss to be interpretable as negative log-likelihood. Understanding the relative merits of walkback training, scheduled denoising autoencoders and composite denoising autoencoders would be an interesting future challenge.

References

[1] Agostinelli, F., Anderson, M.R., Lee, H.: Robust image denoising with multi-column deep neural networks. In: NIPS (2013)
[2] Baldi, P., Hornik, K.: Neural networks and principal component analysis: Learning from examples without local minima. Neural Networks 2 (1989)
[3] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: NIPS (2007)
[4] Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: ICML (2009)
[5] Bengio, Y., Yao, L., Alain, G., Vincent, P.: Generalized denoising auto-encoders as generative models. In: NIPS (2013)
[6] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. SciPy (2010)

[7] Ciresan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: CVPR (2012)
[8] Coates, A., Ng, A.Y., Lee, H.: An analysis of single-layer networks in unsupervised feature learning. In: AISTATS (2011)
[9] Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. JMLR 9 (2008)
[10] Geras, K.J., Sutton, C.: Scheduled denoising autoencoder. In: ICLR (2015)
[11] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: AISTATS (2010)
[12] Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier networks. In: AISTATS (2011)
[13] Glorot, X., Bordes, A., Bengio, Y.: Domain adaptation for large-scale sentiment classification: A deep learning approach. In: ICML (2011)
[14] Gulcehre, C., Bengio, Y.: Knowledge matters: Importance of prior information for optimization. JMLR 17 (2016)
[15] Gulcehre, C., Moczulski, M., Denil, M., Bengio, Y.: Noisy activation functions. arXiv:1603.00391 (2016)
[16] Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Computation 18(7) (2006)
[17] Hyvärinen, A., Dayan, P.: Estimation of non-normalized statistical models by score matching. Journal of Machine Learning Research 6 (2005)
[18] Karklin, Y., Simoncelli, E.P.: Efficient coding of natural images with a population of noisy linear-nonlinear neurons. In: NIPS (2011)
[19] Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)
[20] Le, Q., Sarlós, T., Smola, A.: Fastfood-computing Hilbert space expansions in loglinear time. In: ICML (2013)
[21] LeCun, Y., Huang, F.J., Bottou, L.: Learning methods for generic object recognition with invariance to pose and lighting. In: CVPR (2004)
[22] Memisevic, R., Konda, K., Krueger, D.: Zero-bias autoencoders and the benefits of co-adapting features. In: ICLR (2015)
[23] Mesnil, G., Dauphin, Y., Glorot, X., Rifai, S., Bengio, Y., Goodfellow, I.J., Lavoie, E., Muller, X., Desjardins, G., Warde-Farley, D., Vincent, P., Courville, A.C., Bergstra, J.: Unsupervised and transfer learning challenge: a deep learning approach. In: ICML Unsupervised and Transfer Learning Workshop (2012)
[24] Seung, H.S.: Learning continuous attractors in recurrent networks. In: NIPS (1998)
[25] Snoek, J., Adams, R.P., Larochelle, H.: Nonparametric guidance of autoencoder representations using label information. JMLR 13 (2012)
[26] Tang, Y., Mohamed, A.r.: Multiresolution deep belief networks. In: AISTATS (2012)
[27] Vincent, P.: A connection between score matching and denoising autoencoders. Neural Computation 23 (2011)
[28] Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: ICML (2008)
[29] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. JMLR (2010)
[30] Xie, J., Xu, L., Chen, E.: Image denoising and inpainting with deep neural networks. In: NIPS (2012)
[31] Zhou, G., Sohn, K., Lee, H.: Online incremental feature learning with denoising autoencoders. In: AISTATS (2012)

5.2 Comments on the paper

Jointly training all the parameters in the CDA leads to the optimisation focusing on learning only the features associated with the lowest noise level. We were able to prevent the features associated with high noise levels from undertraining thanks to the cascaded SGD procedure we introduced in the paper. A possible alternative to this method, which might be able to achieve the same goal without alternating between optimising different subsets of parameters of the model, could work in the following manner. We explain it for only two levels of noise for simplicity. The parameters associated with noise level ν_0 are $\mathbf{W}_0$ and $\mathbf{b}_0$ and the parameters associated with noise level $\nu_1 < \nu_0$ are $\mathbf{W}_1$ and $\mathbf{b}_1$. The gradient with respect to the parameters $\mathbf{W}_0$ and $\mathbf{b}_0$ would then be computed by minimising the training objective

$$\mathbb{E}_{(\mathbf{x},\tilde{\mathbf{x}}_{\nu_0},\tilde{\mathbf{x}}_{\nu_1})}\left[L\left(X, h'\left(\mathbf{W}_0^\top h\left(\mathbf{W}_0\tilde{\mathbf{x}}_{\nu_0}+\mathbf{b}_0\right)+\mathbf{W}_1^\top h\left(\mathbf{W}_1\tilde{\mathbf{x}}_{\nu_1}+\mathbf{b}_1\right)+\mathbf{b}'\right)\right)\right]$$

and the gradient with respect to the parameters $\mathbf{W}_1$ and $\mathbf{b}_1$ would be computed by minimising the objective

$$\mathbb{E}_{(\mathbf{x},\tilde{\mathbf{x}}'_{\nu_1},\tilde{\mathbf{x}}_{\nu_1})}\left[L\left(X, h'\left(\mathbf{W}_0^\top h\left(\mathbf{W}_0\tilde{\mathbf{x}}'_{\nu_1}+\mathbf{b}_0\right)+\mathbf{W}_1^\top h\left(\mathbf{W}_1\tilde{\mathbf{x}}_{\nu_1}+\mathbf{b}_1\right)+\mathbf{b}'\right)\right)\right].$$

The difference between the objective is highlighted in blue. The gradient with respect to the bias in the decoder could be simply an average gradient computed for the two objectives. The intuition behind this idea is that corrupting all the copies of the input with low noise when optimising the subset of parameters of the model associated with low noise, would aim to prevent the optimisation from focusing too much on that subset of parameters.

Chapter 6

Blending LSTMs into CNNs

6.1 Introduction to the paper

In this chapter we show how two strong, yet different in their inductive biases, neural network architectures, convolutional neural networks (CNNs) and long short-term memory networks (LSTMs), can be combined to reach a superior performance. We do this model combination with a method we propose which we call model blending. We perform model blending through model compression, which was previously shown to work in an asymmetric setting where the student is much simpler than the teacher. In our work, we use the student and the teacher of comparable capacities and use model blending to average their inductive biases, which resembles the classic technique of ensembling. This allows us to create a student model stronger than both baselines. Remarkably, this is achieved with no extra cost at test time. We use CNNs of vision-inspired architecture, which are more accurate than previous CNN architectures widely used for speech recognition and as accurate as LSTM networks, while being much less expensive to run at test time than the LSTMs. These contributions are of considerable importance for two reasons. Firstly, because of the improved accuracy they yield. Secondly, because they question the common belief in the speech community about how convolutional networks for speech recognition should be designed and what the limits of their performance are in comparison to the LSTMs.

Furthermore, we demonstrate that less than 1% of targets are sufficient to perform model compression. This sheds additional light on the "dark knowledge" hypothesis, which states that a lot of the knowledge about the teacher model is hidden in the low-probability predictions. Our results are especially useful in the settings when the output space is very large.

BLENDING LSTMS INTO CNNS

Krzysztof J. Geras[1], Abdel-rahman Mohamed[2], Rich Caruana[2], Gregor Urban[3], Shengjie Wang[4], Özlem Aslan[5], Matthai Philipose[2], Matthew Richardson[2] & Charles Sutton[1]
[1]University of Edinburgh
[2]Microsoft Research
[3]UC Irvine
[4]University of Washington
[5]University of Alberta

ABSTRACT

We consider whether deep convolutional networks (CNNs) can represent decision functions with similar accuracy as recurrent networks such as LSTMs. First, we show that a deep CNN with an architecture inspired by the models recently introduced in image recognition can yield better accuracy than previous convolutional and LSTM networks on the standard 309h Switchboard automatic speech recognition task. Then we show that even more accurate CNNs can be trained under the guidance of LSTMs using a variant of model compression, which we call *model blending* because the teacher and student models are similar in complexity but different in inductive bias. Blending further improves the accuracy of our CNN, yielding a computationally efficient model of accuracy higher than any of the other individual models. Examining the effect of "dark knowledge" in this model compression task, we find that less than 1% of the highest probability labels are needed for accurate model compression.

1 INTRODUCTION

There is evidence that feedforward neural networks trained with current training algorithms use their large capacity inefficiently (Le Cun et al., 1990; Denil et al., 2013; Dauphin & Bengio, 2013; Ba & Caruana, 2014; Hinton et al., 2015; Han et al., 2016). Although this excess capacity may be necessary for accurate learning and generalization at training time, the function once learned often can be represented much more compactly. As deep neural net models become larger, their accuracy often increases, but the difficulty of deploying them also rises. Methods such as model compression sometimes allow the accurate functions learned by large, complex models to be compressed into smaller models that are computationally more efficient at runtime.

There are a number of different kinds of deep neural networks such as deep fully-connected neural networks (DNNs), convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Different domains typically benefit from deep models of different types. For example, CNNs usually yield highest accuracy in domains such as image recognition where the input forms a regular 1, 2, or 3-D image plane with structure that is partially invariant to shifts in position or scale. On the other hand, recurrent network models such as LSTMs appear to be better suited to applications such as speech recognition or language modeling where inputs form sequences of varying lengths with short and long-range interactions of different scales.

The differences between these deep learning architectures raise interesting questions about what is learnable by different kinds of deep models, and when it is possible for a deep model of one kind to represent and learn the function learned by a different kind of deep model. The success of model compression on feedforward networks raises the question of whether other neural network architectures that embody different inductive biases can also be compressed. For example, can a classification function learnt by an LSTM be represented by a CNN with a wide enough window to span the important long-range interactions?

In this paper we demonstrate that, for a speech recognition task, it is possible to train very accurate CNN models, outperforming LSTMs. This is thanks to CNN architectures inspired by recent developments in computer vision (Simonyan & Zisserman, 2014), which were not previously considered for speech recognition. Moreover, the experiments suggest that LSTMs and CNNs learn different functions when trained on the same data. This difference creates an opportunity: by merging the functions learned by CNNs and LSTMs into a single model we can obtain better accuracy than either model class achieved independently. One way to perform this merge is through a variant of model compression that we call *model blending* because the student and teacher models are of comparable size and have complementary inductive biases. Although at this point model compression is a well established technique for models of different capacity, the question of whether compression is also effective for models of similar capacity has not been explored. For example, by blending an LSTM teacher with a CNN student, we are able to train a CNN that is more accurate because it benefits from what the LSTM learned and also computationally more efficient than an LSTM would be at runtime. This blending process is somewhat analogous to forming an ensemble of LSTMs and CNNs and then training a student CNN to mimic the ensemble, but in blending no explicit ensemble of LSTMs and CNNs need be formed. The blended model is *6.8 times* more efficient at testing time than the analogous ensemble. Blending further improves the accuracy of our convolutional model, yielding a computationally efficient model of accuracy higher than any of the other individual models. Examining the effect of "dark knowledge" (Hinton et al., 2015) in this model compression task, we find that only 0.3% of the highest probability labels are needed during the model compression procedure. Intriguingly, we also find that model blending is even effective in the self-teacheing setting when the student and the teacher are of the same architecture. We show results of an experiment in which we use a CNN to teach another CNN of the same architecture. Such a student CNN is weaker than a CNN student of the LSTMs but still significantly stronger than a baseline trained only with the hard labels in the data set.

2 Background

In model compression (Bucila et al., 2006), one model (a *student*) is trained to mimic another model (a *teacher*). Typically, the student model is small and the teacher is a larger, more powerful model, which has high accuracy but is computationally too expensive to use at test time. For classification, this mimicry can be performed in two ways. One way is to train the student model to match logits (i.e. the values z_i in the output layer of the network, before applying the softmax to compute the output class probabilities $p_i = e^{z_i} / \sum_j e^{z_j}$) predicted by the teacher on the training data, penalising the difference between logits of the two models with a squared loss. Alternatively, compression can be done by training the student model to match class probabilities predicted by the teacher, by penalising cross-entropy between predictions $\mathbf{p}$ of the teacher and predictions $\mathbf{q}$ of the student, i.e. by minimising -$\sum_i p_i(\mathbf{x}) \log q_i(\mathbf{x})$ averaged over training examples. We will refer to predictions made the teacher as *soft labels*. In the context of deep neural networks, this approach to model compression is also known as *knowledge distillation* (Hinton et al., 2015). Additionally, the training loss can also include the loss on the original 0-1 hard labels.

The main advantage of training the student using model compression is that a student trained with knowledge provided by the teacher gets a richer supervision signal than just the hard 0-1 labels in the training data, i.e. for each training example, it gets the information not only about the correct class but also about uncertainty, i.e., how similar the current training example is to those of other classes. Model compression can be viewed as a way to transfer inductive biases between models. For example, in the case of compressing deep models into shallow ones (Ba & Caruana, 2014; Urban et al., 2016), the student is benefiting from the hierarchical representation learned in the deep model, despite not being able to learn it on its own from hard labels.

While model compression can be applied to arbitrary classifiers producing probabilistic predictions, with the recent success of deep neural networks, work on model compression focused on compressing large deep neural networks or ensembles thereof into smaller ones, i.e., with less layers, less hidden units or less parameters. Pursuing that direction, Ba & Caruana (2014) showed that an ensemble of deep neural networks with few convolutional layers can be compressed into a single layer network as accurate as a deep one. In a complementary work, Hinton et al. (2015) focused on compressing ensembles of deep networks into deep networks of the same architecture. They also experimented with softening predictions of the teacher by dividing the logits by a constant greater than one called *temperature*. Using the techniques developed in prior work and adding an extra mimic layer in the middle of the student network, Romero et al. (2014) demonstrated that a moderately deep and wide convolutional network can be compressed into a deeper and narrower convolutional network with much fewer parameters than the teacher network while also increasing accuracy.

2.1 LSTM

One example of a very powerful neural network architecture yielding state-of-the-art performance on a range of tasks, yet expensive to run at test time, is the long short-term memory network (LSTM) (Hochreiter & Schmidhuber, 1997; Graves & Schmidhuber, 2005; Graves et al., 2013), which is a type of recurrent neural network (RNN). The focus of this work is to use this model as a teacher for model compression.

LSTMs exhibit superior performance not only in speech, but also in handwriting recognition and generation (Graves & Schmidhuber, 2009; Graves, 2014), machine translation (Sutskever et al., 2014) and parsing (Vinyals et al., 2015), thanks to their ability to learn longer-range interactions. For acoustic modeling though, the difference between a non-recurrent network and an LSTM using full-length sequences is two fold: the use of longer context while deciding on the current frame label, and the type of processing in each cell (the LSTM cell compared to a sigmoid or ReLU). The LSTM network used in our paper uses a fixed-size input sequence and only predicts the output for the middle item of the input sequence. In this respect, we follow the design proposed in the speech literature by Mohamed et al. (2015). We use acoustic models that use the same context window as a non-recurrent network (limited to about 0.5 s) while using the LSTM cells for processing each frame. The LSTM cells process frames in the same bidirectional manner that any other bidirectional LSTM would do, but they are limited by the size of the contextual window. Details of the LSTM used in this work can be found in the supplementary material. This style of modelling has important benefits. One motivation to use such an architecture is that acoustic modeling labels (i.e. target states) are local in nature with an average duration of about 450 ms. Therefore, the amount of information about the class label decays rapidly as we move away from the target. Long-term relations between labels, on the other hand, are handled using a language model (during testing) or a lattice of competing hypotheses in case of lattice training (Veselý et al., 2013; Kingsbury, 2009). Another motivation to prefer models that utilise limited input windows is faster convergence due to the ability to randomise samples on the frame level rather than on the utterance level. A practical benefit of using a fixed-length window is that using bidirectional architectures becomes possible in real time setups when the delay in response cannot be long.

3 Vision-style CNNs for speech recognition

Convolutional neural networks (LeCun et al., 1998) were considered for speech for many years (LeCun & Bengio, 1998; Lee et al., 2009), though only recently have become very successful (Abdel-Hamid et al., 2012; Sainath et al., 2013; Abdel-Hamid et al., 2014; Sainath et al., 2015). These CNN architectures are quite different from those used in computer vision. They use only two or three convolutional layers with large filters followed by more fully connected layers. They also only use convolution or pooling over one dimension, either time or frequency. When looking at a spectrogram in Figure 3, it is obvious that, like what we observe in vision, similar patterns re-occur both across different points in time and across different frequencies. Using convolution or pooling across only one of these dimensions seems suboptimal. One of the reasons for the success of CNNs is their invariance to small translations and scaling. Intuitively, small translations (corresponding to the pitch of voice) or scaling (corresponding to speaking slowly or quickly) should not change the class assigned to a window of speech. We hypothesise that classification of windows of speech with CNNs can be done more effectively with architectures similar to ones used in object recognition.

Looking at this problem through the lens of computer vision, we use a convolutional network architecture inspired by the work of Simonyan & Zisserman (2014). We only use small convolutional filters of size 3×3, non-overlapping 2×2 pooling regions and our network also has more layers than networks previously considered for the purpose of speech recognition. The same architecture is shared between both baseline and student networks (described in detail in Figure 1, contrasted to a widely applied architecture proposed by Sainath et al. (2013)).

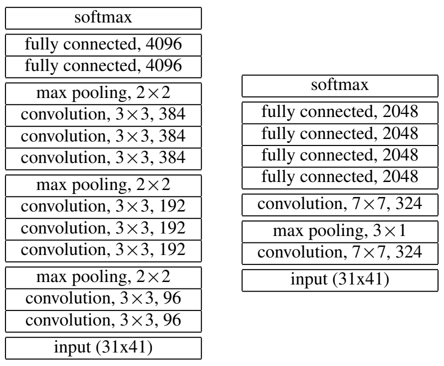


Figure 1: Left panel: configuration of the vision-style convolutional network used in our experiments. We used the stride of two pixels for max-pooling layers and one pixel for convolutional layers. We used no zero-padding in the first two convolutional layers and one pixel zero-padding for all remaining convolutional layers. Right panel: configuration of the CNN very closely resembling the one in the work of Sainath et al. (2013), adjusted to match the number of parameters in our network.

4 COMBINING BIDIRECTIONAL LSTMS WITH VISION-STYLE CNNS

Both LSTMs and CNNs are powerful models, but the mechanisms that guide their learning are quite different. That creates an opportunity to combine their predictions, implicitly averaging their inductive biases. A classic way to perform this is ensembling, that is, to mix posterior predictions of the two models in the following manner:

$$p(y|\mathbf{x}_i) = \gamma p_{\mathrm{LSTM}}(y|\mathbf{x}_i) + (1-\gamma) p_{\mathrm{CNN}}(y|\mathbf{x}_i),$$

where $\gamma \in [0,1]$. The notation $p_{\mathrm{LSTM}}(y|\mathbf{x}_i)$ and $p_{\mathrm{CNN}}(y|\mathbf{x}_i)$ denotes probabilities of class y given a feature vector $\mathbf{x}_i$, respectively for the LSTM and the baseline CNN. It is interesting to combinte these two types of models because they seem to meet the conditions of Dietterich (2000): “a necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the classifiers are accurate and diverse". Although ensembling is known to be very successful, it comes at the cost of executing all models at test time.

We propose an alternative which is to use model compression. Because capacity of the two models is similar we call it “model blending” rather than “model compression”. To combine the inductive biases of both the LSTM and the CNN, we can use a training objective that combines the loss function on the hard labels from the training data with a loss function which penalises deviation from predictions of the LSTM teacher. That is, we optimise

$$L(\lambda) = \lambda \left[-\sum_i \sum_c p_{\mathrm{LSTM}}(c|\mathbf{x}_i) \log q_{\mathrm{CNN}}(c|\mathbf{x}_i) \right] + (1-\lambda) \left[-\sum_i \log q_{\mathrm{CNN}}(y_i|\mathbf{x}_i) \right], \quad (1)$$

where $p_{\mathrm{LSTM}}(c|\mathbf{x}_i)$ is the probability of class c for training example $\mathbf{x}_i$ estimated by the teacher, $q_{\mathrm{CNN}}(c|\mathbf{x}_i)$ is the probability of class c assigned to training example $\mathbf{x}_i$ by the student and y_i is the correct class for $\mathbf{x}_i$. The coefficient $\lambda \in [0,1]$ controls the weight of the errors on soft and hard labels in the objective. When $\lambda = 0$ the network is only learning using the hard labels, ignoring the teacher, while $\lambda = 1$ means that the networks is only learning from the soft labels provided by the teacher, ignoring the hard labels. When $\lambda \in (0,1)$ optimising the objective in Equation 1 yields a form of hybrid model which is learning using the guidance of the teacher, although not depending on it alone. With a symmetric objective, we could train an LSTM using the guidance of the CNN. Instead, we blend into the CNN for efficency at test time.

We motivate the choice of working directly with probabilities instead of logits (cf. section 2) in two ways. First, it is more direct to interpret retaining a subset of predictions of a network when considering probabilities (cf. Equation 2). We can simply look at the fraction of probability mass a subset of outputs covers. This will be necessary in our work (see section 5). Secondly, when using both soft and hard targets, it is easier to find an appropriate λ and learning rate, when the two objectives we are weighting together are of similar magnitudes and optimisation landscapes.

5 EXPERIMENTS

In our experiments we use the Switchboard data set (Godfrey et al., 1992), which consists of 309 hours of transcribed speech. We used 308 hours as training set and kept one hour as a validation

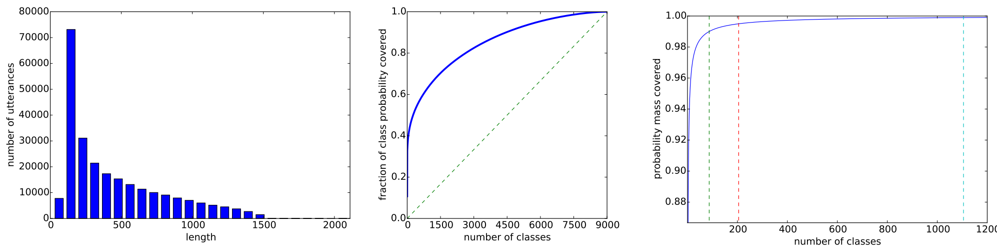


Figure 2: Left: distribution of the lengths (numbers of frames) of utterances in the training data. Center: fraction of class probability in the data covered as a function of the number of most likely classes included. Dashed line indicates what the curve would look like if the labels had a uniform distribution. The distribution is very non-uniform. Right: average fraction of probability mass of the teacher LSTM predictions covered as a function of the number of most probable classes retained. Dashed lines indicate the number of classes necessary to cover 99%, 99.5% and 99.9% of probability mass. Clearly, only very few top classes in the total of 9000 are necessary to cover a large majority of probability mass.

set. The data set is segmented into 248k utterances, i.e. continuous pieces of speech beginning and ending with a clear pause. Each utterance consists of a number of frames, i.e. 25 ms intervals of speech, with a constant shift of 10 ms. For every frame, the extracted features are 31-channel Mel-filterbank parameters passed through a 10-th root nonlinearity. Features for one utterance are visualised in Figure 3. To form our training and validation sets we extract windows of 41 frames, that is, the frame whose label we want to predict, 20 frames before it and 20 frames after it. As shown in Figure 2, distribution of the lengths of utterances is highly non-uniform, therefore to keep the sampling unbiased, we sample training examples by first sampling an utterance proportionally to its length and then sampling a window within that utterance uniformly. To form the validation set we simply extract all possible windows. In both cases, we pad each utterance with zeros at the beginning and at the end so that every frame in each utterance can be drawn as a middle frame. Every frame in the training and validation set has a label. The 9000 output classes represent tied tri-phone states that are generated by an HMM/GMM system (Young et al., 1994). Forced alignment is used to map each input frame to one output state using a baseline DNN model. The distribution of classes in the training data is visualised in Figure 2. We call the frame classification error on the validation set frame error rate (FER).

The test set is a part of the standard Switchboard benchmark (Hub5'00 SW). It was sampled from the same distribution as the training set and consists of 1831 utterances. There are no frame-level labels in the test set and the final evaluation is based on the ability to predict words in the test utterances. To obtain the words predicted by the model, frame label posteriors generated from the neural network are first divided by their prior probabilities then passed to a finite state transducer-based decoder to be combined with 3-gram language model probabilities to generate the most probable word sequences. Hypothesized word sequences are aligned to the human reference transcription to count the number of word insertions (I), deletions (D), and substitutions (S). Word error rate (WER) is defined as $\text{WER} = \frac{S+D+I}{N} \times 100$, where N is the total number of words in the reference transcription.

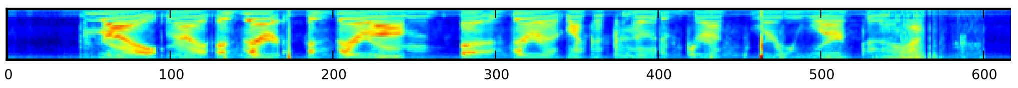


Figure 3: Example utterance from Switchboard. The x-axis indicates time and the y-axis frequency.

Since the teacher model is very slow at prediction time, it is impractical to run a large number of experiments if the teacher must repeatedly be executed to train each student. Unfortunately, running the teacher once and saving its predictions to disk is also problematic — because the output space is large (9000 classes), storing the soft labels for all classes would require a large amount of space ($\approx$ 3.6 TB). Moreover, to sample each minibatch in an unbiased manner, we would need constant random access to disk, which again would make training very slow. To deal with that problem we save predictions only for the small subset of classes with the highest predicted probabilities. To determine whether this is a viable solution, we checked what percentage of the total probability mass, averaged over the examples in the training set, is covered by the C most likely classes according to the teacher model. We denote that set $\mathrm{TOP}_C(\mathbf{x})$. That is, we compute

$$\mathrm{M}(C) = \frac{1}{|\{\mathbf{x}_i\}|} \sum_{\mathbf{x}_i} \sum_{y \in \mathrm{TOP}_C(\mathbf{x}_i)} p(y|\mathbf{x}_i), \tag{2}$$

where $p(y|\mathbf{x})$ denotes the posterior probability of class y given a feature vector $\mathbf{x}_i$. This relationship for one of our LSTM models is shown in Figure 2. We found that, with very few exceptions, posteriors over classes are concentrated on very few values. Therefore, we decided to continue our experiments retaining top classes covering not more than 90 classes for each training example, cutting off after covering 99% of the probability mass. This allows us to store soft labels for the entire data set in the RAM, making unbiased sampling of training data efficient.

5.1 Baseline networks

We used Lasagne, which is based on Theano (Bergstra et al., 2010), for implementing CNNs. We used the architecture described in Figure 1. For training of the CNN baseline we used mini-batches of size 256. Each epoch consisted of 2000 mini-batches. Hyper-parameters of the baseline networks were: initial learning rate (1.7×10^{-2}), momentum coefficient (0.9, we used Nesterov's momentum) and a learning rate decay coefficient (0.7). Because the data set we used is very large (309 hours, 18 GB), the only form of regularisation we used was early stopping in the following form. After every epoch we measured the loss on the validation set. If the loss did not improve for five epochs, we multiplied the learning rate by a learning rate decay coefficient. We stopped the training after the learning rate was smaller than 5×10^{-5}. It took about 200 epochs to finish. We repeated training with three different random seeds. For comparison, we also trained a CNN very similar to the one proposed by Sainath et al. (2013), adjusted to match the number of parameters in our network. We used the same training procedure and hyperparameters.

The bidirectional LSTM teacher networks we used are very similar to the one in the work of Mohamed et al. (2015). We trained three models, all with four hidden layers, two with 512 hidden units for each direction and one with 800 hidden units for each direction. The training starts with the learning rate equal to 0.05 for one of the smaller models and 0.08 for the other ones. We used the standard momentum with the coefficient of 0.9. After three epochs of no improvement of frame error rate on the validation set, the learning rate was multiplied by $\frac{2}{3}$ and training process was rolled back to the last epoch which improved validation error. Training stopped when learning rate was smaller than 10^{-5}. It took about 75 epochs (2000 minibatches of 256 samples) to finish the training.

The results for these models are shown in Table 1. Our vision-style CNN achieved 14.1 WER (averaged over three random seeds), the larger LSTM achieved 14.4 WER, and a CNN of an architecture proposed by Sainath et al. (2013) achieved 15.5 WER. Interestingly, although our LSTM teachers outperform our vision-style CNN trained with hard labels in terms of FER, the results in WER, which is the metric of primary interest, are the opposite. This discrepancy between FER and WER has been observed in the speech community before, for example by Sak et al. (2014) for LSTMs and DNNs. FER and WER are not always perfectly correlated because FER is conditioned on a model that generated the frame alignment in the first place (which might not be correct for all the cases). FER also penalizes misclassifications of boundary frames which might not be of importance as long as the correct target state is recognized. On the other hand, WER is calculated taking into account information about neighbouring frames (i.e. smoothness) as well as external knowledge (e.g. a language model) which corrects many of the misclassifications made locally.

5.2 Ensembles of networks

The first approach we use to combine the two types of models is to create ensembles. The results in Figure 4 and Figure 5 indicate that for the problem we consider it is beneficial to combine neural networks from different families, which have different inductive biases. Even though CNNs are much weaker in FER, combining an LSTM with a vision-style CNN achieves the same FER as an ensemble of two LSTMs (both of which are more accurate than the CNN), and actually yields better WER than ensembles of two CNNs or two (superior) LSTMs. Interestingly, ensembling two CNNs yields almost no benefit in WER. To complete the picture we tried ensembles with more than

Table 1: FER and WER for our models. The numbers for the vision-style CNNs and LSTM→CNN blending are averages over three random seeds. The numbers for smaller LSTM are given for a better of the two on FER, the other one achieved 34.71% FER and the same WER.

	FER	**WER**	model size	execution time
Sainath et al. (2013)-style CNN	37.93%	15.5	$\approx$ 75M	$\times$ 0.75
vision-style CNN	35.51%	14.1	$\approx$ 75M	$\times$ 1.0
smaller LSTM	34.27%	14.8	$\approx$ 30M	$\times$ 3.3
bigger LSTM	34.15%	14.4	$\approx$ 65M	$\times$ 5.8
LSTM + CNN ensemble ($\gamma = 0.5$)	32.4%	13.4	$\approx$ 130M	$\times$ 6.8
LSTM → CNN blending ($\lambda = 0.75$)	34.11%	13.83	$\approx$ 75M	$\times$ 1.0

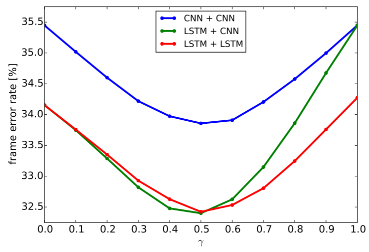


Figure 4: FER of ensembles as a function of γ.

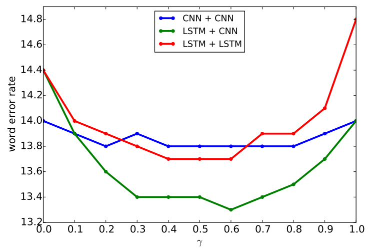


Figure 5: WER of ensembles as a function of γ.

two models. An ensemble of three LSTMs achieved 31.98% FER and 13.7 WER, an ensemble of three CNNs achieved 33.31% FER and 13.9 WER, thus, in both cases, yielding very little gain over ensembles of two models of these types. On the other hand adding a CNN to the ensemble of two LSTMs yielded 31.57% FER and 13.2 WER. The benefits of adding more models to the ensemble appear to be negligible if the ensemble contained at least one model of each type already. Although the ensembles we trained are very effective in terms of WER, it comes at the cost of a large increase of computation at test time compared to the baseline CNN. Because of the cost of the LSTMs, our best two-model ensemble is about 7 times slower than our vision-style CNN (cf. Table 1).

We also compared the errors made by CNNs and LSTMs to see if the models are qualitatively different. We observe that a CNN tends to make similar errors as other CNNs, an LSTM tends to make similar errors as other LSTMs, but CNNs and LSTMs tend to make errors that are less similar to each other than the CNN-CNN and LSTM-LSTM comparisons. Overall, we conclude that the inductive biases of LSTM and CNN are complementary.

5.3 Networks trained with model blending via model compression

The next question we tackle is whether it is possible to achieve an effect similar to creating an ensemble without having to execute all models at prediction time. We attack this with model blending via model compression. To do this, we took predictions of the two best performing LSTMs in Table 1 and averaged their predictions to form a teacher model. Such a teacher model achieves 32.4% FER and 13.4 WER. As we mentioned earlier, it is infeasible to use all predictions during training. Therefore we only store a subset of classes predicted by the teacher for each frame. Table 2 shows what fraction of probability mass is covered when storing different maximum number of predictions (C). It is particularly interesting to understand how many predictions of the teacher model are sufficient to achieve good performance to test the *dark knowledge* hypothesis (Hinton et al., 2015) which states that information about classes predicted with low probability is important to the success of

Table 2: Average fraction of probability mass covered and average number of classes retained after truncating predictions of an ensemble of two LSTMs to fit them in the memory.

maximum number of classes retained	**1**	**3**	**10**	**30**	**90**
average fraction of probability mass covered	73.20%	91.19%	96.68%	98.38%	99.03%
average number of classes retained	1.0	2.89	6.79	13.33	23.96

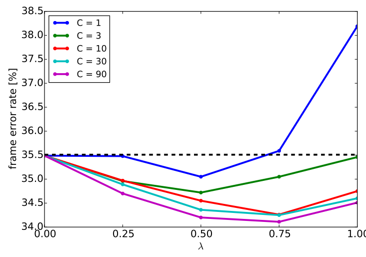


Figure 6: FER of the CNN student as a function of λ as C, the maximum of number of teacher outputs retained, varies from 1 to 90. Dashed line indicates performance of a baseline CNN.

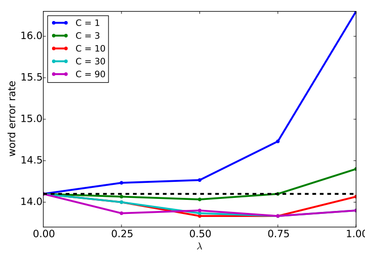


Figure 7: WER of the CNN student as a function of λ as C, the maximum of number of teacher outputs retained, varies from 1 to 90. Dashed line indicates performance of a baseline CNN.

model compression. We use $C \in \{90, 30, 10, 3, 1\}$. We also vary the parameter λ which controls how much the student is learning from the teacher and how much it is learning from hard labels. The architecture and training procedure for the students is the same as for the baseline. For every combination of C and λ we report an average over three random seeds.

The results are shown in Figure 6 and Figure 7. The best model achieved lower FER (34.11%) and lower WER (13.83) than any of the individual models. Furthermore, the blended model has fewer parameters and is 6.8 times faster at test time than the ensemble of the LSTM and the CNN. For all numbers of teacher predictions retained (C) the best performance was achieved for $\lambda \in \{0.25, 0.5, 0.75\}$. That highlights the importance of blending the knowledge extracted from the teacher model with learning from hard labels within the architecture of the student.

Our experiments show that, at least for this task, it is not critical to use the teacher predictions for all classes. Just the 30 most likely predictions ($\frac{1}{300}$ of all classes!) is enough. Bringing the number up to 90 classes did not improve the student WER performance. However, performance deteriorates dramatically when too few predictions are used, suggesting that some dark knowledge is needed.

Finally, we experimented with using a CNN as a teacher for a CNN of the same architecture, i.e. we took predictions of a baseline vision-style CNN and used its predictions to train another CNN of the same architecture (with $\lambda = 0.5$ and $C = 90$). Such a student achieves on average (over 3 random trials) 34.61% FER and 14.1 WER, which is, as we expected, worse than a student of the LSTMs since the two models are more similar, but still significantly better than the baseline in terms of FER. These results are consistent with the results for ensembles (cf. Figure 4 and Figure 5). Clearly, blending dissimilar models like CNNs and LSTMs is stronger.

6 Related work

A few papers have applied model compression in speech recognition settings. The most similar is by Chan et al. (2015) who compressed LSTMs into small DNNs without convolutional layers. Using the soft labels from an LSTM, they were able to show an improvement in WER over the baseline trained with hard labels. The main difference between this work and ours is that their students are non-convolutional and tiny. While this allows for a decent improvement over the baseline, since the student network is much smaller, its performance is still much weaker than performance of a single model of the same type as the teacher. Hence, this work addresses a different question than we do, i.e. whether a network without recurrent structure can perform as well or better as an LSTM when using soft labels provided by the LSTM. Model compression was also successfully applied to speech recognition by Li et al. (2014) who used DNNs without convolutional layers both as a teacher and a student. The architecture of the two networks was the same except that the student had less hidden units in each layer. Finally, work in the opposite direction was done by Wang et al. (2015) and Tang et al. (2015). They demonstrated that when using a small data set for which an LSTM is overfitting, a deep non-convolutional network can provide useful guidance for the LSTM. It can come either in the form of pre-training the LSTM with soft labels from a DNN or training the LSTM optimising a loss mixing hard labels with soft labels from a DNN. We are not aware of previous work on model compression in the setting, in which the student and the teacher are of similar capacity.

7 Discussion

The main contribution of this paper is introducing the use of model compression in an unexplored setting where both the teacher and student architectures are powerful ones, yet with different inductive biases. Thus, rather than calling it model compression we use the term "model blending". We showed that the LSTM and the CNN learn different kinds of knowledge from the data which can be leveraged through simple ensembling or model blending via model compression. We provided experimental evidence that CNNs of appropriate vision-style architecture have the necessary capacity to learn accurate predictors on large speech data sets and gave a simple, practical recipe for improving the performance of CNN-based speech recognition models even further at no cost during test time. We hypothesise that the very recent advances in training even deeper convolutional networks for computer vision (Srivastava et al., 2015; He et al., 2015) will yield improved performance in speech recognition and would further improve our results. Finally, by using a CNN to teach a CNN, we have shown a very easy way of improving a neural network without training networks of more than one architecture or even forming ensembles.

Acknowledgments

We thank Stanisław Jastrzębski for suggesting the name of the paper. We also thank Steve Renals and Paweł Świętojański for insightful comments.

References

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *ICASSP*, 2012.

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *TASLP*, 22, 2014.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NIPS*, 2014.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *SciPy*, 2010.

Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *KDD*, 2006.

William Chan, Nan Rosemary Ke, and Ian Laner. Transferring knowledge from a RNN to a DNN. *arXiv:1504.01483*, 2015.

Yann Dauphin and Yoshua Bengio. Big neural networks waste capacity. *arXiv:1301.3583*, 2013.

Misha Denil, Babak Shakibi, Laurent Dinh, and Nando de Freitas. Predicting parameters in deep learning. In *NIPS*, 2013.

Thomas G. Dietterich. Ensemble methods in machine learning. In *MCS*, 2000.

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. Learning precise timing with LSTM recurrent networks. *JMLR*, 3, 2003.

John J. Godfrey, Edward C. Holliman, and Jane McDaniel. Switchboard: telephone speech corpus for research and development. In *ICASSP*, 1992.

Alex Graves. *Supervised sequence labelling with recurrent neural networks*. 2012.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2014.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6), 2005.

Alex Graves and Jürgen Schmidhuber. Offine handwriting recognition with multidimensional rnns. In *NIPS*, 2009.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8), 1997.

Brian Kingsbury. Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling. In *ICASSP*, 2009.

Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *NIPS*, 1990.

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. In *The Handbook of Brain Theory and Neural Networks*. 1998.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

Honglak Lee, Peter Pham, Yan Largman, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *NIPS*, 2009.

Jinyu Li, Rui Zhao, Jui-Ting Huang, and Yifan Gong. Learning small-size dnn with output-distribution-based criteria. In *INTERSPEECH*, 2014.

Abdel-rahman Mohamed, Frank Seide, Dong Yu, Jasha Droppo, Andreas Stolcke, Geoffrey Zweig, and Gerald Penn. Deep bidirectional recurrent networks over spectral windows. In *ASRU*, 2015.

Adriana Romero, Nicolas Ballas, Samira Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2014.

Tara Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *ICASSP*, 2013.

Tara Sainath, Brian Kingsbury, George Saon, Hagen Soltau, Abdel-rahman Mohamed, George Dahl, and Bhuvana Ramabhadran. Deep convolutional neural networks for large-scale speech tasks. *Neural Networks*, 64, 2015.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv:1402.1128*, 2014.

Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *TSP*, 45(11), 1997.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2014.

Rupesh K. Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *NIPS*, 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Zhiyuan Tang, Dong Wang, Yiqiao Pan, and Zhiyong Zhang. Knowledge transfer pre-training. *arXiv:1506.02256*, 2015.

Gregor Urban, Krzysztof J. Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdel rahman Mohamed, Matthai Philipose, and Matthew Richardson. Do deep convolutional nets really need to be deep (or even convolutional)? In *ICLR (workshop track)*, 2016.

Karel Veselý, Arnab Ghoshal, Lukáŝ Burget, and Daniel Povey. Sequence discriminative training of deep neural networks. In *INTERSPEECH*, 2013.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *NIPS*, 2015.

Dong Wang, Chao Liu, Zhiyuan Tang, Zhiyong Zhang, and Mengyuan Zhao. Recurrent neural network training with dark knowledge transfer. *arXiv:1505.04630*, 2015.

S. J. Young, J. J. Odell, and P. C. Woodland. Tree-based state tying for high accuracy acoustic modelling. In *HLT*, 1994.

SUPPLEMENTARY MATERIAL

DETAILS OF THE LSTM

Given a sequence of input vectors $\boldsymbol{x} = (x_1, \ldots, x_T)$, an RNN computes the hidden vector sequence $\boldsymbol{h} = (h_1, \ldots, h_T)$ by iterating the following from $t = 1$ to T:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right).$$

The W terms denote weight matrices (e.g. W_{xh} is the input-hidden weight matrix), the b terms denote bias vectors (e.g. b_h is hidden bias vector) and $\mathcal{H}$ is the hidden layer function.

While there are multiple possible choices for $\mathcal{H}$, prior work (Graves, 2012; Graves et al., 2013; Sak et al., 2014) has shown that the LSTM architecture, which uses purpose-built *memory cells* to store information, is better at finding and exploiting longer context. The left panel of Figure 8 illustrates a single LSTM memory cell. For the version of the LSTM cell used in this paper (Gers et al., 2003) $\mathcal{H}$ is implemented by the following composite function:

$$\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right),\\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right),\\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right),\\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right),\\
h_t &= o_t \tanh(c_t),
\end{aligned}$$

where σ is the logistic sigmoid function, and i, f, o and c are respectively the *input gate*, *forget gate*, *output gate* and *cell* activation vectors, all of which are the same size as the hidden vector h. The weight matrices from the cell to gate vectors (e.g. W_{ci}) are diagonal, so element m in each gate vector only receives input from element m of the cell vector.

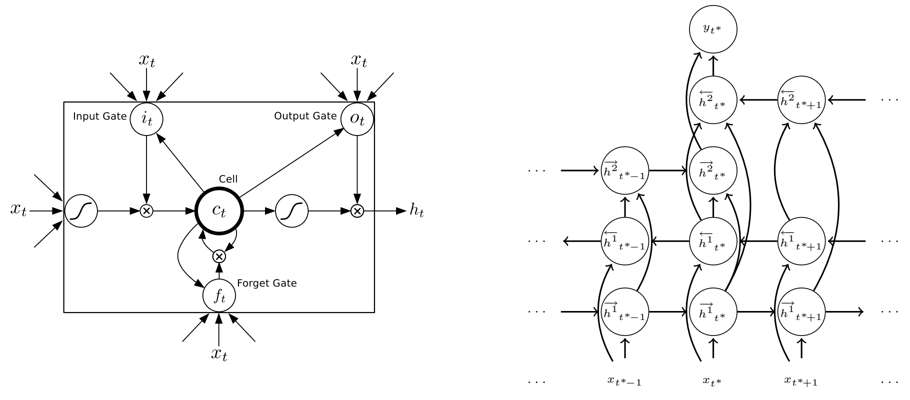

x_t
Input Gate
i_t
Output Gate
o_t
Cell
c_t
h_t
Forget Gate
f_t
y_{t^*}
$\overleftarrow{h^2}_{t^*}$
$\overleftarrow{h^2}_{t^*+1}$
$\overrightarrow{h^2}_{t^*-1}$
$\overrightarrow{h^2}_{t^*}$
$\overleftarrow{h^1}_{t^*-1}$
$\overleftarrow{h^1}_{t^*}$
$\overleftarrow{h^1}_{t^*+1}$
$\overrightarrow{h^1}_{t^*-1}$
$\overrightarrow{h^1}_{t^*}$
$\overrightarrow{h^1}_{t^*+1}$
x_{t^*-1}
x_{t^*}
x_{t^*+1}
...

Figure 8: Left panel: the LSTM cell. Figure from Graves et al. (2013). Right panel: two-layer bidirectional RNN with one output.

One shortcoming of conventional RNNs is that they are only able to make use of previous context. Bidirectional RNNs (BRNNs) (Schuster & Paliwal, 1997) exploit past and future context by processing the data in both directions with two separate hidden layers, which are then fed forwards to the same output layer. A BRNN computes the *forward* hidden sequence $\overrightarrow{\boldsymbol{h}} = (\overrightarrow{h}_1, \ldots, \overrightarrow{h}_T)$ and the *backward* hidden sequence $\overleftarrow{\boldsymbol{h}} = (\overleftarrow{h}_1, \ldots, \overleftarrow{h}_T)$ by iterating from $t = 1$ to T:

$$\begin{aligned}
\overrightarrow{h}_t &= \mathcal{H}\left(W_{x\overrightarrow{h}}x_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right),\\
\overleftarrow{h}_t &= \mathcal{H}\left(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right).
\end{aligned}$$

Combining BRNNs with LSTMs gives the bidirectional LSTM, which can access the context in both directions.

Finally, deep RNNs can be created by stacking multiple RNN hidden layers on top of each other, with the output sequence of one layer forming the input sequence for the next. Assuming the same hidden layer function is used for all N layers in the stack, the hidden vector sequences $\boldsymbol{h}^n$ are iteratively computed from $n = 1$ to N and $t = 1$ to T:

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n\right),$$

where we define $\boldsymbol{h}^0 = \boldsymbol{x}$.

Deep bidirectional RNNs can be implemented by replacing each hidden sequence $\boldsymbol{h}^n$ with the forward and backward sequences $\overrightarrow{\boldsymbol{h}}^n$ and $\overleftarrow{\boldsymbol{h}}^n$, and ensuring that every hidden layer receives input from both the forward and backward layers at the level below. If bidirectional LSTMs are used for the hidden layers we get deep bidirectional LSTMs, the architecture we use as a teacher network in this paper.

In this work, following Mohamed et al. (2015), we only predict the label of the middle frame, hence the network output is computed as $y_{t^*} = W_{h^N y} h_{t^*}^N + b_y$. This also implies that in the last hidden layer the forward sequence runs only from 1 to t^* and the backward sequence runs only from T to t^*. This is illustrated in the right panel of Figure 8.

6.2 Comments on the paper

In one of the experiments in this paper we have showed that blending a CNN into another CNN of the same architecture improves FER but, unfortunately, does not improve the WER. Although this is not very practical result for speech recognition as in that domain the objective is WER and not FER, it is interesting if transferable to other domains, for example to computer vision.

An obvious question to ask in this context is whether the same trick would work more than once. Can we use the CNN which was a student in this experiment as a teacher for another CNN to form a chain of such models? We performed an experiment to verify the answer to this question. In order to do that, we took the best (in terms of FER) of the three CNNs students of the CNN in our paper and used this model as a teacher. FER of this model was 34.43%. We trained three new students with $\lambda = 0.5$ and $C = 90$ using three different random seeds. These models achieved 34.56% FER on average, not improving over their teacher, which seems to indicate that a single blending procedure is enough.

Chapter 7

Conclusions

7.1 Summary

In this thesis we demonstrate how to exploit the concept of diversity for efficient machine learning in a number of contexts. First, in Chapter 3 we show how knowledge of diversity in the structure of the training data set can be exploited to achieve a better estimate of the test error. Secondly, in Chapter 4 and Chapter 5 we demonstrate in the context of unsupervised learning how diversity in the representation learnt can be enforced. Then, finally, in Chapter 6 we show that difference in the inductive biases of a neural network architectures can be exploited to achieve higher accuracy in a supervised task by model blending through model compression.

7.2 Future directions

One of the limitations of the models we considered in Chapter 4 and Chapter 5 is that we are using only single-layer fully-connected networks. While experiments with such networks are sufficient to demonstrate the usefulness of our ideas, modern computer vision tasks require being able to handle much larger inputs, which is not feasible using only fully-connected layers. To do that we would need to use convolutional autoencoders (e.g. Masci et al. [2011], Makhzani and Frey [2015b]) instead. To achieve results competitive with the state-of-the art we should also use deeper networks.

In Chapter 6 we showed that neural network architectures which work very well in computer vision tasks are also very effective for speech recognition and the two domains are quite similar. This suggests that the type of neural network architectures used is not an isolated property transferable between these two modalities. More elements of computer vision are likely to be found useful in the speech community. One particular example of such technique we hypothesise which might have a bigger impact on speech recognition is data augmentation. Another one might be using even deeper convolutional networks [He et al., 2015, Srivastava et al., 2015].

Blending of the LSTM into the CNN we performed in Chapter 6 is not the only possibility. CNN could also be blended into an LSTM. We do not do that because the CNNs are a lot faster at prediction time. One more interesting direction, which was not yet explored, would be to try to use an LSTM as a teacher of a simpler recurrent neural network. This experiment could bring some insight into how critical it is to use an LSTM cells in the recurrent network.

Finally, we hypothesize that soft labels provided by the teacher model can be used not only for model compression or model blending but also to assess difficulties of different training samples. Possible ways of quantifying a difficulty of an example include predicted probability of the correct class, entropy of the predictions or the combination of both. These estimates could be used to create a series of learning problems, starting from easy ones and getting progressively harder. Using this sequence of tasks would be a type of curriculum learning procedure similar to what we proposed in Chapter 4.

Bibliography

The top 20 valuable facebook statistics. `http://zephoria.com/top-15-valuable-facebook-statistics/`. Accessed: 2016-05-25.

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *International Conference on Acoustics, Speech, and Signal Processing*, 2012.

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22, 2014.

Eugene L. Allgower and Kurt Georg. *Numerical continuation methods. An introduction.* Springer-Verlag, 1980.

Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4, 2010.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, 2014.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Representation Learning*, 2014.

Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2, 1989.

Dana H. Ballard. Modular learning in neural networks. In *Association for the Advancement of Artificial Intelligence Conference*, 1987.

Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*, 5, 2003.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, 2007.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning*, 2009.

John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boomboxes and blenders: Domain adaptation for sentiment classification. In *Association for Computational Linguistics*, 2007.

Herve Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4), 1988.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1), 1997.

Yuan Shih Chow and Henry Teicher. *Probability theory: independence, interchangeability, martingales.* Springer, 2012.

Dan Ciresan, Ueli Meier, and Jurgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition*, 2012.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition*, 2005.

Andreas C. Damianou and Neil D. Lawrence. Deep gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, 2013.

Hal Daumé. Frustratingly easy domain adaptation. In *Association for Computational Linguistics*, 2007.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Computer Vision and Pattern Recognition*, 2009.

Thomas G. Dietterich. Ensemble methods in machine learning. In *International Workshop on Multiple Classifier Systems*, 2000.

Eizaburo Doi and Michael S. Lewicki. A simple model of optimal population coding for sensory systems. *PLOS Computational Biology*, 10(8), 2014.

Eizaburo Doi, Doru-Cristian Balcan, and Michael S. Lewicki. A theoretical analysis of robust coding over noisy overcomplete channels. *Advances in Neural Information Processing Aystems*, 18, 2006.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11, 2010.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4), 1980.

Krzysztof J. Geras and Charles Sutton. Composite denoising autoencoders. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2016.

Krzysztof J. Geras and Charles Sutton. Multiple-source cross-validation. In *International Conference on Machine Learning*, 2013.

Krzysztof J. Geras and Charles Sutton. Scheduled denoising autoencoders. In *International Conference on Learning Representations*, 2015.

Krzysztof J. Geras, Abdel-rahman Mohamed, Rich Caruana, Gregor Urban, Shengjie Wang, Ozlem Aslan, Matthai Philipose, Matthew Richardson, and Charles Sutton. Blending lstms into cnns. In *International Conference on Learning Representations (workshop track)*, 2016.

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of Machine Learning Research*, 3, 2003.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. 2016.

Yves Grandvalet and Yoshua Bengio. Hypothesis testing for cross-validation. Technical Report 1285, Département d'informatique et recherche opérationnelle, Université de Montréal, 2006.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2014.

Alex Graves. *Supervised sequence labelling with recurrent neural networks.* 2012.

Alex Graves and Jürgen Schmidhuber. Offine handwriting recognition with multidimensional rnns. In *Advances in Neural Information Processing Systems*, 2009.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *International Conference on Acoustics, Speech, and Signal Processing*, 2013a.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *International Conference on Acoustics, Speech, and Signal Processing*, 2013b.

Caglar Gulcehre and Yoshua Bengio. Knowledge matters: Importance of prior information for optimization. *Journal of Machine Learning Research*, 17(8), 2016.

Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions. *arXiv:1603.00391*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7), 2006.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8), 1997.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Association for Computational Linguistics*, 2014.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Representation Learning*, 2014.

Diederik P. Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

Kai A. Krueger and Peter Dayan. Flexible shaping: How learning in small steps helps. *Cognition*, 110(3), 2009.

Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Rbert E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4), 1989.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1990.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

David G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, 1999.

Alireza Makhzani and Brendan Frey. k-sparse autoencoders. In *International Conference on Representation Learning*, 2015a.

Alireza Makhzani and Brendan J Frey. Winner-take-all autoencoders. In *Advances in Neural Information Processing Systems*, 2015b.

Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, 2011.

Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. In *International Conference on Learning Representations (workshop track)*, 2015.

Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 2011.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 2010.

Ben Poole, Jascha Sohl-Dickstein, and Surya Ganguli. Analyzing noise in autoencoders and deep networks. *arXiv:1406.1831*, 2014.

Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, 2015.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *International Conference on Machine Learning*, 2011.

Tara Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *International Conference on Acoustics, Speech, and Signal Processing*, 2013.

Tara Sainath, Brian Kingsbury, George Saon, Hagen Soltau, Abdel-rahman Mohamed, George Dahl, and Bhuvana Ramabhadran. Deep convolutional neural networks for large-scale speech tasks. *Neural Networks*, 64, 2015.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv:1402.1128*, 2014.

Robert E. Schapire. The strength of weak learnability. *Machine learning*, 5(2), 1990.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *arXiv:1404.7828*, 2014.

Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11), 1997.

H. Sebastian Seung. Learning continuous attractors in recurrent networks. In *Advances in Neural Information Processing Systems*, 1998.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Representation Learning*, 2014.

B. F. Skinner. Reinforcement today. *American Psychologist*, 13(3), 1958.

Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing.* California Technical Publishing, 1997.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, 2015.

Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2), 1974.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014.

Yichuan Tang and Abdel-rahman Mohamed. Multiresolution deep belief networks. In *AISTATS*, 2012.

Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, 1996.

Gregor Urban, Krzysztof J. Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdel-rahman Mohamed, Matthai Philipose, and Matthew Richardson. Do deep convolutional nets really need to be deep (or even convolutional)? In *ICLR (workshop track)*, 2016.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 2010.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, 2015.

Jason Weston, Frédéric Ratle, and Ronan Collobert. Deep learning via semi-supervised embedding. In *International Conference on Machine Learning*, 2008.

Richard S. Zemel. *A Minimum Description Length Framework for Unsupervised Learning*. PhD thesis, Dept. of Computer Science, University of Toronto, 1993.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, 2015.