New Techniques in Deep Representation Learning

Galen Andrew

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2016

Reading Committee:

Emanuel Todorov, Chair

Emily Fox

Carlos Guestrin

Luke Zettlemoyer

Program Authorized to Offer Degree:
UW Computer Science and Engineering

University of Washington

Abstract

New Techniques in Deep Representation Learning

Galen Andrew

Chair of the Supervisory Committee:
Associate Professor Emanuel Todorov
CSE, joint with AMATH

The choice of feature representation can have a large impact on the success of a machine learning algorithm at solving a given problem. Although human engineers employing task-specific domain knowledge still play a key role in feature engineering, automated domain-independent algorithms, in particular methods from the area of *deep learning*, are proving more and more useful on a variety of difficult tasks, including speech recognition, image analysis, natural language processing, and game playing. This document describes three new techniques for automated domain-independent deep representation learning:

- *Sequential deep neural networks* (SDNN) learn representations of data that is continuously extended in time such as audio. Unlike "sliding window" neural networks applied to such data or convolutional neural networks, SDNNs are capable of capturing temporal patterns of arbitrary span, and can encode that discovered features should exhibit greater or lesser degrees of continuity through time.

- *Deep canonical correlation analysis* (DCCA) is a method to learn parametric nonlinear transformations of multiview data that capture latent shared aspects of the views so that the learned representation of each view is maximally predictive of (and predicted by) the other. DCCA may be able to learn to represent abstract properties when the two views are not superficially related.

- The *orthant-wise limited-memory quasi-Newton* algorithm (OWL-QN) can be employed to train any parametric representation mapping to produce parameters that are sparse (mostly zero), resulting in more interpretable and more compact models. If the prior assumption that parameters should be sparse is reasonable for the data source, training with OWL-QN should also improve generalization.

Experiments on many different tasks demonstrate that these new methods are computationally efficient relative to existing comparable methods, and often produce representations that yield improved performance on machine learning tasks.

TABLE OF CONTENTS

Page

List of Figures . . . iii

Glossary . . . iv

Chapter 1: Introduction . . . 1
- 1.1 Contributions and thesis overview . . . 4

Chapter 2: The current state of research in deep representation learning . . . 7
- 2.1 Parameter optimization for deep representation learning models . . . 7
- 2.2 Specialized architectures for structured data . . . 14
- 2.3 Computational considerations . . . 21
- 2.4 Conclusion . . . 22

Chapter 3: Learning deep representations for continuously extended data . . . 23
- 3.1 Continuously extended data . . . 23
- 3.2 The sequential restricted Boltzmann machine (SRBM) . . . 28
- 3.3 The sequential deep neural network (SDNN) . . . 34
- 3.4 Backpropagation algorithm . . . 36
- 3.5 Experiments . . . 40
- 3.6 Conclusion . . . 43

Chapter 4: Learning deep representations for multiview data . . . 45
- 4.1 Multiview data . . . 45
- 4.2 Related work . . . 47
- 4.3 Background: CCA, KCCA, and deep representations . . . 47
- 4.4 Deep canonical correlation analysis (DCCA) . . . 50
- 4.5 Experiments . . . 56

4.6 Conclusion 63

Chapter 5: Learning sparsely connected neural representation models with L_1-regularization 64

5.1 Introduction 64

5.2 Quasi-Newton algorithms and L-BFGS 68

5.3 Orthant-wise limited-memory quasi-Newton (OWL-QN) for convex objectives 69

5.4 Experiments with convex loss 73

5.5 OWL-QN for non-convex loss functions 82

5.6 Experiments with non-convex loss 83

5.7 Conclusions 86

Chapter 6: Conclusions 88

6.1 The future of representation learning 89

Bibliography 93

Appendix A: Proof of claim in derivation of SDNN gradient 115

Appendix B: Derivation of DCCA gradient 117

Appendix C: Termination of OWL-QN projected line search 120

LIST OF FIGURES

Figure Number — Page

- 2.1 Illustration of multilayer perceptron 8
- 2.2 The basic restricted Boltzmann machine (RBM) graphical model structure. . . 9
- 2.3 The basic autoencoder network. 11
- 2.4 Illustration of convolutional neural network. 15
- 2.5 Illustration of recurrent neural network. 17
- 2.6 Long short-term memory unit 19
- 3.1 Illustration of sequential restricted Boltzmann machine structure 31
- 3.2 Illustration of sequential deep neural network structure 35
- 3.3 Sequential deep neural network performance on TIMIT 42
- 4.1 Illustration of deep canonical correlation analysis structure 51
- 4.2 Modified cube-root nonlinearity 55
- 4.3 Examples of left and right views of split MNIST dataset. 58
- 4.4 DCCA/KCCA comparison as representation size varies 61
- 5.1 Algorithmic description of OWL-QN 72
- 5.2 Performance of L_1- and L_2-regularized models on parse reranking 77
- 5.3 Objective value during optimization on parse reranking 79
- 5.4 Comparing AlgLib's L-BFGS with our own implementation 80
- 5.5 Number of non-zero weights during training of parse reranking model 81
- 5.6 MNIST test set errors during training with OWL-QN. 84
- 5.7 Number of non-zero weights during training of classifier for MNIST. 85

GLOSSARY

AE:	Autoencoder
CCA:	Canonical correlation analysis
CD:	Contrastive divergence
CNN:	Convolutional neural network
CRF:	Contional random field
DAE:	Denoising autoencoder
DCCA:	Deep canonical correlation analysis
DNN:	Deep neural network
GPU:	Graphics processing unit
HMM:	Hidden Markov model
KCCA:	Kernel canonical correlation analysis
L-BFGS:	Limited-memory BFGS (BroydenFletcherGoldfarbShanno)
LSTM:	Long short-term memory
MFCCS:	Mel-frequency cepstral coefficients
MLP:	Multilayer perceptron
MRF:	Markov random field
MT:	Machine translation

NLP:	Natural language processing
NN:	Neural network
OWL-QN:	Orthant-wise limited-memory quasi-Newton
PCA:	Principal components analysis
RBF:	Radial basis function
RBM:	Restricted Boltzmann machine
RELU:	Rectified linear unit
RKHS:	Reproducing kernel Hilbert space
RL:	Representation learning
RNN:	Recurrent neural network
SDNN:	Sequential deep neural network
SGD:	Stochastic gradient descent
SRBM:	Sequential restricted Boltzmann machine

ACKNOWLEDGMENTS

The author wishes to express his sincere appreciation to the professors, colleagues and students with whom I have collaborated on the research presented in this thesis, and from whom I have learned so much, including Jianfeng Gao, Kristina Toutanova, Mark Johnson, Eric Brill, Chris Burgess, Raman Arora, Karen Livescu, Rishabh Iyer, Igor Mordatch, and Emo Todorov.

This research presented here was supported by Microsoft Research (where I was employed), the University of Washington, and by grants from the NSF (IIS-0905633) and the Intel/UW ISTC.

DEDICATION

to Kristina, the best co-parent I could imagine

Chapter 1

INTRODUCTION

The ultimate success of a machine learning algorithm applied to a given task is often determined at least as much by the choice of data representation (i.e., feature representation) as by the choice of algorithm or algorithm hyperparameters. A machine learning expert usually cannot achieve the best results without collaborating with a domain expert to identify which aspects of the problem would most fruitfully be included in the input. Yet, natural learning systems, such as animal brains and in particular the human brain, seem to be able to determine which aspects of their high-dimensional input are worth focusing on with comparatively little guidance. This difference in the feature representation, as much as differences in the "learning algorithm" employed by artificial vs. natural learning agents to make predictions based on those features, may underlie the difficulty in creating computer learning systems that can flexibly respond to high-dimensional sensory input and perform "hard AI" tasks, such as communicating fluently in natural language, human-level image and audio understanding, and robot control in noisy and uncertain environments.

In this thesis, we will take *representation learning* to mean any technique for constructing a mapping from one data representation to another, with the intention that the altered data would be more useful as input to a machine learning algorithm. Typically the input data would be low-level, raw and unabstracted, such as pixel intensities for visual input, or local air pressure deviation readings as measured by a set of microphones for audio. The job of a representation learning algorithm is to determine functions of the raw input that compute higher-level properties to aid in downstream tasks such as classification. For images, the representation might include edges, color or texture patches, or object components; for audio, we might have amplitude of certain frequency bands or the existence of component sounds from a dictionary. If the data is discrete, as in a set of independent images, the representation learning algorithm may operate on instances independently of each other. In the case of streaming data like continually processed audio, the representation mapping could act as a *filter*, where the representation at a given time depends on the history of the signal up to that time.

Some classical dimensionality reduction techniques, such as Principle Components Analysis (PCA) [140], k-means clustering [112], and canonical correlation analysis (CCA) [77, 3] may be considered to be forms of automatic representation learning. Such simple methods are often useful as a preprocessing step for many learning algorithms. In the past few decades, many, many other more sophisticated techniques have been proposed for inducing useful features. To name just a few: there is manifold learning [178, 150], sparse coding [138, 100, 47], spectral clustering [159, 118], (single-layer) autoencoder networks and variations [187], and probabilistic latent factor models of many different flavors, such as latent Dirichlet allocation (LDA) [26], sigmoid belief networks [131], and restricted Boltzmann machines (RBMs) [167].

What all of the approaches just mentioned have in common is that they may be viewed as producing a single new representation “all at once”, without intermediate layers of representation. In contrast, so-called “deep” representation learning constructs features at multiple levels, with higher-level features constructed as functions of lower-level ones. Theoretical and empirical evidence suggests that useful and compact representations for some hard problems may require multiply nested layers of representation [73, 183]. Considering biological learning systems, the primary visual cortex in mammals is known to have a hierarchical organization with earlier stages of processing (area V1) identifying points, edges, and lines, which are later used to detect more complex features in area V2 [81].[1] Håstad and Goldmann [66] demonstrate that there exist efficiently computable binary functions that would require exponentially sized circuits if the circuit depth were bounded.

[1]The visual cortex also makes use of *feedback* connections, where “higher level” features can influence the perception of “lower level” ones. Some machine learning researchers have attempted modeling such connections, (see, e.g., Stollenga et al. [174]), but all models in widespread use today are purely feedforward.

Bengio et al. [20] discuss some properties we may demand of a representation learning algorithm in order that it perform well on hard AI tasks, including *expressivity* (useless information should be discarded), *disentangling factors of variation* (induced features should vary independently of each other), and most critically *abstraction* (identifying meaningful, predictive features, even if two inputs sharing some feature may be distant in the input space). A hierarchy of successively more abstract features enables the recognition of properties shared between instances that are superficially dissimilar. It promotes the re-use of features lower in the hierarchy, increasing the efficiency of the representation, particularly when the same representation is to be used for multiple tasks, as in transfer learning. A hierarchical representation may make adaptation to new tasks or changes in the data distribution easier, so long as most features remain identifiable and useful. Concepts that may require an impossible amount of data to learn as functions of the raw input may be have a much simpler expression in terms of higher level features.

Representation learning algorithms that aim to create such a feature hierarchy have been referred to as "deep learning" algorithms, invoking the depth of the induced feature hierarchy. Many people these days equate deep learning with artificial neural networks, but the basic principle that more abstract features should be composed of more primitive ones applies to a much richer set of models. Probabilistic models that posit the existence of at least one "layer" of latent (unobserved) variables whose values influence the observations have been used in statistics and machine learning for a long time, and multilayer probabilistic models like the deep belief network also exist. Hierarchical representation learning methods based on such models may be appealing in that they are grounded in the theory of probabilistic reasoning, unlike neural models that involve seemingly arbitrary choices like the form of the nonlinear activation function. Unfortunately, exact inference in probabilistic models with multiple layers of latent variables is generally intractable, and fitting their parameters to data is, if anything, more difficult.[2] It is deep neural networks that have been shown to scale

[2]The exception that proves the rule is deep belief networks, a probabilistic model with multiple layers of latent variables which can be usefully approximated by neural networks.

up to very high-dimensional data (such as images, with millions of pixels and millions of induced features) and that have empirically proven useful on so many tasks compared to non-hierarchical approaches.

In this thesis, we will be concerned with the family of deep representation learning systems based on artificial neural networks. As we will see, while all widely used deep RL models are essentially descendants of the feedforward neural network architecture known as the multilayer perceptron, there are many successful variations on the basic algorithm improving its performance or specializing it to particular forms of data.

1.1 Contributions and thesis overview

In recent years, deep neural networks have advanced the state-of-the-art on several classically difficult machine learning tasks, such as speech recognition [1], image and character recognition [38, 94], natural language parsing [171], language modeling [120], machine translation [14], pixels-to-controls video-game playing [123], and playing the challenging game of go [162], among others. The fact that all of the currently highest-performing systems on these tasks make use of an induced deep feature hierarchy is itself evidence that learning multiple layers of intermediate representations is a successful general strategy for complex tasks.

The details of the architectures and training methods used in all of these cases vary considerably. The basic idea of composing feature mappings in multiple layers is so general that there is a huge landscape of possible variations to explore, and different problem types may benefit from different algorithms. For example, in image processing, convolutional networks, in which parameters are shared between feature detectors at different locations throughout the image, are typically used to obtain the best results [94, 99]. The convolutional architecture itself has been further articulated in many ways to increase performance, including max pooling (downsampling by retaining the maximum feature value over possibly overlapping regions) and the use of fully-connected (non-convolutional) layers before the output.

To obtain optimal results on other tasks, different architectural variations and training techniques are used. The field remains highly empirical and a certain amount of hands-on experience is necessary to be able to successfully choose from the many available tools and apply them optimally toward solving a given problem on a given computing architecture. New techniques are still being proposed at a rapid pace, and it will be a continuing empirical question, as well as an art, to determine which techniques, in which combinations, obtain the best results on any task.

In the present work, we consider three learning scenarios in which there exist aspects of the problem that we believe current deep representation learning models have not yet taken full advantage of: learning representations of continuous-time data (such as audio), learning representations of multiview data (consisting of simultaneous data from different sensory modalities), and learning compact representation models for application on low-power, low-storage devices. Our specialized models can be used in combination with many existing techniques.

This thesis is organized as follows. In the following chapter, we will review the state of the art in deep representation learning, describing some of the important developments that have led to the field's resurgence and record-breaking performance on so many tasks.

Next, we will present our first model, the *sequential deep neural network* (SDNN), a deep representation learning algorithm developed for continuous, discretized data streams such as audio or video. Unlike a static network, or even a convolutional network, the SDNN allows explicit temporal connections between corresponding components of the representation at adjacent time frames. These connections give the SDNN the ability to directly model the tendency for the components of the learned representation to exhibit continuity (or lack thereof) through time. It also enables the representation to capture aspects of the input that span arbitrarily long temporal intervals. In chapter 3 we describe the SDNN model in detail and explore its performance and properties.

In chapter 4, we describe a form of network training appropriate for data with multiple views. The *deep canonical correlation analysis model* (DCCA) uncovers deep representation mappings of two different data views that expose latent correlated states (and suppress uncorrelated "noise"). It is therefore a generalization of linear CCA and a parametric counterpart to nonlinear kernel CCA in which, unlike these methods, the learned representation mapping has explicit hierarchical structure. The intuition motivating the approach is that if correspondences exist between deep level features of the different views, then those features are likely to correspond to "real" latent aspects of the scene and, as such, may be considered more likely to form part of a useful abstract representation.

In chapter 5 we present a training algorithm for deep neural networks (or for that matter any other parametric representation learning model trained to optimize a nonlinear function whose gradient can be evaluated) that employs L_1 regularization to set some synapse weights to exactly zero. Using L_1 regularization produces a sparse parameter vector, encoding the prior assumption that on average each component of the learned representation should depend on only a few of its possible inputs. We first describe the *orthant-wise limited-memory quasi-Newton* (OWL-QN) algorithm for convex training objectives, then we present an extension that allows it to run on non-convex objectives. Networks trained with OWL-QN exhibit competitive performance with only a tiny fraction of the weights, making representation models that are far more compact and efficiently computable.

The field of deep neural network learning is extremely broad and varied. Architectures like convolutional networks have been developed to specialize learning to particular forms of input data, while techniques such as dropout training and novel transfer functions like rectified linear units apply more generally. Many of the techniques that have been proposed combine synergistically with each other, while others may substitute for each other if they perform similar functions or have similar effects. Due to the highly empirical nature of the field and the rich set of possibilities afforded by combining available techniques, it is generally not possible to prove that any one method dominates another in all situations. Our purpose in the present work is to contribute to the pool of available methods and to provide evidence that our new models have desirable properties in some contexts. Future work by machine learning practitioners employing our methods in combination with others should more clearly define the circumstances under which our models are most useful.

Chapter 2

THE CURRENT STATE OF RESEARCH IN DEEP REPRESENTATION LEARNING

As introduced in the last chapter, a deep representation learning algorithm automatically induces a hierarchy of features. Recent successes in the field have come from the burgeoning arena of novel architectures, some tailored to particular forms of input data, and also from the development of novel training methods. In this chapter I will survey some of the most important recent developments that have not only made deep representation learning possible, but have turned it into an essential component of systems that perform at the state-of-the-art on many tasks.

2.1 Parameter optimization for deep representation learning models

Feedforward neural networks with multiple layers (shown in figure 2.1) were proposed long ago [82, 84], but challenges in finding good parameters for such models kept them from being useful in practice. With one or at most two hidden layers, initializing the network with small, random parameters and then training them with stochastic gradient descent can find reasonably well-performing models [152], but for deeper networks this simple strategy fails. The loss function of deep neural networks is highly non-convex, is riddled with plateaus and bad local optima, and may exhibit pathological curvature, making it impossible to optimize with simple gradient methods from a random initialization[50, 116, 20]. Recent methods to overcome these difficulties come in three major flavors.

The first generally effective methods for training many-layered networks to be discovered involved initializing the network parameters through optimization of an unsupervised, generative, layer-by-layer training criterion [69, 70, 19]. Initializing the parameters this way

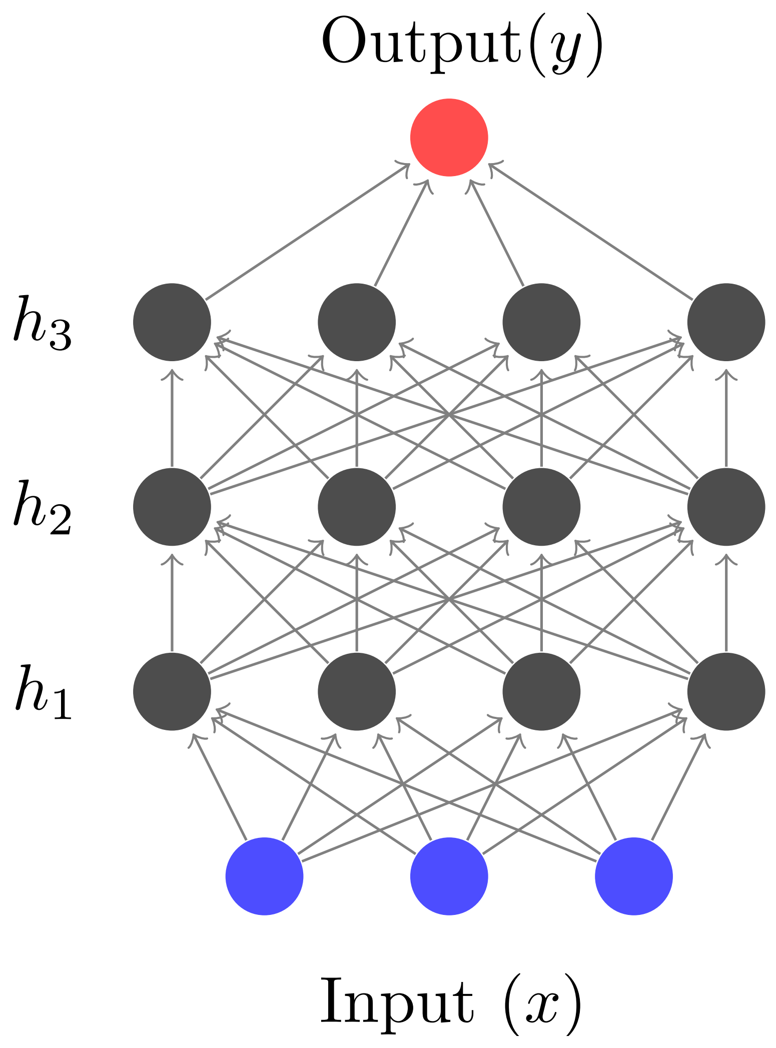


Figure 2.1: The basic multilayer perceptron (MLP) architecture, shown here with three hidden layers. The lowest layer h_1 of feature representation is computed directly from the input x according to $h_1 = \sigma(W_1 x + b_1)$, where W_1 and b_1 are parameters and σ is a nonlinear activation function applied componentwise. Then $h_2 = \sigma(W_2 h_1 + b_2)$ is computed the same way using parameters W_2, b_2, and so on, until the output $y = W_y h_3 + b_y$ is formed as an affine combination of the final representation layer. Given a differentiable loss function $l(y; \hat{y})$ evaluating the output y compared to the ideal output $\hat{y}$, the gradient of l with respect to the parameters at all layers can be computed using the backpropagation algorithm. A set of parameters locally minimizing the average loss on a training set can be found using stochastic gradient descent.

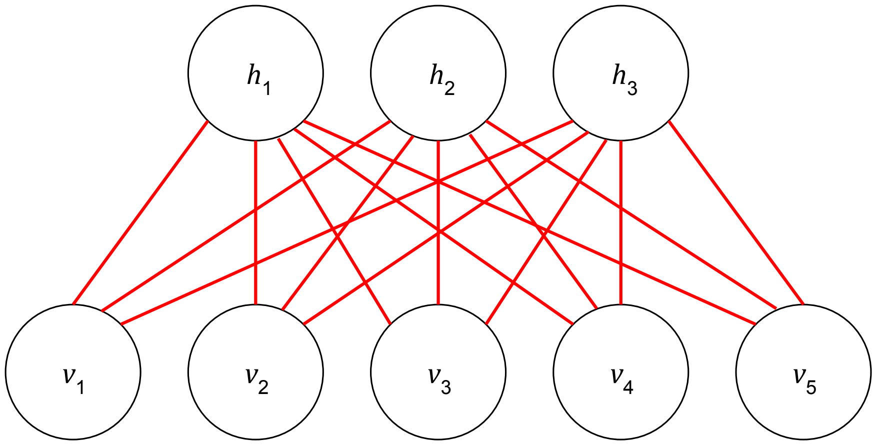


Figure 2.2: The basic restricted Boltzmann machine (RBM) graphical model structure, modeling the joint distribution between the observations $v \in \{0,1\}^{n_v}$ and the hidden variables $h \in \{0,1\}^{n_v}$, shown here with $n_v = 5$ and $n_h = 3$. The probability of a joint configuration (v, h) is given by $\Pr(v, h) \propto \exp(h'Wv + a'h + b'v)$ for parameters W, a, b.

before training to optimize the discriminative criterion of interest is known as “pre-training”. An early successful approach came from attempts to learn parameters for a truly generative probabilistic model called a *deep belief network* [69, 75]. To induce features at the first level, one models the distribution over inputs using a Restricted Boltzmann Machine (RBM) architecture, shown in figure 2.2. An RBM is a binary-valued, bipartite Markov random field graphical model structure [69]. Although exact computation of the likelihood and its gradient is intractable in an RBM, effective and very fast gradient approximations exist, in particular, contrastive divergence (CD), which uses truncated Gibbs sampling to estimate the gradient [74]. Once the RBM has been trained, the expectation over hidden values conditioned on the observations (which *can* be efficiently computed) form the first level feature representation.[1] The dataset can then be mapped to the new representation, which becomes the visible layer for another RBM stacked on top of the first. In this way, a hierarchy

[1]Alternatively a stochastic representation can be formed by *sampling* the hidden variables conditioned on the observations, also a tractable operation.

of features is constructed.

Conveniently, the expectation over hidden values in an RBM conditioned on the observations is computed simply as the logistic sigmoid function $\sigma(t) = (1 + \exp(-t))^{-1}$ applied componentwise to the inputs.[2] Thus the RBM parameters have a natural relationship to a single layer of a feedforward neural network with the logistic sigmoid activation function: computing the induced representations layer by layer is equivalent to forward propagation through a multilayer perceptron. To perform, for example, classification of the input, one could apply a simple linear classifier to the final layer representation and train it for a discriminative criterion. However, this would leave the representation mapping unchanged, bearing no relationship to the end task. Better results can be obtained by “fine-tuning” the parameters of the entire representation hierarchy using backpropagation, in effect learning not only the final classifier but adjusting the intermediate representations to model more precisely what is necessary for the classification task at hand.

A related generative pretraining strategy uses autoencoder (AE) networks (also called autoassociators). Autoencoder networks (illustrated in figure 2.3) are two-layer MLPs mapping the input v to a reconstruction $r(v)$, via a hidden layer $h(v)$. An AE network is trained to minimize a reconstruction criterion $l(r(v); v)$, optimizing the network's ability to produce output values that closely reconstruct the input, thus forcing the intermediate layer to capture as much of the information contained in the input as the network's architecture allows. As with RBMs, autoencoders can be stacked to learn a representation hierarchy, and then all parameters can be fine-tuned jointly for a discriminative criterion [19]. AE and RBM pretraining methods are closely related, as the AE network reconstruction error can be seen as an approximation to the KL-divergence of the corresponding RBM model to the empirical distribution of the training data [18]. One might expect the deterministic AE network to be a more effective pretraining strategy, on the grounds that it is superficially more closely

[2]If the hidden variables take values $\{-1, 1\}$ instead of $\{0, 1\}$, the conditional expectation corresponds to the tanh function applied to the inputs. We discuss this property further in chapter 3 as it is used in our SDNN model.

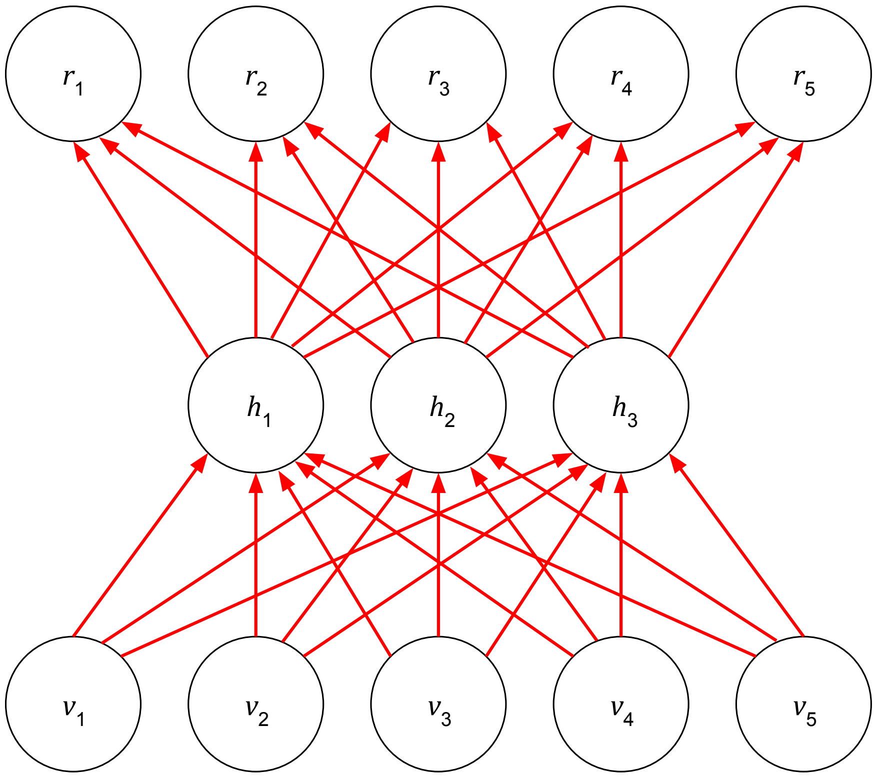


Figure 2.3: The autoencoder network structure. The observations v are used to compute the representation h according to $h(v) = \sigma(W_h v + b_h)$ where σ is a nonlinear activation function applied componentwise. The representation h is then in turn used to make the reconstruction $r(v) = \sigma(W_r h(v) + b_h)$. The parameters W_h, b_h, W_r, and b_r are learned to attempt to minimize a reconstruction error criterion $l(v; r(v))$, making r close to v on average over the training data. In a common variation called the *tied-weights* model, the reconstruction weights W_r are constrained to equal the transpose of the hidden weights, $W_r = W_h'$. In a *denoising* autoencoder, v is corrupted stochastically to $\tilde{v}$ before presentation to the network, so the training criterion becomes $l(v; r(\tilde{v}))$.

related to a layer of (deterministic) multilayer perceptron than is an RBM, but empirically RBM pretraining tends to work better [188].

A stochastic elaboration of the AE called the *denoising* autoencoder (DAE) appears to close the gap with RBM pretraining performance [187, 50]. In a DAE, the input v is corrupted stochastically into $\tilde{v}$ before presentation to the network (for example, by independently setting some components to zero, or adding independent Gaussian noise), so the loss to minimize becomes $l(r(\tilde{v}; v))$. The hidden representation then has to not only retain the information about v but also to capture relationships allowing it to undo the effect of the noise. In particular, adding noise renders the trivial solution in which $h(v) = v$ suboptimal. The DAE is discussed further in chapter 4 as it is used as a component of our deep canonical correlation analysis model. Other successful pretraining strategies include sparse RBMs [101], mean-covariance RBMs [146], and contractive autoencoders [148, 149].

Several researchers have attempted to answer the question of why pretraining deep neural networks improves performance of deep neural networks so dramatically. One answer is that it may regularize them toward more general solutions earlier in training, thereby reducing the oversized influence of early examples [50]. It may also reduce the pathological curvature of the objective observed around random initial parameters generated by standard schemes [116]. With random initialization, the error gradient of deep networks is of significant magnitude only for the final layers; the early layers' parameters are left essentially unchanged by stochastic gradient descent training. Pretraining not only gives the deeper representations a reasonable starting point in which they at least model some aspects of the input distribution, it also allows the error gradient to propagate all the way down to the lowest layers.

Subsequent to the discovery of pretraining methods, it was found that good results could be obtained without pretraining by using new classes of activation functions. Historically, sigmoidal functions like hyperbolic tangent (tanh) or the logistic sigmoid $\sigma(x) = (1 + exp(-1))^{-1}$ have been favored in neural network research and applications. New classes of functions, like *rectified linear units* (ReLU) using the function $\sigma_{\mathrm{ReLU}}(x) = \max(0, x)$, or *maxout* units which take the maximum over a set of inputs, can be effective even when training from random parameters [130, 56]. These activation functions have broad regions on which the function is linear and therefore the first derivative is constant (and non-zero), whereas the classical functions saturate to maximal and minimal values, at which point the derivative vanishes. Using piecewise linear activation functions thus lets the error gradient propagate to the earlier layers, just as pretraining does, which may at least partially account for their utility.

Finally, some excellent results have recently been obtained by moving from simple first-order stochastic optimization training to more sophisticated second-order (often full-batch) methods. Martens [116] used a Hessian-free algorithm to approximate Newton's method over minibatches of training data. The method was used to train recurrent neural networks (discussed in more detail in the next section) by Martens and Sutskever [117]. This line of research supports the view that the primary difficulty in training deep networks from a random initialization is the pathological curvature, which makes first-order optimization methods impractically slow.

The development of the aforementioned training techniques demonstrated finally that it was possible to learn deep representation hierarchies (in particular, deep neural networks) that perform very well. There has also been a significant amount of research into stochastic regularization methods that, while perhaps not sufficient on their own without pretraining, have been shown to improve the learned models. Perhaps the most significant such development has been *dropout* training, in which a random set of internal units are deactivated during the presentation of each training instance (or minibatch) during the fine-tuning phase [72, 173]. Such training may encourage the learned features to be more robust, not relying too heavily on information coming from only one or two inputs. Dropout has been compared to learning and averaging a large ensemble of networks, one for each possible set of deactivated units. It has been shown to be equivalent to an adaptive L_2-regularization that promotes rare but useful features [191]. A popular variant of unit dropout randomly deactivates connections rather than whole units [192]. Noise distributions other than the multiplicative Bernoulli noise used by Hinton et al. [72] have been used, for example additive Gaussian noise [128].

Stochastic downsampling in convolutional neural networks (discussed shortly) has also been shown to yield improvements compared to the more typical max pooling [201].

2.2 Specialized architectures for structured data

So far, I have discussed some of the new training methods that have made it possible to learn high-performing deep networks, or when used in combination, improve their performance on some tasks. These methods apply to multilayer perceptrons, the basic feedforward neural network architecture arranged in discrete layers, with a single-dimensional output, used for classification or regression problems. In addition to advances in training algorithms, there has also been significant research into developing alternative architectures for situations where the input or output of the network exhibits special structure.

Consider a visual image presented to a machine learning algorithm. Most simply, the image can be represented as an unstructured vector describing all of the pixel intensities, but this ignores the fact that pixels in an image are naturally arranged in a two dimensional array. Prior to the success of deep representation learning on images, specialized feature detectors for edges [31], corners, textures [63] and other local features [109] had been successfully applied to image processing tasks, all of which are critically dependent on the spacial arrangement of the pixels. Convolutional neural networks (CNN), developed prior to the advances in deep network training just described but enjoying particular success recently, make two reasonable assumptions about how features should depend on pixels: first, that each feature should depend only on pixels within a relatively small window, and second, that the same feature detector can be applied to every patch of the image to detect localized features [52, 163]. A multilayer CNN applies the same assumptions that were made regarding pixels to the mid-level features themselves, creating a deep hierarchy of feature detectors, each applied over regions to compute local feature values. If those assumptions are reasonable for the data source, the system should be able to learn from a smaller amount of training data due to the sharing of parameters. A diagram of a convolutional neural network is shown in figure 2.4. In the last decade, deep CNNs have rapidly advanced the state-of-the-art in image

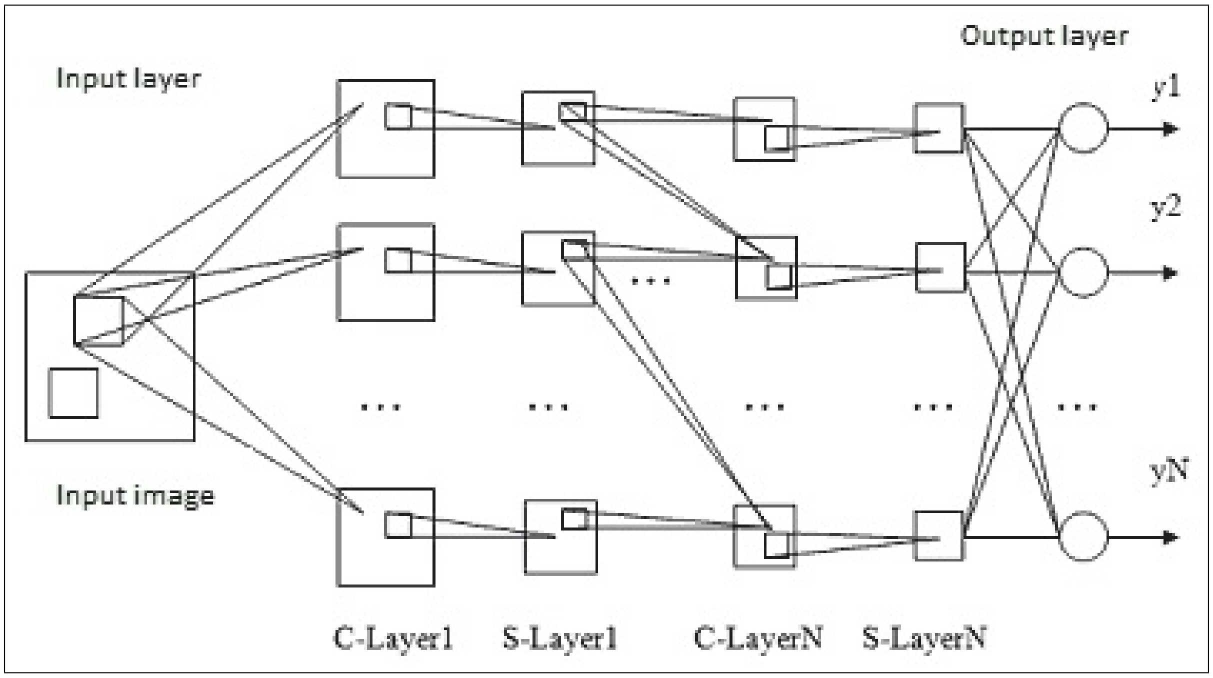


Figure 2.4: An illustration of a convolutional neural network for image processing. In the convolutional layers, local feature detectors are applied to patches that span the image, producing maps of localized feature values. In the subsampling layers (labeled "S-Layer"), aggregate values (typically the maximum value, although averaging or stochastic selection are also used) of the local features within small patches are computed, reducing the dimensionality of the representation. Just before the output, a fully connected layer is applied to detect global properties.

processing [39, 38, 94, 88, 55, 164].

Beyond applying the convolutional technique at multiple layers of feature hierarchy, a variety of further elaborations to the CNN architecture have been developed to achieve higher performance on image processing tasks. It is often helpful to reduce the dimensionality of the image as we ascend the feature hierarchy by alternating convolution layers with "pooling" layers, in which small patches of the image features are downsampled by retaining only a single aggregate measure of the feature of each patch (typically the maximum value) for further processing [145] (labeled "S-Layer" in figure 2.4). The pooling operation is inspired by similar computations performed in popular non-neural image feature detection algorithms such as *Gist* [160] and SIFT [110, 109], as well as biological models of the visual cortex [147]. The intuition behind such an operation is that if some feature has been detected within a small patch of image, its precise location within the patch is not important, the more so as we ascend into the higher-level, more abstract features. Pooling therefore increases the invariance of the representation to small perturbations in the location of features, as well as further reducing the number of trainable parameters.

Another common architectural choice for image-processing CNNs is to use fully connected (non-convolutional) layers at the deepest levels of the hierarchy, just before classification in the final layer (also shown in figure 2.4). This enables the discovery of properties of the image as a whole. The best-performing image classification models combine all of these specialized model choices, as well as algorithmic tricks discussed in the last section, like dropout training [94, 164].

Specialized network forms have also been developed for data with a temporal component, such as audio and natural language. In speech processing, convolutional architectures similar to those used for images but with the convolution applied along the time dimension have been fruitfully employed [1], as have various forms of recurrent neural networks (RNNs). The basic form of an RNN is a feedforward neural network that propagates activations forward in time. (See figure 2.5.) At each frame, a new representation is computed using a (possibly multilayered) network that takes as input the current frame's observations and the previous

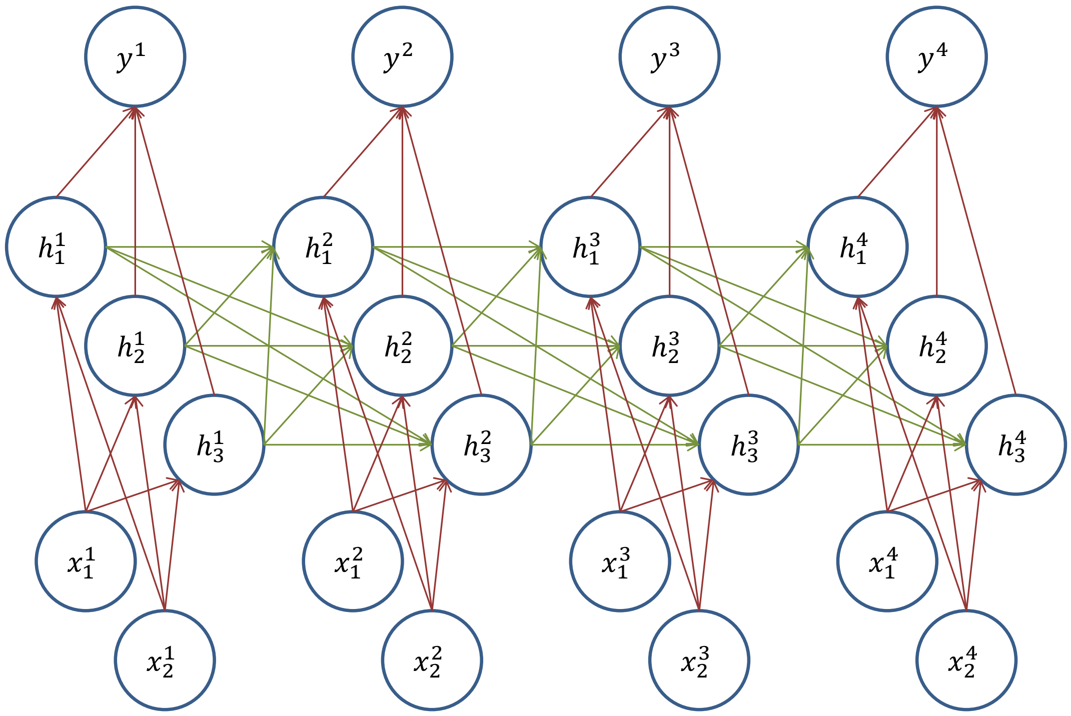


Figure 2.5: A simple recurrent neural network architecture for a sequential labeling task, unrolled in time. At frame t, the hidden state vector h^t is computed as a function of the current observation x^t and the previous hidden state h^{t-1}, and the label y^t is computed from h^t. An unrolled RNN can be seen as an extremely deep neural network "turned on its side". Training an RNN therefore has similar difficulties to training deep NNs.

frame's representation [161]. Learning in RNNs is considered difficult for many of the same reasons as static deep networks [197, 195]. However a special form of network called "long short-term memory" (LSTM) that employs multiplicative connections to gate the input and output of each unit (illustrated in figure 2.6) has proved easier to train [76]. Several recent successful representation learning models for audio processing (phone and speech recognition) stack LSTMs to produce the feature hierarchy. In order that the component of a representation at a particular frame depend not only on the inputs at previous frames but also on future frames, bidirectional LSTMs are used, in which one LSTM computes features forward in time, and one backward, at every layer [58, 153]. After generative pretraining, all parameters of the LSTMs at all layers can be fine-tuned jointly for the discriminative criterion of interest. Deep bidirectional LSTM networks are an excellent example of just how elaborate some of the top-performing architectures are, despite being able to trace their roots to the humble multilayer perceptron model.

Natural language data (in textual form) has a particularly interesting and rich structure that can be exploited by specialized deep representation learning algorithms. Most trivially, language has a discrete temporal form, with either letters or words as basic units. Simple recurrent neural networks, using words as tokens, have been used with great success for language modeling [120, 121]. Sutskever et al. [176] train an RNN to generate character sequences (using a factored tensor representation of the hidden-to-hidden transition matrix and second order training methods), producing surprisingly fluent text. Similar results were obtained using deep LSTM networks trained on character sequences [59].

More interestingly, natural language has a recursive structure of nested constituents, and this structure has also been exploited. *Recursive* neural networks (note the distinction between *recurrent* and *recursive*) determine vector representations of each node in a tree structure by applying the same transformation to the representation of each pair of daughter nodes to produce the representation of the parent. The depth of the representation therefore corresponds to the depth of the tree structure. Representations produced by recursive NNs have proven useful in natural language parsing [171, 168] and sentiment analysis [170].

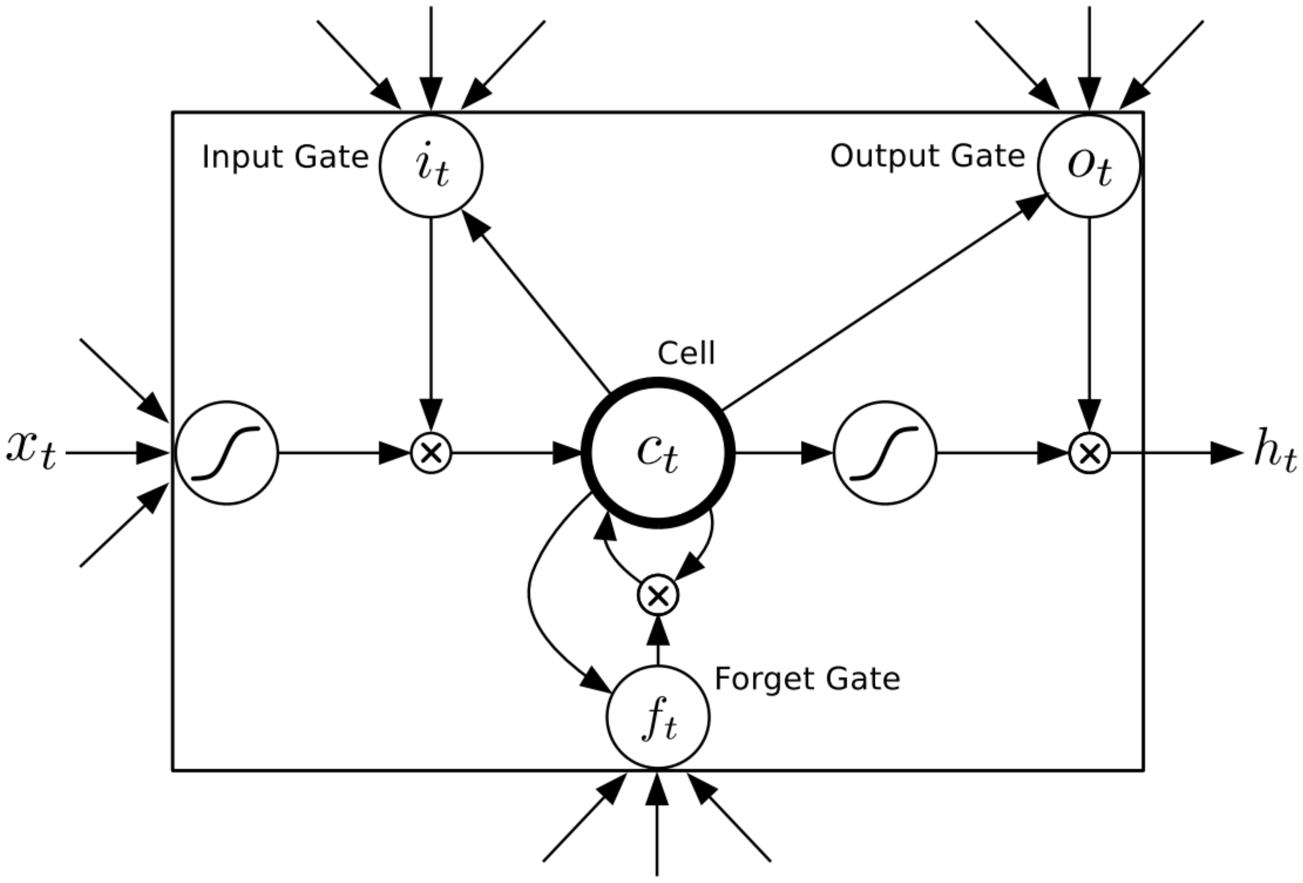


Figure 2.6: A long short-term memory recurrent neural network uses multiplicative gates to control the input and output of each recurrent unit. The figure shows a single LSTM “neuron”. A complete LSTM network would include a vector of such units for each time step, each taking inputs from the units at the previous frame, as well as the current frame’s observations. (Illustration borrowed from `http://blog.otoro.net/2015/05/14/long-short-term-memory`.)

Recursive NNs have also been used to identify compositional structure in natural images [169].

Specialized deep representation learning algorithms have also been developed for the holy grail of NLP tasks: machine translation. In sequence-to-sequence ML, an LSTM network is trained to scan the source sentence and output a fixed-length representation as its final state. That representation is then decoded by another, jointly trained LSTM network to produce the target sentence [177, 36]. Kalchbrenner and Blunsom [86] describe a similar model that uses a CNN to produce the fixed-length representation of the source sentence (but still an RNN for the output). These models makes use of the linear structure of natural language in building the representation of the source sentence and in generating the output, however the explicit structure is lost when the sentence is compressed into a fixed-length representation.

A recent approach that does not contain a fixed-length bottleneck is Bahdanau et al. [14]. A (bidirectional) recurrent network is learned to produce a sequential, variable-length representation of the source, with a vector representation of each word in context. Then a decoder recurrent network generates the translation sequentially while jumping around the source representation according to the alignment given by a separate network, in what is called an attentional mechanism.[3] These specialized deep representation learning systems seem to be poised to overtake traditional phrase-based translation systems in the near future, as evidenced by the performance of Bahdanau et al. [14] in the 2015 Workshop on Statistical MT, coming in among the best systems in several languages [27].

So far I have discussed deep learning architectures that were tailored toward learning from a particular form of data: CNNs for spatially arranged data, RNNs and LSTMs for temporally extended data, recursive NNs for tree-structured data. There has also been research into systems to process complex data of more than one form simultaneously. Ngiam et al. [134] describe a model called the bimodal deep autoencoder for learning a shared representation of synchronous audio and video data. Initial, shallow layers of representation are learned

[3]The alignment is "soft", meaning that the representation s_i of the target word generated at each step is a function of the weighted sum $c_i = \sum_j \alpha_{ij} h_j$ where h_j is the source word representation and α_{ij} is the alignment probability.

from each modality separately, using an autoencoder network, while in deeper layers the representation is combined: an autoencoder taking both representations as input produces a single, joint representation. In image captioning, the input consists of an image, while the output is text. A recent successful approach consists of learning a high-level representation of the image using a CNN which then becomes the input to an RNN that generates the textual caption [114, 87, 190]. This can be seen as similar to the sequence-to-sequence MT system of Sutskever et al. [177] discussed earlier, only the system is "translating" images to captions.

2.3 Computational considerations

There is one more major factor contributing to the recent success of deep representation learning algorithms: the ever-increasing speed of computation and the availability of new hardware such as programmable GPUs that are exceptionally well suited to accelerating existing algorithms. Deep MLPs with very wide hidden layers require matrix-matrix products to compute the activations of each minibatch during SGD training. CNNs employ convolution operators. Nonlinear activation functions of all units in a layer can be computed independently in parallel. All of these operations can exploit the massive parallelism afforded by GPUs to achieve major speedups [51]. New libraries like Caffe [83], CuDNN [35] and Theano [23, 16] make it easier for researchers and practitioners to take full advantage of modern hardware.

Developments in computing architecture allow far larger models that can train for a far greater number of iterations than was possible only a few years ago, and we probably would not see nearly as much impact of the other strategies mentioned in this chapter if they were forced to run on hardware of the nineties. Some techniques, like dropout training and Hessian-free optimization, have been shown to yield higher performing models, but only at the expense of requiring much greater amount of computation. (In the case of dropout, each training iteration is not much more expensive, but the greater variance in the update means that many more passes over the data are required to achieve the best results.) Therefore it is an important consideration that new techniques be well-suited to efficient implementation on contemporary hardware.

2.4 Conclusion

In this chapter I have touched on some of the most important specialized architectures and training algorithms used to learn deep representations of a variety of different types of data. The models described in this chapter have been used as components in systems that advanced the state of the art in many tasks. However, it would take a much larger survey to present in any detail the record-breaking systems that have been built out of the huge number of possible permutations on the basic architecture afforded by combining these and other contributions. My intention has been to describe some of the most important and useful variations of the basic deep multilayer perceptron, illustrating the diversity of the field, and putting my own contributions, which follow, into context.

Chapter 3

LEARNING DEEP REPRESENTATIONS FOR CONTINUOUSLY EXTENDED DATA

3.1 Continuously extended data

Many machine learning tasks operate on data that is extended continuously in time. In speech recognition, for example, a five-second audio clip might be discretized into 25ms frames and presented to the ML algorithm as a matrix where each column provides features for a given frame. Other examples include music analysis, object tracking from video or other continuous-time inputs, and activity recognition. In all of these cases, the input features are organized into frames that are inherently ordered according to the dimension of time, and the values of the features within each frame come from a discretization of an underlying continuous signal. Thus, the closer together two frames are in time, the more likely they are to be "similar", either in the raw input space (e.g., the corresponding pixels of two adjacent frames of video are likely to have nearly the same colors), or in more abstract ways (e.g., the same object is present at nearly the same location, and the activity being performed in two adjacent frames is probably the same). [1]

There are several common ways of using deep networks to learn representations of continuously extended data. The simplest way is to feed each frame to a static network, independently of the rest of the sequence so that a single network with a single set of weights is applied to every frame. Although the representations of the hidden layers at different frames cannot directly influence each other, the representation of adjacent frames may still be highly correlated due to the similarity of the input. If the end goal is to segment and label

[1]The work presented in this chapter was originally published as Andrew and Bilmes [4] and Andrew and Bilmes [5].

the frames, the final layers' representations can still be combined by a sequence classifier like a hidden Markov model or conditional random field (discussed shortly) to at least model dependencies between nearby labels.

In many cases better results can be obtained by simply expanding the input of the static network to include a sliding window of frames on either side of the center, target frame. If two frames on either side are included, for example, then the network is applied with inputs $[x_{t-2}\ x_{t-1}\ x_t\ x_{t+1}\ x_{t+2}]$ concatenated into a single vector to obtain the representation of frame t. The same network (with the same parameters) would be applied to every window of five frames. This allows the output to depend on aspects of the input that are nearby in time. For example, a speech recognition system with a sliding window model could model the fact that when the input features are characteristic of a *t* or *d* phoneme having recently completed, the current frame may more likely to be labeled as a vowel. Such a "sliding window" network may also learn to approximately represent the rates of change of each input feature, which could be highly informative. Without context frames in the input, there is no way to recognize first- or higher-order differential behavior of the signal unless derivatives are explicitly included as features.

Another generally helpful technique for modeling continuously extended data is to use a convolutional architecture, where the convolution is performed along the time dimension. In a convolutional network, not only the input data, but also the hidden representations are explicitly arrayed in time. Whereas in the sliding window network, the inputs $[x_{t-2}\ x_{t-1}\ x_t\ x_{t+1}\ x_{t+2}]$ produce a first layer representation h_t^1 and then the next layer's representation h_t^2 is a function of h_t^1 alone, now the concatenated vector $[h_{t-2}^1\ h_{t-1}^1\ h_t^1\ h_{t+1}^1\ h_{t+2}^1]$ may be used to produce h_t^2, and so on for each layer. Such a convolutional architecture has the advantages of the sliding window model *at every layer*, not just with respect to the input. In our example, the speech recognition system could determine the likely existence of a *t* or *d* based on higher level features, not just the raw input features like the sliding window model. It can also model the rates of change of the hidden representations. Moreover, a deep convolutional architecture in effect allows deeper layers to depend on a *larger* window of input than a simple sliding window model: if the representation h_t^l at each layer can depend on δ frames to each side of the center frame of its input, then the output at layer L is ultimately influenced by $2L\delta + 1$ frames of the input. A sliding window network would require far more (mostly redundant) parameters to enable dependency on such distant frames. The convolutional network can therefore be seen as more efficient.

In the present work I take the next step to describe a further elaboration of the convolutional network that provides an additional advantage on continuously extended data. When the input is known *a priori* to come from a continuous signal, it may be reasonable to expect that some features of the representation should also be more or less continuous, in the sense that the values at nearby frames should be close. The representations discovered by a convolutional architecture (or for that matter, any of the aforementioned architectures) may in many cases turn out to be more or less continuous, just based on the fact that their inputs are continuous. However the model has no way of explicitly encouraging continuity in the hidden representations. If the input signal is noisy or corrupted, the hidden representations may exhibit higher levels of discontinuity. In that case it may be particularly advantageous to give the model the capacity to smooth the hidden representations.

Furthermore, our model implicitly allows a particular form of dependence of each hidden representation on arbitrarily distant input features. While the convolutional network can, at layer l recognize aspects of the input that span $2l\delta + 1$ input frames, it cannot hope to recognize properties of larger spans, as frames outside of this window have no influence on it. Our contribution is to enable hidden units at adjacent frames to *directly* influence each other, not only via their overlapping inputs, thereby allowing the model to explicitly encourage smoothness in the hidden representations as well as the potential to recognize properties of the input of arbitrary temporal extent.

The model we will introduce is applicable whenever continuously extended data is input to a representation learning algorithm. In this work we will assume the continuous dimension is time, but in general the ideas presented in this chapter would apply to any type of data in which a one-dimensional continuum of values is discretized into pieces small enough that the variation between adjacent pieces is low. It may also be extended to multi-dimensional continua, for example high-resolution images (two continuous dimensions) or video (three continuous dimensions). Our algorithms may also be useful for sequential data that is inherently discrete, such as natural language (text) or bioinformatics data, so long as it is useful for the induced representations to recognize features that occur over significantly longer temporal extents than the underlying discrete bits.

Although we do not assume anything about the goal of the induced representation, a particularly common usage case for continuously extended data is sequence labeling. For concreteness, we will define our model and conduct experiments in this problem setting.

3.1.1 Sequence labeling

A frequent goal of ML algorithms processing continuously extended data is to segment the data into coherent labeled pieces: given entire continuous (but discretized) sequences of inputs over observations $\{x_1, x_2, \ldots, x_T\}$, we want to produce sequences of labeled segments $\{(b_1, e_1, y_1), (b_2, e_2, y_2), \ldots, (b_n, e_n, y_n)\}$, where $b_k < e_k$ are the beginning and ending frame of the k^{th} segment, and y_k is its label. The length of the sequence T, as well as the number of segments n may vary from instance to instance. A *static* (non-sequential) learning algorithm might be used to classify each frame x_t separately, but we will consider a true sequential labeling algorithm to be one that makes use of dependencies between observations or labels at different positions within the sequence to produce a better labeling.

In conditional random fields (CRFs) [95], a Markov random field (MRF) is defined over the label sequence whose parameters depend on the input. Typically, the graphical model structure is a linear chain so that globally conditioned on the input, each label is conditionally independent of the other labels given its neighboring labels. CRFs allow features of the input to be defined at arbitrary distance from the associated label, but the user must consciously design such features to allow long-distance dependencies and, moreover, the feature functions may only be fixed length. Feature design is a difficult, task-specific problem, and it is especially difficult to hand-design effective long-range features for tasks such as speech recognition, where the input is a relatively low-level representation of the acoustic signal. On the other hand, results in speech science suggest that longer-range features (that is, longer than the typical 25 ms frame width) may be useful for speech perception, particularly in noisy environments [68]. In any case, the basic CRF model is a shallow architecture in which there is no feature hierarchy at all. As argued in chapter 1 on general grounds, we would expect a deeper representation learning algorithm to be very helpful for speech recognition.

Several methods have been proposed to introduce hidden variables to CRFs that might be capable of modeling regularities in the data that are not explicit in the features but nevertheless aid in classification. We focus here on approaches that were applied to phone recognition, which is a prototypical sequential labeling task over continuously extended data, and the subject of our experiments. The hidden CRF (HCRF) appends a multinomial hidden state to each phone class and optimizes the marginal likelihood [60], so that subclasses may be induced that are easier to recognize than the original classes. Another successful approach models each phone as a sequence of three subphones, the boundaries of which are latent [175]. Other work uses a multi-layer CRF in which the data is mapped through various layers of multinomial sequences that may be either Markov order-1, or order-0 (conditionally independent given the input) [199]. All of these approaches are more effective than a single-layer CRF with no feature hierarchy, but much better results have been obtained with a richer latent representation via deep networks.

Several researchers have employed deep MLPs to learn feature representations for use in phone recognition [144, 72]. Preceding the deep NN revolution of the 2000s, shallow networks had been combined with hidden Markov models for use in phone recognition [28].[2] Mohamed et al. [125] use deep networks combined with HMMs to achieve excellent results on phone recognition. In later work, a deep MLP phone classifier is trained jointly with a CRF taking the top hidden layer of the network as input [126]. Veselỳ et al. [186] describe a model where

[2]While HMMs require the emission probability $P(x_t|y_t)$, a neural network can be trained to estimate the class posterior $P(y_t|x_t)$. Using Bayes' rule one can estimate $P(x_t|y_t) = \frac{P(y_t|x_t)P(x_t)}{P(y_t)}$, where the prior label probability $P(y_t)$ can be estimated by relative frequency, and the observation prior $P(x_t)$ can be discarded for the purposes of discriminating between label sequences.

gradients based on a sequence error in an HMM-based system are passed down through shared-parameter deep neural networks at each time frame. All of these models are of the "sliding window" type mentioned earlier. Convolutional networks have also recently been applied to phone recognition [1].

In the present work, we present our sequential deep neural network (SDNN) model, which introduces true sequence models in each of the hidden layers of a deep MLP [4]. Each layer is modeled with an MRF structure called a sequential restricted Boltzmann machine (SRBM) that allows dependencies between corresponding hidden units at adjacent time frames. Exact sampling of hidden structures given the input and computation of conditional expectations remains tractable in the SRBM—it involves only matrix multiplication and forward-backward computations—so CD training is still possible. As with RBMs, we can stack SRBMs and append a discriminative sequence classifier (in particular, a CRF) atop the final layer. Finally, using a backpropagation-like algorithm, we can compute the error gradient exactly to discriminatively fine-tune all parameters of the representation hierarchy jointly.

3.1.2 Notation

We employ the following notation in the remainder of the chapter. If X is a matrix, the $(i,j)^{\text{th}}$ entry is X_{ij} and the j^{th} column (as a column vector) is X_{*j}. The submatrix of columns j through k is $X_{*(j:k)}$. The matrix transpose is denoted X'. If X and Y are matrices (or vectors) of the same dimension, $\langle X, Y\rangle$ denotes $\text{tr}(X'Y)$. If X and Y have the same number of rows, $[X|Y]$ denotes their horizontal concatenation.

3.2 The sequential restricted Boltzmann machine (SRBM)

The restricted Boltzmann machine (RBM), shown earlier in Figure 2.2, is a graphical model structure often used for representation learning [167]. The RBM consists of two vectors of variables, the visible units $v_1, \ldots, v_{n_v}$ and the hidden units $h_1, \ldots, h_{n_h}$. The visible units v_i may be binary-valued or real-valued, while the hidden units h_i are typically binary-valued. The joint distribution is defined by a Markov random field model with the given graphical structure. Thus, the hidden units are independent given the visible units, and vice-versa. Although exact computation of the likelihood and its gradient is intractable, successful approximation schemes exist, including contrastive divergence updating and variants [74, 181].

Suppose that we are employing an RBM where both the visible and hidden units are random variables taking values in ± 1. Then for parameters $W \in \mathbb{R}^{n_v \times n_h}$, $a \in \mathbb{R}^{n_v}$ and $b \in \mathbb{R}^{n_h}$, the joint distribution is given by $\Pr(v, h) \propto \exp(h'W'v + v'a + h'b)$. Conditioned on observed visible units $\hat{v}$, the distribution of hidden unit h_i is

$$\Pr(h_i|\hat{v}) = \frac{\exp(h_i(W'_{*i}\hat{v} + b))}{\exp(W'_{*i}\hat{v} + b) + \exp(-(W'_{*i}\hat{v} + b))}.$$

Writing the input to unit i as $\alpha_i = W'_{*i}\hat{v} + b_i$, this simplifies to

$$\Pr(h_i|\hat{v}) = \frac{\exp(h_i\alpha_i)}{\exp(\alpha_i) + \exp(-\alpha_i)}.$$

Therefore, the expected value of h_i given $\bar{v}$ is

$$\mathbb{E}[h_i|\bar{v}] = \frac{\exp(\alpha_i) - \exp(-\alpha_i)}{\exp(\alpha_i) + \exp(-\alpha_i)} = \tanh(\alpha_i).$$

This is how RBMs can be used to initialize the parameters of one layer of a neural network using the hyperbolic tangent activation function: given inputs $\alpha = W\hat{v} + b$, the activations of the neural network hidden layer are the expected value of the hidden layer under the RBM model, $\tanh(\alpha)$. If we had used Bernoulli variables with $h_i \in \{0, 1\}$ instead of $h_i \in \{-1, 1\}$, we would have logistic sigmoid units instead of tanh units. This motivates our use of an extended RBM model for sequential data, in which the hidden units are not completely independent given the visible units. Still, their expected value, taking into account the dependencies, will become the activation of the next layer of a sequential neural network.

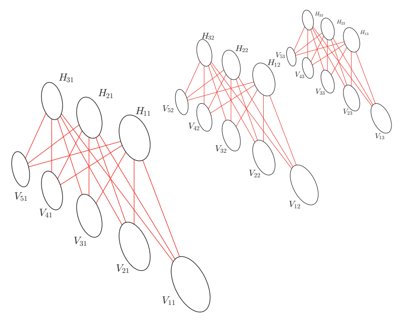
We now introduce a generalization of the RBM intended for sequential data, called a sequential restricted Boltzmann machine (SRBM). An SRBM defines a joint distribution over two matrix-valued layers, a visible layer $V \in \mathbb{R}^{n_v \times T}$ and a hidden layer $H \in \mathbb{R}^{n_h \times T}$. Conditioned on the hidden layer, all variables of the visible layer are independent (just as in a standard RBM). Conditioned on the visible layer, all *rows* of the hidden layer are independent of each other, but we allow Markov interactions within each row (see Figure 3.1). Allowing dependencies between variables within each row of the hidden layer lets the SRBM potentially model long-range dependencies between widely separated time frames of the visible layer, while retaining the tractability of important operations like marginalizing and sampling.

While an RBM typically has dense connections between the visible and hidden layers, an SRBM has only edges that are local in time. Specifically, we use edges between V_{it} and $H_{j(t+\delta)}$ for all i, j, t and for $|\delta| \leq \delta_{\max}$. The weights on the edges linking the visible and hidden variables are summarized in the matrices $W_\delta \in \mathbb{R}^{n_v \times n_h}$, where $(W_\delta)_{ij}$ is the weight on all edges $(V_{it}, H_{j(t+\delta)})$. The hidden layer of the SRBM also has a vector of transition parameters $\theta \in \mathbb{R}^{n_h}$ which govern the interactions between adjacent frames within each row of H, as we describe shortly. We intentionally disallow edges between observed units, in order to encourage the hidden layer to model any dependencies between time frames of the observations. In the experiments we also include three vectors of bias terms: one for H_{*1}, one for H_{*T} and one that is shared by all columns of H. We omit these from the exposition to keep the formulas uncluttered.

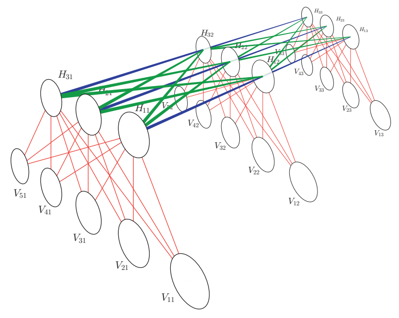
We assume the hidden variables are always binary, meaning $H \in \{\pm 1\}^{n_h \times T}$, and the observed variables are either binary ($V \in \{\pm 1\}^{n_v \times T}$) or real-valued Gaussian ($V \in \mathbb{R}^{n_v \times T}$). For $\delta_{\max} = 1$, the energy of a configuration is defined in terms of the matrix $A^h \in \mathbb{R}^{n_h \times T}$:

$$A^h = \left[W'_{-1} V_{*(2:T)} \middle| \mathbf{0} \right] + W'_0 V + \left[\mathbf{0} \middle| W'_1 V_{*(1:T-1)} \right]. \tag{3.1}$$

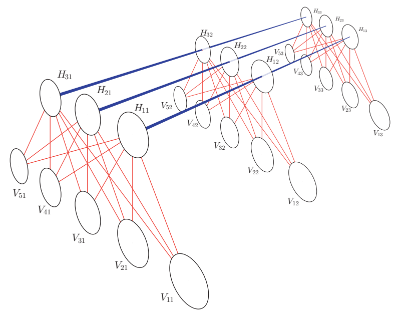
In (3.1), the middle term $W'_0 V$ produces the matrix of inputs to each hidden unit coming from the visible units at the same time frame (the red edges in the figure). The other two terms add the influence of visible units at the preceding and subsequent frame (the purple edges). The computation of A^h is essentially a single-dimensional convolution operation. The generalization to $\delta_{\max} > 1$ is straightforward.



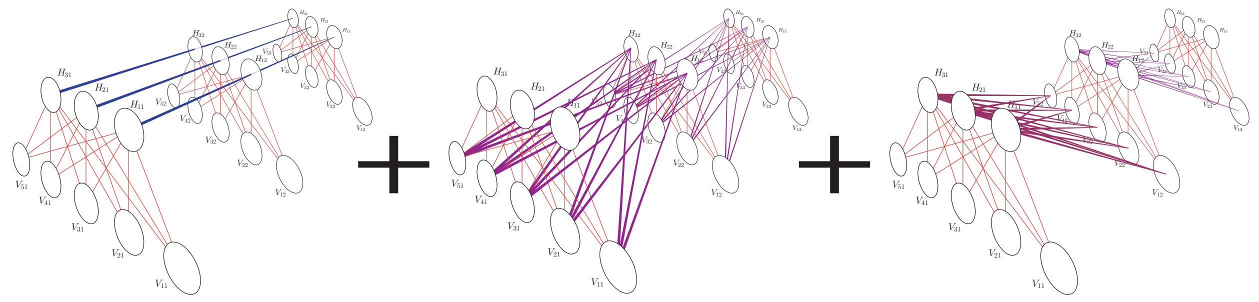
(a) Independent RBMs



(b) Intractable



(c) SRBM with $\delta_{\max} = 0$



(d) Purple edges in the center figure correspond to W_1, and those on the right, to W_{-1}. The union of the three graphs is an SRBM with $\delta_{\max} = 1$.

Figure 3.1: Illustrations of SRBM with $T = 3$ time frames, $n_1 = 5$ input units and $n_2 = 3$ hidden units per frame. Fig. 3.1a shows a sequence of repeated, independent (non-sequential) RBMs. Fig. 3.1b shows a model with dense connections in the hidden layer, which may seem desirable from a modeling perspective, but including all green edges would render contrastive divergence training and computation of the conditional expectations $\mathbb{E}[H|\hat{V}]$ intractable. Fig. 3.1c shows an SRBM with $\delta_{\max} = 0$. The red edges correspond to the weights of the matrix W_0, while the blue edges have weights given by θ. Fig. 3.1d shows the edges corresponding to $W_{\pm 1}$ when $\delta_{\max} > 0$ in purple.

If both layers are binary, the joint distribution is given by

$$\Pr(V, H) \propto \exp\left(\langle H, A^h \rangle + \sum_{j=1}^{n_h} \sum_{t=1}^{T-1} \theta_j H_{jt} H_{j(t+1)}\right). \tag{3.2}$$

An examination of (3.2) reveals the function of the θ parameters. If $\theta_j = 0$, there are no terms involving the products $H_{jt}H_{j(t+1)}$, so the hidden states in row j are independent. If $\theta_j > 0$, the model prefers configurations where $H_{jt} = H_{j(t+1)}$, and the values are more likely to be continuous through time. In the unlikely case that $\theta_j < 0$, "flip-flopping" hidden state configurations would be preferred. Because θ is a free parameter, just like the W_δ parameters, the model is able to learn on a row-by-row basis precisely how important it is for each hidden state to be continuous through time. The addition of the θ parameters is what differentiates our SRBM model from the convolutional RBM described by Lee et al. [102].

Defining the matrix of visible unit inputs

$$A^v = \left[\mathbf{0} \middle| W_{-1} H_{*(1:T-1)}\right] + W_0 H + \left[W_1 H_{*(2:T)} \middle| \mathbf{0}\right],$$

note that

$$\Pr(V|H) \propto \exp\langle H, A^h \rangle \propto \exp\langle V, A^v \rangle,$$

so the V_{it} are independent given H, with $\Pr(V_{it}|H) \propto \exp A^v_{it} V_{it}$, or $\Pr(V_{it}|H) = \sigma(2V_{it}A^v_{it})$ where σ is the logistic sigmoid: $\sigma(x) = (1 + \exp -x)^{-1}$.

We also allow the visible layer to be real-valued with a fixed-variance Gaussian distribution. In that case, then the joint density is

$$f(V, H) \propto \exp\left(\langle H, A^h \rangle + \sum_{j=1}^{n_h} \sum_{t=1}^{T-1} \theta_j H_{jt} H_{j(t+1)} - \frac{1}{2}\langle V, V \rangle\right).$$

Now $f(V|H) \propto \exp\left(\langle V, A^v \rangle - \frac{1}{2}\sum_{it} V_{it}^2\right)$, so the V_{it} are independent and Gaussian-distributed given H, with $V_{it}|H \sim \mathcal{N}(A^v_{it}, 1)$.

Regardless of the type of visible layer, $\Pr(H|V)$ factorizes into terms involving individual

H_{jt} and terms involving $H_{jt}H_{j(t+1)}$:

$$\begin{aligned}\Pr(H|V) &\propto \exp\Big(\langle H, A^h\rangle + \sum_{t=1}^{T-1}\sum_{j=1}^{n_h}\theta_j H_{jt}H_{j(t+1)}\Big)\\ &= \prod_{j=1}^{n^h}\exp\Big(\langle H_{j*}, A^h_{j*}\rangle + \sum_{t=1}^{T-1}\theta_j H_{jt}H_{j(t+1)}\Big)\\ &= \prod_{j=1}^{n^h}\Big(\prod_{t=1}^{T}\exp H_{jt}A^h_{jt}\Big)\Big(\prod_{t=1}^{T-1}\exp\theta_j H_{jt}H_{j(t+1)}\Big). \qquad (3.3)\end{aligned}$$

So given V, the rows of H are independent Markov order-1 sequences with binary states. Therefore the Baum-Welch (or "forward-backward") algorithm can be used to sample from $\Pr(H|V)$ and to determine $\mathbb{E}[H|V]$, which we will need for training.

As with a standard RBM, we would like to train the SRBM to maximize the likelihood of the data, marginalizing over the hidden values: $\prod_i \Pr(V = \hat{V}^{(i)})$. It is not hard to show that the gradient of the log-likelihood $\log\Pr(V = \hat{V})$ with respect to the W_δ has the following form, similar to a standard RBM:

$$\begin{aligned}\nabla_{W_0} &= \hat{V}\Big(\mathbb{E}\big[H' \mid V = \hat{V}\big] - \mathbb{E}\big[H'\big]\Big)\\ \text{and, e.g.,}\quad \nabla_{W_1} &= \hat{V}_{*(1:T-1)}\Big(\mathbb{E}\big[H'_{*(2:T)} \mid V = \hat{V}\big] - \mathbb{E}\big[H'_{*(2:T)}\big]\Big).\end{aligned}$$

Also, the gradient with respect to θ_j is

$$\nabla_{\theta_j} = \sum_{t=1}^{T-1}\Big(\mathbb{E}\big[H_{jt}H_{j(t+1)} \mid V = \hat{V}\big] - \mathbb{E}\big[H_{jt}H_{j(t+1)}\big]\Big).$$

The positive terms (the conditional expectations) can all be computed exactly with Baum-Welch. Just as with the basic RBM, it is intractable to compute the negative terms (the unconditional expectations) exactly. To approximate them, we follow the successful contrastive divergence algorithm for training RBMs. Specifically, we sample $\tilde{V}$ by running two steps of blocked Gibbs sampling, from $\hat{V}$ to H and back, and then replace the unconditional expectations with the corresponding conditional expectations given $\tilde{V}$:

$$\nabla_{W_0} \approx \hat{V}\Big(\mathbb{E}\big[H' \mid V = \hat{V}\big] - \mathbb{E}\big[H' \mid V = \tilde{V}\big]\Big).$$

3.3 *The sequential deep neural network (SDNN)*

An L-layer SDNN is formed by stacking multiple layers of SRBMs. For $l = 1 \dots L-1$, the hidden layer at level l is a binary matrix $H^l \in \{\pm 1\}^{n_l \times T}$ with weight matrices W^l_δ and transition parameters θ^l. We define $V^l \in \mathbb{R}^{n_l \times T}$ for $l = 0 \dots L-1$ to be a matrix of features at layer l. In case $l = 0$ (the input), the features are assumed to be real values that are defined by the user in a task-specific way. For the hidden layers ($l = 1 \dots L-1$), we specify $V^l = \mathbb{E}\left[H^l | V^{l-1}\right]$, where $\Pr(H^l|V^{l-1})$ is defined as in Eq. (3.3), using the input matrix A^l of the l^{th} layer as defined in Eq. (3.1).

For a sequential labeling task, the output layer of the SDNN is a conditional random field taking features over local configurations of the top hidden layer features V^{L-1}. $\{y_1 \dots y_T\}$ is assumed to be a sequence of integer labels, with $y_i \in \{1 \dots n_L\}$. Let $Y \in \mathbb{R}^{n_L \times T}$ be the matrix where $Y_{it} = 1$ if $y_t = i$, and 0 otherwise. We have weight matrices W^L_δ just as with the hidden layers, and the input matrix A^L is formed applying Eq. (3.1) to the features V^{L-1} of the deepest hidden layer. However now instead of a vector θ of transition parameters, we have a full matrix $U \in \mathbb{R}^{n_L \times n_L}$ containing values that represent the affinities between each pair of labels. The distribution is similar to Eq. 3.3:

$$\Pr(Y|V^{L-1}) \propto \exp\left(\langle Y, A^L\rangle + \sum_{t=1}^{T-1} Y'_{*t} U Y_{*(t+1)}\right) \tag{3.4}$$

only here instead of a set of independent binary Markov sequences, $\Pr(Y|V^{L-1})$ defines a single Markov sequence over multinomials with n_L values. The normalization is done by summing over all possible labelings $\{y_1 \dots y_T\}$ which can be performed efficiently with standard algorithms. An illustration of the SDNN structure is shown in Figure 3.2.

The temporal edges at internal layers of an SDNN can potentially offer distinct advantages in modeling capacity. Consider, for example, a CRF that utilizes features with a fixed temporal span over the input, for example, derived from a static DNN. The only hope to recognize patterns that occur over larger spans is via the temporal integration at the output CRF layer. A SDNN, by contrast, has the ability, starting at $l = 2$, for its hidden units to

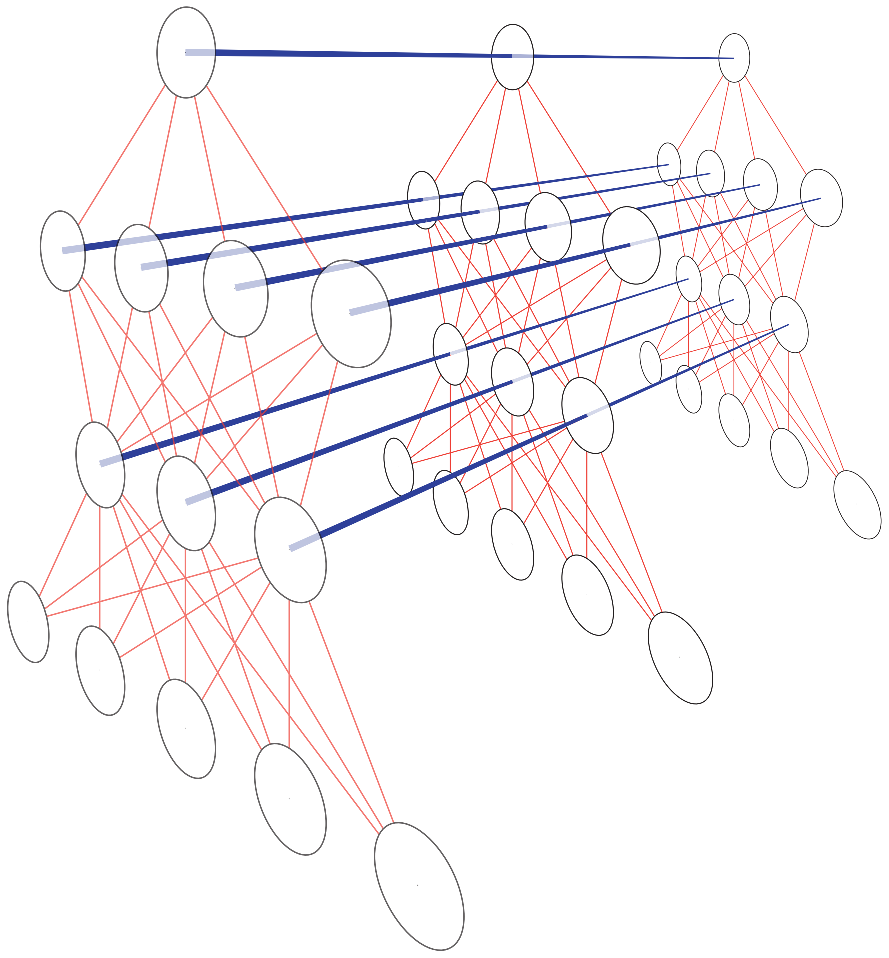


Figure 3.2: Illustration of sequential deep neural network structure with $T = 3$, $L = 3$, $\delta_{\max} = 0$.

detect the presence of an arbitrarily long temporal pattern, or even properties of the entire sequence, owing to the earlier layers' Baum-Welch stages that can pass information over an arbitrary temporal extent.

One way to conceptualize the effect of the message passing at intermediate layers is that unlike a "sliding-window" network that labels each center frame independently given the input in a fixed-size context window, or even a convolutional neural network, the SDNN is able to enforce continuity of the hidden states through time. If the parameter θ_j is large, only configurations in which H_{jt} and $H_{j(t+1)}$ have the same sign will have significant probability, so V_{jt} and $V_j(t+1)$ are likely to be close in value. In the limit as $\theta_j \to \infty$, only two configurations will be allowed, in which either all H_{jt} are $+1$, or all are -1. Then the V_{jt} will exhibit maximal continuity, all having the same value: the difference in the probabilities of the two configurations. Giving the model a way to specify that certain values should be more continuous through time would seem to be particularly advantageous for continuously extended data, where we know *a priori* that the data varies continuously in time, and therefore it may be reasonable to assume that our induced feature representations should as well. If the input data is noisy or corrupted, the capacity to smooth out hidden values could prove especially useful.

3.4 Backpropagation algorithm

To fine tune the SDNN parameters, we use a procedure similar to error backpropagation in a static deep network to compute the gradient of the log-likelihood $\ell = \log \Pr(\hat{Y}|V^{L-1})$. The computation has algorithmic properties that favor efficient implementation. Matrix-matrix multiplication is used for both the upward and downward passes, enabling the use of fast matrix multiplication routines. In addition, both passes require Baum-Welch-like procedures that operate independently on rows of the matrix, and that can make efficient use of distributed processing or vectorized arithmetic. (The reader who is uninterested in the mathematical details may skip to the summary at the end of the section on page 40.)

The gradient of the log-likelihood with respect to U is

$$\begin{aligned}\nabla_U \ell &= \sum_{t=1}^{T-1} \hat{Y}_{*t}\hat{Y}'_{*(t+1)} - \mathbb{E}\Big[\sum_{t=1}^{T-1} Y_{*t}Y'_{*(t+1)}\Big] \\ &= \sum_{t=1}^{T-1}\Big(\hat{Y}_{*t}\hat{Y}'_{*(t+1)} - \mathbb{E}\Big[Y_{*t}Y'_{*(t+1)}\Big]\Big) \end{aligned} \tag{3.5}$$

For W_0^L, we have that

$$\nabla_{W_0^L}\ell = V^{L-1}\nabla_{A^L}\ell = V^{L-1}(\hat{Y}' - \mathbb{E}\big[Y'\big]) = V^{L-1}(D^L)', \tag{3.6}$$

where $D^L \triangleq \nabla_{A^L}\ell$, and the expressions for W_1^L and W_{-1}^L are similar. Note that a different objective function ℓ' could be used, in which case we just need to replace D^L with $\nabla_{A^L}\ell'$, and the rest of the derivation is unchanged.

By the chain rule, the derivative with respect to some value ρ at or below level l is

$$\frac{\partial \ell}{\partial \rho} = \sum_{i=1}^{n_l}\sum_{t=1}^{T}\frac{\partial \ell}{\partial V_{it}^l}\frac{\partial V_{it}^l}{\partial \rho}. \tag{3.7}$$

Define ϵ^l to be the matrix with $\epsilon_{it}^l = \frac{\partial \ell}{\partial V_{it}^l}$. Then

$$\epsilon^{L-1} = \big[\mathbf{0} \mid W_{-1}^L D_{*(1:T-1)}^L\big] + W_0^L D^L + \big[W_1^L D_{*(2:T)}^L \mid \mathbf{0}\big].$$

Because V^l is the conditional expected value of a log-linear distribution, it follows that

$$\frac{\partial V_{it}^l}{\partial \rho} = \mathbb{E}\Big[H_{it}^l\Big(\mathbb{E}\big[\frac{\partial}{\partial \rho}E(V^{l-1}, H^l)\big] - \mathbb{E}\big[\frac{\partial}{\partial \rho}E(V^{l-1}, H^l) \mid H_{it}^l\big]\Big)\Big]. \tag{3.8}$$

Now consider $\rho = (W_0^l)_{jk}$. Since $\frac{\partial}{\partial \rho}E(V^{l-1}, H^l) = -\sum_{\tau=1}^{T} V_{j\tau}^{l-1}H_{k\tau}^l$,

$$\begin{aligned}\frac{\partial V_{it}^l}{\partial \rho} &= \sum_{\tau=1}^{T} V_{j\tau}^{l-1}\mathbb{E}\Big[H_{it}^l\Big(\mathbb{E}\big[H_{k\tau}^l \mid H_{it}^l\big] - \mathbb{E}\big[H_{k\tau}^l\big]\Big)\Big] \\ &= \sum_{\tau=1}^{T} V_{j\tau}^{l-1}\Big(\mathbb{E}\big[H_{it}^l H_{k\tau}^l\big] - \mathbb{E}\big[H_{it}^l\big]\big[H_{k\tau}^l\big]\Big) \\ &= \sum_{\tau=1}^{T} V_{j\tau}^{l-1}\,\mathrm{Cov}(H_{it}^l, H_{k\tau}^l),\end{aligned}$$

which is zero when $k \neq i$ because the rows of H are independent. Plugging this into Eq. (3.7) only the $i = k$ terms remain, yielding

$$\frac{\partial \ell}{\partial (W_0^l)_{jk}} = \sum_{t=1}^{T} \epsilon_{kt}^l \sum_{\tau=1}^{T} V_{j\tau}^{l-1} \operatorname{Cov}(H_{kt}^l, H_{k\tau}^l) = \sum_{\tau=1}^{T} V_{j\tau}^{l-1} \sum_{t=1}^{T} \epsilon_{kt}^l \operatorname{Cov}(H_{kt}^l, H_{k\tau}^l).$$

Defining $D_{i\tau}^l \triangleq \sum_t \epsilon_{it}^l \operatorname{Cov}(H_{it}^l, H_{i\tau}^l)$ we can write the gradient with respect to the entire matrix W_0^l compactly as $\nabla_{W_0^l} \ell = V^{l-1}(D^l)'$ (exactly as (3.6)). The gradients with respect to W_δ^l can also be expressed in terms of D^l.

The case of θ_i^l is similar. Since $\frac{\partial}{\partial \theta_i^l} E(V^{l-1}, H^l) = -\sum_{\tau=1}^{T-1} H_{i\tau}^l H_{i(\tau+1)}^l$, from (3.8)

$$\begin{aligned}
\frac{\partial V_{it}^l}{\partial \theta_i^l} &= \mathbb{E}\Big[H_{it}^l\Big(\mathbb{E}\big[\textstyle\sum_{\tau=1}^{T-1} H_{i\tau}^l H_{i(\tau+1)}^l \mid H_{i\tau}^l\big] - \mathbb{E}\big[\textstyle\sum_{\tau=1}^{T-1} H_{i\tau}^l H_{i(\tau+1)}^l\big]\Big)\Big] \\
&= \sum_{\tau=1}^{T-1} \Big(\mathbb{E}\big[H_{it}^l H_{i\tau}^l H_{i(\tau+1)}^l\big] - \mathbb{E}\big[H_{it}^l\big] E\big[H_{i\tau}^l H_{i(\tau+1)}^l\big]\Big) \\
&= \sum_{\tau=1}^{T-1} \operatorname{Cov}(H_{it}^l, H_{i\tau}^l H_{i(\tau+1)}^l),
\end{aligned}$$

so

$$\begin{aligned}
\nabla_{\theta_i^l} \ell &= \sum_{\tau=1}^{T-1} \sum_{t=1}^{T} \epsilon_{it}^l \operatorname{Cov}(H_{it}^l, H_{i\tau}^l H_{i(\tau+1)}^l) \\
&\triangleq \sum_{\tau=1}^{T-1} F_{i\tau}.
\end{aligned}$$

Finally, by setting $\rho = V_{it}^{l-1}$ in (3.7), we can also derive

$$\epsilon^{l-1} = \big[\mathbf{0}\big|W_{-1}^l D_{*(1:T-1)}^l\big] + W_0^l D^l + \big[W_1^l D_{*(2:T)}^l\big|\mathbf{0}\big]$$

which allows us to recursively compute the derivatives with respect to lower-level parameters.

At first glance, it may appear that it requires $\mathcal{O}(n_l T^2)$ operations to compute D^l and F^l, since each entry is a sum over T weighted covariances. In fact, it is possible to compute all entries of D^l and F^l in time linear in T with an algorithm that bears a striking resemblance to Baum-Welch. Analogously to the forward and backward probabilities of Baum-Welch, we define

$$\alpha^l_{i\tau} = \sum_{t=1}^{\tau-1} \epsilon^l_{it} \frac{\operatorname{Cov}(H^l_{it}, H^l_{i\tau})}{\operatorname{Var} H^l_{i\tau}} \quad \text{and} \quad \beta^l_{i\tau} = \sum_{t=\tau+1}^{T} \epsilon^l_{it} \frac{\operatorname{Cov}(H^l_{i\tau}, H^l_{it})}{\operatorname{Var} H^l_{i\tau}},$$

so that $D^l_{i\tau} = \operatorname{Var}(H^l_{i\tau})(\alpha^l_{i\tau} + \epsilon^l_{i\tau} + \beta^l_{i\tau})$. For $F^l_{i\tau}$ we have

$$\begin{aligned} F^l_{i\tau} &= \sum_{t=1}^{\tau-1} \epsilon^l_{it} \operatorname{Cov}(H^l_{it}, H^l_{i\tau} H^l_{i(\tau+1)}) \\ &\quad + \sum_{t=\tau}^{\tau+1} \epsilon^l_{it} \operatorname{Cov}(H^l_{it}, H^l_{i\tau} H^l_{i(\tau+1)}) \\ &\quad + \sum_{t=\tau+2}^{T} \epsilon^l_{it} \operatorname{Cov}(H^l_{it}, H^l_{i\tau} H^l_{i(\tau+1)}). \end{aligned}$$

Considering the first term,[3]

$$\begin{aligned} \sum_{t=1}^{\tau-1} \epsilon^l_{it} \operatorname{Cov}(H^l_{it}, H^l_{i\tau} H^l_{i(\tau+1)}) &= \sum_{t=1}^{\tau-1} \epsilon^l_{it} \frac{\operatorname{Cov}(H^l_{it}, H^l_{i\tau}) \operatorname{Cov}(H^l_{i\tau}, H^l_{i\tau} H^l_{i(\tau+1)})}{\operatorname{Var} H^l_{i\tau}} \\ &= \operatorname{Cov}(H^l_{i\tau}, H^l_{i\tau} H^l_{i(\tau+1)}) \alpha^l_{i\tau}, \end{aligned}$$

so that

$$F^l_{i\tau} = \operatorname{Cov}(H^l_{i\tau}, H^l_{i\tau} H^l_{i(\tau+1)})(\alpha^l_{i\tau} + \epsilon^l_{i\tau}) + \operatorname{Cov}(H^l_{i\tau} H^l_{i(\tau+1)}, H^l_{i(\tau+1)})(\beta^l_{i(\tau+1)} + \epsilon^l_{i(\tau+1)}).$$

To compute α^l_{i1}, we can set $\alpha^l_{i1} = 0$, and recursively apply

$$\begin{aligned} \alpha^l_{i(\tau+1)} &= \frac{1}{\operatorname{Var} H^l_{i(\tau+1)}} \left(\sum_{t=1}^{\tau-1} \epsilon^l_{it} \operatorname{Cov}(H^l_{it}, H^l_{i(\tau+1)}) + \epsilon^l_{i\tau} \operatorname{Cov}(H^l_{i\tau}, H^l_{i(\tau+1)}) \right) \\ &= \frac{1}{\operatorname{Var} H^l_{i(\tau+1)}} \left(\sum_{t=1}^{\tau-1} \epsilon^l_{it} \frac{\operatorname{Cov}(H^l_{it}, H^l_{i\tau}) \operatorname{Cov}(H^l_{i\tau}, H^l_{i(\tau+1)})}{\operatorname{Var} H_{i\tau}} + \epsilon^l_{i\tau} \operatorname{Cov}(H^l_{i\tau}, H^l_{i(\tau+1)}) \right) \\ &= \frac{\operatorname{Cov}(H^l_{i\tau}, H^l_{i(\tau+1)})}{\operatorname{Var} H^l_{i(\tau+1)}} \left(\alpha^l_{i\tau} + \epsilon^l_{i\tau} \right) \\ &= \frac{1}{2} \left(\mathbb{E}[H^l_{i\tau} | H^l_{i(\tau+1)} = 1] - \mathbb{E}[H^l_{i\tau} | H^l_{i(\tau+1)} = -1] \right) \left(\alpha^l_{i\tau} + \epsilon^l_{i\tau} \right), \end{aligned} \tag{3.9}$$

[3]Here and in the recursion for α below, use the identity $\operatorname{Cov}(A, C) = \frac{\operatorname{Cov}(A,B)\operatorname{Cov}(B,C)}{\operatorname{Var} B}$ which holds for ± 1-valued variables where A is independent of C given B. The proof of this claim is given in Appendix A.

and symmetrically

$$\beta^l_{i(\tau-1)} = \frac{1}{2}\big(\mathbb{E}[H^l_{i\tau}|H^l_{i(\tau-1)} = 1] - \mathbb{E}[H^l_{i\tau}|H^l_{i(\tau-1)} = -1]\big)\left(\beta^l_{i\tau} + \epsilon^l_{i\tau}\right),$$

yielding a dynamic program for computing α^l and β^l (and therefore D^l and F^l) in linear time. Note also that the normalization (dividing by the variance) from the definition of α^l is not necessary in (3.9) greatly increasing the numerical accuracy of the algorithm, considering that $\operatorname{Var} H^l_{i\tau}$ may often be numerically zero.

The entire SDNN backpropagation procedure is summarized as follows.

1. (Upward pass.) Given V^0, for $l = 1 \dots L-1$ compute the inputs A^l from the previous layer's activations V^{l-1}, and then V^l from A^l using Baum-Welch on each row. Compute A^L from V^{L-1} and the label marginals $\mathbb{E}[Y_{it}]$ and $\mathbb{E}[Y_{it}Y_{j(t+1)}]$ with Baum-Welch over the multinomial sequence.

2. (Downward pass.) Construct D^L from the labels $\hat{Y}$ and marginals, and update the weights at the output layer L. Then for $l = L-1 \dots 1$, backpropagate the error ϵ^l from D^{l+1}, compute D^l and F^l from ϵ^l with the Baum-Welch-like algorithm above, and finally update the weights at layer l.

Since the message passing operations in both passes require only an amount of work that is linear in the size of the layers, computation time for larger models will still be dominated by the matrix multiplications to compute the inputs and backpropagate the errors. In other words, the SDNN does not require significantly more computation than a CNN model with the same architectural parameters.

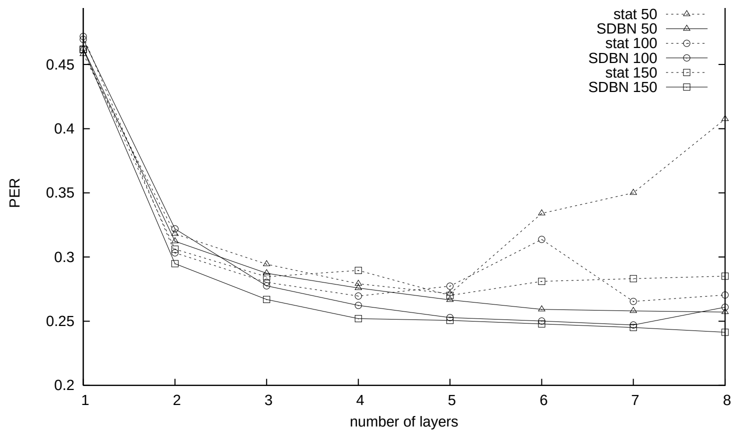
3.5 Experiments

We tested the SDNN on the TIMIT phone recognition dataset. We use the standard train/test split for phone recognition experiments: removing all SA records from training, and testing on the core test set of 24 speakers. The dataset has 3696 train utterances and 192 test utterances, with an average of 304 frames per utterance. The input features were 12th order mel-frequency cepstral coefficients (MFCCs) [44] and energy over 25 ms windows, plus the first-order temporal differences, giving 26 total features per 10 ms frame. The outputs are sequences of the standard 39-phone set of Lee and Hon [104].

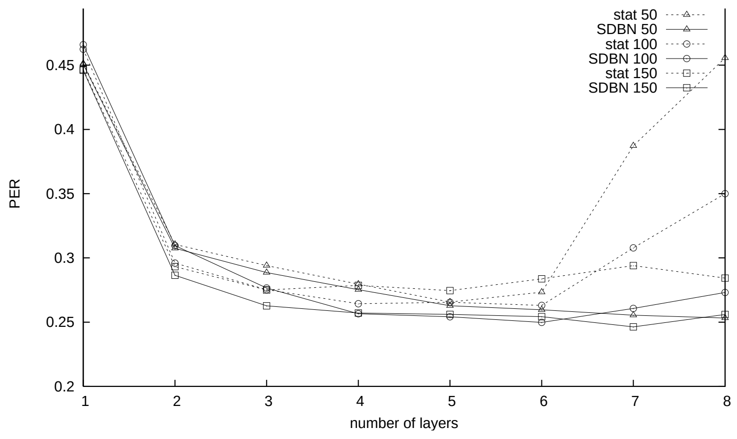
We divide each phone into two subphone states, so frames are labeled as either the beginning or ending substate of a phone, giving a total of 78 labels. This allows us to model repeated phones and also to account for changes in the acoustic signal during pronunciation of a single phone. We constrain the model to require traversal through each substate of each phone. The boundaries between subphones are kept latent, that is, we follow the gradient of $\log \sum_{Y_{\text{sub}}\,:\,\text{phones}(Y_{\text{sub}})=\hat{Y}} \Pr(Y_{\text{sub}})$. This only requires a small change to the usual forward-backward algorithm on the output layer.

To establish the value of the primary innovation of the SDNN—that it makes use of sequence models at all layers—we compared the complete SDNN to a CNN model with the same architectural parameters. The CNN uses a sequence classifier at the top level, but no sequential model at any hidden layer, exactly as if θ^l were constrained to be zero for $l < L$. We compare the models over a range of configurations: we vary the model depth from one layer to eight, the half-width $\delta_{\max}$ of the input window (at all layers) from one to four, and the number of hidden units per frame (in each layer) from 50 to 150. Each stage of training (that is, pre-training each layer with CD and also joint training of all parameters with BP) continued until the training criterion (squared reconstruction error for CD, log-likelihood for BP) failed to improve over five epochs, at which point the learning rate was annealed linearly to zero over another five epochs. Weight decay is applied after each update, and the amount of decay is scaled proportionally with T. The initial learning rates, weight decays and momentum parameters were estimated using random search [22] to maximize phone error rate (PER) on a randomly selected 10% the training set, which was added back before training the final model for test results.

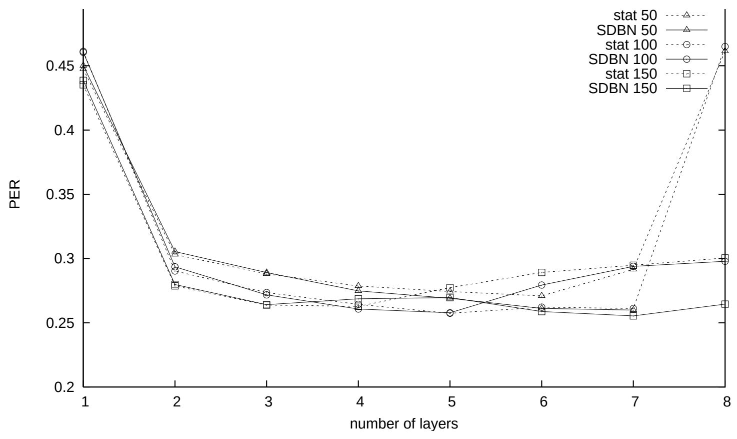
The results of the experiment are summarized in Figure 3.3. Comparing the complete SDNN to the baseline CNN model, it is apparent that using full sequence information at all



(a) $\delta_{\max} = 1$



(b) $\delta_{\max} = 2$



(c) $\delta_{\max} = 4$

Figure 3.3: Average test set PER of the SDNN and baseline CNN model (labeled "stat") over a range of number of layers, n_h and input window width $\delta_{\max}$.

layers is beneficial across nearly all configurations, and the gains are more significant as the number of hidden layers increases.

Interestingly, the results also indicate that the use of temporal hidden units may make a very wide input window unnecessary: our best results are obtained with $\delta_{\max} = 1$, whereas $\delta_{\max}$ from 5 to as high as 11 is more common with MLPs or DNNs for phone recognition [42, 72]. This suggests that with temporal connections the hidden units are able to represent longer-range features that would otherwise require large input windows to capture.

Our best performing configuration (150 units/frame, 8 layers, $\delta_{\max} = 1$) achieved a PER of 24.2, which surpasses many systems that are highly tailored to the phone-recognition task [90, 41, 34, 129, 91, 175, 158]. Other recent developments using deep networks have improved somewhat on those scores [42, 72, 144, 182], and in future work it would be interesting to try combining some of those successful techniques with the SDNN model, in particular, the use of triphone states (rather than our biphones), dropout training and larger numbers of hidden units per frame. It is significant that the SDNN performs so well with only 150 hidden units per frame, over an order of magnitude fewer than state-of-the-art neural systems (1500 in Dahl et al. [42], 2000 in Tóth [182] and 4000 in Hinton et al. [72]), as well as much smaller input windows. However our goal in the present work is not to build a state-of-the-art system but to demonstrate that the use of temporal connections between hidden units improves upon a strong baseline model that does not.

3.6 Conclusion

In this chapter we introduced the sequential deep neural network for inducing features of continuously extended data. Unlike previous approaches such as a sliding-window network or a convolutional network, the SDNN can capture temporal patterns of arbitrary extent, and explicitly model the tendency of each feature to be continuous through time. We demonstrated that the essential innovation of the SDNN—the use of temporal connections between hidden units—can improve performance on a sequential task. We hope that the SDNN will prove to be a useful component of the deep representation learning toolbox when the data has one or more continuous dimensions.

The SDNN model could be extended in several ways that may increase its effectiveness at learning useful representations for continuously extended data. Although using dense connections between hidden representations at adjacent frames would be impractical, dense connections between small subsets of units (but larger than one) could give the model additional expressive power at marginal computational cost. Another useful direction would be to extend the model to different nonlinear activation functions than the hyperbolic tangent. Something like rectified linear units could perhaps be used, but it is not clear how that would fit into the formulation of the SRBM. These would be interesting directions for future research.

Chapter 4 LEARNING DEEP REPRESENTATIONS FOR MULTIVIEW DATA

4.1 Multiview data

Some data is inherently multi-viewed, that is, the features of the input representation are conveniently partitioned into two or more groups such that features within each group are more tightly related than features in different groups. For example if a learning agent takes input simultaneously from multiple sensory modalities such as vision and audio, the visual components of the image might be considered to be one data view, while the acoustic features of the audio form another. Within each view different features may be simply related: adjacent pixels are likely to have similar values, and acoustic energy at some frequency is likely to be predictive of energy at harmonic frequencies. However across views whatever relationships exist are more likely to be complex and abstract. Continuing with our running example of audio and video, the presence of a blue pixel may not be particularly associated with any aspects of the audio, but if we can recognize abstract properties like the presence of a car in the image, this could be predictive of the presence of a car's engine sounds or other traffic noises in the audio (which features are themselves at a fairly abstract level). The intuition behind the model presented in this chapter is that if such correspondences exist between deep level features of the different views, then those features are likely to correspond to "real" latent aspects of the scene. Such features may be considered more likely to form part of a useful abstract representation.[1]

Canonical correlation analysis (CCA) [77, 3] is a standard statistical technique for finding linear projections of two random vectors that are maximally correlated. Kernel canonical

[1]The work presented in this chapter was originally published as Andrew et al. [7].

correlation analysis (KCCA) [2, 119, 12, 64] is an extension of CCA in which maximally correlated *nonlinear* projections, restricted to reproducing kernel Hilbert spaces with corresponding kernels, are found. Both CCA and KCCA are techniques for learning representations of two data views, such that each view's representation is simultaneously the most predictive of, and the most predictable by, the other. CCA and KCCA have been used for unsupervised data analysis when multiple views are available [65, 189, 46]; learning features for multiple modalities that are then fused for prediction [154]; learning features for a single view when another view is available for representation learning but not at prediction time [25, 33, 8]; and reducing sample complexity of prediction problems using unlabeled data [85]. The applications range broadly across a number of fields, including medicine, meteorology [3], chemometrics [127], biology and neurology [185, 65], natural language processing [189, 62, 46], speech processing [37, 151, 9], computer vision [92], and multimodal signal processing [154, 166]. An appealing property of CCA for prediction tasks is that, if there is noise in either view that is uncorrelated with the other view, the learned representations should not contain the noise.

While kernel CCA allows learning of nonlinear representations, it has the drawback that the representation is limited by the fixed kernel. Also, as it is a nonparametric method, the time required to train a KCCA model or compute the representations of new datapoints scales poorly with the size of the training set. In this chapter, we consider an alternative method for learning flexible nonlinear correlated representations using deep neural networks. Deep networks do not suffer from the aforementioned drawbacks of nonparametric models, and given the empirical success of deep neural network representations on a wide variety of tasks, we may expect to be able to learn more highly correlated representations, particularly as in many cases whatever correspondences do exist between highly dissimilar data views will likely be highly abstract and less recognizable by shallow architectures. In this work we introduce *deep CCA* (DCCA), which simultaneously learns two deep nonlinear representation mappings of two views that are maximally correlated.

4.2 Related work

Some researchers have attempted to learn correlated representations via neural networks in the past [10, 78, 79, 80]. However these early approaches used ad-hoc training procedures for which it is not clear whether the multivariate total correlation is actually being maximized. In contrast, we give an exact expression for the gradient of the correlation objective for use in gradient-based optimization. The cited works had other weaknesses addressed by the current work on DCCA: they did not regularize the models in a principled way, they were tested only on very small (two-to-three-dimensional) artificial datasets, and were evaluated only on the training set correlation, so it is not clear how well they generalize to unseen data.[2]

The most closely related work is that of Ngiam et al. [134] on multimodal autoencoders and Srivastava and Salakhutdinov [172] on multimodal restricted Boltzmann machines. In these approaches, there is a single network being learned with one or more layers connected to both views (modalities); in the absence of one of the views, it can be predicted from the other view using the learned network. The key difference is that in our approach we learn two separate deep encodings, with the objective that the learned encodings are as correlated as possible. These different objectives may have advantages in different settings. In the current work, we are interested specifically in the correlation objective, that is in extending CCA with learned nonlinear mappings. Our approach is therefore directly applicable in all of the settings where CCA and KCCA are used, and we compare its ability relative to CCA and KCCA to generalize the correlation objective to new data, showing that DCCA achieves much better results.

4.3 Background: CCA, KCCA, and deep representations

Let $(X_1, X_2) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2}$ denote random vectors with covariances $(\Sigma_{11}, \Sigma_{22})$ and cross-covariance Σ_{12}. CCA finds pairs of linear projections of the two views, $(w_1' X_1, w_2' X_2)$ that

[2]There have also been attempts to approximate the linear CCA variates with MLPs, a task of questionable utility which should not be confused with actually learning nonlinear representations [96].

are maximally correlated:

$$(w_1^*, w_2^*) = \operatorname*{argmax}_{w_1, w_2} \ \text{corr}(w_1' X_1, w_2' X_2) \tag{4.1}$$

$$= \operatorname*{argmax}_{w_1, w_2} \frac{w_1' \Sigma_{12} w_2}{\sqrt{w_1' \Sigma_{11} w_1 w_2' \Sigma_{22} w_2}}. \tag{4.2}$$

Since the objective is invariant to scaling of w_1 and w_2, we can constrain the projections to have unit variance:

$$(w_1^*, w_2^*) = \operatorname*{argmax}_{w_1' \Sigma_{11} w_1 = w_2' \Sigma_{22} w_2 = 1} w_1' \Sigma_{12} w_2 \tag{4.3}$$

When finding multiple pairs of vectors (w_1^i, w_2^i), subsequent projections are also constrained to be uncorrelated with previous ones, that is $w_1^i \Sigma_{11} w_1^j = w_2^i \Sigma_{22} w_2^j = 0$ for $i < j$. Assembling the top k projection vectors w_1^i into the columns of a matrix $A_1 \in \mathbb{R}^{n_1 \times k}$, and similarly placing w_2^i into $A_2 \in \mathbb{R}^{n_2 \times k}$, we obtain the following formulation to identify the top $k \leq \min(n_1, n_2)$ projections:

$$\begin{aligned} \text{maximize:} \quad & \text{tr}(A_1' \Sigma_{12} A_2) \\ \text{subject to:} \quad & A_1' \Sigma_{11} A_1 = A_2' \Sigma_{22} A_2 = I. \end{aligned} \tag{4.4}$$

There are several ways to express the solution to this objective; we follow the one in Mardia et al. [115]. Define $T \triangleq \Sigma_{11}^{-1/2} \Sigma_{12} \Sigma_{22}^{-1/2}$, and let U_k and V_k be the matrices of the first k left- and right- singular vectors of T. Then the optimal objective value is the sum of the top k singular values of T (the Ky Fan k-norm of T) and the optimum is attained at $(A_1^*, A_2^*) = (\Sigma_{11}^{-1/2} U_k, \Sigma_{22}^{-1/2} V_k)$. Note that this solution assumes that the covariance matrices Σ_{11} and Σ_{22} are nonsingular, which is satisfied in practice because they are estimated from data with regularization: given centered data matrices $\bar{H}_1 \in \mathbb{R}^{n_1 \times m}, \bar{H}_2 \in \mathbb{R}^{n_2 \times m}$, one can estimate, e.g.

$$\hat{\Sigma}_{11} = \frac{1}{m-1} \bar{H}_1 \bar{H}_1' + r_1 I, \tag{4.5}$$

where $r_1 > 0$ is a regularization parameter. Estimating the covariance matrices with regularization also reduces the detection of spurious correlations in the training data, a.k.a. overfitting [45].

4.3.1 Kernel CCA

Kernel CCA finds pairs of nonlinear projections of the two views [64]. The reproducing kernel Hilbert spaces (RKHS) of functions on $\mathbb{R}^{n_1}, \mathbb{R}^{n_2}$ are denoted $\mathcal{H}_1$, $\mathcal{H}_2$ and the associated positive definite kernels are denoted κ_1, κ_2. The optimal projections are those functions $f_1^* \in \mathcal{H}_1, f_2^* \in \mathcal{H}_2$ that maximize the correlation between $f_1^*(X_1)$ and $f_2^*(X_2)$:

$$\begin{aligned}(f_1^*, f_2^*) &= \underset{f_1 \in \mathcal{H}_1, f_2 \in \mathcal{H}_2}{\operatorname{argmax}} \ \operatorname{corr}\left(f_1(X_1), f_2(X_2)\right) \qquad (4.6)\\ &= \underset{f_1 \in \mathcal{H}_1, f_2 \in \mathcal{H}_2}{\operatorname{argmax}} \frac{\operatorname{Cov}\left(f_1(X_1), f_2(X_2)\right)}{\sqrt{\operatorname{Var}\left(f_1(X_1)\right) \operatorname{Var}\left(f_2(X_2)\right)}},\end{aligned}$$

To solve the nonlinear KCCA problem, the "kernel trick" is used: Since the nonlinear maps $f_1 \in \mathcal{H}_1$, $f_2 \in \mathcal{H}_2$ are in RKHS, the solutions can be expressed as linear combinations of the kernels evaluated at the data: $f_1(x) = \alpha_1' \kappa_1(x, \cdot)$, where $\kappa_1(x, \cdot)$ is a vector whose i^{th} element is $\kappa_1(x, x_i)$ (resp. for $f_2(x)$). KCCA can then be written as finding vectors $\alpha_1, \alpha_2 \in \mathbb{R}^m$ that solve the optimization problem

$$\begin{aligned}(\alpha_1^*, \alpha_2^*) &= \underset{\alpha_1, \alpha_2}{\operatorname{argmax}} \frac{\alpha_1' K_1 K_2 \alpha_2}{\sqrt{(\alpha_1' K_1^2 \alpha_2)\ (\alpha_1' K_2^2 \alpha_2)}}\\ &= \underset{\alpha_1' K_1^2 \alpha_1 = \alpha_2' K_2^2 \alpha_2 = 1}{\operatorname{argmax}} \quad \alpha_1' K_1 K_2 \alpha_2, \qquad (4.7)\end{aligned}$$

where $K_1 \in \mathbb{R}^{m \times m}$ is the centered Gram matrix $K_1 = K - K\mathbf{1} - \mathbf{1}K + \mathbf{1}K\mathbf{1}$, $K_{ij} = \kappa_1(x_i, x_j)$ and $\mathbf{1} \in \mathbb{R}^{m \times m}$ is an all-1s matrix, and similarly for K_2. Subsequent vectors (α_1^j, α_2^j) are solutions of (4.7) with the constraints that $(f_1^j(X_1), f_2^j(X_2))$ are uncorrelated with the previous ones.

Careful regularization is critical to the performance of KCCA, since the spaces $\mathcal{H}_1$, $\mathcal{H}_2$ could have high complexity. Since $\alpha_1' f_1(\cdot)$ plays the role of w_1 in KCCA, the generalization of $w_1' w_1$ would be $\alpha_1' K_1 \alpha$. Therefore the correct generalization of (4.5) is to use $K_1^2 + r_1 K_1$ in place of K_1^2 in the constraints of (4.7), for regularization parameter $r_1 > 0$ (resp. for K_2^2).

The optimization is in principle simple: The objective is maximized by the top eigenvectors of the matrix

$$(K_1 + r_1 I)^{-1} \ K_2 \ (K_2 + r_2 I)^{-1} \ K_1. \qquad (4.8)$$

The regularization coefficients r_1 and r_2, as well as any parameters of the kernel in KCCA, can be tuned using held-out data. Often a further regularization is done by first projecting the data onto an intermediate-dimensionality space, between the target and original dimensionality [49, 8]. In practice solving KCCA may not be straightforward, as the kernel matrices become very large for real-world data sets of interest, and iterative SVD algorithms for the initial dimensionality reduction can be used [8].

4.3.2 Deep learning

Deep neural networks, having more than two hidden layers, are capable of representing nonlinear functions involving multiply nested high-level abstractions of the kind that may be necessary to accurately model complex real-world data. As discussed in chapter 2, there has been a resurgence of interest in such models following the advent of various successful unsupervised methods for initializing the parameters ("pretraining") in such a way that a useful solution can be found [71, 70]. The growing availability of both data and compute resources also contributes to the resurgence, because empirically the performance of deep networks seems to scale very well with data size and complexity.

While deep networks are more commonly used for learning classification labels or mapping to another vector space with supervision, here we use them to learn nonlinear transformations of two datasets to a space in which the data is highly correlated, just as KCCA does. The same properties that may account for deep networks' success in other tasks—high model complexity, the ability to concisely represent a hierarchy of features for modeling real-world data distributions—could be particularly useful in a setting where the output space is significantly more complex than a single label.

4.4 Deep canonical correlation analysis (DCCA)

Deep CCA computes representations of the two views by passing them through multiple stacked layers of nonlinear transformation (see Figure 4.1). Assume for simplicity that each intermediate layer in the network for the first view has c_1 units, and the final (output) layer

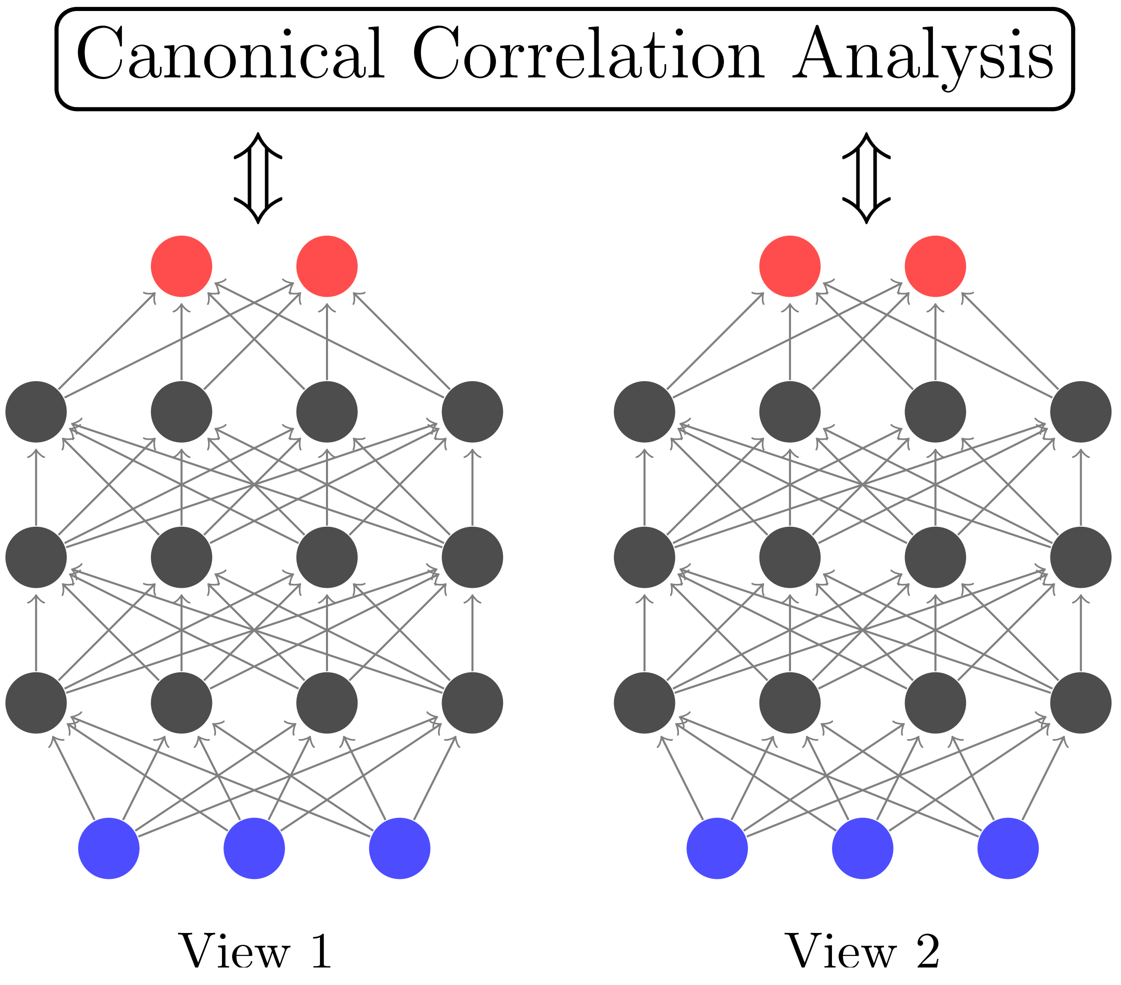


Figure 4.1: A schematic of deep CCA, consisting of two deep networks learned so that the output layers (topmost layer of each network) are maximally correlated. Blue nodes correspond to input features ($n_1 = n_2 = 3$), grey nodes are hidden units ($c_1 = c_2 = 4$), and the output layer is red ($o = 2$). Both networks have $d = 4$ layers.

has o units. Let $x_1 \in \mathbb{R}^{n_1}$ be an instance of the first view. The outputs of the first layer for the instance x_1 are $h_1 = s(W_1^1 x_1 + b_1^1) \in \mathbb{R}^{c_1}$, where $W_1^1 \in \mathbb{R}^{c_1 \times n_1}$ is a matrix of weights, $b_1^1 \in \mathbb{R}^{c_1}$ is a vector of biases, and $s : \mathbb{R} \mapsto \mathbb{R}$ is a nonlinear function applied componentwise. The outputs h_1 may then be used to compute the outputs of the next layer as $h_2 = s(W_2^1 h_1 + b_2^1) \in \mathbb{R}^{c_1}$, and so on until the final representation $f_1(x_1) = s(W_d^1 h_{d-1} + b_d^1) \in \mathbb{R}^o$ is computed, for a network with d layers. Given an instance x_2 of the second view, the representation $f_2(x_2)$ is computed the same way, with different parameters W_l^2 and b_l^2 (and potentially different architectural parameters c_2 and d). The goal is to jointly learn parameters for both views W_l^v and b_l^v such that $\text{corr}(f_1(X_1), f_2(X_2))$ is as high as possible. If θ_1 is the vector of all parameters W_l^1 and b_l^1 of the first view for $l = 1, \ldots, d$, and similarly for θ_2, then

$$(\theta_1^*, \theta_2^*) = \underset{(\theta_1, \theta_2)}{\text{argmax}} \, \text{corr}(f_1(X_1; \theta_1), f_2(X_2; \theta_2)). \tag{4.9}$$

To find (θ_1^*, θ_2^*), we follow the gradient of the correlation objective as estimated on the training data. Let $H_1 \in \mathbb{R}^{o \times m}, H_2 \in \mathbb{R}^{o \times m}$ be matrices whose columns are the top-level representations produced by the deep models on the two views, for a training set of size m. Let $\bar{H}_1 = H_1 - \frac{1}{m} H_1 \mathbf{1}$ be the centered data matrix (resp. $\bar{H}_2$), and define $\hat{\Sigma}_{12} = \frac{1}{m-1} \bar{H}_1 \bar{H}_2'$, and $\hat{\Sigma}_{11} = \frac{1}{m-1} \bar{H}_1 \bar{H}_1' + r_1 I$ for regularization constant r_1 (resp. $\hat{\Sigma}_{22}$). Assume that $r_1, r_2 > 0$ so that $\hat{\Sigma}_{11}, \hat{\Sigma}_{22}$ are positive definite.

As discussed in section 4.3 for CCA, the total correlation of the top k components of H_1 and H_2 is the sum of the top k singular values of the matrix $T = \hat{\Sigma}_{11}^{-1/2} \hat{\Sigma}_{12} \hat{\Sigma}_{22}^{-1/2}$. If we take $k = o$, then this is exactly the matrix trace norm of T, or[3]

$$\text{corr}(H_1, H_2) = ||T||_{\text{tr}} = \text{tr}(T'T)^{1/2}. \tag{4.10}$$

The parameters W_l^v and b_l^v of DCCA are trained to optimize this quantity using gradient-based optimization. To compute the gradient of $\text{corr}(H_1, H_2)$ with respect to all parameters W_l^v and b_l^v, we can compute its gradient with respect to H_1 and H_2 and then use backpropagation.

[3]Here we abuse notation slightly, writing $\text{corr}(H_1, H_2)$ as the empirical correlation of the data represented by the matrices H_1 and H_2.

If the singular value decomposition of T is $T = UDV'$, then

$$\frac{\partial \text{corr}(H_1, H_2)}{\partial H_1} = \frac{1}{m-1}\left(2\nabla_{11}\bar{H}_1 + \nabla_{12}\bar{H}_2\right). \tag{4.11}$$

where

$$\nabla_{12} = \hat{\Sigma}_{11}^{-1/2} UV' \hat{\Sigma}_{22}^{-1/2} \tag{4.12}$$

and

$$\nabla_{11} = -\frac{1}{2}\hat{\Sigma}_{11}^{-1/2} UDU' \hat{\Sigma}_{11}^{-1/2}, \tag{4.13}$$

and $\partial \text{corr}(H_1, H_2)/\partial H_2$ has a symmetric expression. The derivation of the gradient is not entirely straightforward (involving, for example, the gradient of the trace of the matrix square-root, which we could not find in standard references such as Petersen and Pedersen [142]) and is given in appendix B. We also regularize (4.10) by adding to it a quadratic penalty with weight $\lambda_\text{b} > 0$ for all parameters.

Because the correlation objective is a function of the entire training set that does not decompose into a sum over data points, it is not clear how to use a stochastic optimization procedure that operates on data points one at a time. We experimented with a stochastic method based on mini-batches, but obtained much better results with full-batch optimization using the L-BFGS second-order optimization method [135] which has been found to be useful for deep learning in other contexts [97].

As discussed in section 4.3.2 for deep models in general, the best results will in general not be obtained if parameter optimization is started from random initialization—some form of pretraining is necessary. In the experiments of this chapter, we initialize the parameters of each layer with a denoising autoencoder [187]. Given centered input training data assembled into a matrix $X \in \mathbb{R}^{n \times m}$, a distorted matrix $\tilde{X}$ is created by adding i.i.d. zero-mean Gaussian noise with variance σ_a^2. For parameters $W \in \mathbb{R}^{c \times n}$ and $b \in \mathbb{R}^c$, the reconstructed data $\hat{X} = W's(W\tilde{X} + b\bar{1}')$ is formed.[4] Then we use L-BFGS to find a local minimum of the total

[4]This is the so-called "tied-weights" version of the autoencoder network where the same weights W, transposed, are used for mapping to the hidden layer and mapping to the reconstruction layer.

squared error from the reconstruction to the original data, plus a quadratic penalty:

$$l_{\mathrm{a}}(W, b) = ||\hat{X} - X||_F^2 + \lambda_{\mathrm{a}}(||W||_{\mathrm{F}}^2 + ||b||_2^2), \tag{4.14}$$

where $|| \cdot ||_{\mathrm{F}}$ is the matrix Frobenius norm. The minimizing values W^* and b^* are used to initialize optimization of the DCCA objective, and to produce the representation for pretraining the next layer. σ_{a}^2 and λ_{a} are treated as hyperparameters, and optimized on a development set, as described in section 4.5.

4.4.1 Non-saturating nonlinearity

Any form of sigmoid nonlinearity could be used to determine the output of the nodes in a DCCA network, but in our experiments we obtained the best results using a novel non-saturating sigmoid function based on the cube root. If $g : \mathbb{R} \mapsto \mathbb{R}$ is the function $g(y) = y^3/3 + y$, then our function is $s(x) = g^{-1}(x)$. Like the more popular logistic (σ) and tanh nonlinearities, s has sigmoid shape and has unit slope at $x = 0$. However, logistic and tanh approach their asymptotic value very quickly, at which point the derivative drops to essentially zero (i.e., they saturate). On the other hand, s is not bounded, and its derivative falls off much more gradually with x. We hypothesize that these properties make s better-suited for batch optimization with second-order methods which might otherwise get stuck on a plateau early during optimization. In figure 4.2 we plot s alongside tanh for comparison.

Another property that our nonsaturating sigmoid function shares with logistic and tanh is that its derivative is a simple function of its value. For example, $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, and $\tanh'(x) = 1 - \tanh^2(x)$. This property is convenient for neural network implementations, because it means the input to a unit can be overwritten by its output, which is used both for forward propagation and for computing the derivative in the backpropagation phase. Also, as it turns out, it is more efficient to compute the derivatives as a function of the value in all of these cases (e.g., given $y = \tanh(x)$, $1 - y^2$ can be computed more efficiently than $1 - \tanh^2(x)$). In the case of s, we have $s'(x) = (s^2(x) + 1)^{-1}$ as is easily shown with implicit

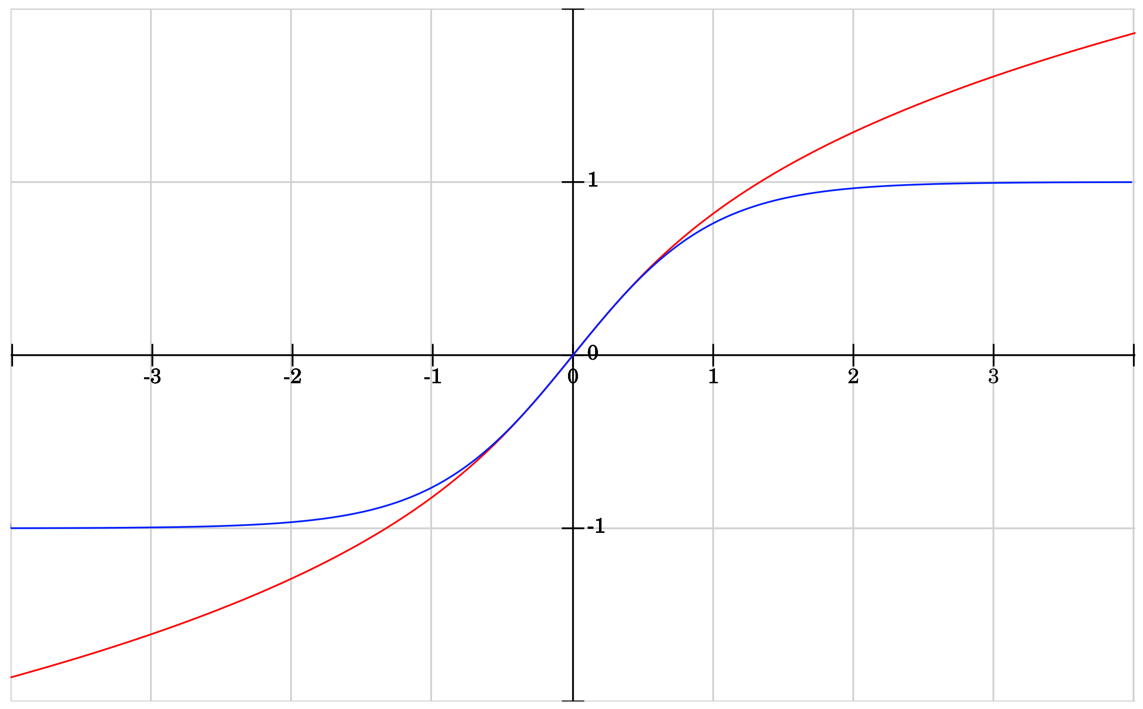


Figure 4.2: Comparison of our modified cube-root sigmoid function (red) with the more standard tanh (blue).

differentiation. If $y = s(x)$, then

$$x = y^3/3 + y,$$

so,

$$\frac{dx}{dy} = y^2 + 1,$$

and,

$$\frac{dy}{dx} = \frac{1}{y^2 + 1}.$$

To compute $s(x)$, we use Newton's method. To solve for $g(y) - x = 0$, iterate

$$\begin{aligned} y_{n+1} &= y_n - \frac{g(y_n) - x}{g'(y_n)} \\ &= y_n - \frac{y_n^3/3 + y_n - x}{y_n^2 + 1} \\ &= \frac{2y_n^3/3 + x}{y_n^2 + 1}. \end{aligned}$$

For positive x, initializing $y_0 = x$, the iteration decreases monotonically, so convergence is guaranteed.[5] When x is negative, we use the property that $s(x) = -s(-x)$. In the range of values in our experiments, the iteration converges to machine precision in just a few iterations. As a further optimization, we wrote a vectorized CPU implementation.

4.5 Experiments

We perform experiments on two datasets to demonstrate that DCCA learns transformations that are not only dramatically more correlated than a linear CCA baseline, but also significantly more correlated than well-tuned KCCA representations. We refer to a DCCA model with an output size of o and d layers (including the output) as DCCA-o-d.

[5]Another reasonable initialization would be $y_0 = \sqrt[3]{3x}$ which is a tighter upper bound for $s(x)$ and so would require fewer iterations, but the cube root evaluation may be expensive on some architectures.

Because the total correlation of two transformed views grows with dimensionality, it is important to compare only equal-dimensionality representations. In addition, in order to compare the test correlation of the top k components of two representations of dimensionality $o_1, o_2 \geq k$, the components must be ordered by their correlation on the training data. In the case of CCA and KCCA, the dimensions are always ordered in this way by default, but in DCCA, there is no ordering to the output nodes. Therefore, we derive such an ordering by performing a final (linear) CCA on the output layers of the two views on the training data. This final CCA produces two projection matrices A_1, A_2, which are applied to the DCCA test output before computing test set correlation. Another way would be to compute a new DCCA representation at each target dimensionality; this is not done here for expediency but should, if anything, improve performance.

Each of the DCCA models we tested has a fixed number of layers and output size, and the parameters W_l^v and b_l^v are trained as discussed in section 4.4. Several other values are treated as hyperparameters. Specifically, for each view, we have σ_{a}^2 and λ_{a} for autoencoder pretraining, c, the width of all hidden layers (a large integer parameter treated as a real value) and r, the CCA regularization hyperparameter. Finally there is a single hyperparameter λ_{b}, the fine-tuning regularization weight. These values were chosen to optimize total correlation on a development set using a derivative-free optimization method.

4.5.1 MNIST handwritten digits

For our first experiments, we learn correlated representations of the left and right halves of handwritten digit images, as shown in figure 4.3. While in this case the two views come from the same sensory modality (vision), we would nevertheless expect that there exist more highly correlated properties between abstract features like digit identity and stroke width than simply linear combinations of pixel intensities. We use the MNIST handwritten image dataset [98], which consists of 60,000 train images and 10,000 test images. We randomly selected 10% (6,000) images from the training set to use for hyperparameter tuning. Each image is a 28x28 matrix of pixels, each representing one of 256 grayscale values. The left and

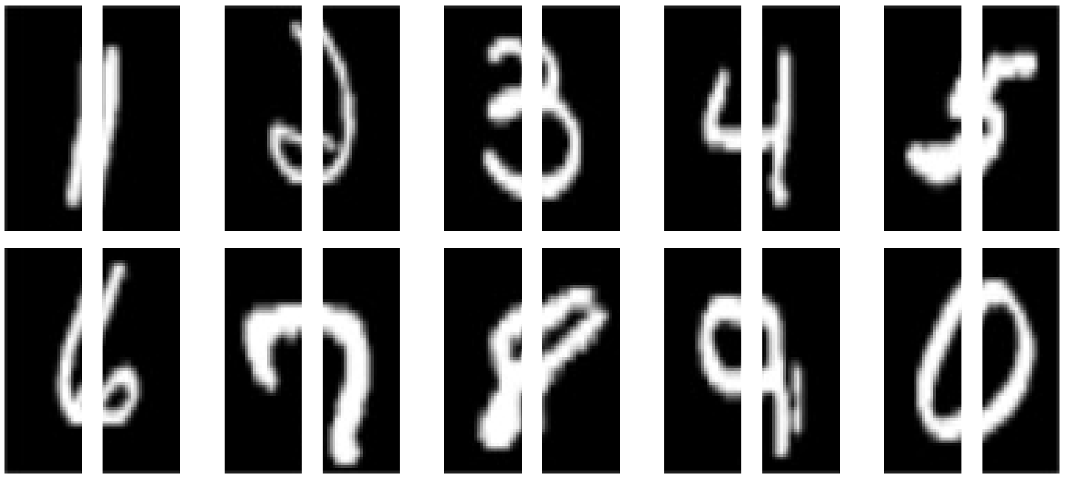


Figure 4.3: Examples of left and right views of split MNIST dataset.

right 14 columns are separated to form the two views, making 392 features in each view. For KCCA, we use a radial basis function (RBF) kernel for both views: $k_1(x_i, x_j) = e^{-\|x_i - x_j\|^2/2\sigma_1^2}$ and similarly for k_2. The bandwidth parameters σ_1, σ_2 are tuned over the range $[0.25, 64]$. Regularization parameters r_1, r_2 for CCA and KCCA are tuned over the range $[10^{-8}, 10]$. The four parameters were jointly tuned to maximize correlation at $k = 50$ on the development set. We use a scalable KCCA algorithm based on incremental SVD [8]. The selected widths of the hidden layers for the DCCA-50-2 model were 2038 (left half-images) and 1608 (right half-images). Table 4.1 compares the total correlation on the development and test sets obtained for the 50 most correlated dimensions with linear CCA, KCCA, and DCCA. DCCA is able to find far greater generalizable correlation between the two halves.

4.5.2 Articulatory speech data

In our next set of experiments, the two views come from completely different sensory modalities, hence we would expect even more value from using deep representations. We use speech data from the Wisconsin X-ray Microbeam Database (XRMB) of simultaneous acoustic and articulatory recordings [196]. The articulatory data consist of horizontal and

	CCA	KCCA (RBF)	DCCA (50-2)
Dev	28.1	33.5	**39.4**
Test	28.0	33.0	**39.7**

Table 4.1: Correlation captured in the 50 most correlated dimensions on the split MNIST dataset.

vertical displacements of eight pellets on the speaker's lips, tongue, and jaws, yielding a 16-dimensional vector at each time point. The baseline acoustic features consist of standard 13-dimensional mel-frequency cepstral coefficients (MFCCs) [44] and their first and second derivatives computed every 10ms over a 25ms window. The articulatory measurements are downsampled to match the MFCC frame rate.

The input features X_1 and X_2 to CCA/KCCA/DCCA are the acoustic and articulatory features concatenated over a 7-frame window around each frame, giving acoustic vectors $X_1 \in \mathbb{R}^{273}$ and articulatory vectors $X_2 \in \mathbb{R}^{112}$. We discard frames that are missing any of the articulatory data (e.g., due to mistracked pellets), resulting in $m \approx 50,000$ frames for each speaker. For KCCA, besides an RBF kernel (described in the previous section) we also use a polynomial kernel of degree d, with $k_1(x_i, x_j) = \left(x_i^T x_j + c\right)^d$ and similarly for k_2.

We run five independent experiments, each using 60% of the utterances for learning projections, 20% for tuning hyperparameters (regularization parameters and kernel bandwidths), and 20% for final testing. For this set of experiments, kernel bandwidths for the RBF kernel were fixed at $\sigma_1 = 4 \times 10^6, \sigma_2 = 2 \times 10^4$ to match the variance in the un-normalized data. For the polynomial kernel we tuned the degree d over the set $\{2, 3\}$ and the offset parameter c over the range $[0.25, 2]$ to optimize development set correlation at $k = 110$. Hyperparameter optimization selected the number of hidden units per layer in the DCCA-50-2 model as 1641 and 1769 for the MFCC and XRMB views respectively. In the DCCA-112-3 model, 1811 and

	CCA	KCCA (RBF)	KCCA (Poly)	DCCA (50-2)
Fold 1	16.8	29.2	32.3	**38.2**
Fold 2	15.8	25.3	29.1	**34.1**
Fold 3	16.9	30.8	34.0	**39.4**
Fold 4	16.6	28.6	32.4	**37.1**
Fold 5	16.2	26.2	29.9	**34.0**

Table 4.2: Correlation captured in the 50 most correlated dimensions on the articulatory dataset.

1280 units per layer, respectively, were chosen. The widths for the DCCA-112-8 model were fixed at 781 and 552 as discussed in the last paragraph of this section.

Table 4.2 compares total correlation captured in the top 50 dimensions on the test data for all five folds with CCA, KCCA with both kernels, and DCCA-50-2. The pattern of performance across the folds is similar for all four models, and DCCA consistently uncovers a greater amount of total correlation.

We next examine how the total correlation changes as the dimensionality of the output representation increases. Figure 4.4 shows correlation obtained using linear CCA, KCCA with RBF kernel, KCCA with a polynomial kernel ($d = 2, c = 1$), and various topologies of deep CCA, on the test set of one of the folds as a function of number of dimensions. The KCCA models tend to detect slightly more correlation in the first few components, after which the deep CCA models outperform them by a large margin. Apparently after finding a few correlated components, CCA and DCCA have difficulty identifying useful new features that are uncorrelated (within each view) to the previous ones. DCCA, on the other hand, can continue to uncover more and more abstract properties that are highly correlated. We note that DCCA may particularly have an advantage when k is equal to the number of output

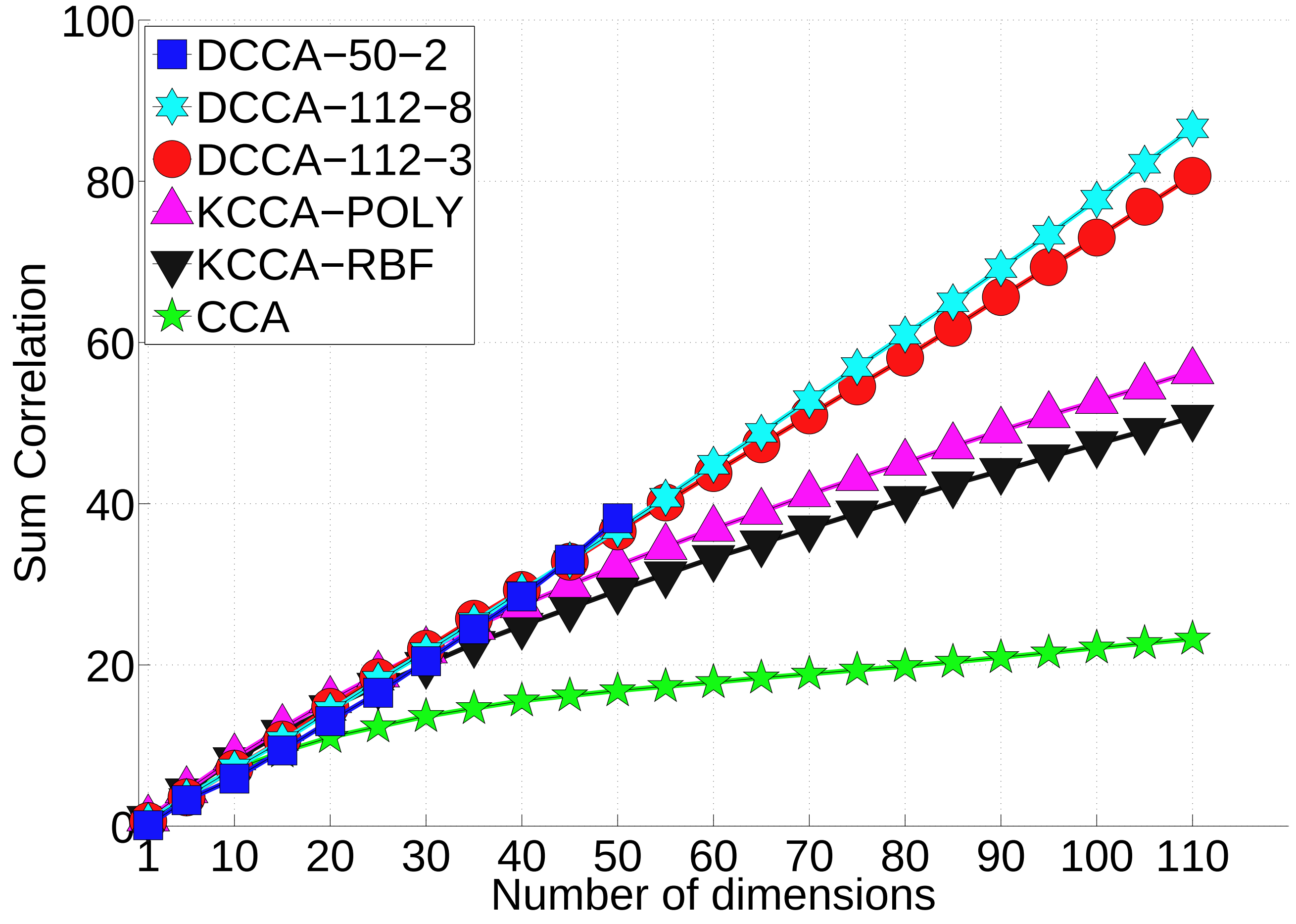


Figure 4.4: Test correlation as a function of number of dimensions. Note that DCCA-50-2 is truncated at $k = o = 50$.

layers (d)	3	4	5	6	7	8
Dev set	66.7	68.1	70.1	72.5	76.0	**79.1**
Test set	80.4	81.9	84.0	86.1	88.5	**88.6**

Table 4.3: Total correlation captured, on one of the folds, by DCCA-112-d, for d ranging from three to eight.

units o. We found that DCCA models with only two or three output units can indeed find more correlation than the top two or three components of KCCA (results not shown). This is also consistent with the observation that DCCA-50-2 has the highest performance of any model at $k = 50$.

Finally, we test the hypothesis that the advantage of DCCA comes from its ability to induce more abstract features. Conceptually, CCA is a single-layer architecture, since its features are formed as simple linear combinations of the input features. KCCA may be considered to be a two-layer architecture, where the kernel computes the first layer features, and the output features are a linear combination of those. In DCCA, however, model depth is a tunable parameter. To determine the impact of model depth on performance, we conducted an experiment in which we increased the number of layers from three to eight, while reducing the number of hidden units in each layer in order to keep the total number of parameters approximately constant. The output width was fixed at 112, and all hyperparameters other than the number of hidden units were kept fixed at the values chosen for DCCA-112-3. Table 4.3 gives the total correlation on the first fold as a function of the number of layers. The total correlation of both datasets increases monotonically with the depth of DCCA, indicating that the multiple levels of nonlinear transformation afforded by deeper representation mappings are indeed helpful.

4.6 Conclusion

We have shown that deep CCA can obtain improved representations with respect to the correlation objective measured on unseen data. Our experiments indicate that when the two views of the data come from different sensory modalities, much of the latent correlation lies in deeper, more abstract properties that are not available to the shallow representation mappings used by CCA or KCCA. Another appealing feature of DCCA as a flexible non-linear alternative to KCCA is that it does not require an inner product. As a parametric model, representations of unseen datapoints can be computed without reference to the training set, and training time should scale asymptotically at most linearly with the size of the training set. We hope DCCA will prove to be another useful method in the toolkit of representation learning methods when multiple views of a dataset are available.

In many applications of CCA, such as classification and regression, maximizing the correlation is not the final goal and the correlated representations are used in the service of another task. A natural next step is therefore to test the representations produced by deep CCA in the context of prediction tasks and to compare against other nonlinear multi-view representation learning approaches that optimize other objectives, e.g., Ngiam et al. [134], Srivastava and Salakhutdinov [172]. Some work has already been done in this direction, for example Wang et al. [193] demonstrate that DCCA features can improve performance of an HMM-GMM phone recognizer.

Chapter 5

LEARNING SPARSELY CONNECTED NEURAL REPRESENTATION MODELS WITH L_1-REGULARIZATION

5.1 Introduction

As the preceding two chapters demonstrate, representation transformations often take the form of parametric functions $x_{\text{new}} = k_\theta(x_{\text{orig}})$. For example, an autoencoder network (or one layer of a deep autoencoder network) performs the mapping $a' = k_\theta(a) = \sigma(Wa + b)$, where σ is a nonlinear activation function and the parameters θ are (W, b). When the function is a deep neural network, the parameters θ are all of the weights and biases of the network. The parameters of such models are often trained to minimize an objective function

$$f(\theta) = \ell(\theta) + r(\theta), \tag{5.1}$$

where ℓ is a loss measure of the mapping on a (possibly labelled) training set, and r is a regularization term that favors "simpler" models. Taking the autoencoder network again as an example, a typical loss function over a dataset D would be the squared reconstruction error, $\ell(\theta) = \sum_{x \in D} ||x - W'k_\theta(x)||_2^2$, while the most common regularizer is probably the weighted squared L_2 norm, $r(\theta) = C||\theta||_2^2$.[1]

It is well-known that the use of a non-trivial regularization term $r(\theta)$ is often necessary to achieve a representation model that generalizes well to unseen data, particularly if the number of parameters is very high relative to the amount of training data. Regularization with a weighted penalty term is a convenient means to tune the complexity of the model to balance flexibility and tendency to overfit (bias/variance). A choice of regularizer that has

[1]The work presented in this chapter was originally published as Andrew and Gao [6]

received increasing attention in recent years is the weighted L_1-norm of the parameters

$$r(\theta) = C\|\theta\|_1 = C\sum_i |\theta_i|$$

for some constant $C > 0$. Popularized in the context of regression by Tibshirani [180] where it is known as the *LASSO* estimator, the L_1 regularizer enjoys several favorable properties compared to other regularizers such as L_2. It has been shown experimentally and theoretically to be capable of learning good models when most features are irrelevant [133]. It also typically produces *sparse* parameter vectors in which many of the parameters are exactly zero, which makes for models that are more interpretable and computationally manageable [106].

An autoencoder or deep neural network with a sparse parameter vector has the intuitively appealing property that each intermediate level feature (each hidden layer activation) depends on a small number of input features. If one is constructing a hierarchy of feature representations at different levels of abstraction (imagine detecting edges from pixels or objects from parts), it seems reasonable that the presence of a feature at one level might only depend on the presence of a few of the very high number of possible features at the previous level. To detect the presence of an oriented edge at a particular location in an image, one may safely ignore most of the pixels of the image outside of a small window.

The L_1 regularizer tends to produce sparse parameters as a consequence of the fact that its first partial derivative with respect to each variable is constant as the variable moves toward zero, "pushing" the value all the way to zero if possible. The partial derivative of the L_2 regularizer, by contrast, diminishes as the parameter value moves toward zero, producing parameters that are close to, but not exactly, zero. Unfortunately, this fact about L_1 also means that the partial derivative is discontinuous at zero, where it jumps from a negative to a positive value, so the objective function cannot be minimized with general-purpose gradient-based optimization algorithms for smooth functions such as the L-BFGS quasi-Newton method [136], which has been shown to be superior at training large-scale L_2-regularized log-linear models by Malouf [113] and Minka [122].

Several special-purpose algorithms have been designed to overcome this difficulty. Perkins and Theiler [141] propose an algorithm called *grafting*, in which variables are added one-at-a-time, and then the weights are reoptimized with respect to the current set of variables. Goodman [57] and Kazama and Tsujii [89] (independently) reframe the problem as a constrained optimization, and solve it with a modification of generalized iterative scaling (GIS) [43] and BLMVM [21], a quasi-Newton algorithm for problems with bound constraints, respectively. Both of these algorithms require doubling the number of variables in the general case.

Lee et al. [105] propose the algorithm *IRLS-LARS*, inspired by Newton's method, which iteratively minimizes the function's second order Taylor expansion, subject to linear constraints. The quadratic program at each iteration is efficiently solved using the LARS algorithm (LASSO variant) of Efron et al. [48]. They compare the approach to the other aforementioned algorithms (except that of Kazama & Tsujii) on small- to medium-scale logistic regression problems, and show that in most cases it is much faster. Unfortunately IRLS-LARS cannot be used to train very large-scale log-linear models involving millions of variables and training instances, such as are commonly encountered, for example, in vision, speech, and natural language processing. Although worst-case bounds are not known, under charitable assumptions the LASSO variant of LARS used in the inner loop of IRLS-LARS may require as many as $\mathcal{O}(mn^2)$ operations, where m is the number of variables and n is the number of training instances. Indeed, the only test problems considered by Lee et al. [105] in which another algorithm approached or surpassed IRLS-LARS were also the largest, with thousands of variables.

In this chapter, we propose a new algorithm based on L-BFGS for training large-scale representation mappings with L_1 regularization, which we call *orthant-wise limited-memory quasi-Newton* (OWL-QN). At each iteration, our algorithm computes a search direction by moving toward the minimum of a quadratic function that models the objective over a set containing the previous point and on which the objective is differentiable (an orthant). It adjusts the search direction and the line search to ensure that it considers only points on the set for which the quadratic approximation is valid. Unlike previous approaches, the algorithm scales well to very high-dimensional training objectives. It is also quite easy to implement:

starting from an implementation of standard L-BFGS, our version of OWL-QN required changing only about 30 lines of code.

First we will develop the algorithm using the simplifying assumption that the loss function $\ell(\theta)$ is convex, and discuss experiments on a large-scale parse reranking task with a convex objective. Then (beginning in section 5.5) we will show how the basic algorithm can be modified to find a local minimum of the regularized objective even for non-convex loss functions such as deep neural networks.

5.1.1 Notation

Let us establish some notation and a few definitions that will be used in the remainder of the chapter. Suppose $f(\theta) = \ell(\theta) + C\|\theta\|_1$ where $\ell : \mathbb{R}^n \mapsto \mathbb{R}$ is a smooth function and $C > 0$. We will let $\partial_i^+ f(\theta)$ denote the right partial derivative of f at θ with respect to θ_i:

$$\partial_i^+ f(\theta) = \lim_{\alpha \downarrow 0} \frac{f(\theta + \alpha e_i) - f(\theta)}{\alpha},$$

where e_i is the i^{th} standard basis vector, with the analogous left variant

$$\partial_i^- f(\theta) = \lim_{\alpha \downarrow 0} \frac{f(\theta) - f(\theta - \alpha e_i)}{\alpha}.$$

Since $\|\cdot\|_1$ is convex, $\partial_i^- f(\theta) \leq \partial_i^+ f(\theta)$, for any θ and i. The *directional derivative* of f at θ in direction $d \in \mathbb{R}^n$ is denoted $f'(\theta; d)$, and is defined as

$$f'(\theta; d) = \lim_{\alpha \downarrow 0} \frac{f(\theta + \alpha d) - f(\theta)}{\alpha}\ .$$

A vector d is referred to as a *descent direction* at θ if $f'(\theta; d) < 0$.

We will also find it convenient to define a few special functions. The *signum* function $\operatorname{sgn} : \mathbb{R} \mapsto \mathbb{R}$ is defined as

$$\operatorname{sgn}(t) \triangleq \begin{cases} -1 & \text{if } t < 0, \\ 0 & \text{if } t = 0, \\ 1 & \text{if } t > 0. \end{cases}$$

The function $\pi : \mathbb{R}^n \mapsto \mathbb{R}^n$ is parameterized by $\xi \in \{-1, 0, 1\}^n$, where

$$\pi_i(\theta; \xi) \triangleq \begin{cases} \theta_i & \text{if } \operatorname{sgn}(\theta_i) = \operatorname{sgn}(\xi_i), \\ 0 & \text{otherwise,} \end{cases} .$$

and can be interpreted as the orthogonal projection of θ onto an orthant defined by ξ. We will use the word *orthant* to refer to any set Ω_ξ for which there exists a sign vector ξ such that

$$\Omega_\xi = \{\theta : \pi(\theta; \xi) = \theta\}.$$

Note that our definition of orthant is slightly more general than the standard definition, which is what would be obtained if we restricted $\xi_i \in \pm 1$. With our definition, if $\xi_i = 0$ then $\theta_i = 0$ for all $\theta \in \Omega_\xi$.

5.2 Quasi-Newton algorithms and L-BFGS

We begin our discussion of OWL-QN with a description of its parent, the L-BFGS quasi-Newton algorithm for unconstrained optimization of a smooth function.

Like Newton's method, *quasi-Newton* algorithms iteratively construct a local quadratic approximation to a function, and then conduct a line search in the direction of the point that minimizes the approximation. If H_k is the (perhaps approximated) Hessian matrix of a smooth function f at the point θ^k, and g^k is the gradient of f at θ^k, the function is locally modelled by

$$Q(\theta) = f(\theta^k) + (\theta - \theta^k)^\top g^k + \frac{1}{2}(\theta - \theta^k)^\top H_k(\theta - \theta^k).$$

If H^k is positive definite, the value θ^* that minimizes Q can be computed analytically according to

$$\theta^* = \theta^k - H_k^{-1} g^k. \tag{5.2}$$

A quasi-Newton method will then explore along the ray $\phi(\alpha) = \theta^k - \alpha H_k^{-1} g^k$ for $\alpha \in (0, \infty)$ to obtain the next point θ^{k+1}.

While pure Newton's method uses the exact second-order Taylor expansion at each point, quasi-Newton algorithms approximate the Hessian using first-order information gathered from previously explored points. L-BFGS, as a *limited-memory* quasi-Newton algorithm, maintains only curvature information from the most recent m points. Specifically, at step k, it records the displacement $s^k \triangleq \theta^k - \theta^{k-1}$ and the change in gradient $y^k \triangleq g^k - g^{k-1}$, discarding the corresponding vectors from iteration $k - m$. It then uses $\{s^i\}$ and $\{y^i\}$ to estimate H_k, or more precisely, to estimate the search direction $-H_k^{-1} g^k$, since the full Hessian matrix (which may be unmanageably large) is not explicitly computed or inverted. The time and memory requirements of the computation are linear in the number of variables. The details of this process are not important for the purposes of this paper, and we refer the interested reader to Nocedal and Wright [136].

5.3 Orthant-wise limited-memory quasi-Newton (OWL-QN) for convex objectives

Our algorithm is motivated by the following observation about the L_1 norm: when restricted to any given orthant, it is differentiable, and in fact is a linear function of its argument. Hence the second-order behavior of the regularized objective f on a given orthant is determined by the loss component alone. This consideration suggests the following strategy: construct a quadratic approximation that is valid for some orthant containing the current point using the inverse Hessian estimated from the loss component alone, then search in the direction of the minimum of the quadratic, restricting the search to the orthant on which the approximation is valid.

5.3.1 Choosing an orthant and search direction

We propose that a natural choice of orthant to explore is the one containing the current point θ^k, and into which the direction of steepest descent of f leads. The complete characterization of the direction of steepest descent and the chosen orthant is as follows. First, the left and right partial derivatives of f are given by

$$\partial_i^{\pm} f(\theta) = \frac{\partial}{\partial \theta_i}\ell(\theta) + \begin{cases} C\,\mathrm{sgn}(\theta_i) & \text{if } \theta_i \neq 0, \\ \pm C & \text{if } \theta_i = 0. \end{cases}$$

The subdifferential at θ is the Cartesian product of intervals $\partial f(\theta) = \times_{i=1}^n J_i(\theta)$, where $J_i(\theta) = [\partial_i^- f(\theta), \partial_i^+ f(\theta)]$. (When $\theta_i \neq 0$, $\partial_i^- f(\theta) = \partial_i^+ f(\theta)$, so $J_i(\theta)$ contains a single point.) For any direction d, the directional derivative is[2]

$$f'(\theta; d) = \sup_{g \in \partial f(\theta)} g'd = \sum_i \begin{cases} \partial_i^- f(\theta) v_i & \text{if } v_i < 0, \\ \partial_i^+ f(\theta) v_i & \text{if } v_i > 0, \\ 0 & \text{if } v_i = 0. \end{cases}$$

The direction of steepest descent at θ, which we will denote as $\Diamond f(\theta)$, is then

$$\Diamond_i f(\theta) = \begin{cases} -\partial_i^- f(\theta) & \text{if } \partial_i^- f(\theta) > 0, \\ -\partial_i^+ f(\theta) & \text{if } \partial_i^+ f(\theta) < 0, \\ 0 & \text{if } \partial^- f(\theta) \leq 0 \leq \partial^+ f(\theta)\ . \end{cases} \tag{5.3}$$

Note that $\Diamond f(\theta) = 0$ if and only if $0 \in J_i(\theta)$ for all i, hence $0 \in \partial f(\theta)$, which means that θ is the global optimum. Now supposing $\Diamond f(\theta^k) \neq 0$, define the sign vector $\xi^k \in \{-1, 0, 1\}^n$ according to

$$\xi_i^k = \begin{cases} -1 & \text{if } \theta_i^k < 0, \text{ or } \theta_i^k = 0 \text{ and } \Diamond_i f(\theta^k) < 0, \\ 1 & \text{if } \theta_i^k > 0, \text{ or } \theta_i^k = 0 \text{ and } \Diamond_i f(\theta^k) > 0, \\ 0 & \text{if } \theta_i^k = 0 \text{ and } v_i^k = 0. \end{cases}$$

Then the set of points explored by OWL-QN at iteration k is the orthant

$$\Omega^k \triangleq \{\theta \in \mathbb{R}^n : \pi(\theta; \xi^k) = \theta\}.$$

For all $\theta \in \Omega^k$, $f(\theta) = \ell(\theta) + C\theta'\xi^k$. Furthermore, this definition of Ω^k means that for all directions q^k within Ω^k (i.e., such that $\theta^k + \alpha q^k \in \Omega^k$ for some $\alpha > 0$) the directional derivative

[2]This formula for the directional derivative of a convex function comes from Boyd et al. [29].

is given by $f'(\theta^k; q^k) = -(q^k)'\Diamond f(\theta^k)$. So $\Diamond f(\theta^k)$ plays precisely the role of the negative gradient of a differentiable function. Now using H_k^{-1}, the L-BFGS approximation to the inverse Hessian of the loss, and $\Diamond f(\theta^k)$, the direction of steepest descent of f at θ^k, we can approximate f on Ω^k with a quadratic function Q^k as in (5.2), and search in the direction of the minimum of Q^k, which is given by $p^k = H_k^{-1}\Diamond f(\theta^k)$.

5.3.2 Constrained line search for convex functions

During the line search, in order to ensure that we do not leave the region on which Q^k is valid, we project each explored point orthogonally back on to the set Ω^k. That is, we explore points

$$\theta^{k+1} = \pi(\theta^k + \alpha p^k; \xi^k),$$

which amounts to setting to zero any coordinate that moves from positive to negative or vice-versa. While our line search does not allow any coordinate to cross zero, note that it is still possible for a coordinate to change sign in two iterations, by moving first from a negative value (say) to zero, and then moving to a positive value on a subsequent iteration.

Any number of methods could be used to choose α, but for convex loss functions it is sufficient to use a simple backtracking line search designed to satisfy the a variant of the Armijo condition. For fixed constants $\beta, \gamma \in (0, 1)$ and for $n = 0, 1, 2, \ldots$, we accept the first step size $\alpha = \beta^n$ such that

$$f(\theta^{k+1}) \leq f(\theta^k) - \gamma(\theta^{k+1} - \theta^k)'\Diamond f(\theta^k). \tag{5.4}$$

This line search exactly generalizes the familiar Armijio condition for a differentiable objective

$$f(\theta^{k+1}) \leq f(\theta^k) + \gamma\alpha\nabla f(\theta^k)'p^k$$

in the following way. Defining the *effective direction* $q^k = \frac{1}{\alpha}(\theta^{k+1} - \theta^k)$ so that $\theta^{k+1} = \theta^k + \alpha q^k$, our condition can be rewritten as

$$f(\theta^{k+1}) \leq f(\theta^k) - \gamma\alpha(q^k)'\Diamond f(\theta^k).$$

```
Choose initial point $\theta^0$
$S \Leftarrow \{\}, Y \Leftarrow \{\}$
for $k = 0$ to MaxIters do
  Compute steepest descent direction $\lozenge f(\theta^k)$                (1)
  Compute $p^k \Leftarrow H_k^{-1} \lozenge f(\theta^k)$ using $S$ and $Y$
  Find $\theta^{k+1}$ with constrained line search                         (2)
  if termination condition satisfied then
    Stop with $\theta^* = \theta^{k+1}$
  end if
  Update $S$ with $s^k = \theta^{k+1} - \theta^k$
  Update $Y$ with $y^k = \nabla\ell(\theta^{k+1}) - \nabla\ell(\theta^k)$   (3)
end for
```

Figure 5.1: Algorithmic description of OWL-QN

In the standard Armijo condition for a true line search on a differentiable function, $\nabla f(\theta^k)' p^k = f'(\theta^k; p^k)$ is the directional derivative at θ^k in the search direction p^k, while in ours $-(q^k)' \lozenge f(\theta^k) = f'(\theta^k; q^k)$ is the directional derivative at θ^k in the effective search direction q^k. In appendix C we prove that this line search is guaranteed to terminate in a finite number of steps.

L-BFGS requires that at every step k, it holds that

$$(s^k)' y^k = (\theta^{k+1} - \theta^k)'(\nabla\ell(\theta^{k+1}) - \nabla\ell(\theta^k)) > 0$$

in order to ensure that the update is stable and the Hessian approximation H_k is positive definite. This is known as the *curvature condition*. For strongly convex objectives, this condition is always satisfied when s^k and y^k are generated by any two points θ^k, θ^{k+1}. In section 5.5 we will discuss how a more sophisticated line search can ensure the curvature condition holds for non-convex objectives.

A pseudo-code description of OWL-QN is given in Algorithm (5.1). In fact, only a few steps of the standard L-BFGS algorithm have been changed. The only differences have been marked in the figure:

1. The steepest descent vector $\diamondsuit f(\theta^k)$ of the regularized objective is used in place of the negative gradient in (5.2).

2. During the line search, each search point is projected onto the chosen orthant, by zeroing out coordinates that change sign.

3. The gradient of the unregularized loss alone is used to construct the vectors y^k used to approximate the Hessian.

Starting with an implementation of L-BFGS, altering it to implement OWL-QN requires changing only about 30 lines of code.

5.4 Experiments with convex loss

We evaluated the algorithm OWL-QN on the task of training conditional log-linear model for parse reranking. Following Collins [40], the setup is as follows. We are given:

- a procedure $\texttt{GEN}$ that generates a nonempty set of candidate parses $\texttt{GEN}(x) \subseteq Y$ for each sentence $x \in X$.

- training samples (x_i, y_i) for $i = 1 \dots m$, where $x_i \in X$ is a sentence, and $y_i \in \texttt{GEN}(x_i)$ is the gold-standard parse for that sentence, and

- a feature mapping $\Phi : X \times Y \mapsto \mathbb{R}^n$, which maps each pair (x, y) to a vector of feature values.

For any parameter vector $w \in \mathbb{R}^n$ and sentence x, we define a distribution over parses in $\texttt{GEN}(x)$ according to the log-linear model

$$P_w(y|x) = \frac{\exp w'\Phi(x,y)}{\sum_{y' \in \texttt{GEN}(x)} \exp w'\Phi(x,y')}.$$

Our task is to minimize

$$\ell(w) + C||w||_1,$$

where the loss term $\ell(w)$ is the negative conditional log-likelihood of the training data:

$$\ell(w) = -\sum_i P(y_i|x_i).$$

We follow the experimental paradigm of parse reranking outlined by Charniak and Johnson [32]. We use the same generative baseline model for generating candidate parses, and nearly the same feature set, which includes the log probability of the parse according to the baseline model plus 1,219,272 additional features. We optimized the model parameters on sections 2-19 of the Penn Treebank, used sections 20-21 to select the regularization weight C, and finally evaluated the models on section 22.[3] The training set contains 36K sentences, while the held-out set and the test set have 4K and 1.7K, respectively.

We compared OWL-QN to a fast implementation of the only other special-purpose algorithm for L_1 regularized minimization of which we are aware that can feasibly run at this scale: that of Kazama and Tsujii [89], hereafter called "K&T". In K&T, each weight w_i is represented as the difference of two values: $w_i = w_i^+ - w_i^-$, with $w_i^+ \geq 0, w_i^- \geq 0$. The L_1 penalty term then becomes simply $||w||_1 = \sum_i w_i^+ + w_i^-$. Thus, at the cost of doubling the number of parameters, we have a constrained optimization problem with a differentiable objective that can be solved with general-purpose numerical optimization software. In our experiments, we used the AlgLib implementation of the L-BFGS-B algorithm of Byrd et al. [30], which is a C++ port of the FORTRAN code by Zhu et al. [202].[4] We also ran two implementations of L-BFGS (AlgLib's and our own, on which our implementation of OWL-QN is based) on the L_2-regularized problem, to show that they are compatible in terms of performance.

[3]Since we are not interested in parsing performance *per se*, we did not evaluate on the standard test corpus used in the parsing literature (section 23).

[4]The original FORTRAN implementation can be found at `http://users.iems.northwestern.edu/~nocedal/lbfgsb.html`, while the AlgLib C++ port is available at `www.alglib.net`.

We used the memory parameter $m = 5$ for all four algorithms. For the timing results, we first ran each algorithm until the relative change in the function value, averaged over the previous five iterations, dropped below $\tau = 10^{-5}$. We report the CPU time and the number of function evaluations required for each algorithm to reach within 1% of the smallest value found by either of the two algorithms. We also report the number of function evaluations so that the algorithms can be compared in an implementation-independent way, as typically function evaluation is by far the most expensive operation.

Although we are primarily interested in the efficiency of training, we first report the performance of the learned parse models, to demonstrate that the models are competitive. Performance is measured with the *PARSEVAL* metric, i.e., the F-score over labeled brackets. These results are summarized in Table 5.1. In the table, "Baseline" refers to the generative model used by `GEN`. "Oracle" shows the topline of ideal performance if the best parse from `GEN` (according to F-score) were always selected by the re-ranking model. Both types of model performed significantly better than the baseline, and may indeed be considered close to the state-of-the-art. For comparison, the model of Charniak and Johnson [32] also achieved 91.6% F-score on the same test set.[5] Interestingly, the two regularizers performed almost identically. However, as we will see, the trained L_1 model was much sparser and in fact trained more quickly.

Figure 5.2 shows the F-scores of the the trained models as the regularization constant C is varied by an order of magnitude. Both types of regularizer have a fairly broad region of near optimal performance on this task, although it seems L_2 is slightly more forgiving of deviations from the maximum.

The results of CPU timing experiments using the same values of C are shown in Table 5.2. We stopped K&T after 946 iterations when it had reached the value 7.34×10^4, still 5.7% higher than the best value found by OWL-QN. The difference in both runtime and number of function evaluations between K&T and OWL-QN is quite dramatic. Perhaps

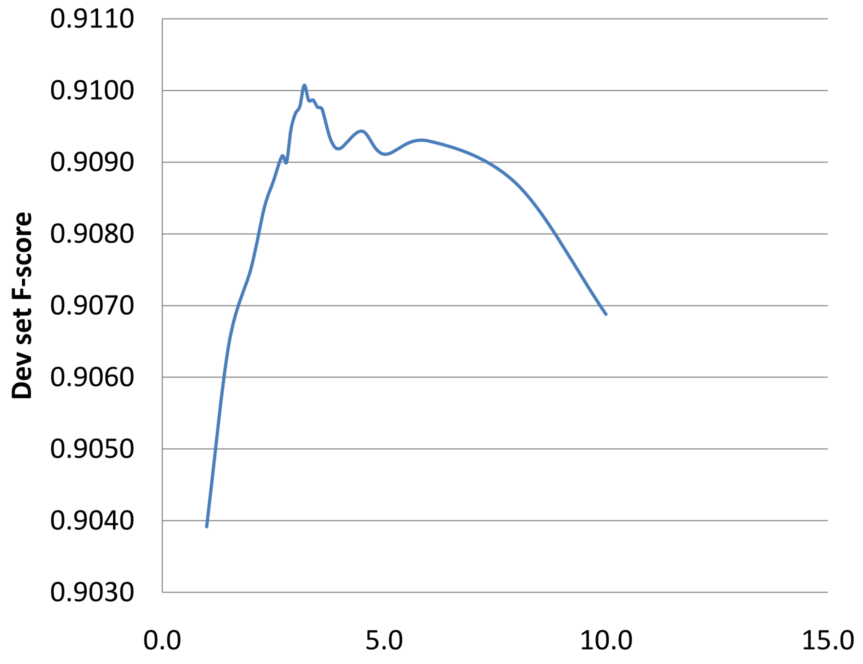
[5]Mark Johnson, personal communication, May 2007.

	C	dev	test
Baseline		0.8925	0.8986
Oracle		0.9632	0.9687
L_2 (L-BFGS)	13.0	0.9103	0.9163
L_1 (OWL-QN)	3.2	0.9101	0.9165

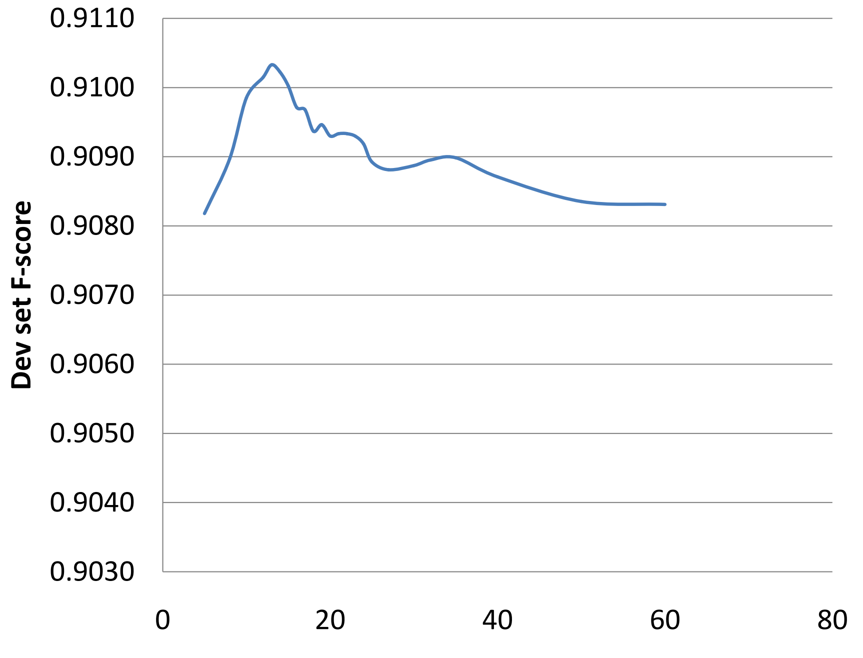
Table 5.1: Chosen value of C and F-scores of the models used in the study.

	func evals	func eval time	L-BFGS dir time	other time	total time
OWL-QN	54	707 (97.7)	10.4 (1.4)	6.9 (1.0)	724
K&T (AlgLib)	>946	16043 (91.2)	1555 (8.8)		>17598
L-BFGS (our impl.)	109	1400 (97.7)	22.4 (1.5)	10 (0.7)	1433
L-BFGS (AlgLib)	107	1384 (83.4)	276 (16.6)		1660

Table 5.2: CPU time and function evaluations to reach within 1% of the best value found by any algorithm. All times are in seconds. Values in parentheses show percentages of total time.



(a) With L_1 regularization.



(b) With L_2 regularization.

Figure 5.2: Dev set performance of L_1- and L_2-regularized models on parse reranking task as a function of regularization weight C.

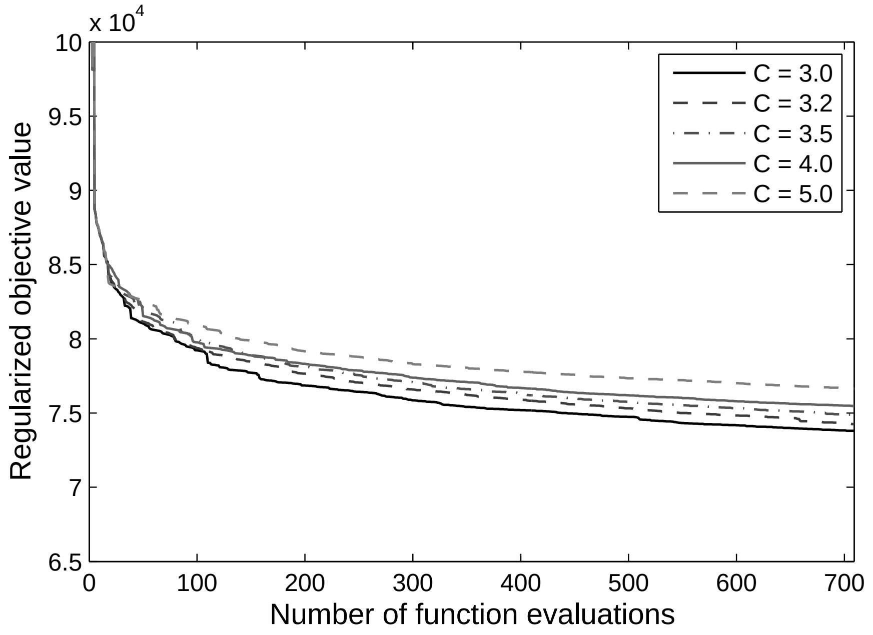
surprisingly, OWL-QN converges even faster than our implementation of L-BFGS run on the L_2-regularized objective. Note also that the runtime of all algorithms is dominated by evaluations of the objective function, and that otherwise the most expensive step of OWL-QN is the computation of the L-BFGS direction, an operation that is shared with L-BFGS. The operations that are unique to OWL-QN, specifically the orthant projection and computation of the steepest descent direction, took an almost negligible amount of time.

A more complete picture of the training efficiency of the two models can be gleaned by plotting the value of the objective as a function of the number of function calls, shown in figure 5.3. Comparing OWL-QN to K&T in figure 5.3b to 5.3a we see that OWL-QN reaches a near optimal solution very quickly while K&T does not appear to have converged after being allowed far greater run time.

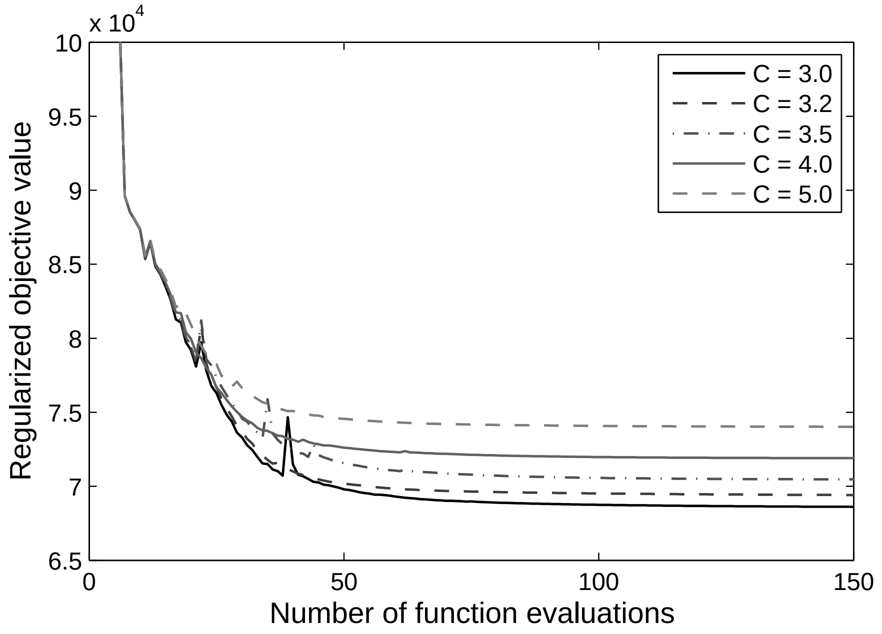
Since our experiments on OWL-QN and K&T are based on different implementations of L-BFGS, it is important to establish that the observed differences in performance are due to the algorithms themselves and not to their differing implementations. We can do this by comparing the performance of the two implementations on the L_2-regularized version of the objective to ensure they are comparable, shown in figure 5.4. Comparing figures 5.4b and 5.4a, we see that our own implementation of L-BFGS is very closely comparable to AlgLib's implementation. Therefore the differences between 5.3b amd 5.3a can safely be attributed to the algorithmic differences between OWL-QN and K&T, not to significant differences in the efficiency of our own L-BFGS implementation compared to AlgLib's.

Since its ability to learn a sparse parameter vector is an important advantage of the L_1 regularizer, we examine how the number of non-zero weights changes during the course of optimization in figure 5.5. Again, note the change in x-axis. Both algorithms start with a significant fraction of features (5%-12%) and prune them away as the algorithm progresses, with OWL-QN producing a sparse model rather more quickly. It is clear from this figure that OWL-QN not only finds a near optimal solution very quickly, but it also prunes away the vast majority of unneeded features.

Although here we have focused on the runtime behavior of OWL-QN for a single convex

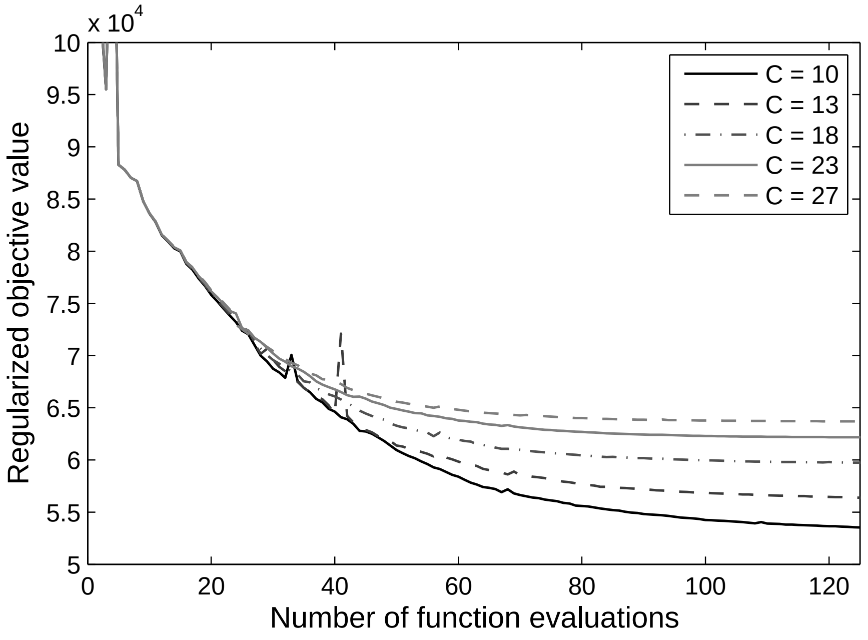


(a) K&T

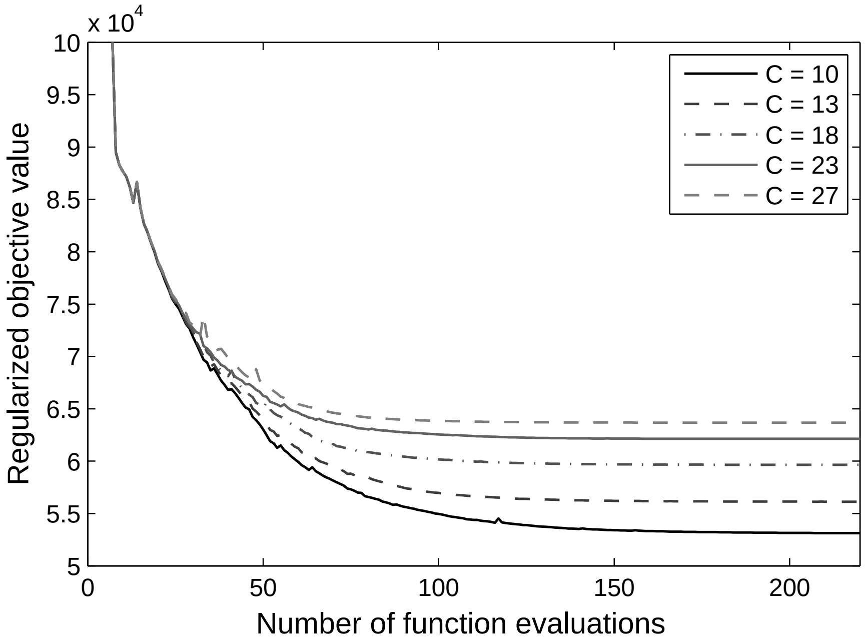


(b) OWL-QN

Figure 5.3: L_1-regularized objective value during optimization with K&T and OWL-QN for various values of regularization weight C. Note the difference in scale of the x-axis.

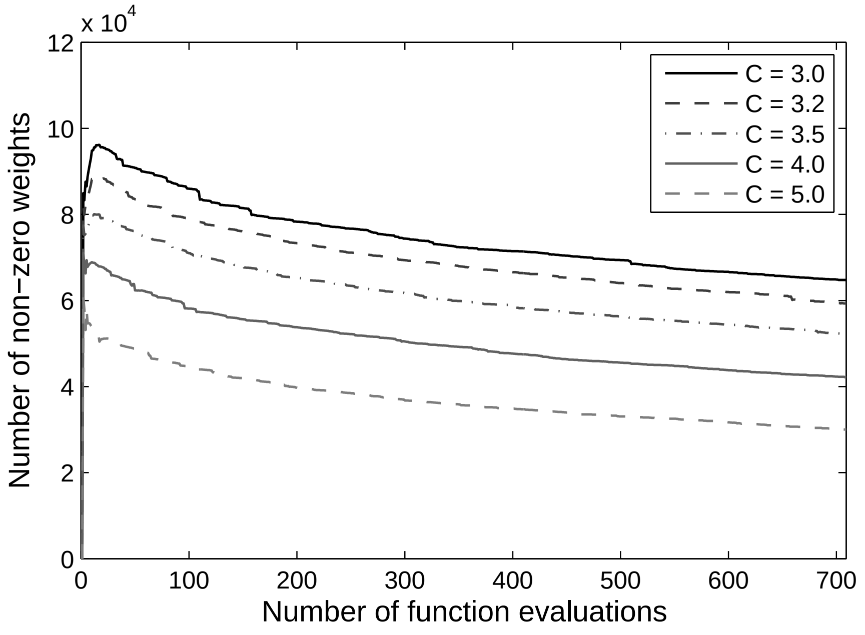


(a) L-BFGS (AlgLib's implementation)

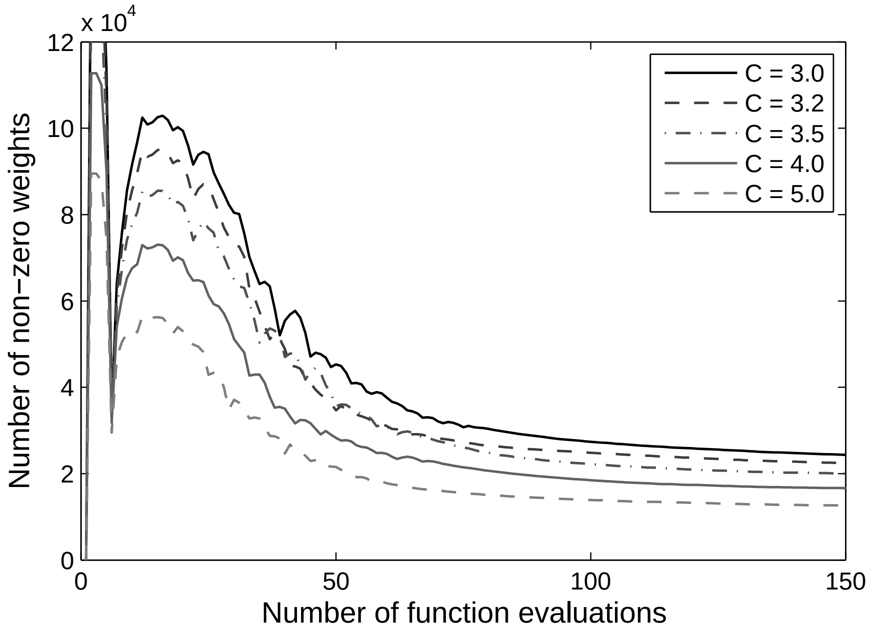


(b) L-BFGS (our own implementation)

Figure 5.4: L_2-regularized objective value during optimization with L-BFGS, comparing our own implementation to AlgLib's.



(a) K&T



(b) OWL-QN

Figure 5.5: Number of non-zero weights during optimization with K&T vs. OWL-QN for various values of regularization weight C. Note the difference in scale of the x-axis.

problem, we have also used it to train L_1-regularized log-linear models with up to ten million variables for a variety of other problems in NLP. In Gao et al. [53] we applied OWL-QN to the problems of part-of-speech tagging, Chinese word segmentation and language modeling. In all cases, OWL-QN trained faster than L-BFGS on the L_2-regularized problem, ended up with only a small fraction of the available features, and achieved nearly identical evaluation performance.

5.5 *OWL-QN for non-convex loss functions*

Many loss functions of interest in representation learning, in particular the error of a deep neural network, are very far from convex. However an extension of the basic OWL-QN algorithm allows it to run on instances where the loss function ℓ is non-convex. The issue, as discussed in section 5.3.1, is that in order for the L-BFGS update to be stable and produce a positive-definite Hessian approximation H_k, it must be the case at every iteration that

$$(s^k)'y^k = (\theta^{k+1} - \theta^k)'(\nabla\ell(\theta^{k+1}) - \nabla\ell(\theta^k)) > 0. \tag{5.5}$$

This holds trivially when ℓ is strongly convex, but if ℓ is non-convex we must perform a more careful line search to ensure that it holds.

Whereas our backtracking line search guaranteed only the sufficient decrease condition (5.6) below, we will now seek a point θ^{k+1} that also satisfies the curvature condition (5.7):

$$f(\theta^{k+1}) \leq f(\theta^k) - \gamma_1\alpha(q^k)'\Diamond f(\theta^k) \tag{5.6}$$

$$|(q^k)'(\nabla\ell(\theta^{k+1}) + C\xi^k)| \leq \gamma_2(q^k)'\Diamond f(\theta^k), \tag{5.7}$$

where again $q^k = \frac{1}{\alpha}(\theta^{k+1} - \theta^k)$ is the "effective search direction", and $\gamma_1 \in (0,1)$ and $\gamma_2 \in (\gamma_1, 1)$ are constants. Since q^k is a descent direction within Ω^k, (5.7) implies that

$$\begin{aligned}(q^k)'(\nabla\ell(\theta^{k+1}) + C\xi^k) &\geq \gamma_2(q^k)'(\nabla\ell(\theta^k) + C\xi^k) \\ &> (q^k)'(\nabla\ell(\theta^k) + C\xi^k),\end{aligned}$$

and therefore $(q^k)'\nabla\ell(\theta^{k+1}) > (q^k)'\nabla\ell(\theta^k)$, implying (5.5). Conditions (5.6) and (5.7) are analogous to the strong Wolfe conditions for smooth optimization. Nocedal and Wright [136] describe a line search for the Wolfe conditions that can be used to satisfy our modified conditions. The algorithm is somewhat involved so we will not reproduce it here. To use it, it is only necessary to substitute $-(q^k)'\Diamond f(\theta^k)$ in place of $(p^k)'\nabla f(\theta^k)$, and $(q^k)'(\nabla\ell(\theta^{k+1})+C\xi^k)$ in place of $(p^k)'\nabla f(\theta^{k+1})$.

5.6 Experiments with non-convex loss

We demonstrate the effectiveness of OWL-QN on a non-convex objective by optimizing the loss of a deep feedforward neural network. The network is trained on the MNIST dataset to minimize the softmax error. We use three hidden layers with 1024 hidden units each. The activation function of all hidden units is the modified cube-root function described in section 4.4.1. Both L_2 and L_1 regularization were applied by adding the squared L_2 penalty to the loss ℓ and treating the L_1 penalty with OWL-QN.[6] The network was pretrained with a denoising autoencoder, including dropout noise in the encoding layer, using stochastic gradient descent over minibatches of size 64 for ten epochs. After fine-tuning, the error of the resulting classifier was surprisingly insensitive to the weights of the regularizers within a range of two orders of magnitude. Therefore our results are based on the values that produced the maximal parameter sparsity.

In figure 5.6 we show the number of errors on the test set during training. Our purpose is to demonstrate that OWL-QN can optimize a highly non-convex function, not to achieve state-of-the-art results on MNIST. Nevertheless our simple multilayer perceptron trained with OWL-QN achieves a final error rate of around 2%, which is reasonable for a baseline classifier.

More interesting is the degree of sparsity shown to be achievable. Although classification performance remains essentially constant after about 200 epochs, the number of active

[6] The sum of squared-L_2 and L_1 penalties is known as the “elastic net” regularizer [203].

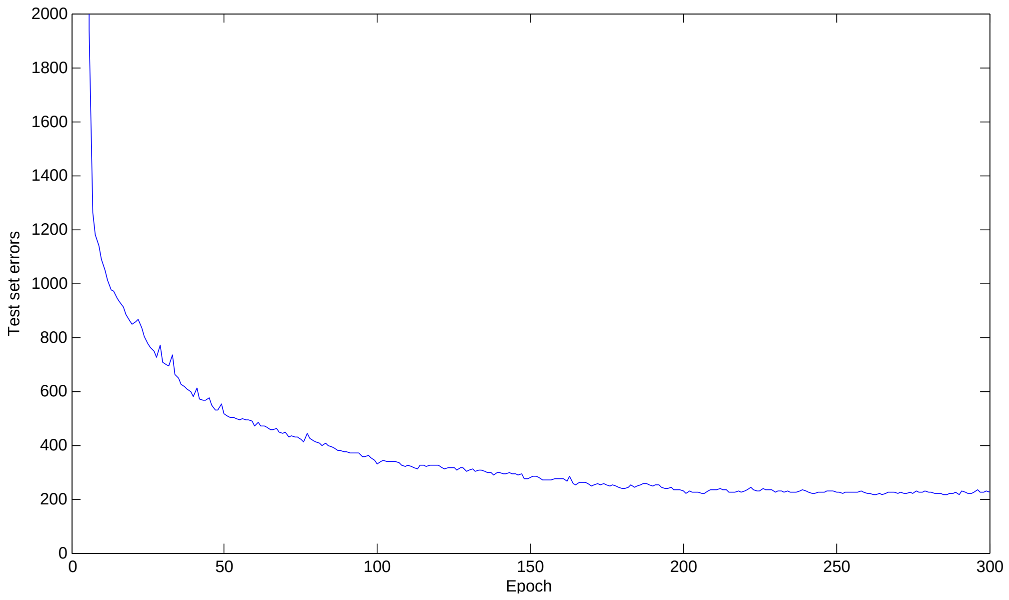
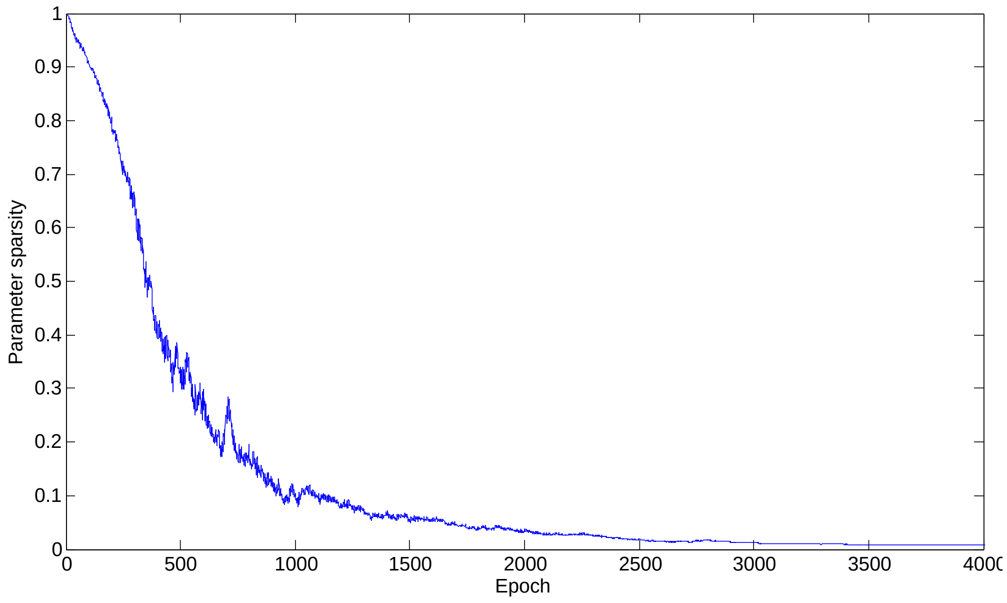


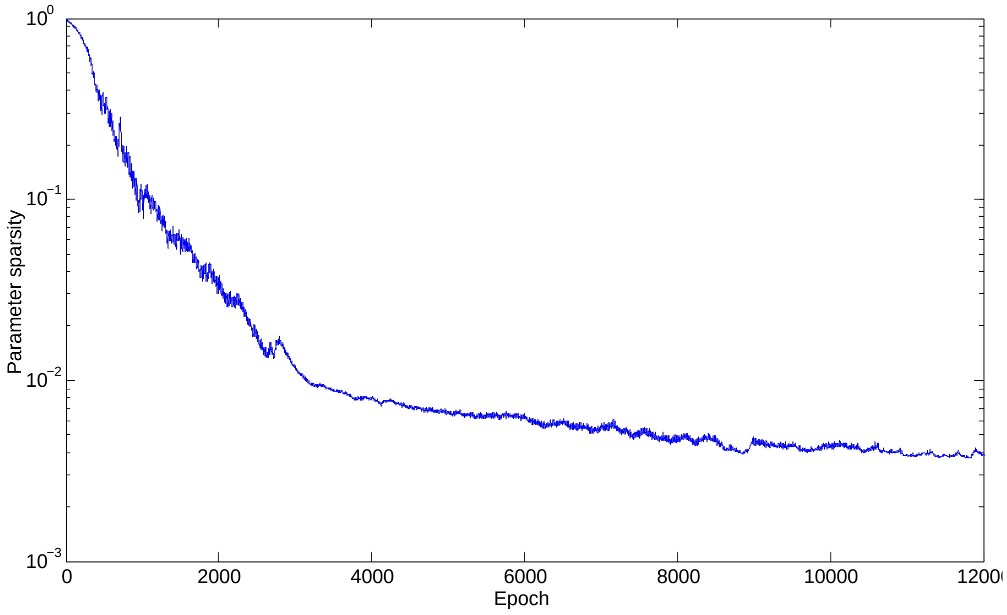
Figure 5.6: MNIST test set errors during training with OWL-QN.

synapses is reduced dramatically as training continues. Ultimately the algorithm learns a competitive classifier that uses only about 0.4% of the available connections. Astonishingly, this means that *each unit only receives input from about four others*, on average. Figure 5.7 shows the number of non-zero weights during training, with linear- and log-scaled y-axes to illustrate performance both at the beginning and in the tail. After 1000 epochs, only 10% of the connections remain, and by 3000 only 1% remain. Further gains are slow, but possible given enough training time. The sparsity bottoms out at 0.36% after around 12000 function evaluations.

Although the L_1 regularization did not appear to yield gains in classification accuracy on this task, the remarkable degree of sparsity may be useful in its own right. The resulting model has been compressed to a tiny fraction of the original size, which may be of significant utility when storage space is an issue as in with mobile devices. Also, if a sparse representation of the MLP weight matrices were employed, classification could be performed with significantly less computation. This property could be important to enable such networks to run on low-power



(a) Sparisty (linear y-axis)



(b) Sparisty (log y-axis)

Figure 5.7: Number of non-zero weights during training of classifier for MNIST. Plots with both linear and logarithmic y-axes are shown to observe behavior early in optimization as well as in the tail.

portable devices.

5.7 Conclusions

We have presented an algorithm OWL-QN for efficiently training L_1-regularized representation learning models with up to millions of variables. We tested the algorithm on several very large-scale NLP tasks, and found that it was considerably faster than an alternative algorithm for L_1 regularization, and even somewhat faster than L-BFGS on the analogous L_2-regularized problem. An extension of the algorithm allows it to run when the objective is non-convex. Our experiments with a multilayer perceptron classifier showed that it can find a competitive parameter vector that uses only a miniscule fraction of the weights.

Since its publication [6], OWL-QN has been widely verified as among the fastest existing algorithms for large-scale L_1-regularized convex optimization [53, 156, 157, 200]. New algorithms have also been developed that may rival OWL-QN on very large problems. In particular, accelerated gradient algorithms for non-smooth optimization [132] seem very promising, as well as proximal quasi-Newton algorithms [17, 103]. For an overview of some of these methods (including many others that probably would not scale to very large problems, like coordinate-descent-type algorithms), see Bach et al. [13]. Becker and Fadili [17] do conduct an empirical comparison with OWL-QN on two medium-scale (thousands of variables) problems and find their algorithm to be faster than OWL-QN on one of them. However, an empirical comparison using large-scale (millions of variables) objectives has not yet been performed. Also we are not aware of any newer algorithms that have been used to find a local minimum of a large scale *non-convex* objective with L_1 regularization, as we have done with OWL-QN.

In future work, it would be valuable to perform a careful empirical comparison of OWL-QN to newer algorithms on real large-scale problems of practical interest. Another important contribution would be to prove asymptotic convergence of the non-convex formulation of OWL-QN to a local minimum. Unfortunately we are not optimistic that any useful result can be proved regarding the *rate of convergence* of OWL-QN, since even L-BFGS, a thirty-five year old algorithm, is not known to converge any faster than gradient descent, although in practice it is much more efficient.

Chapter 6
CONCLUSIONS

The past decade (spanning 2006–2016) has seen an explosion in the number of proposed variations on deep neural network architectures and training methods, starting with the first effective greedy layerwise generative pretraining methods in 2006, and continuing through the development of specialized architectures for images, audio, and text, as well as minor modifications that can dramatically improve performance, such as dropout training and the use of rectified linear units. In this thesis I have attempted to contribute to this research agenda by describing and justifying several novel techniques to advance the state of the art in deep representation learning.

In the conclusions to chapters 3–5, I discussed some possible extensions and future research directions. Meanwhile, other researchers have already begun to extend the work presented here and apply it in novel scenarios. DCCA was recently used to learn word embeddings of multilingual data [111], and applied to images with captions (including convolutional layers for the images) for caption ranking [198]. Wang et al. [193] describe an extension of DCCA called *deep canonically correlated autoencoder* that attempts to learn representations of two views that are not only correlated, but also retain information about their input, via an autoencoder term in the training objective. This may make the learned representations more useful for some tasks. A stochastic optimization procedure for DCCA, alleviating its large memory requirements, was recently described [194].

Orthant-wise quasi-Newton optimization has been widely verified as among the fastest existing algorithms for large-scale L_1-regularized convex optimization [53, 156, 157, 200] and used to learn sparse representation models for many different tasks (although not as yet for models with non-convex objectives like the MLP used in chapter 5, since the published paper describing OWL-QN presented only the version for convex objectives) [139, 155, 143, 54, 184]. It has also been extended to L_p pseudo-norm regularization for $p < 1$ [93].

6.1 The future of representation learning

It is my intention and hope that the work described here will continue to be tested on new tasks and in combination with other advancements in the field and perhaps inspire new directions toward more effective deep representations for hard AI tasks. At the same time, I believe that the most important developments in the near future of representation learning will come from an area that I did not have the opportunity to address in this thesis: reinforcement learning.

The majority of work in the resurgence of neural networks of the last decade has focused on *static* tasks, where data such as images or sentences are presented to the learning algorithm in discrete, disconnected units. Even networks operating on data with a temporal component, such as audio, typically operate on an entire clip at once (like our SDNN) rather than processing audio continuously in real time. In contrast, humans presumably maintain an evolving representation through time. The major exception is game playing, whether video game playing [123, 124, 61] or board games like *go* [162], in which deep reinforcement learning is already effectively used to determine actions in sequence. Very recently, similar approaches have been extended to continuous control tasks [108, 15, 67]. Part of what makes processing in continuous time difficult is that it is not sufficient to make decisions independently: one must also take into account the effect of current decisions on future outcomes. Two problems that are considered difficult for classical reinforcement learning are *credit assignment* (determining which of all past decisions is responsible for the current reward) and the *exploration vs. exploitation tradeoff* (should I do what worked well before, or try something new that may be better?). Similar issues confront a representation learning module that is expected to operate in real time. Which aspects of past representations could have predicted the current reward or informed the current choice of action? Is my current representation mapping sufficient to make optimal decisions, or should I attempt to model other aspects of the situation that might turn out to be useful?

Reinforcement learning has a rich history and current developments are occurring at a rapid pace; I could not possibly hope to give an adequate survey of the area in this conclusion. Instead I would like to argue at a high level why I believe neural systems operating in real time are the future of deep representation learning.

It is clear that real-time decision making systems are necessary to solve real-time tasks like robot control and game playing. But it is my belief that even tasks that have so far been addressed as static, such as image and speech recognition, will be ultimately solved by real-time systems that make sequential decisions with an evolving representation. Steps in this direction have already been taken, for example the machine translation system of Bahdanau et al. [14] (described earlier in section 2.2), which makes sequential decisions about which tokens of the target language to generate based on an attentional mechanism that is allowed to skip around the source sentence. Humans performing almost any complex task to which machine learning has been applied certainly employ a temporally evolving representation.

Take for example image recognition: upon presentation, a human will scan around the image, focusing on various salient areas, where earlier impressions impact the trajectory taken by the attentional mechanism in service of the goal to understand and describe the scene. If something about the scene is confusing, further processing is directed toward resolution of the ambiguity. If we imagine the task to be labeling the image, the final answer might be given when some threshold of confidence has been reached or further scanning is not revealing any helpful changes to the representation. This is all in strong contrast to current state-of-the-art static convolutional neural networks for image recognition, which compute a precisely defined function in a constant amount of time mapping the image to a label.

It is true that regarding some tasks with a temporal component that have so far been addressed with static methods, like speech recognition, directed attentional scanning may not be as important for humans. Even so, it is clear that chopping the input into discrete units (utterances, in the case of speech) that have no relationship to each other misses important dependencies that must be incorporated for optimal results. Humans understand speech or video in a continuous way, with earlier representations having a major influence on later representations, via attention, or association, or otherwise. Removing long-term temporal dependencies from consideration is a major simplifying assumption. It is clear that at some point systems for such tasks must add time back into consideration for machine learning systems to rival human abilities.

What would a representation learning module that operates in real-time look like? Some contenders have already been proposed, for example recurrent neural networks, or elaborations like the long short-term memory RNN. Sutskever et al. [176] use RNNs to generate text, maintaining an evolving neural representation, although the model cannot truly be said to operate in continuous time as it reads and generates one character of text per frame. *Spiking* neural models, which can be trained using *spike-timing-dependent plasticity*, a model of learning in biological neurons [165] are another candidate. Representation models using spiking neurons are already beginning to show promise for tasks like image recognition where static deep networks are currently the best performing models [137, 179, 24]. In any case, following the success of generative pretraining strategies for learning static representations, I believe the most promising general approach for learning dynamic representations would involve generatively modeling the input stream, producing representations that better enable the model to predict upcoming values. In the case of a hierarchical representation, this would mean having higher level representations that are useful in predicting future values of the lower level representation. In a reinforcement learning context, there may be a tradeoff between representations that help to predict the input and those that help to choose optimal actions. It could also be the case that a generative approach alone will yield a sufficiently rich representation to make good decisions, particularly if the reward signal and the agent's own action sequence are also generatively modeled along with the observations.

In my emphasis on real-time learning systems for the future I don't mean to imply that there is no need for further research on static models. First of all, given the great difficulty of reinforcement learning on complex tasks and many other challenges ahead, it may be many years before true temporally evolving representation learning models outperform static models, and so the latter will continue to be useful in the interim. Also methods developed for static learning tasks may still be useful as a component in time-dynamic systems. For example, DeepMind's champion AlphaGo *go* playing system uses static deep neural networks to evaluate board position and propose moves [162].

To summarize, I believe further research into static deep representation learning systems is necessary to improve on recent breakthroughs, to develop better-performing systems for use in the short term, and perhaps to create methods and techniques that will be used as components of temporally embodied representation learning systems of the future. I see my contributions as described in this thesis as a strand in that agenda. At the same time, I believe the most important directions for high-impact research are toward deep reinforcement learning and temporally evolving representation learning modules. Such systems will form the backbone of machine learning systems of the future that will surpass human-level performance at the most difficult tasks.

Bibliography

[1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, 22(10):1533–1545, 2014.

[2] S. Akaho. A kernel method for canonical correlation analysis. In *Proc. Int'l Meeting on Psychometric Society*, 2001.

[3] T. W. Anderson. *An Introduction to Multivariate Statistical Analysis (2nd edition)*. John Wiley and Sons, 1984.

[4] Galen Andrew and Jeff Bilmes. Sequential deep belief networks. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4265–4268. IEEE, 2012.

[5] Galen Andrew and Jeff Bilmes. Backpropagation in sequential deep neural networks. In *NIPS*, 2013.

[6] Galen Andrew and Jianfeng Gao. Scalable training of l1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40. ACM, 2007.

[7] Galen Andrew, Raman Arora, Jeff Bilmes, and Karen Livescu. Deep canonical correlation analysis. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1247–1255, 2013.

[8] R. Arora and K. Livescu. Kernel CCA for multi-view learning of acoustic features using articulatory measurements. In *Symp. on Machine Learning in Speech and Language Processing*, 2012.

[9] R. Arora and K. Livescu. Multi-view CCA-based acoustic features for phonetic recognition across speakers and domains. In *Int. Conf. on Acoustics, Speech, and Signal Processing*, 2013.

[10] Hideki Asoh and Osamu Takechi. An approximation of nonlinear canonical correlation analysis by multilayer perceptrons. In *ICANN94*, pages 713–716. Springer, 1994.

[11] F. R. Bach. Consistency of trace norm minimization. *J. Mach. Learn. Res.*, 9:1019–1048, June 2008.

[12] F. R. Bach and M. I. Jordan. Kernel independent component analysis. *J. Mach. Learn. Res.*, 3:1–48, 2002.

[13] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning*, 4(1):1–106, 2012.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[15] David Balduzzi and Muhammad Ghifary. Compatible value gradients for reinforcement learning of continuous deep policies. *arXiv preprint arXiv:1509.03005*, 2015.

[16] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[17] Stephen Becker and Jalal Fadili. A quasi-newton proximal splitting method. In *Advances in Neural Information Processing Systems*, pages 2618–2626, 2012.

[18] Y. Bengio and O. Delalleau. Justifying and generalizing contrastive divergence. *Neural Computation*, 21(6):1601–1621, 2009.

[19] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, Universit De Montral, and Montral Qubec. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press, 2007.

[20] Yoshua Bengio, Aaron Courville, and Pierre Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013.

[21] J. S. Benson and J. J. More. A limited memory variable metric method for bound constraint minimization, 2001.

[22] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[23] 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 *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX, 2010.

[24] Olivier Bichler, Damien Querlioz, Simon J Thorpe, Jean-Philippe Bourgoin, and Christian Gamrat. Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. *Neural Networks*, 32:339–348, 2012.

[25] M. B. Blaschko and C. H. Lampert. Correlational spectral clustering. In *CVPR*, 2008.

[26] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[27] Ondrej Bojar, Rajen Chatterjee, Christian Federman, Barry Haddow, Matthias Huck, Chris Hokamp, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Carolina Scarton, Lucia Specia, and Marco Turchi. Findings of the 2015 workshop on statistical machine translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 1–46. Association for Computational Linguistics, 2015.

[28] Hervé Bourlard and Christian J Welleken. Links between markov models and multilayer perceptrons. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(12): 1167–1178, 1990.

[29] S. Boyd, J. Duchi, and L. Vandenberghe. Lecture notes for ee364b: Subgradients. `http://web.stanford.edu/class/ee364b/lectures/subgradients_notes.pdf`, 2015. Accessed 2016-1-13.

[30] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[31] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[32] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL*, 2005.

[33] K. Chaudhuri, S. M. Kakade, K. Livescu, and K. Sridharan. Multi-view clustering via canonical correlation analysis. In *ICML*, 2009.

[34] C.-C. Cheng, F. Sha, and L. K. Saul. A fast online algorithm for large margin training of continuous-density hidden markov models. In *Interspeech*, 2009.

[35] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[36] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[37] K. Choukri and G. Chollet. Adaptation of automatic speech recognizers to new speakers using canonical correlation analysis techniques. *Speech Comm.*, 1:95–107, 1986.

[38] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[39] Dan C. Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237, 2011.

[40] Michael Collins. Discriminative reranking for natural language parsing. In *ICML*, pages 175–182, 2000. ISBN 1-55860-707-2.

[41] K. Crammer. Efficient online learning with individual learning-rates for phoneme sequence recognition. *Journal of Machine Learning Research*, 7, 2006.

[42] G. E. Dahl, M. Ranzato, A. Mohamed, and G. Hinton. Phone recognition with the mean-covariance restricted boltzmann machine. In *Advances in Neural Information Processing Systems 23*, 2010.

[43] J. Darroch and D. Ratcliff. Generalised iterative scaling for log-linear models. *Annals of Mathematical Statistics*, 1972.

[44] S. B. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Trans. Acoustics, Speech, and Signal Proc.*, 28(4):357–366, 1980.

[45] T. De Bie and B. De Moor. On the regularization of canonical correlation analysis. In *Proc. Int'l Conf. on Independent Component Analysis and Blind Source Separation*, 2003.

[46] P. Dhillon, D. Foster, and L. Ungar. Multi-view learning of word embeddings via CCA. In *NIPS*, 2011.

[47] David L. Donoho, Michael Elad, and Vladimir N. Temlyakov. Stable recovery of sparse overcomplete representations in the presence of noise. *Information Theory, IEEE Transactions on*, 52(1):6–18, 2006.

[48] B. Efron, T. Hastie, I. Johnstone, and R Tibshirani. Least angle regression. *Annals of Statistics*, 2004.

[49] C. H. Ek, P. H. Torr, , and N. D. Lawrence. Ambiguity modelling in latent spaces. In *MLMI*, 2008.

[50] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.

[51] Kayvon Fatahalian, Jeremy Sugerman, and Pat Hanrahan. Understanding the efficiency of gpu algorithms for matrix-matrix multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 133–137. ACM, 2004.

[52] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[53] Jianfeng Gao, Galen Andrew, Mark Johnson, and Kristina Toutanova. A comparative study of parameter estimation methods for statistical natural language processing. In *ANNUAL MEETING-ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, volume 45, page 824, 2007.

[54] Robert Gens and Domingos Pedro. Learning the structure of sum-product networks. In *Proceedings of The 30th International Conference on Machine Learning*, pages 873–880, 2013.

[55] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[56] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[57] J. Goodman. Exponential priors for maximum entropy models. In *ACL*, 2004.

[58] Alan Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013.

[59] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[60] A. Gunawardana, M. Mahajan, A. Acero, and J. C. Platt. Hidden conditional random fields for phone classification. In *Interspeech*, 2005.

[61] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in Neural Information Processing Systems*, pages 3338–3346, 2014.

[62] A. Haghighi, P. Liang, T. Berg-Kirkpatrick, and D. Klein. Learning bilingual lexicons from monolingual corpora. In *ACL-HLT*, 2008.

[63] Robert M. Haralick, Karthikeyan Shanmugam, and Its' Hak Dinstein. Textural features for image classification. *Systems, Man and Cybernetics, IEEE Transactions on*, (6): 610–621, 1973.

[64] D. R. Hardoon, S. Szedmák, and J. Shawe-Taylor. Canonical correlation analysis: An overview with application to learning methods. *Neural Computation*, 16(12):2639–2664, 2004.

[65] D. R. Hardoon, J. Mourao-Miranda, M. Brammer, and J. Shawe-Taylor. Unsupervised analysis of fMRI data using kernel canonical correlation. *NeuroImage*, 37(4):1250–1259, 2007.

[66] Johan Håstad and Mikael Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1(2):113–129, 1991.

[67] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2926–2934, 2015.

[68] H. Hermansky and S. Sharma. Temporal patterns (TRAPS) in ASR of noisy speech. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 289–292. IEEE, 2002. ISBN 0780350413.

[69] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18, 2006.

[70] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[71] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[72] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[73] Geoffrey E. Hinton. Connectionist learning procedures. *Artificial intelligence*, 40(1): 185–234, 1989.

[74] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[75] Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

[76] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[77] H. Hotelling. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377, 1936.

[78] William W Hsieh. Nonlinear canonical correlation analysis by neural networks. *Neural Networks*, 13(10):1095–1105, 2000.

[79] William W Hsieh. Nonlinear canonical correlation analysis of the tropical pacific climate variability using a neural network approach. *Journal of Climate*, 14(12):2528–2539, 2001.

[80] William W Hsieh. Nonlinear multivariate and time series analysis by neural network methods. *Reviews of Geophysics*, 42(1), 2004.

[81] David H. Hubel and Torsten N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1): 106–154, 1962.

[82] A. G. Ivakhnenko. *Cybernetic predicting devices.* CCM Information Corporation, 1965.

[83] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[84] R. D. Joseph. *Contributions to perceptron theory*. PhD thesis, Cornell University, 1961.

[85] S. M. Kakade and D. P. Foster. Multi-view regression via canonical correlation analysis. In *COLT*, 2007.

[86] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, volume 3, page 413, 2013.

[87] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[88] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[89] Jun'ichi Kazama and Jun'ichi Tsujii. Evaluation and extension of maximum entropy models with inequality constraints. In *EMNLP*, 2003.

[90] J. Keshet, S. Shalev-Shwartz, S. Bengio, Y. Singer, and D. Chazan. Discriminative kernel-based phoneme sequence recognition. In *Interspeech*, 2006.

[91] J. Keshet, D. McAllester, and T. Hazan. Pac-bayesian approach for minimization of phoneme error rate. In *International Conference on Acoustics Speech and Signal Processing*, 2011.

[92] T. K. Kim, S. F. Wong, and R. Cipolla. Tensor canonical correlation analysis for action classification. In *CVPR*, 2007.

[93] Takehiko Kobayashi. L p-regularized optimization by using orthant-wise approach for inducing sparsity. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3327–3331. IEEE, 2013.

[94] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[95] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, 2001.

[96] Pei Ling Lai and Colin Fyfe. A neural implementation of canonical correlation analysis. *Neural Networks*, 12(10):1391–1397, 1999.

[97] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng. On optimization methods for deep learning. In *ICML*, 2011.

[98] Y. LeCun and C. Cortes. The MNIST database of handwritten digits, 1998.

[99] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[100] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pages 801–808, 2006.

[101] Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pages 873–880, 2008.

[102] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[103] Jason Lee, Yuekai Sun, and Michael Saunders. Proximal newton-type methods for convex optimization. In *Advances in Neural Information Processing Systems*, pages 836–844, 2012.

[104] K.F. Lee and H.-W. Hon. Speaker-independent phone recognition using hidden markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37, 1989.

[105] S.-I. Lee, H. Lee, P. Abbeel, and A.Y. Ng. Efficient L1 regularized logistic regression. In *AAAI-06*, 2006.

[106] Su-In Lee, Varun Ganapathi, and Daphne Koller. Efficient structure learning of markov networks. using L1 regularization. In *NIPS*, 2006.

[107] A. S. Lewis. Derivatives of spectral functions. *Mathematics of Operations Research*, 21 (3):576–588, 1996.

[108] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[109] David G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[110] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[111] Ang Lu, Weiran Wang, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Deep multilingual correlation for improved word embeddings. In *Proceedings of NAACL*, 2015.

[112] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and*

Probability, Volume 1: Statistics, pages 281–297, Berkeley, Calif., 1967. University of California Press. URL `http://projecteuclid.org/euclid.bsmsp/1200512992`.

[113] Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *CONLL*, 2002.

[114] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, and Alan L Yuille. Explain images with multimodal recurrent neural networks. *arXiv preprint arXiv:1410.1090*, 2014.

[115] K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate Analysis*. Academic Press, 1979.

[116] James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 735–742, 2010.

[117] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040, 2011.

[118] Marina Meila and Jianbo Shi. Learning segmentation by random walks. In *In Advances in Neural Information Processing Systems*, pages 873–879. MIT Press, 2001.

[119] T. Melzer, M. Reiter, and H. Bischof. Nonlinear feature extraction using generalized canonical correlation analysis. In *ICANN*, 2001.

[120] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, page 3, 2010.

[121] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Honza Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011.

[122] Thomas P. Minka. A comparison of numerical optimizers for logistic regression. Technical report, 2003.

[123] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[124] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

[125] A. Mohamed, G. Dahl, and G. Hinton. Deep belief networks for phone recognition. In *Advances in Neural Information Processing Systems 22 (Workshops)*, 2009.

[126] A. Mohamed, D. Yu, and L. Deng. Investigation of full-sequence training of deep belief networks. In *Interspeech*, 2010.

[127] L. Montanarella, M. Bassami, and O. Breas. Chemometric classification of some European wines using pyrolysis mass spectrometry. *Rapid Communications in Mass Spectrometry*, 9(15):1589–1593, 1995.

[128] Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V Todorov. Interactive control of diverse complex characters with neural networks. In *Advances in Neural Information Processing Systems*, pages 3114–3122, 2015.

[129] J. Morris and E. Fosler-Lussier. Conditional random fields for integrating local discriminative classifiers. *IEEE Trans. on Acoustics, Speech, and Language Processing*, 16(3), 2008.

[130] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[131] Radford M. Neal. Connectionist learning of belief networks. *Artificial intelligence*, 56 (1):71–113, 1992.

[132] Yurii Nesterov et al. Gradient methods for minimizing composite objective function. Technical report, UCL, 2007.

[133] Andrew Y. Ng. Feature selection, L1 vs. L2 regularization, and rotational invariance. In *ICML*, 2004.

[134] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

[135] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[136] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, 1999.

[137] Peter OConnor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Neuromorphic Engineering Systems and Applications*, page 61, 2015.

[138] Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[139] Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. Improved part-of-speech tagging for online conversational text with word clusters. Association for Computational Linguistics, 2013.

[140] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572, 1901. doi: 10.1080/14786440109462720.

[141] S. Perkins and J. Theiler. Online feature selection using grafting. In *ICML*, 2003.

[142] K. B. Petersen and M. S. Pedersen. The matrix cookbook, Nov 2012. URL `http://www2.imm.dtu.dk/pubdb/p.php?3274`.

[143] Slav Petrov and Dan Klein. Discriminative log-linear grammars with latent variables. In *Advances in Neural Information Processing Systems*, pages 1153–1160, 2007.

[144] Christian Plahl, Tara N. Sainath, Bhuvana Ramabhadran, and David Nahamoo. Improved pre-training of deep belief networks using sparse encoding symmetric machines. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4165–4168. IEEE, 2012.

[145] Marc Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[146] Marc'Aurelio Ranzato and Geoffrey E Hinton. Modeling pixel means and covariances using factorized third-order boltzmann machines. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2551–2558. IEEE, 2010.

[147] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999.

[148] Salah Rifai, Grégoire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. Higher order contractive auto-encoder. In *Machine Learning and Knowledge Discovery in Databases*, pages 645–660. Springer, 2011.

[149] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 833–840, 2011.

[150] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[151] F. Rudzicz. Adaptive kernel canonical correlation analysis for estimation of task dynamics from acoustics. In *ICASSP*, 2010.

[152] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[153] Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*, pages 338–342, 2014.

[154] M. E. Sargin, Y. Yemez, and A. M. Tekalp. Audiovisual synchronization and fusion using canonical correlation analysis. *IEEE. Trans. Multimedia*, 9(7):1396–1403, 2007.

[155] Mark Schmidt. *Graphical model structure learning with l1-regularization.* PhD thesis, UNIVERSITY OF BRITISH COLUMBIA (Vancouver, 2010.

[156] Mark Schmidt, Glenn Fung, and Romer Rosales. Optimization methods for l1-regularization. *University of British Columbia, Technical Report TR-2009*, 19, 2009.

[157] Mark W Schmidt, Ewout Berg, Michael P Friedlander, and Kevin P Murphy. Optimizing costly functions with simple constraints: A limited-memory projected quasi-newton algorithm. In *International Conference on Artificial Intelligence and Statistics*, page None, 2009.

[158] F. Sha and L. K. Saul. Comparison of large margin training to other discriminative methods for phonetic recognition by hidden markov models. In *International Conference on Acoustics Speech and Signal Processing*, 2007.

[159] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[160] Christian Siagian and Laurent Itti. Rapid biologically-inspired scene classification using

features shared with visual attention. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(2):300–312, 2007.

[161] BY HAVA EVE TOVA SIEGELMANN. *Foundations of recurrent neural networks*. PhD thesis, Citeseer, 1993.

[162] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[163] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *null*, page 958. IEEE, 2003.

[164] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[165] J. Sj ostr om and W. Gerstner. Spike-timing dependent plasticity. 5(2):1362, 2010.

[166] M. Slaney and M. Covell. FaceSync: A linear operator for measuring synchronization of video facial images and audio tracks. In *NIPS*, 2000.

[167] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. 1986.

[168] Richard Socher, Eric H. Huang, Jeffrey Pennin, Christopher D. Manning, and Andrew Y. Ng. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, pages 801–809, 2011.

[169] Richard Socher, Cliff C. Lin, Christopher D. Manning, and Andrew Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

[170] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 151–161. Association for Computational Linguistics, 2011.

[171] Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. Parsing with compositional vector grammars. In *ACL (1)*, pages 455–465, 2013.

[172] N. Srivastava and R. Salakhutdinov. Multimodal learning with deep Boltzmann machines. In *NIPS*, 2012.

[173] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[174] Marijn F. Stollenga, Jonathan Masci, Faustino Gomez, and Jürgen Schmidhuber. Deep networks with internal selective attention through feedback connections. In *Advances in Neural Information Processing Systems*, pages 3545–3553, 2014.

[175] Y.-H. Sung and D. Jurafsky. Hidden conditional random fields for phone recognition. In *Automatic Speech Recognition and Understanding*, 2009.

[176] Ilya Sutskever, James Martens, and Geoffrey E. Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[177] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[178] Joshua B. Tenenbaum, Vin De Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[179] Simon J Thorpe. Spike-based image processing: can we reproduce biological vision in hardware? In *Computer Vision–ECCV 2012. Workshops and Demonstrations*, pages 516–521. Springer, 2012.

[180] Rob Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B*, 1996.

[181] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1064–1071, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390290. URL `http://doi.acm.org/10.1145/1390156.1390290`.

[182] László Tóth. Convolutional deep rectifier neural nets for phone recognition. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[183] Paul E. Utgoff and David J. Stracuzzi. Many-layered learning. *Neural Computation*, 14 (10):2497–2529, 2002.

[184] Douglas L Vail and Manuela M Veloso. Feature selection for activity recognition in multi-robot domains. In *AAAI*, volume 8, pages 1415–1420, 2008.

[185] J.-P. Vert and M. Kanehisa. Graph-driven features extraction from microarray data using diffusion kernels and kernel CCA. In *NIPS*, 2002.

[186] Karel Veselỳ, Arnab Ghoshal, Lukáš Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In *Interspeech*, Lyon, France, 2013.

[187] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*. ACM, 2008.

[188] 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. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

[189] A. Vinokourov, J. Shawe-Taylor, and N. Cristianini. Inferring a semantic representation of text via cross-language correlation analysis. In *NIPS*, 2003.

[190] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

[191] Stefan Wager, Sida Wang, and Percy S. Liang. Dropout training as adaptive regularization. In *Advances in neural information processing systems*, pages 351–359, 2013.

[192] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[193] Weiran Wang, Raman Arora, Karen Livescu, and Jeff Bilmes. On deep multi-view representation learning. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1083–1092, 2015.

[194] Weiran Wang, Raman Arora, Karen Livescu, and Nathan Srebro. Stochastic optimization for deep cca via nonlinear orthogonal iterations. *arXiv preprint arXiv:1510.02054*, 2015.

[195] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[196] J. R. Westbury. *X-ray microbeam speech production database user's handbook*. Waisman Center on Mental Retardation & Human Development, U. Wisconsin, Madison, WI, version 1.0 edition, June 1994.

[197] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[198] Fei Yan and Krystian Mikolajczyk. Deep correlation for matching images and text. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3441–3450, 2015.

[199] D. Yu, S. Wang, and L. Deng. Sequential labeling using deep-structured conditional random fields. *IEEE Journal of Selected Topics in Signal Processing*, 4(6), 2010.

[200] Jin Yu, SVN Vishwanathan, Simon Günter, and Nicol N Schraudolph. A quasi-newton approach to nonsmooth convex optimization problems in machine learning. *The Journal of Machine Learning Research*, 11:1145–1200, 2010.

[201] Matthew D. Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

[202] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

[203] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2): 301–320, 2005.

Appendix A

PROOF OF CLAIM IN DERIVATION OF SDNN GRADIENT

Proposition 1. *If A, B, and C are ± 1-valued random variables with A independent of C given B, then*

$$\operatorname{Cov}(A, C) = \frac{\operatorname{Cov}(A, B)\operatorname{Cov}(B, C)}{\operatorname{Var} B}.$$

Proof. The following parameterization of the joint distribution over A, B, and C will simplify the proof. Let $b = \Pr(B = 1)$ and,

$$a^+ = \mathbb{E}[A|B=1], \qquad c^+ = \mathbb{E}[C|B=1],$$
$$a^- = \mathbb{E}[A|B=-1], \qquad c^- = \mathbb{E}[C|B=-1].$$

Now we can derive

$$\mathbb{E}[B] = 2b - 1,$$
$$\mathbb{E}[A] = ba^+ + (1-b)a^-,$$
$$\mathbb{E}[C] = bc^+ + (1-b)c^-,$$

and

$$\mathbb{E}[AB] = ba^+ - (1-b)a^-,$$
$$\mathbb{E}[AC] = ba^+c^+ + (1-b)a^-c^-,$$

and

$$\operatorname{Var}(B) = 4b(1-b).$$

So

$$\begin{aligned}\operatorname{Cov}(A,B) &= \mathbb{E}[AB] - \mathbb{E}[A]\,\mathbb{E}[B] \\ &= ba^+ - (1-b)a^- - (ba^+ + (1-b)a^-)(2b-1) \\ &= 2b(1-b)(a^+ - a^-),\end{aligned}$$

and

$$\begin{aligned}\operatorname{Cov}(A,C) &= \mathbb{E}[AC] - \mathbb{E}[A]\,\mathbb{E}[C] \\ &= ba^+c^+ + (1-b)a^-c^- - (ba^+ + (1-b)a^-)(bc^+ + (1-b)c^-) \\ &= b(1-b)(a^+ - a^-)(c^+ - c^-).\end{aligned}$$

Therefore

$$\begin{aligned}\operatorname{Cov}(A,B)\operatorname{Cov}(B,C) &= 2b(1-b)(a^+ - a^-)\cdot 2b(1-b)(c^+ - c^-) \\ &= 4b(1-b)\cdot b(1-b)(a^+ - a^-)(c^+ - c^-) \\ &= \operatorname{Var}(B)\operatorname{Cov}(A,C).\end{aligned}$$

$\square$

Appendix B
DERIVATION OF DCCA GRADIENT

To perform backpropagation, we must be able to compute the gradient of $f = \mathrm{corr}(H_1, H_2)$ defined in Equation (4.10). Denote by ∇_{ij} the matrix of partial derivatives of f with respect to the entries of $\hat{\Sigma}_{ij}$. Let the singular value decomposition of $T = \hat{\Sigma}_{11}^{-1/2}\hat{\Sigma}_{12}\hat{\Sigma}_{22}^{-1/2}$ be given as $T = UDV'$. First we will show that

$$\nabla_{12} = \hat{\Sigma}_{11}^{-1/2}UV'\hat{\Sigma}_{22}^{-1/2} \tag{B.1}$$

and

$$\nabla_{11} = -\frac{1}{2}\hat{\Sigma}_{11}^{-1/2}UDU'\hat{\Sigma}_{11}^{-1/2} \tag{B.2}$$

(resp. ∇_{22}). To prove (B.1), we use the fact that for a matrix X, $\nabla||X||_{\mathrm{tr}} = UV'$, where $X = UDV'$ is the singular value decomposition of X [11]. Using the chain rule:

$$\begin{aligned}
(\nabla_{12})_{ab} &= \frac{\partial f}{\partial(\hat{\Sigma}_{12})_{ab}} \\
&= \sum_{cd} \frac{\partial f}{\partial T_{cd}} \cdot \frac{\partial T_{cd}}{\partial(\hat{\Sigma}_{12})_{ab}} \\
&= \sum_{cd} (UV')_{cd} \cdot (\hat{\Sigma}_{11}^{-1/2})_{ca}(\hat{\Sigma}_{22}^{-1/2})_{bd} \\
&= (\hat{\Sigma}_{11}^{-1/2}UV'\hat{\Sigma}_{22}^{-1/2})_{ab}
\end{aligned}$$

For (B.2) we use the identity $\nabla \operatorname{tr} X^{1/2} = \frac{1}{2}X^{-1/2}$. This is easily derived from Theorem 1 of Lewis [107]:

Theorem 1. *A matrix function f is called a* spectral function *if it depends only on the set of eigenvalues of its argument. That is, for any positive definite matrix X and any unitary matrix V, $f(X) = f(VXV')$. If f is a spectral function, and X is positive definite with* *eigendecomposition* $X = UDU'$, *then*

$$\frac{\partial f(X)}{\partial X} = U \operatorname{diag} \frac{\partial f(D)}{\partial D} U'. \tag{B.3}$$

Now we can proceed

$$\begin{aligned}
(\nabla_{11})_{ab} &= \frac{\partial f}{\partial (\hat{\Sigma}_{11})_{ab}} \\
&= \sum_{cd} \frac{\partial f}{\partial (T'T)_{cd}} \frac{\partial (T'T)_{cd}}{\partial (\hat{\Sigma}_{11})_{ab}} \\
&= \sum_{cd} \left(\frac{1}{2} (T'T)^{-1/2} \right)_{cd} \frac{\partial (T'T)_{cd}}{\partial (\hat{\Sigma}_{11})_{ab}}
\end{aligned} \tag{B.4}$$

Since $T'T = \hat{\Sigma}_{22}^{-1/2} \hat{\Sigma}_{21} \hat{\Sigma}_{11}^{-1} \hat{\Sigma}_{12} \hat{\Sigma}_{22}^{-1/2}$, and using Eq. 60 from Petersen and Pedersen [142] for the derivative of an inverse,

$$\begin{aligned}
\frac{\partial (T'T)_{cd}}{\partial (\hat{\Sigma}_{11})_{ab}} &= \sum_{ij} \frac{\partial (T'T)_{cd}}{\partial (\hat{\Sigma}_{11}^{-1})_{ij}} \frac{\partial (\hat{\Sigma}_{11}^{-1})_{ij}}{\partial (\hat{\Sigma}_{11})_{ab}} \\
&= -\sum_{ij} (\hat{\Sigma}_{22}^{-1/2} \hat{\Sigma}_{21})_{ci} (\hat{\Sigma}_{12} \hat{\Sigma}_{22}^{-1/2})_{jd} (\hat{\Sigma}_{11}^{-1})_{ia} (\hat{\Sigma}_{11}^{-1})_{bj} \\
&= -(\hat{\Sigma}_{22}^{-1/2} \hat{\Sigma}_{21} \hat{\Sigma}_{11}^{-1})_{ca} (\hat{\Sigma}_{11}^{-1} \hat{\Sigma}_{12} \hat{\Sigma}_{22}^{-1/2})_{bd} \\
&= -(T' \hat{\Sigma}_{11}^{-1/2})_{ca} (\hat{\Sigma}_{11}^{-1/2} T)_{bd}
\end{aligned}$$

So continuing from (B.4),

$$\begin{aligned}
(\nabla_{11})_{ab} &= -\frac{1}{2} \sum_{cd} (T' \hat{\Sigma}_{11}^{-1/2})_{ca} (T'T)_{cd}^{-1/2} (\hat{\Sigma}_{11}^{-1/2} T)_{bd} \\
&= -\frac{1}{2} \sum_{cd} (\hat{\Sigma}_{11}^{-1/2} T)_{ac} (T'T)_{cd}^{-1/2} (T' \hat{\Sigma}_{11}^{-1/2})_{db} \\
&= -\frac{1}{2} \left(\hat{\Sigma}_{11}^{-1/2} T (T'T)^{-1/2} T' \hat{\Sigma}_{11}^{-1/2} \right)_{ab} \\
&= -\frac{1}{2} \left(\hat{\Sigma}_{11}^{-1/2} UDV' (VD^{-1}V') VDU' \hat{\Sigma}_{11}^{-1/2} \right)_{ab} \\
&= -\frac{1}{2} \left(\hat{\Sigma}_{11}^{-1/2} UDU' \hat{\Sigma}_{11}^{-1/2} \right)_{ab}
\end{aligned}$$

Using ∇_{12} and ∇_{11}, we are ready to compute $\partial f/\partial H_1$. First (temporarily moving subscripts on H_1 and $\hat{\Sigma}_{11}$ to superscripts so subscripts can index into matrices)

$$\frac{\partial \hat{\Sigma}^{11}_{ab}}{\partial H^1_{ij}} = \begin{cases} \frac{2}{m-1}\big(H^1_{ij} - \frac{1}{m}\sum_k H^1_{ik}\big) & \text{if } a = i, b = i \\ \frac{1}{m-1}\big(H^1_{bj} - \frac{1}{m}\sum_k H^1_{bk}\big) & \text{if } a = i, b \neq i \\ \frac{1}{m-1}\big(H^1_{aj} - \frac{1}{m}\sum_k H^1_{ak}\big) & \text{if } a \neq i, b = i \\ 0 & \text{if } a \neq i, b \neq i \end{cases}$$
$$= \frac{1}{m-1}\big(1_{\{a=i\}}\bar{H}^1_{bj} + 1_{\{b=i\}}\bar{H}^1_{aj}\big).$$

Also,

$$\frac{\partial \hat{\Sigma}^{12}_{ab}}{\partial H^1_{ij}} = \frac{1}{m-1} 1_{\{a=i\}} \left(H^2_{bj} - \frac{1}{m}\sum_k H^2_{bk} \right)$$
$$= \frac{1}{m-1} 1_{\{a=i\}} \bar{H}^2_{bj}.$$

Putting this together, we obtain

$$\frac{\partial f}{\partial H^1_{ij}} = \sum_{ab} \nabla^{11}_{ab} \frac{\partial \hat{\Sigma}^{11}_{ab}}{\partial H^1_{ij}} + \sum_{ab} \nabla^{12}_{ab} \frac{\partial \hat{\Sigma}^{12}_{ab}}{\partial H^1_{ij}}$$
$$= \frac{1}{m-1}\left(\sum_b \nabla^{11}_{ib}\bar{H}^1_{bj} + \sum_a \nabla^{11}_{ai}\bar{H}^1_{aj} + \sum_b \nabla^{12}_{ib}\bar{H}^2_{bj} \right)$$
$$= \frac{1}{m-1}\Big(\big(\nabla_{11}\bar{H}_1\big)_{ij} + \big(\nabla'_{11}\bar{H}_1\big)_{ij} + \big(\nabla_{12}\bar{H}_2\big)_{ij} \Big).$$

Using the fact that ∇_{11} is symmetric, this can be written more compactly as

$$\frac{\partial f}{\partial H_1} = \frac{1}{m-1}\left(2\nabla_{11}\bar{H}_1 + \nabla_{12}\bar{H}_2\right).$$

Appendix C

TERMINATION OF OWL-QN PROJECTED LINE SEARCH

Proposition 2. *If x^k is not a critical point, then there exists a step size $\alpha > 0$ such that $\pi(x^k + \alpha p^k; \xi^k) \neq x^k$.*

Proof. Since x^k is not a critical point, $\Diamond f(x^k) \neq 0$. The only way for the line search to make no progress would be if for all i, either $p_i^k = 0$, or $x_i^k = 0$ and $p_i^k \Diamond_i f(x^k) < 0$, in which case the i^{th} coordinate would be constrained to zero. But then

$$(\Diamond f(x^k))' H_k^{-1} \Diamond f(x^k) = (p^k)' \Diamond f(x^k) = \sum_i p_i^k \Diamond_i f(x^k) \leq 0,$$

which is impossible because H_k is positive definite. □

Proposition 3. *If x^k is not a critical point, the line search described in 5.3.2 will terminate in a finite number of steps.*

Proof. Let ϕ be the function $\phi(\alpha) = f(\pi(x^k + \alpha p^k; \xi^k))$ for $\alpha \geq 0$. Let P^k be the set of coordinates that are not constrained at step k. Prop. 2 guarantees us that $P^k \neq \emptyset$. For each $i \in P^k$, define

$$\alpha_i^* = \begin{cases} -x_i^k / p_i^k & \text{if } x_i^k p_i^k < 0, \\ 1 & \text{otherwise,} \end{cases}$$

and let $\alpha^* = \min_i \alpha_i^*$. Thus for steps smaller than α^*, none of the coordinates of P^k are projected, so ϕ is continuously differentiable in $[0, \alpha^*)$. If the backtracking line search gets to this region, it becomes the standard backtracking line search for the Armijo condition on a smooth function, which is guaranteed to terminate. □