UNSUPERVISED FEATURE LEARNING VIA SPARSE HIERARCHICAL REPRESENTATIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Honglak Lee

August 2010

Abstract

Machine learning has proved a powerful tool for artificial intelligence and data mining problems. However, its success has usually relied on having a good feature representation of the data, and having a poor representation can severely limit the performance of learning algorithms. These feature representations are often hand-designed, require significant amounts of domain knowledge and human labor, and do not generalize well to new domains.

To address these issues, I will present machine learning algorithms that can automatically learn good feature representations from unlabeled data in various domains, such as images, audio, text, and robotic sensors. Specifically, I will first describe how efficient sparse coding algorithms — which represent each input example using a small number of basis vectors — can be used to learn good low-level representations from unlabeled data. I also show that this gives feature representations that yield improved performance in many machine learning tasks.

In addition, building on the deep learning framework, I will present two new algorithms, sparse deep belief networks and convolutional deep belief networks, for building more complex, hierarchical representations, in which more complex features are automatically learned as a composition of simpler ones. When applied to images, this method automatically learns features that correspond to objects and decompositions of objects into object-parts. These features often lead to performance competitive with or better than highly hand-engineered computer vision algorithms in object recognition and segmentation tasks. Further, the same algorithm can be used to learn feature representations from audio data. In particular, the learned features yield improved performance over state-of-the-art methods in several speech recognition tasks.

Acknowledgements

Most of all, I would like to thank my advisor Andrew Ng. It has been a privilege and truly an honor to have him as a mentor. Andrew has been an amazing mentor and advisor, not only in research, but also other aspects of academic life. I cannot thank you enough.

I also would like to thank all my committee members, Daphne Koller and Krishna Shenoy who were my reading committee, as well as Jay McClelland and Kai Yu who were my defense committee. It has been truly a great privilege and honor to have them as mentors, and I received invaluable advice and constructive feedback for the research. Thank you so much.

I also would like to thank all the lab members of Andrew Ng's machine learning group, especially Rajat Raina whom I did lots of collaborations with. I also thank other former and current lab members: Pieter Abbeel, Ashustosh Saxena, Tom Do, Zico Kolter, Morgan Quigley, Adam Coates, Quoc Le, Olga Russakovsky, Jiquan Ngiam, and Andrew Maas. I thank my friends and colleagues at Stanford: Su-In Lee, Stephen Gould, Alexis Battle, Ben Packer, Suchi Saria, Varun Ganapathi, Alex Teichman, Jenny Finkel, Yun-Hsuan Sung, David Jackson, David Stavens, Roger Grosse, Chaitu Ekanadham, Rajesh Ranganath, Peter Pham, Yan Largman, Jaewon Yang, and Myunghwan Kim, Dongjun Shin, and Jinsung Kwon. It has been such a wonderful experience and privilege to know and work with you.

Finally, I thank my parents and family for their love and support. Without their support, this thesis would not have been possible. Especially, I thank my wife Youngjoo, who has always been supportive and encouraging.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The goal of machine learning is to develop algorithms that can learn (e.g., recognize patterns) from complex data and make accurate predictions for previously unseen data. In the past few decades, machine learning has been successful in tackling many real-world artificial intelligence (AI) problems. For example, it has been successfully applied in optical character recognition, face detection, and speech recognition. Machine learning has also been a powerful tool in tackling challenging problems in computer vision, natural language understanding, autonomous car driving, data mining of biological data, medical imaging, financial engineering, and web search/information retrieval.

However, there are two issues that may need to be addressed in order for machine learning to be more successful. First, the success of machine learning systems often requires a large amount of labeled data. While it is known that a larger amount of training data is better [10, 11, 28], it is expensive to obtain a large amount of labels since it often requires significant human labor. In contrast, unlabeled data are cheap to obtain because a virtually unlimited amount of them can be easily obtained over the web. Therefore, it is desirable to use a large amount of unlabeled data with only a small amount of labeled data. Furthermore, many real-world machine learning applications require a good "feature" representation to be successful. Here we define features as properties or attributes that can be computed from the input data, with a hope that these feature representations are somewhat easier for learning algorithms. However, it is not straightforward to obtain good feature representations; unfortunately, such processes are usually done by human efforts, as further described in this thesis.

Feature engineering

The requirement of good feature representations comes from the fact that real-world data are complex and highly variable. For instance, computer vision is a challenging problem where current state-of-the-art algorithms do not match the level of human performance. The challenge for computers lies in the fact that images are high-dimensional (images are often represented by hundreds of thousands of pixels) and highly variable (due to many factors of variations, such as viewpoint changes, photometric effects, shape variation, and context variation). Hence, it is very difficult for learning algorithms to capture the structure of raw image data (i.e., pixel intensities) and to generalize well from training images to new images. This difficulty also holds true for most other domains, including speech and text, where input data is high-dimensional. Thus, it is often necessary to develop clever ways of constructing features to make learning problems easier to solve.

As a motivating example, consider a problem of classifying images into motorcycles or non-motorcycles. First, consider a setting where we use the pixel representation of images, and put the pixel features into a learning algorithm, such as the support vector machine (SVM) [26, 41], to classify whether the image contains motorcycles and other objects. As shown in Figure 1.1, if we pick two pixels and draw a scatter plot, motorcycles and non-motorcycles are difficult to separate. This issue is also true when we consider tens of thousands or hundreds of thousands of pixels.

Now, consider another setting where we have some magical features that give useful information about the image. For example, suppose that we have a feature mapping that computes "whether there exists a motorcycle wheel or motorcycle handle-bars." Given these higher-level features, the binary classification problem can be better separated. In other words, if we know that motorcycle wheels and motorcycle handlebar exist in an image, then there will be a high chance that the image contains motorcycles.

This illustration shows the powerful impact of good features on machine learning and its applications. However, it is not very practical to hand specify these high-level features

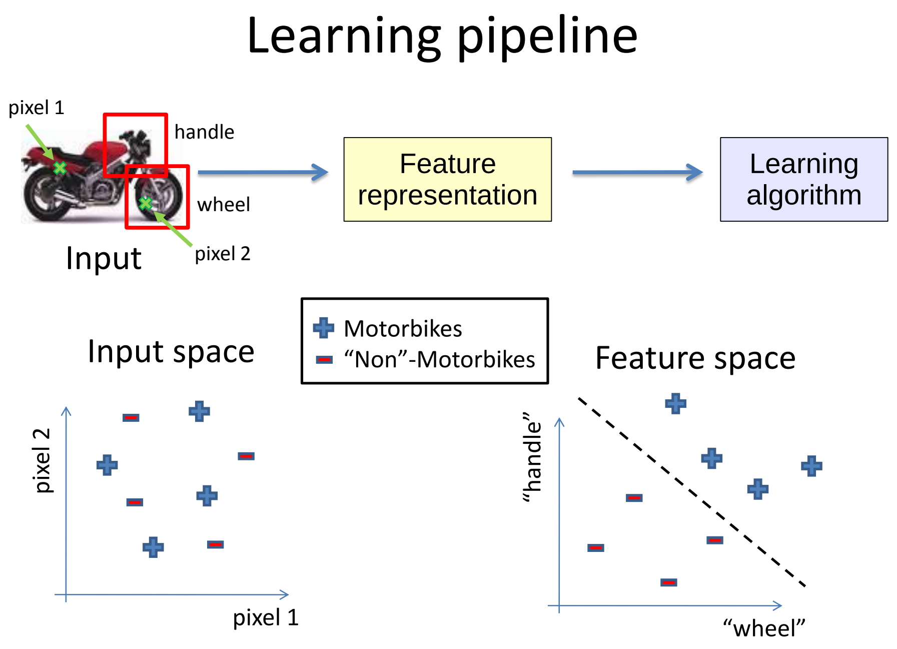


Figure 1.1: Illustration comparing pixel representation and higher-level feature representation. The upper part shows a typical pipeline of a machine learning application. For example, when we use pixel values as features, the two classes are not well separated (the bottom left plot). However, the separation becomes clearer when using a feature mapping from pixels to high-level features, such as "is there a handle bar in the image?" or "are there any motorcycle wheels in the image?" (the bottom right plot). See text for details.

because it requires much human labor (mostly spent on trials and errors). Furthermore, this process becomes more expensive and cumbersome if we have a large number of object categories to classify. More importantly, this approach is not feasible when we have little domain knowledge about the data.

Currently, most state-of-the-art systems in machine learning applications (e.g., computer vision, speech recognition, etc.) rely on these types of hand-tuned feature representations. These features are then used as inputs for machine learning algorithms (e.g., classification or regression). Typically, these hand-tuned features are fairly low-level because we try to hand-encode much prior knowledge, but at the same time we want to make these features work reasonably well in general cases. In other words, these hand-tuned features are domain-specific, but not data-specific. For example, in a visual object recognition task, we can use edge detectors to compute the boundaries of the object, and then put this feature representation ("edge detection" output) into a supervised learning algorithm (e.g., SVM). Similarly, in a speaker identification task, we can convert the waveform data into a spectrogram representation, which is essentially a Fourier transform of the time-series data (waveform intensities) by taking a small time window, and then put these features into a learning algorithm to identify the speaker.

Indeed, many of these hand-engineered features have been developed over last several decades. Examples of widely used computer vision features include Geometric Blur [18], SIFT [106], Shape Context [15], HOG [42], Textons [102], Spin images [74], RIFT [87], SURF [12], and GLOH [115]. Similarly in audio processing, a number of features have been developed, such as Spectrogram [44], MFCC [113], Spectral rolloff [146], Spectral flux [46], Zero-crossing rate (ZCR) [34]. Although such hand-engineered features are extensively used in practice, there are problems with these features.

First, they need a huge amount of domain-expert knowledge and time consuming hand-tuning. These feature representations often have to be obtained by human efforts, meaning that many domain experts (e.g., computer vision experts, speech recognition experts, natural language processing experts, etc.) have to spend a significant amount of their time (weeks, months, or even years) on "hand-tuning" the features based on their knowledge and numerous trials and errors. Second, the features in one domain do not generalize to other domains. Both of these issues are universally true in most domains. Therefore, it is desirable to have a way of automatically learning these feature representations from data, instead of hand-tuning.

In this thesis, I will investigate the problem of learning feature representations in an unsupervised way, without much hand-tuning. Addressing this issue will dramatically improve the current state of machine learning and its applications by achieving better prediction performance while using much less labeled data.

1.2 Unsupervised Feature Learning

Given the above issues, consider the problem of learning feature representations from unlabeled data. We are interested in primarily using unlabeled data because we can easily obtain a virtually unlimited amount of unlabeled data, such as images, speech, video, text, and other domains. In fact, even though we do not have labels, there often exist rich structures in unlabeled data. For example, if we look at images of a specific object (e.g. face), we can easily discover high-level structures such as object parts (e.g. face parts). Given natural images, we may be able to discover low-level structures such as edges, as well as high-level structures such as corners, local curvatures, and shapes. Similarly, given speech data, we may be able to discover not only low-level structures such as gamma-tone filters, but also higher-level structures that correspond to phonemes or words.

This thesis assumes that structures in unlabeled data can be useful in machine learning tasks. For example, if the input data have structures generated from specific object classes (such as cars vs. faces), then discovering class-specific patterns (such as car wheels or face parts) will be useful for classification, possibly together with a very small amount of labeled data. How can we discover these high-level features from unlabeled data? Unfortunately, it is difficult to learn such high-level features, which is the main problem that this thesis aims to address.

1.3 Related work

This section describes related work on learning features in a supervised or unsupervised way, using unlabeled data to help supervised learning tasks, and learning hierarchical representations.

1.3.1 Learning features from labeled data

Learning features from labeled data is related to several research areas in machine learning. I will briefly describe some of them, including multiple kernel learning, neural networks, multi-task learning, and transfer learning.

Recently, there has been much research in “learning kernels.” One representative approach is called multiple kernel learning (MKL) [79, 150, 161, 8], where the goal is to learn a convex combination of multiple kernels to obtain improved results for training data. Here, each kernel is typically derived from a specific feature representation; thus, MKL can be viewed as learning feature representations by combining features. However, it requires that multiple basic feature representations are initially given.

Multi-layer neural network [20] consists of hidden or output variables (also called artificial neurons) connected in a feed-forward manner across multiple layers. Each value of hidden or output variables is computed as a nonlinear function of the weights and the variables in the previous layer. The objective function to minimize is a loss function between the output and the labels in the training data, and the parameters (weights) of the network are trained via backpropagation [141]. In the context of feature learning, the hidden variables in the neural network can be viewed as features; therefore, training a neural network corresponds to learning the feature representation to improve the objective function. In particular, convolutional neural network has been successful in computer vision tasks [89]. The main idea is to share the parameters across hidden variables, and this weight-sharing has the advantage of reducing the number of parameters. However, convolutional neural networks do not work well given a small number of training examples [173, 133].

Multi-task learning [30] is a framework that aims to optimize multiple related (often similar) tasks simultaneously. The main idea is to share the intermediate feature representations for multiple tasks, thereby allowing the learning algorithms to exploit commonality among the tasks. Multi-task learning can be formulated in several different ways. For example, Argyriou et al. [6] formulated as regularizing with a composite norm (e.g., 1-2 norm) of the weight matrix (where each column represents a weight vector for each task). In this setting, the algorithm chooses a subspace of the original feature representation, and the weights for individual tasks are learned within this learned subspace. In another example, multi-task learning is implemented in multi-layer neural networks, where multiple output variables correspond to multiple tasks [30, 39]. In this setting, the hidden variables in the intermediate layers are shared and tuned for the multiple tasks simultaneously.

In a broader perspective, transfer learning is also related to the problem of feature learning [156, 30]. A complete review of transfer learning literature is beyond the scope of this thesis. (See [126] for a survey.) Briefly speaking, transfer learning assumes that there are source tasks and a target task (or sometimes multiple target tasks). The goal is to transfer knowledge or information from the source tasks to the target task. In particular, there are several transfer learning approaches based on transferring knowledge of feature representations from source tasks to the target task [6, 98, 71]. In addition, transfer learning and multi-task learning are closely related. Multi-task learning is informally considered as a sub-field of transfer learning although technically, multi-task learning aims to optimize for the multiple tasks simultaneously instead of having source and target tasks.

Although these methods show promise, all above-mentioned supervised feature learning methods typically require a large number of labeled training examples. Therefore, these algorithms may suffer when there is a small amount of labels. In this thesis, we mostly focus on settings without a large amount of labels. Thus, a question naturally follows: how can we learn feature representations in an unsupervised way?

1.3.2 Using unlabeled data to improve supervised learning tasks

There has been research on using unlabeled data to help supervised learning tasks. I will briefly describe semi-supervised learning (SSL) and a few specific SSL-based algorithms, such as co-occurrence heuristics [97, 62, 104], LDA [21], Universum [163], and Alternating Structural Optimization [4].

Semi-supervised learning [119, 33, 177] attempts to improve classifiers by using a large amount of unlabeled data together with the labeled data. A complete review of semi-supervised learning is beyond the scope of this thesis. (For details, see [33] or [176] for a survey.) Roughly speaking, semi-supervised learning algorithms are based on several different assumptions. One common assumption, often called "cluster assumption," is that the decision boundary should avoid the high density region of the input data. Examples of the models based on this assumption are mixture models (e.g., Gaussian mixtures) [119], transductive support vector machines [73], information regularization [152], and Gaussian processes-based models [86]. Other methods based on different assumptions include self-training [169], co-training [24], and graph-based methods [177, 23].

There are a few domain specific methods that are based on modeling the distribution of the unlabeled data. For example, co-occurrence statistics [97, 62, 104] is a heuristic widely used in natural language processing. The main idea is to compute co-occurrence statistics of the word pairs in unlabeled documents, and use the co-occurrence statistics for representing the context of a word in the documents. This heuristic is shown to improve performance in clustering similar words [104] and disambiguation problems [127]; however, it is not obvious how to apply this heuristic to other domains.

Latent Dirichlet Allocation (LDA) [21] is also a very popular algorithm in text processing. LDA assumes that the "topics" (where each topic represents a multinomial distribution over words) of each document is drawn from a Dirichlet distribution. LDA can learn topics based on bag-of-word representations. After training, the posterior over the topics for each document can be used as a feature representation. LDA and its variants (e.g., [22]) have been widely applied to information retrieval [162]. However, LDA is restricted to input domains that are represented as bag-of-words, so it cannot be applied to real-valued data.

Universum [163] assumes that the unlabeled data are distributed near the decision boundary of the classification task. Informally speaking, unlabeled data (assuming a binary classification task) are interpreted as "neither positive nor negative" or "between positive and negative classes." Weston et al. demonstrated improved performance by carefully constructing classification tasks that satisfy this assumption. However, the assumption about unlabeled data is fairly restrictive; therefore, the Universum method may not work well in cases where unlabeled data share the same labels with labeled data.

Ando and Zhang [4] proposed the Alternating Structural Optimization (ASO) algorithm. Specifically, they constructed auxiliary tasks using unlabeled data and showed that these auxiliary tasks are beneficial for the supervised task. In practice, constructing these auxiliary tasks significantly affect the classification performance, and the authors proposed a clever heuristics for natural language processing tasks. However, it is not obvious how to construct such auxiliary tasks in general settings.

1.3.3 Generic unsupervised learning algorithms

Many unsupervised learning algorithms can be used to learn features from unlabeled data. I will briefly discuss those algorithms, such as clustering methods (e.g., K-means and Gaussian mixture model), matrix factorization methods (e.g., PCA and other variants, ICA, NMF, and sparse coding), and nonlinear embeddings (e.g., ISOMAP, Locally Linear Embedding, Laplacian Eigenmaps, restricted Boltzmann machines, etc.).

K-means [107] is one of the most widely used unsupervised algorithms for clustering. Given a set of training examples (in a d-dimensional vector space $\mathbb{R}^d$), K-means learns a set of K centroids such that each example is assigned to the closest centroid. K-means can be viewed as an algorithm for vector quantization that partitions the input vector space (as Voronoi tessellation), where each centroid corresponds to the "center" of each partition [48]. A potential limitation of K-means is that the number of centroid points often needs to be very large as the input dimension grows. Thus, K-means may not be scalable to high-dimensional input data, especially when the number of examples is large.

Gaussian Mixture Model (GMM) [158] is also widely used for clustering and density estimation. The assumption for GMM is that the density (or distribution) of the input data is well modeled by a mixture of "clusters" (Gaussian distributions). In GMMs, it is easy to compute the posterior probability of mixture components. In the context of feature learning, this posterior can be used as a feature representation for a supervised learning task.[1] GMMs are closely related to K-means in that, GMMs can be viewed as "soft-assignment" of each input example to multiple clusters, whereas K-means performs "hard-assignment" of each input example to exactly one of the clusters. Similar to K-means, GMM may not be applicable when input data is high-dimensional and the number of examples is large.

Principal component analysis (PCA) is one of the most widely used algorithms for

[1]The assumption here can be stated as follows: If two examples x and y "belong" to the same Gaussian (i.e., x and y have high posterior probability of belonging to the same cluster), then there is a high chance that these two examples will have the same labels. This is somewhat consistent with the "cluster assumption" in semi-supervised learning. Here the cluster assumption (considering a binary classification problem without loss of generality) can be stated as: the decision boundary of the binary classification problem passes through the low density region of the data. In this setting, the positive and negative classes should be "well clustered," and it is sensible to use the posterior probability of mixture components as features.

dimensionality reduction and preprocessing. PCA extracts a set of basis vectors from unlabeled data, which collectively maximizes the variance of the projected subspace (spanned by the basis vectors). The basis vectors are obtained by the computing eigenvectors of the covariance matrix of the input data. Informally speaking, PCA retains the "variability" of the input data, and the projected coordinates can recover the original data quite accurately. However, PCA is limited in that (1) it is an orthogonal linear transformation, and (2) PCA is not optimal for discovering structures for input data with a highly non-Gaussian distribution.

Algorithms such as PCA can be viewed as matrix factorization methods. In general, matrix factorization attempts to decompose the input data $X \in \mathbb{R}^{d \times m}$ (where d is the dimension of the data, m is the number of examples, and each column of X corresponds to an individual input example) as $X \approx BS$ (where $B \in \mathbb{R}^{d \times n}, S \in \mathbb{R}^{n \times m}$, n is the number of basis vectors, and each column of B corresponds to an individual basis vector). Matrix factorization methods can be viewed as decomposing each input example as a linear combination of basis vectors. Therefore, each column of S represents "coefficients" of the basis vectors for each input example. There are several variants of matrix factorization, such as sparse matrix factorization [178] (e.g., assuming the entries of B and S are sparse), factor analysis [84], and probabilistic matrix factorization [144].

Other notable matrix factorization methods include independent component analysis (ICA) and non-negative matrix factorization (NMF). ICA is also a linear model which assumes the coefficients of the basis vectors are independent [40, 69]. ICA attempts to maximize non-Gaussianity (e.g., kurtosis) of the coefficients (or the linear filter output) for the basis vectors. ICA is widely used for blind source separation [69]. On the other hand, NMF [90] attempts to decompose the input data X (with non-negative values) into a product of two non-negative matrices (i.e., non-negative basis vectors and non-negative coefficients). NMF has been applied to text mining [165] and spectral data analysis [19].

In this thesis, we consider sparse coding [122] as an algorithm for unsupervised feature learning. Sparse coding can be also viewed as matrix factorization where the coefficients are assumed to be sparse. One advantage of sparse coding is that it can learn succinct representations of the input data, where each input example can be represented as a linear combination of small number of basis vectors. As a result, the basis vectors (also called "dictionary") are forced to capture salient patterns in the data, thus the basis vectors and the coefficients are often easy to interpret. Sparse coding allows for an overcomplete basis set (i.e., large dictionary), where the number of basis vectors is larger than the dimension of the input vector. Finally, sparse coding is robust to noise in the data or "change" in the input distributions. This is because the coefficients are not determined as a simple linear function of the input, but it goes through a sparsification process due to the sparse prior on the coefficients. This property makes sparse coding a good candidate in settings where (1) unlabeled data and labeled data have different distributions or (2) input data contain much noise. We will investigate the former setting (also called as self-taught learning framework in this thesis) in detail.

Compared to other generic matrix factorization methods (e.g., PCA, factor analysis, sparse matrix factorization, etc.), sparse coding fits better to the statistics of the natural stimulus data (e.g., natural images, sound, speech, and video). It is true that ICA can learn sparse representations as sparse coding. However, unlike sparse coding, the ICA output is simply a linear function of input. Therefore, ICA may not be robust to noise in the data or generalize well to different input distributions. NMF can be beneficial when the input data is non-negative, but NMF usually does not lead to sparse representations. However, we note that these matrix factorization methods may also be used for feature learning, depending on the input data.

The above-mentioned matrix factorization methods can be viewed as some sort of linear embeddings. A more flexible representation may be obtained by learning nonlinear embedding. In fact, numerous algorithms exist for nonlinear embedding, such as Kernel PCA [114], ISOMAP [155], Locally Linear Embedding [140], Laplacian Eigenmaps [13], Gaussian process latent variable models [85], autoencoders [16], restricted Boltzmann machines (RBMs) [149, 52], and deep belief networks (DBNs) [65]. (For a survey of nonlinear embedding algorithms, see [96] or [168].) In particular, autoencoders, RBMs, and DBNs can be used for building embeddings with multiple layers of abstractions, which will be further discussed in the next section.

1.3.4 Deep learning

In recent years, "deep learning" approaches have gained significant interest as a way of building hierarchical representations from unlabeled data [65, 16, 136, 134, 83]. Deep architectures attempt to learn hierarchical structures and seem promising in learning simple concepts first and then successfully building up more complex concepts by composing the simpler ones together. For example, Hinton et al. [65] proposed an algorithm based on learning individual layers of a hierarchical probabilistic graphical model from the bottom up. Bengio et al. [16] proposed a similarly greedy algorithm based on autoencoders. Ranzato et al. [136] developed an energy-based hierarchical algorithm, using a sequence of sparsified autoencoders/decoders. Here, I will briefly describe restricted Boltzmann machines and deep belief networks.

The restricted Boltzmann machine (RBM) is a bipartite undirected graphical model (Markov Random Field) with two layers, where a set of latent (or hidden) binary random variables are densely connected to a set of input (or visible) variables. However, there are no connections between hidden variables or between visible variables. The joint distribution is described as an energy function (as in standard log-linear models), which is simply a sum of products between the hidden, visible units, and the weights between the two layers. The RBM is an instance of products-of-experts (PoE) models [63, 125]. Roughly speaking, the RBM represents the input data using binary latent variables, and these binary latent variables or their posterior probabilities (given the input data) can be used as a feature representation.

The deep belief network is a graphical model with multiple layers. Hinton et al. proposed an algorithm for training deep belief networks, based on greedy layer-wise training of RBMs. In a DBN, the top two layers have undirected connections, and the connections below are directed (downward connections). Recently, many variants of this greedy learning algorithm have been successfully applied for classification tasks [65, 16, 136, 83], object recognition [135, 17, 1], dimensionality reduction [66], information retrieval [137, 159, 143], and human motion analysis [154, 153]. Although these learning algorithms show promise, the dimensionality of the input data is fairly limited (e.g., a few thousands at most); therefore, it is nontrivial to apply these algorithms to learn from high-dimensional data (e.g. tens or hundreds of thousands). This thesis will address this problem by scaling up deep belief networks via incorporating convolutional structures.

1.4 Proposed approach

The main goal of this thesis is to develop algorithms that can extract rich, high-level structures from unlabeled data. To learn from unlabeled data, we build probabilistic generative models so that we can perform probabilistic inference over this data.

This thesis will describe three ingredients to build such generative models. The first ingredient is sparsity. By incorporating sparsity, we can learn a succinct representation of the input data. Further, sparsity often allows us to learn readily interpretable and discriminative features. The second ingredient is hierarchy. We learn simple features first and then learn more abstract and hierarchical features by compositing simpler features. Specifically, I will present a method to incorporate sparsity into deep belief networks. The final ingredient is scaling up by convolutional structure. This thesis will present a method to incorporate convolutional structure into deep belief networks that achieves both computational efficiency and local invariance. As a result, this algorithm can learn high-level feature representations from unlabeled data that can be very useful for many machine learning tasks. To summarize, sparsity, hierarchy, and scaling up by convolutional structure will be the three ideas that this thesis combines. In more general settings where convolutional structures do not apply, the four principles, sparsity, hierarchy, invariance, and computational efficiency, can be the main ingredients for learning good feature representations.

For intuitive understanding, I provide a preview of the key results. For example, this thesis presents algorithms that allow us to learn high-level features from images, as shown in Figure 1.2. Specifically, starting from pixel-level representation, the presented algorithm will learn "edge detectors" in first layer representation; these edge detectors turn out to be similar to what many other computer vision features represent. Furthermore, I will show that the presented algorithm will learn higher-level features going beyond this level, such as learning object parts in the intermediate layer and full objects in the top layer. Moreover, I will show that the learned high-level features can directly improve machine learning algorithms. Finally, these methods are not limited to computer vision, but can also

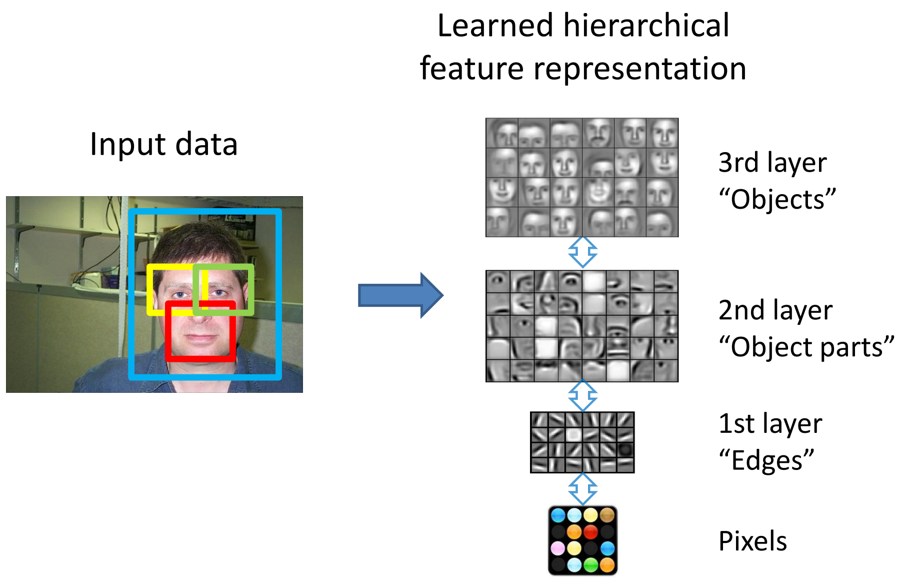


Figure 1.2: Illustration of the main results. In this thesis, we explore algorithms for building hierarchical representations from unlabeled data. We then demonstrate the usefulness of these learned feature representations in a number of classification tasks.

apply to other domains. Specifically, we show that the same algorithm can be applied to audio data, achieving excellent performance in audio classification tasks.

1.5 Summary of contributions

The main contribution of my thesis are as follows:

- We formulated sparse coding problem as a well-defined objective, proposed efficient algorithms, and showed how it can be applied to machine learning tasks. In addition, we developed generalized sparse coding algorithms that extend the range of data types that sparse coding can be applied.
- By developing sparse deep belief networks, this work first incorporated sparsity into deep belief networks. More specifically, we introduced sparsity regularization into

deep networks, which are useful for learning succinct and discriminative features in many applications.

- We developed convolutional deep belief networks. Our method is the first algorithm that scales up the unsupervised deep learning algorithm into large sized images (e.g., 200x200 pixels). We also demonstrated excellent performance in computer vision tasks, such as object recognition and multi-class segmentation.
- We applied convolutional deep belief networks for audio recognition and demonstrated state-of-the-art performance in speech and music classification tasks.

1.6 First published appearances of the described contributions

Most of the contributions described in this thesis have first appeared as various publications. The following list describes the representative publications roughly corresponding to the chapters in this thesis:

- Chapter 2: H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems 19*, 2007.

- Chapter 3: H. Lee, R. Raina, A. Teichman, and A. Y. Ng. Exponential family sparse coding with applications to self-taught learning. In *Proceedings of the international Joint Conference on Artifical Intelligence*, 2009.

- Chapter 4: H. Lee, C. Ekanadham, and A. Y. Ng. Sparse deep belief network model for visual area V2. In *Advances in Neural Information Processing Systems 20*, 2008.

- Chapter 5: H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the International Conference on Machine Learning*, 2009.

- Chapter 6: H. Lee, P. Pham, Y. Largman, and A. Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems 22*, 2009.

1.7 Organization

This thesis is organized as follows. Chapter 2 will present efficient sparse coding algorithms that can be used to learn sparse representations from the input data, which can also help the supervised classification tasks. Chapter 3 will present an extension of the original sparse coding formulation (which assumes real-valued inputs) to more general types of input data (e.g., binary or integer values), with application to text classification and 3d point-cloud classification. Chapter 4 will present a way of incorporating sparsity into hierarchical representations, building upon deep learning algorithms. Chapter 5 will present an algorithm that scales the deep belief networks to high-dimensional data, such as from small (e.g., 14x14 pixels) image patches to realistic-sized (e.g., 200x200 pixels) images, with extensive experimental results in computer vision tasks. Chapter 6 will present an application of the same algorithm to audio data, achieving excellent performance in speech and music classification tasks. Chapter 7 will provide a summary and the conclusion.

Chapter 2

Efficient Sparse Coding Algorithms

2.1 Introduction

Originated from computational neuroscience, sparse coding provides a class of algorithms for finding succinct representations of stimuli; given only unlabeled input data, it learns basis functions that capture higher-level features in the data. When a sparse coding algorithm is applied to natural images, the learned bases resemble the receptive fields of neurons in the visual cortex [122, 123]. In addition, sparse coding produces localized bases when applied to other natural stimuli such as speech and video [103, 121]. Unlike some other unsupervised learning techniques such as PCA, sparse coding can be applied to learning overcomplete basis sets, in which the number of bases is greater than the input dimension. Sparse coding can also model inhibition between the bases by sparsifying their activations. Similar properties have been observed in biological neurons, thus making sparse coding a plausible model of the visual cortex [123, 124].

Despite the rich promise of sparse coding models, we believe that their development has been hampered by their expensive computational cost. In particular, learning large, highly overcomplete representations has been extremely expensive. In this chapter, we develop a class of efficient sparse coding algorithms that are based on alternating optimization over two subsets of the variables. The optimization problems over each of the subsets of variables are convex; in particular, the optimization over the first subset is an L_1-regularized least squares problem; the optimization over the second subset of variables is an L_2-constrained least squares problem. We describe each algorithm and empirically analyze their performance. Our method is significantly faster than the existing algorithms and allows us to efficiently learn large overcomplete bases from natural images. We demonstrate that the resulting learned bases exhibit (i) end-stopping [145] and (ii) modulation by stimuli outside the classical receptive field [31]. Thus, sparse coding may also provide a partial explanation for these phenomena in V1 neurons. Further, in related work [131], we show that the learned succinct representation captures higher-level features that can then be applied to supervised classification tasks.

2.2 Formulation

Sparse coding is a method for discovering salient basis vectors automatically using only unlabeled data. The goal of sparse coding is to represent input vectors approximately as a weighted linear combination of a small number of (unknown) "basis vectors." Concretely, each input vector $\vec{x} \in \mathbb{R}^d$ is succinctly represented using basis vectors $\vec{b}_1, \ldots, \vec{b}_n \in \mathbb{R}^d$ and a sparse vector of weights or "coefficients" $\vec{s} \in \mathbb{R}^n$ such that $\vec{x} \approx \sum_j \vec{b}_j s_j$. Here, the coefficients $\vec{s} \in \mathbb{R}^n$ are called the "activations" corresponding to the input $\vec{x}$, and are encouraged to be sparse (i.e., to have many of the activations exactly equal to zero). The basis set can be overcomplete ($n > d$), and can thus capture a large number of patterns in the input data. Sparse coding often learns a compact, succinct representation because only a small number of coefficients are used to represent the data. At the same time, the basis vectors thus capture salient patterns in the input data.

A standard generative model for sparse coding assumes that the reconstruction error $\vec{x} - \sum_j \vec{b}_j s_j$ is distributed as a zero-mean Gaussian distribution with covariance $\sigma^2 I$. To favor sparse coefficients, the prior distribution for each coefficient s_j is defined as: $P(s_j) \propto \exp(-\beta\phi(s_j))$, where $\phi(\cdot)$ is a sparsity function and β is a constant. For example, we can use one of the following:

$$\phi(s_j) = \begin{cases} \|s_j\|_1 & (\mathrm{L}_1 \text{ penalty function}) \\ (s_j^2+\epsilon)^{\frac{1}{2}} & (\text{epsilonL}_1 \text{ penalty function}) \\ \log(1+s_j^2) & (\log \text{ penalty function}). \end{cases} \tag{2.1}$$

In this chapter, we will use the L_1 penalty unless otherwise mentioned; L_1 regularization is known to produce sparse coefficients and can be robust to irrelevant features [118].

Consider a training set of m unlabeled input vectors $\vec{x}^{(1)}, ..., \vec{x}^{(m)}$, and their (unknown) corresponding coefficients $\vec{s}^{(1)}, ..., \vec{s}^{(m)}$. The *maximum a posteriori* estimate of the bases and coefficients, assuming a uniform prior on the bases, is the solution to the following optimization problem:

$$\text{maximize}_{\{\vec{b}_j\},\{\vec{s}^{(i)}\}} \prod_i \; P\left(\vec{x}^{(i)}|\{\vec{b}_j\},\{\vec{s}^{(i)}\}\right) \, P\left(\vec{s}^{(i)}\right). \tag{2.2}$$

This further reduces to the following optimization problem for sparse coding:

$$\begin{aligned} \text{minimize}_{\{\vec{b}_j\},\{\vec{s}^{(i)}\}} \quad & \textstyle\sum_{i=1}^m \frac{1}{2\sigma^2}\|\vec{x}^{(i)} - \sum_{j=1}^n \vec{b}_j s_j^{(i)}\|^2 + \beta \sum_{i=1}^m \sum_{j=1}^n \phi(s_j^{(i)}) & (2.3) \\ \text{subject to} \quad & \|\vec{b}_j\|^2 \le c, \; \forall j = 1, ..., n. & (2.4) \end{aligned}$$

Here, we imposed a norm constraint for bases: $\|\vec{b}_j\|^2 \le c, \forall j = 1, ..., n$ for some constant c. This norm constraint is necessary because, otherwise, there always exists a linear transformation of $\vec{b}_j$'s and $\vec{s}^{(i)}$'s which keeps $\sum_{j=1}^n \vec{b}_j s_j^{(i)}$ unchanged, while making $s_j^{(i)}$'s approach zero.

This problem can be written more concisely in matrix form: let $X \in \mathbb{R}^{d\times m}$ be the input matrix (each column is an input vector), let $B \in \mathbb{R}^{d\times n}$ be the basis matrix (each column is a basis vector), and let $S \in \mathbb{R}^{n\times m}$ be the coefficient matrix (each column is a coefficient vector). Then, the optimization problem above can be written as:

$$\begin{aligned} \text{minimize}_{B,S} \quad & \textstyle\frac{1}{2\sigma^2}\|X - BS\|_\mathrm{F}^2 + \beta \sum_{i,j} \phi(S_{i,j}) & (2.5) \\ \text{subject to} \quad & \textstyle\sum_i B_{i,j}^2 \le c, \; \forall j = 1, ..., n. \end{aligned}$$

Assuming the use of either L_1 penalty or $\mathrm{epsilon}\mathrm{L}_1$ penalty as the sparsity function, the optimization problem is convex in B (while holding S fixed) and convex in S (while holding B fixed), but not convex in both simultaneously. In this chapter, we iteratively optimize the above objective by alternatingly optimizing with respect to B (bases) and S (coefficients) while holding the other fixed.

For learning the bases B, the optimization problem is a least squares problem with quadratic constraints. There are several approaches to solving this problem, such as generic convex optimization solvers (e.g., QCQP solver) as well as gradient descent using iterative projections [32]. However, generic convex optimization solvers are too slow to be applicable to this problem, and gradient descent using iterative projections often shows slow convergence. In this chapter, we derive and solve the Lagrange dual, and show that this approach is much more efficient than gradient-based methods.

For learning the coefficients S, the optimization problem is equivalent to a regularized least squares problem. For some differentiable sparsity functions, we can use gradient-based methods (e.g., conjugate gradient). However, for the L_1 sparsity function, the objective is not continuously differentiable and the most straightforward gradient-based methods are difficult to apply. In this case, the following approaches have been used: generic QP solvers (e.g., CVX [59, 58]), Chen et al.'s interior point method [35], a modification of least angle regression (LARS) [49], or grafting [128]. In this chapter, we present a new algorithm for solving the L_1-regularized least squares problem and show that it is more efficient for learning sparse coding bases.

In previous work, Olshausen and Field [122] presented a sparse coding algorithm with the same form of objective function as in the Equation (2.3). However, they used gradient-based algorithms to solve both for the coefficients and the basis vectors, respectively. Specifically, they used conjugate gradient for solving the coefficients. However, using conjugate gradient is not efficient for the L_1 sparsity function. Also, a non-convex sparsity function (the log penalty) was used in their implementation, so conjugate gradient may get stuck in local optima. For solving the basis vectors, they applied a single gradient descent step followed by a heuristic for normalization. In detail, to prevent the norm of the basis vectors going to infinity, they used a heuristic which retains the variance of the coefficients (for every basis) at the same level. Overall, this algorithm is slow and does not scale well for a large number of basis vectors. In this chapter, we present new algorithms for sparse coding that is significantly faster than Olshausen and Field's method.

2.3 L$_1$-regularized least squares: The feature-sign search algorithm

Consider solving the optimization problem (2.3) with an L$_1$ penalty over the coefficients $\{s_j^{(i)}\}$ while keeping the bases fixed. This problem can be solved by optimizing over each $\vec{s}^{(i)}$ individually:

$$\text{minimize}_{\vec{s}^{(i)}} \|\vec{x}^{(i)} - \sum_j \vec{b}_j s_j^{(i)}\|^2 + (2\sigma^2\beta)\sum_j |s_j^{(i)}|. \tag{2.6}$$

Notice now that if we know the signs (positive, zero, or negative) of the $s_j^{(i)}$'s at the optimal value, we can replace each of the terms $|s_j^{(i)}|$ with either $s_j^{(i)}$ (if $s_j^{(i)} > 0$), $-s_j^{(i)}$ (if $s_j^{(i)} < 0$), or 0 (if $s_j^{(i)} = 0$). Considering only nonzero coefficients, this reduces (2.6) to a standard, unconstrained quadratic optimization problem (QP), which can be solved analytically and efficiently. Our algorithm, therefore, tries to search for, or "guess," the signs of the coefficients $s_j^{(i)}$; given any such guess, we can efficiently solve the resulting unconstrained QP. Further, the algorithm systematically refines the guess if it turns out to be initially incorrect.

To simplify notation, we present the algorithm for the following equivalent optimization problem (e.g., we use x and s to denote $\vec{x}$ and $\vec{s}$, respectively):

$$\text{minimize}_s f(s) \equiv \|x - Bs\|^2 + \gamma\|s\|_1, \tag{2.7}$$

where γ is a constant. The feature-sign search algorithm is shown in Algorithm 1. It maintains an *active set* of potentially nonzero coefficients and their corresponding signs—all other coefficients must be zero—and systematically searches for the optimal active set and coefficient signs.

The algorithm proceeds in a series of "feature-sign steps": on each step, it is given a current guess for the active set and the signs, and it computes the analytical solution

Algorithm 1 Feature-sign search algorithm

```
1: Initialize $s := \vec{0}, \theta := \vec{0}$, and *active set* $:= \{\}$, where $\theta_i \in \{-1, 0, 1\}$ denotes $\text{sign}(s_i)$.
2: From zero coefficients of $s$, select $i = \arg\max_j \frac{\partial \|x-Bs\|^2}{\partial s_j}$.
  Activate $s_i$ (add $i$ to the *active set*) only if it locally improves the objective, namely:
    If $\frac{\partial \|x-Bs\|^2}{\partial s_i} > \gamma$, then set $\theta_i := -1$, *active set* $:= \{i\} \cup$ *active set*.
    If $\frac{\partial \|x-Bs\|^2}{\partial s_i} < -\gamma$, then set $\theta_i := 1$, *active set* $:= \{i\} \cup$ *active set*.
3: Feature-sign step:
  Let $\hat{B}$ be a submatrix of $B$ that contains only the columns corresponding to the *active set*.
  Let $\hat{s}$ and $\hat{\theta}$ be subvectors of $s$ and $\theta$ corresponding to the *active set*.
  Compute the analytical solution to the resulting unconstrained QP:
    $\hat{s}_{new} := \arg\min_{\hat{s}} \|x - \hat{B}\hat{s}\|^2 + \gamma\hat{\theta}^\top \hat{s} = (\hat{B}^\top \hat{B})^{-1}(\hat{B}^\top x - \gamma\hat{\theta}/2)$.
  Perform a discrete line search on the closed line segment from $\hat{s}$ to $\hat{s}_{new}$:
    Check the objective value at $\hat{s}_{new}$ and all points where any coefficient changes sign.
    Update $\hat{s}$ (and the corresponding entries in $s$) to the point with the lowest objective value.
  Remove zero coefficients of $\hat{s}$ from the *active set* and update $\theta := \text{sign}(s)$.
4: Check the optimality conditions:
  (a) Optimality condition for nonzero coefficients: $\frac{\partial \|x-Bs\|^2}{\partial s_j} + \gamma \, \text{sign}(s_j) = 0, \forall s_j \neq 0$
    If condition (a) is not satisfied, go to Step 3 (without any new activation);
    else check condition (b).
  (b) Optimality condition for zero coefficients: $\left|\frac{\partial \|x-Bs\|^2}{\partial s_j}\right| \leq \gamma, \forall s_j = 0$
    If condition (b) is not satisfied, go to Step 2; otherwise return $s$ as the solution.
```

$\hat{s}_{new}$ to the resulting unconstrained QP; it then updates the solution, the active set and the signs using an efficient discrete line search between the current solution and $\hat{s}_{new}$ (details in Algorithm 1).[1] We will show that each such step reduces the objective $f(s)$, and that the overall algorithm always converges to the optimal solution.

To sketch the proof of convergence, let a coefficient vector s be called *consistent* with a given active set and sign vector θ if the following two conditions hold for all i: (i) If i is in the active set, then $\text{sign}(s_i) = \theta_i$, and, (ii) If i is not in the active set, then $s_i = 0$.

[1]A technical detail has been omitted from the algorithm for simplicity, as we have never observed it in practice. In Step 3 of the algorithm, in case $\hat{B}^\top \hat{B}$ becomes singular, we can check if $q \equiv \hat{B}^\top x - \gamma\hat{\theta}/2 \in \mathcal{R}(\hat{B}^\top \hat{B})$. If yes, we can replace the inverse with the pseudo-inverse to minimize the unconstrained QP; otherwise, we can update $\hat{s}$ to the first zero-crossing along any direction z such that $z \in \mathcal{N}(\hat{B}^\top \hat{B}), z^\top q \neq 0$. Both these steps are still guaranteed to reduce the objective; thus, the proof of convergence is unchanged.

Lemma 2.3.1. *Consider optimization problem (2.7) augmented with the additional constraint that s is consistent with a given active set and sign vector. Then, if the current coefficients s_c are consistent with the active set and sign vector, but are not optimal for the augmented problem at the start of Step 3, the feature-sign step is guaranteed to strictly reduce the objective.*

Proof (sketch): Let $\hat{s}_c$ be the subvector of s_c corresponding to coefficients in the given active set. In Step 3, consider a smooth quadratic function $\tilde{f}(\hat{s}) \equiv \|x-\hat{B}\hat{s}\|^2+\gamma\hat{\theta}^\top\hat{s}$. Since $\hat{s}_c$ is not an optimal point of $\tilde{f}$, we have $\tilde{f}(\hat{s}_{new}) < \tilde{f}(\hat{s}_c)$. Now consider the two possible cases: (i) if $\hat{s}_{new}$ is consistent with the given active set and sign vector, updating $\hat{s} := \hat{s}_{new}$ strictly decreases the objective; (ii) if $\hat{s}_{new}$ is not consistent with the given active set and sign vector, let $\hat{s}_d$ be the first zero-crossing point (where any coefficient changes its sign) on a line segment from $\hat{s}_c$ to $\hat{s}_{new}$, then clearly $\hat{s}_c \neq \hat{s}_d$, and $\tilde{f}(\hat{s}_d) < \tilde{f}(\hat{s}_c)$ by convexity of $\tilde{f}$, thus we finally have $f(\hat{s}_d) = \tilde{f}(\hat{s}_d) < \tilde{f}(\hat{s}_c) = f(\hat{s}_c)$.[2] Therefore, the discrete line search described in Step 3 ensures a decrease in the objective value. □

Lemma 2.3.2. *Consider optimization problem (2.7) augmented with the additional constraint that s is consistent with a given active set and sign vector. If the coefficients s_c at the start of Step 2 are optimal for the augmented problem, but are not optimal for problem (2.7), the feature-sign step is guaranteed to strictly reduce the objective.*

Proof (sketch): Since s_c is optimal for the augmented problem, it satisfies optimality condition (a), but not (b); thus, in Step 2, there is some i, such that $\left|\frac{\partial\|x-Bs\|^2}{\partial s_i}\right| > \gamma$; this i-th coefficient is activated, and i is added to the active set. In Step 3, consider the smooth quadratic function $\tilde{f}(\hat{s}) \equiv \|x-\hat{B}\hat{s}\|^2+\gamma\hat{\theta}^\top\hat{s}$. Observe that (i) since a Taylor expansion of $\tilde{f}$ around $\hat{s} = \hat{s}_c$ has a first order term in s_i only (using condition 4(a) for the other coefficients), we have that any direction that locally decreases $\tilde{f}(\hat{s})$ must be consistent with the sign of the activated s_i, and, (ii) since $\hat{s}_c$ is not an optimal point of $\tilde{f}(\hat{s})$, $\tilde{f}(\hat{s})$ must decrease locally near $\hat{s} = \hat{s}_c$ along the direction from $\hat{s}_c$ to $\hat{s}_{new}$. From (i) and (ii), the line search direction $\hat{s}_c$ to $\hat{s}_{new}$ must be consistent with the sign of the activated s_i. Finally, since $\tilde{f}(\hat{s}) = f(\hat{s})$ when $\hat{s}$ is consistent with the active set, either $\hat{s}_{new}$ is consistent, or

[2]To simplify notation, we reuse $f(\cdot)$ even for subvectors such as $\hat{s}$; in the case of $f(\hat{s})$, we consider only the coefficients in $\hat{s}$ as variables, and all coefficients not in the subvector can be assumed constant at zero.

the first zero-crossing from $\hat{s}_c$ to $\hat{s}_{new}$ has a lower objective value (similar argument to Lemma 2.3.1). □

Theorem 2.3.3. *The feature-sign search algorithm converges to a global optimum of the optimization problem (2.7) in a finite number of steps.*

Proof (sketch): From the above lemmas, it follows that the feature-sign steps always strictly reduce the objective $f(s)$. At the start of Step 2, s either satisfies optimality condition 4(a) or is $\vec{0}$; in either case, s is consistent with the current active set and sign vector, and must be optimal for the augmented problem described in the above lemmas. Since the number of all possible active sets and coefficient signs is finite, and since no pair can be repeated (because the objective value is strictly decreasing), the outer loop of Steps 2–4(b) cannot repeat indefinitely. Now, it suffices to show that a finite number of steps is needed to reach Step 4(b) from Step 2. This is true because the inner loop of Steps 3–4(a) always results in either an exit to Step 4(b) or a decrease in the size of the active set. □

While the above convergence proof does not provide a strong guarantee on the convergence rate, we often observe that feature-sign search algorithm finds optimal solution very efficiently when the optimal solution is sparse. Further, note that initialization with arbitrary starting points requires a small modification: after initializing θ and the active set with a given initial solution, we need to start with Step 3 instead of Step 1. Specifically, if the algorithm terminates without reaching Step 2, we are done; otherwise, once the algorithm reaches Step 2, the same argument in the proof applies. When the initial solution is near the optimal solution, feature-sign search can often obtain the optimal solution more quickly than when starting from $\vec{0}$.

2.4 Learning bases using the Lagrange dual

In this subsection, we present a method for solving optimization problem (2.5) over bases B given fixed coefficients S. This reduces to the following problem:

$$\begin{array}{rc} \text{minimize} & \|X - BS\|_{\text{F}}^2 \\ \text{subject to} & \sum_{i=1}^{d} B_{i,j}^2 \leq c, \forall j = 1, ..., n. \end{array} \tag{2.8}$$

This is a least squares problem with quadratic constraints. In general, this constrained optimization problem can be solved using gradient descent with iterative projection [32]. However, it can be much more efficiently solved using a Lagrange dual. First, consider the Lagrangian:

$$\mathcal{L}(B, \vec{\lambda}) = \mathrm{trace}\left((X - BS)^\top (X - BS)\right) + \sum_{j=1}^{n} \lambda_j \left(\sum_{i=1}^{d} B_{i,j}^2 - c\right), \tag{2.9}$$

where each $\lambda_j \geq 0$ is a dual variable. Minimizing over B analytically, we obtain the Lagrange dual:

$$\mathcal{D}(\vec{\lambda}) = \min_B \mathcal{L}(B, \vec{\lambda}) = \mathrm{trace}\left(X^\top X - XS^\top (SS^\top + \Lambda)^{-1} (XS^\top)^\top - c\,\Lambda\right), \tag{2.10}$$

where $\Lambda = \mathrm{diag}(\vec{\lambda})$. The gradient and Hessian of $\mathcal{D}(\vec{\lambda})$ are computed as follows:

$$\frac{\partial \mathcal{D}(\vec{\lambda})}{\partial \lambda_i} = \|XS^\top (SS^\top + \Lambda)^{-1} e_i\|^2 - c \tag{2.11}$$

$$\frac{\partial^2 \mathcal{D}(\vec{\lambda})}{\partial \lambda_i \partial \lambda_j} = -2 \left((SS^\top + \Lambda)^{-1} (XS^\top)^\top XS^\top (SS^\top + \Lambda)^{-1}\right)_{i,j} \left((SS^\top + \Lambda)^{-1}\right)_{i,j}, \tag{2.12}$$

where $e_i \in \mathbb{R}^n$ is the i-th unit vector. Now, we can optimize the Lagrange dual (2.10) using Newton's method or conjugate gradient. After maximizing $\mathcal{D}(\vec{\lambda})$, we obtain the optimal bases B as follows:

$$B^\top = (SS^\top + \Lambda)^{-1} (XS^\top)^\top. \tag{2.13}$$

The advantage of solving the dual is that it uses significantly fewer optimization variables than the primal. For example, optimizing $B \in \mathbb{R}^{1,000 \times 1,000}$ requires only 1,000 dual variables. Note that the dual formulation is independent of the sparsity function (e.g., L_1, $\mathrm{epsilonL}_1$, or other sparsity function), and can be extended to other similar models such as "topographic" cells [68].[3]

[3]The sparsity penalty for topographic cells can be written as $\sum_l \phi((\sum_{j \in \mathrm{cell}\, l} s_j^2)^{\frac{1}{2}})$, where $\phi(\cdot)$ is a sparsity function and cell l is a topographic cell (e.g., group of 'neighboring' bases in 2-D torus representation).

	natural image	speech	stereo	video
	196×512	500×200	288×400	512×200
Feature-sign	2.16 (0)	0.58 (0)	1.72 (0)	0.83 (0)
LARS	3.62 (0)	1.28 (0)	4.02 (0)	1.98 (0)
Grafting	13.39 (7e-4)	4.69 (4e-6)	11.12 (5e-4)	5.88 (2e-4)
Chen et al.'s	88.61 (8e-5)	47.49 (8e-5)	66.62 (3e-4)	47.00 (2e-4)
QP solver (CVX)	387.90 (4e-9)	1108.71 (1e-8)	538.72 (7e-9)	1219.80 (1e-8)

Table 2.1: The running time in seconds (and the relative error in parentheses) for coefficient-learning algorithms applied to different natural stimulus datasets. For each dataset, the input dimension d and the number of bases n are specified as $d \times n$. The relative error for an algorithm was defined as $(f_{obj} - f^*)/f^*$, where f_{obj} is the final objective value attained by that algorithm, and f^* is the best objective value attained among all the algorithms.

2.5 Experimental results

2.5.1 The feature-sign search algorithm

We evaluated the performance of our algorithms on four natural stimulus datasets: natural images (`http://redwood.berkeley.edu/bruno/sparsenet`), speech (TIDIGITS dataset [101]), stereo images (`http://www.pokescope.com/geology/geology.html`), and natural image videos (`http://hlab.phys.rug.nl/imlib/`). All experiments were conducted on a Linux machine with AMD Opteron 2GHz CPU and 2GB RAM.

First, we evaluated the feature-sign search algorithm for learning coefficients with the L_1 sparsity function. We compared the running time and accuracy to previous state-of-the-art algorithms: a generic QP solver (CVX [59]), a modified version of LARS [49] with early stopping,[4] grafting [128], and Chen et al.'s interior point method [35];[5] all the algorithms were implemented in MATLAB. For each dataset, we used a test set of 100 input vectors and measured the running time and the objective function at convergence.

[4]LARS (with LASSO modification) provides the entire regularization path with discrete L_1-norm constraints; we further modified the algorithm so that it stops upon finding the optimal solution of the Equation (2.6).

[5]MATLAB code is available at `http://www-stat.stanford.edu/~atomizer/`.

For each dataset/algorithm combination, we report the average running time over 20 trials. Table 2.1 shows both the running time and accuracy (measured by the relative error in the final objective value) of different coefficient learning algorithms. Over all datasets, feature-sign search achieved the best objective values as well as the shortest running times. Feature-sign search and modified LARS produced more accurate solutions than the other methods. Specifically, a general-purpose QP package (such as CVX) does not explicitly take the sparsity of the solutions into account. Thus, its solution tends to have many very small nonzero coefficients; as a result, the objective values obtained from CVX were always slightly worse than those obtained from feature-sign search or LARS. Further, feature-sign search was an order of magnitude faster than both Chen et al.'s algorithm and the generic QP solver, and it was also significantly faster than modified LARS and grafting. In addition, feature-sign search has the crucial advantage that it can be initialized with arbitrary starting coefficients (unlike LARS); we will demonstrate that feature-sign search leads to even further speedup over LARS when applied to iterative coefficient learning.

2.5.2 Total time for learning bases

The Lagrange dual method for one basis learning iteration was much faster than gradient descent with iterative projections. Below, we present results for the overall time taken by sparse coding for learning bases from natural stimulus datasets.

We evaluated different combinations of coefficient learning and basis learning algorithms: the fastest coefficient learning methods from our experiments (feature-sign search, modified LARS and grafting for the L_1 sparsity function, and conjugate gradient for the $\mathrm{epsilonL}_1$ sparsity function) and the basis learning methods (gradient descent with iterative projection and the Lagrange dual formulation). We used a training set of 1,000 input vectors for each of the four natural stimulus datasets. We initialized the bases randomly and ran each algorithm combination (by alternatingly optimizing the coefficients and the bases) until convergence. In detail, we ran each algorithm combination until the relative change of the objective per iteration became less than 10^{-6} (i.e., $|(f_{new} - f_{old})/f_{old}| < 10^{-6}$). To compute the running time to convergence, we first computed the "optimal" (minimum) objective value achieved by any algorithm combination. Then, for each combination, we

L_1 sparsity function

Coeff. / Basis learning	natural image	speech	stereo	video
Feature-sign / LagDual	**260.0**	**248.2**	**438.2**	**186.6**
Feature-sign / GradDesc	1093.9	1280.3	950.6	933.2
LARS / LagDual	666.7	1697.7	1342.7	1254.6
LARS / GradDesc	13085.1	17219.0	12174.6	11022.8
Grafting / LagDual	720.5	1025.5	3006.0	1340.5
Grafting / GradDesc	2767.9	8670.8	6203.3	3681.9

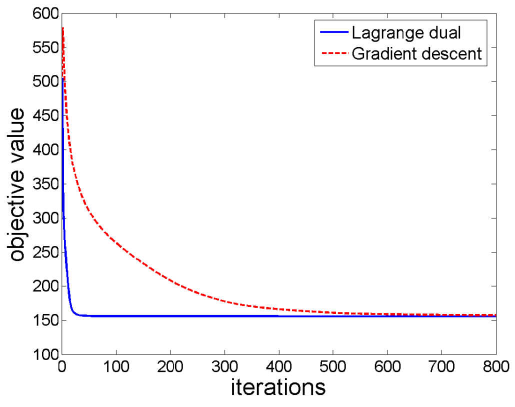
$\mathrm{epsilonL}_1$ sparsity function

Coeff. / Basis learning	natural image	speech	stereo	video
ConjGrad / LagDual	**1286.6**	**544.4**	**1942.4**	**1461.9**
ConjGrad / GradDesc	5047.3	11939.5	3435.1	2479.2

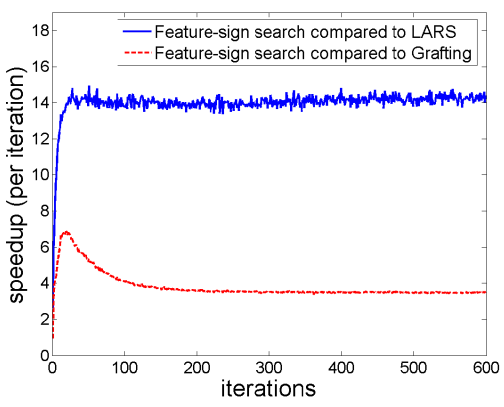
Table 2.2: The running time (in seconds) for different algorithm combinations using different sparsity functions.

defined the convergence point as the point at which the objective value reaches within 1% relative error of the observed "optimal" objective value. The running time measured is the time taken to reach this convergence point. We truncated the running time if the optimization did not converge within 60,000 seconds.

Table 2.2 shows the running times for different algorithm combinations. First, we observe that the Lagrange dual method significantly outperformed gradient descent with iterative projections for both L_1 and $\mathrm{epsilonL}_1$ sparsity; a typical convergence pattern is shown in Figure 2.1(a). Second, we observe that for L_1 sparsity, feature-sign search significantly outperformed both modified LARS and grafting. We also evaluated a generic conjugate gradient implementation based on the L_1 sparsity function; however, it did not converge even after 60,000 seconds. Figure 2.1(b) shows the running time per iteration for modified LARS and grafting as a multiple of that for feature-sign search (using the same gradient descent algorithm for basis learning), demonstrating significant efficiency gains at later iterations; note that feature-sign search (and grafting) can be initialized with the coefficients obtained in the previous iteration, whereas modified LARS cannot. This result



(a)



(b)

Figure 2.1: Demonstration of speedup. Left: Comparison of convergence between the Lagrange dual method and gradient descent for learning bases. Right: The running time per iteration for modified LARS and grafting as a multiple of the running time per iteration for feature-sign search.

demonstrates that feature-sign search is particularly efficient for the alternating optimization procedure, such as learning sparse coding bases.

2.5.3 Learning highly overcomplete natural image bases

Using our efficient algorithms, we were able to learn highly overcomplete bases of natural images. For example, we were able to learn a set of 1,024 bases (each 14×14 pixels) in about 2 hours and a set of 2,000 bases (each 20×20 pixels) in about 10 hours. We used Lagrange dual formulation for learning bases, and both conjugate gradient with $\mathrm{epsilonL}_1$ sparsity as well as the feature-sign search with L_1 sparsity for learning coefficients. The bases learned from both methods showed qualitatively similar receptive fields. The bases shown in the Figure 2.2 were learned using $\mathrm{epsilonL}_1$ sparsity function and 4,000 input image patches randomly sampled for every iteration. In contrast, the gradient descent method for basis learning did not result in any reasonable bases even after running for 24 hours.

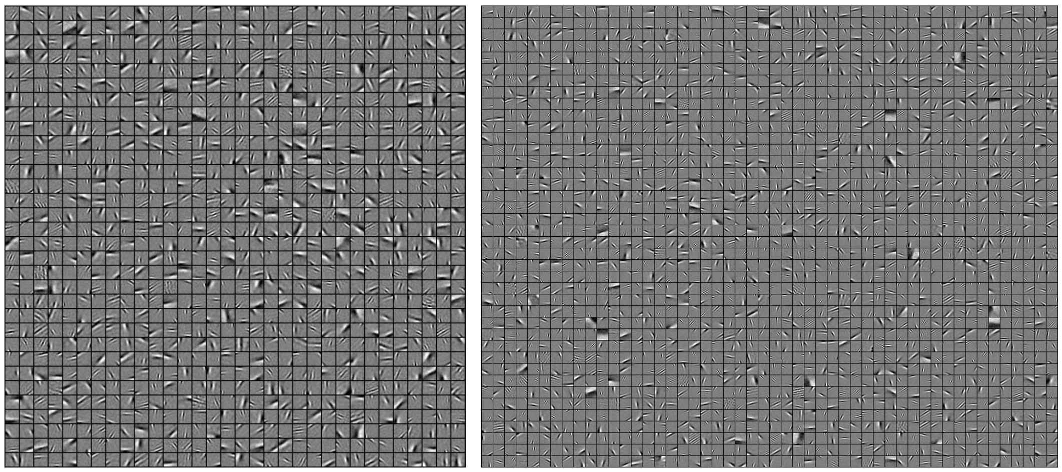

(a) 1,024 bases (each 14×14 pixels) (b) 2,000 bases (each 20×20 pixels)

Figure 2.2: Learned overcomplete natural image bases.

2.5.4 Replicating complex neuroscience phenomena

Several complex phenomena of V1 neural responses are not well explained by simple linear models (in which the response is a linear function of the input). For instance, many visual neurons display “end-stopping,” in which the neuron’s response to a bar image of optimal orientation and placement is actually suppressed as the bar length exceeds an optimal length [145]. Sparse coding can model the interaction (inhibition) between the bases (neurons) by sparsifying their coefficients (activations), and our algorithms enable these phenomena to be tested with highly overcomplete bases. In addition, center-surround non-classical receptive field effects explain inhibition between biological V1 neurons that have similarly-oriented, spatially-close receptive fields (see [31] for details). In the previous publication of this chapter [91], we applied the sparse coding algorithms to learn highly overcomplete (4x) bases from natural images and demonstrated that the inferred sparse codes exhibit end-stopping and non-classical receptive field surround suppression. Thus, sparse coding may also provide a partial explanation for these phenomena in V1 neurons.

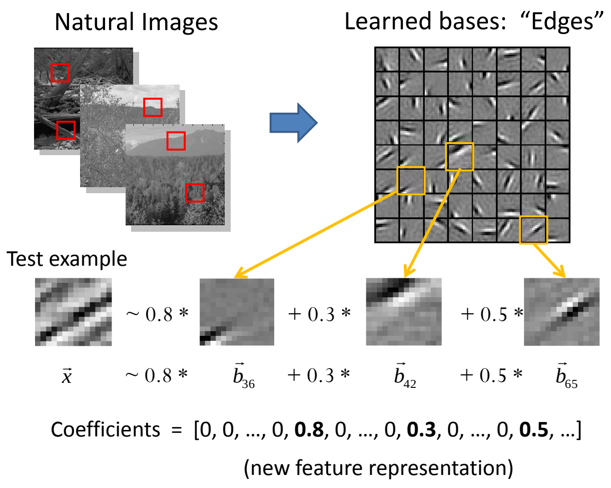

Natural Images
Learned bases: "Edges"
Test example
~ 0.8 *
+ 0.3 *
+ 0.5 *
$\vec{x}$ ~ 0.8 * $\vec{b}_{36}$ + 0.3 * $\vec{b}_{42}$ + 0.5 * $\vec{b}_{65}$
Coefficients = [0, 0, …, 0, **0.8**, 0, …, 0, **0.3**, 0, …, 0, **0.5**, …]
(new feature representation)

Figure 2.3: Illustration of sparse coding applied to natural images. In the upper left, small red squares represent 14x14 image patches randomly sampled from natural images. The upper right shows the learned bases from natural images via sparse coding. The below illustrates of how sparse coding decomposes a new 14x14 image patch into a linear combination of a few basis vectors. The resulting sparse coefficients can be used as features representing 14x14 pixels.

2.6 Application to self-taught learning

Sparse coding forces each input $\vec{x}$ to be "explained" using a small number of basis vectors, and the activations often capture certain higher-level features of the input. For example, when the inputs $\vec{x}$ consist of small images (represented as vectors of pixel intensities), the basis vectors learned using Equation (2.2) capture various kinds of edges, as shown in Figure 2.3. Effectively, the procedure converts the pixel-based representation $\vec{x}$ to a succinct, slightly higher-level, edge-based representation $\vec{s}$, and this new feature representation can be useful for a variety of tasks.

In related work [131], we apply this to *self-taught learning*, a new machine learning formalism in which we are given a supervised learning problem together with additional unlabeled instances that may not have the same class labels or the same generative distribution as the labeled instances. The main motivation for self-taught learning is to learn a useful representation from a huge amount of mildly-related unlabeled data and to use this learned representation for a supervised learning task. For example, one may wish to learn to distinguish between cars and motorcycles given images of each, and additional—and in practice readily available—unlabeled images of various natural scenes. This approach contrasts the much more restrictive semi-supervised learning problem, which would require that the unlabeled examples also be of cars or motorcycles only. We apply our sparse coding algorithms to the unlabeled data to learn bases, which gives us a higher-level representation for images, thus making the supervised learning task easier. More concretely, the following describes a self-taught learning algorithm using sparse coding:

1. Use the unlabeled data to learn basis vectors $\{\vec{b}_j\}$ by solving the problem in Equation (2.3).

2. Now fix the basis vectors, and compute the "activations" for labeled examples x_l by computing $\vec{s}^* = \arg\min_{\vec{s}} \frac{1}{2\sigma^2}\|\vec{x}_l - \sum_j \vec{b}_j s_j\|^2 + \beta \sum_j |s_j|$.

3. Finally, use the activations $\vec{s}^*$ as features to train a standard, off-the-shelf classifier (such as an SVM) using the labeled data. The classifier can then be applied to predict test data.

On a variety of problems including object recognition, audio classification, and text categorization, this approach leads to 11–36$\%$ reductions in test error. For details, see [131] or Rajat Raina's Ph.D. thesis [130]. In the next chapter, we will develop a generalized version of sparse coding that can be applied to self-taught learning given a wider range of input data types.

Self-taught learning does not necessarily assume sparse coding for the unsupervised feature learning step; however, its good performance was demonstrated via sparse coding. Indeed, in empirical comparisons, sparse coding outperformed other baseline methods such as PCA (which can be viewed as another unsupervised feature learning algorithm) [131, 95]. Therefore, the initial success of self-taught learning partly comes from the good feature representations that can be learned via sparse coding.

2.7 Other related work and applications

In this chapter, we formulated sparse coding as optimization with a well-defined objective function and proposed efficient algorithms. Since the original publication of this work, many other authors have built upon this work (e.g., online sparse coding [108, 109], discriminative sparse coding [111], and other variants of sparse feature learning [80, 7, 9, 72, 43]).

The sparse coding implementation described in this chapter is available at:

`http://ai.stanford.edu/~hllee/`,

and has been applied to various applications. For example, Yang et al. (2009) [167] used feature-sign search in learning sparse coding bases for SIFT descriptors. Their results are considered state-of-the-art in image classification, outperforming the classical SIFT features. The sparse coding implementation also has been applied to image super resolution [166], mass spectroscopy [3, 2], and autonomous altitude estimation of a UAVs via learning basis vectors for aerial images [36].

2.8 Summary

In this chapter, we formulated sparse coding as a combination of two convex optimization problems and presented efficient algorithms for each: the feature-sign search for solving the L_1-least squares problem to learn coefficients, and a Lagrange dual method for the L_2-constrained least squares problem to learn the bases for any sparsity penalty function. We test these algorithms on a variety of datasets, and show that they give significantly better performance compared to previous methods. Our algorithms can be used to learn an overcomplete set of bases, and show that sparse coding could partially explain the phenomena of end-stopping and non-classical receptive field surround suppression in biological V1 neurons. In related work on self-taught learning, we also show that this algorithm can be used for learning useful feature representations that achieve good performance in image classification, audio classification, and text classification.

Chapter 3

Exponential Family Sparse Coding

3.1 Introduction

In this chapter, we consider the self-taught learning problem, in which we are given limited labeled data from a classification task, as well as large amounts of unlabeled data that is only mildly related to the task [163, 131]. Raina et al. [131] show that with a specific sparse coding model, such mildly-related unlabeled data can help improve classification accuracy on some tasks. However, the previous sparse coding model assumes that the inputs consist of real-valued vectors, and that the vectors can be well described using a Gaussian noise model (details in Section 2.2). In our view, and as demonstrated by our experiments, this severely limits the applicability of sparse coding in a general self-taught learning algorithm.

As a running example, consider the application of self-taught learning to text classification: suppose we would like to classify sports webpages as "Baseball" or "Football" using only very few labeled webpages and many unlabeled text documents, obtained randomly from the Internet (say). The natural representation of text documents is often as a binary "bag-of-words" vector $x \in \{0, 1\}^d$, where the i-th feature is 1 if the i-th word in our vocabulary occurred in the document, or as a word-counts vector $x \in \{0, 1, 2, \dots\}^d$, where the i-th feature represents the number of times the i-th word occurred in the document. In either case, such input vectors are very poorly modeled by a continuous Gaussian distribution (which could take fractional, or negative values). It is thus hardly surprising that when sparse coding is applied to a self-taught learning task involving text data, it only leads to very small improvements in accuracy.

The above problem is not unique to text classification. Sparse coding with the Gaussian noise distribution assumption may be too restrictive to model the wide variety of inputs that we might encounter in machine learning problems, including point clouds or depth maps, discrete data, etc.

To address this problem, we generalize the Gaussian probabilistic model behind sparse coding in a principled way to include most standard distributions. We draw on the widely studied idea of the "exponential family" of distributions. This class of distributions includes the Gaussian, Bernoulli and Poisson distribution, among others, while still providing guarantees useful for efficient learning and inference. Our generalization is analogous to the way in which generalized linear models (GLMs) generalize least squares regression (which relies on a Gaussian assumption) to other kinds of regression, including logistic regression (for binary inputs) or softmax regression (for multivalued inputs) [112]. We call our model *exponential family sparse coding*, and to differentiate it from the previous model, we henceforth call that model Gaussian sparse coding.

Our generalization makes the parameter learning problem significantly harder. However, we show that the optimization problem can be solved via a series of L_1-regularized least squares problems, each of which can be solved using algorithms similar to the ones used for Gaussian sparse coding. In fact, our optimization procedure can also be applied to other L_1-regularized optimization problems, and is especially efficient for problems that have very sparse optimal solutions.

We apply our model successfully to two self-taught learning problems—text classification and a robotic perception task—even though Gaussian sparse coding produces poor performance for both.

3.2 Self-taught Learning for Discrete Inputs

We consider a classification problem with a small labeled training set $\{(x_l^{(1)}, y^{(1)}), (x_l^{(2)}, y^{(2)}), \ldots, (x_l^{(m)}, y^{(m)})\}$ drawn i.i.d. from an unknown distribution $\mathcal{D}$. Each input $x_l^{(i)} \in \mathcal{X}$ is assigned a class label $y^{(i)} \in \mathcal{Y} = \{1, 2, \ldots, K\}$. We do not place the additional restriction that $\mathcal{X} = \mathbb{R}^d$. For example, text documents can be represented as a binary vector

$x_l^{(i)} \in \mathcal{X} = \{0,1\}^d$ or as an integer-valued vector $x_l^{(i)} \in \mathcal{X} = \{0,1,2,\dots\}^d$.

Following the self-taught learning framework, we assume that we are also given a large set of unlabeled examples $\{x^{(1)}, x^{(2)}, \dots, x^{(r)}\}$ (without the subscript "l"). The inputs $x^{(i)} \in \mathcal{X}$ are only mildly constrained to be of the same "input type" as the labeled examples, but need not belong to any of the labels in the classification task, and need not arise from the same distribution $\mathcal{D}$.

To motivate the algorithms introduced in this chapter, consider the application of the above self-taught learning algorithm to binary input vectors $x \in \{0,1\}^d$, say for text classification. The Gaussian sparse coding model makes the probabilistic assumption that $P(x \mid \eta = \sum_j b_j s_j)$ is a Gaussian distribution, which is a poor fit to binary data. Stated otherwise, the Gaussian sparse coding model tries to enforce the decomposition $x \approx \sum_j b_j s_j$, even though the unconstrained sum $\sum_j b_j s_j$ is a particularly poor approximation to a binary vector x. Thus, a straightforward application of Gaussian sparse coding does not lead to very useful basis vectors or features.

Instead, we might want to find an approximation of the form $x \approx \sigma(\sum_j b_j s_j)$, where $\sigma(v) = [\frac{1}{1+e^{-v_1}}, \frac{1}{1+e^{-v_2}}, \dots]$ represents the element-wise logistic function for a vector v. This promises to be a better formulation, since the logistic function always lies in $(0,1)$.

We now present a systematic generalization of the Gaussian sparse coding model. Our generalization includes both Gaussian sparse coding and our seemingly arbitrary logistic function approximation as special cases, and will directly suggest models for other input types, such as when the inputs consist of nonnegative integer counts $x \in \{0,1,2,\dots\}^d$.

3.3 Exponential Family Sparse Coding

The exponential family is a widely used class of distributions in statistics, and in its most general form, is represented as:

$$P(x|\eta) = h(x)\exp(\eta^\top T(x) - a(\eta)). \tag{3.1}$$

Here, η represents the natural parameter for the model, and the functions h, T and a together define a particular member of the family.

For example, the multivariate Gaussian distribution $\mathcal{N}(\mu, I)$ with fixed covariance I and (unknown) mean parameter $\mu \in \mathbb{R}^d$ can be written as follows:

$$P(x; \mu, I) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{\|x-\mu\|^2}{2}\right) \tag{3.2}$$

This is equivalent to the exponential family distribution defined by $h(x) = e^{-\|x\|^2/2}/(2\pi)^{d/2}$, $T(x) = x$, $\eta = \mu$, and $a(\eta) = \eta^\top \eta/2$.

As another example, a Bernoulli distribution for a random variable $x \in \{0, 1\}$ with probability p can be written as follows:

$$\begin{aligned} P(x; p) &= \exp(x \log(p) + (1-x)\log(1-p)) && (3.3) \\ &= \exp\left(x \log(p/(1-p)) + \log(1-p)\right) && (3.4) \end{aligned}$$

This can be written as an exponential family distribution with $h(x) = 1$, $T(x) = x$, $\eta = \log(p/(1-p))$, and $a(\eta) = -\log(1-p) = \log(1 + \exp(\eta))$.

Finally, a Poisson distribution for a random variable $x \in \{0, 1, 2, ...\}$ with rate λ can be written as follows:

$$\begin{aligned} P(x; \lambda) &= \frac{1}{x!}\lambda^x \exp(-\lambda) && (3.5) \\ &= \frac{1}{x!}\exp(x \log \lambda - \lambda) && (3.6) \end{aligned}$$

We can simply describe this as an exponential family distribution with $h(x) = 1/x!$, $T(x) = x$, $\eta = \log \lambda$, and $a(\eta) = \lambda = \exp(\eta)$.

To summarize, the exponential family of distributions is broad enough to include most standard distributions (Gaussian, Bernoulli, Poisson, exponential, multinomial, and others), but also restrictive enough to provide useful guarantees. For example, it guarantees that the log-likelihood is concave in the natural parameter η, making maximum likelihood learning of parameters tractable [112].

In our approach, we modify the Gaussian sparse coding model to allow any distribution from the exponential family:

$$P(x|b,s) = h(x)\exp(\eta^\top T(x) - a(\eta)), \quad \eta = \textstyle\sum_j b_j s_j \tag{3.7}$$

where we use the basis vectors b_j and the activations s_j to construct the natural parameter η for the family. Since the Gaussian distribution is a member of the exponential family, our new generative model includes the earlier Gaussian generative model for $P(x|\eta)$ as a special case. In fact, our model extends Gaussian sparse coding in the same way that generalized linear models (GLMs) extend the notion of regression with least squares loss to other loss functions, including logistic regression and softmax regression as special cases.

Given unlabeled examples $\{x^{(1)}, x^{(2)}, \ldots, x^{(r)}\}$, we can apply Equation (2.2) to compute the *maximum a posteriori* estimates of the basis vectors b_j and the activations $s^{(i)}$ as follows:[1]

$$\begin{aligned} &\underset{B,\{s^{(i)}\}}{\text{minimize}} && \textstyle\sum_i -\log h(x^{(i)}) - {s^{(i)}}^\top B^\top T(x^{(i)}) + a(Bs^{(i)}) + \beta \sum_{i,j} |s_j^{(i)}| && (3.8)\\ &\text{subject to} && \|b_j\|^2 \le c,\ \forall j = 1, ..., n, && (3.9)\end{aligned}$$

where we define the basis matrix B such that its j-th column is the basis vector b_j, implying that $\eta = \sum_j b_j s_j = Bs$. By definition, we want the model to produce sparse activations, so we set β large enough to produce only a small number of nonzero values per activation vector $s^{(i)}$ on average.

Since the exponential family guarantees convexity of $\log P(x|\eta)$ with respect to η, we can show that the above optimization problem is convex with respect to s for fixed B, and with respect to B for fixed s (though it is not jointly convex). This again suggests an alternating minimization procedure iterating the following two steps till convergence: (i) fix the activations s, and compute the optimal bases B; and, (ii) fix these bases B, and compute the optimal activations s.

Step (i) involves a constrained optimization problem over B with a differentiable objective function. We can thus apply projective gradient descent updates, where at each iteration we perform a line search along the direction of the (negative) gradient, projecting

[1]As in Gaussian sparse coding, we assume a Laplacian prior on s: $P(s) \propto \prod_j \exp(-\beta|s_j|)$, and a uniform prior on b_j.

onto the feasible set before evaluating the objective function during the line search. In our case, the projection operation is especially simple: we just need to rescale each basis vector to have norm c if its norm is greater than c. In our experiments, we find that such a projective gradient descent scheme is sufficiently fast for basis learning. We thus focus now on the algorithm for computing the optimal activations in Step (ii).

Step (ii) computes the optimal activation s given fixed basis vectors. The resulting problem involves a non-differentiable L$_1$-regularized objective function, to which straightforward gradient descent methods are not applicable. Recently, many sophisticated algorithms have been developed for L$_1$-regularized optimization, including specialized interior point methods [78], quasi-Newton methods [5, 172] and coordinate descent methods [53]. When used for computing activations with 1000 basis vectors, these methods find the optimal solution in a few seconds *per unlabeled example*. Since we often need to solve for the activations of tens of thousands of unlabeled examples repeatedly in the inner loop of the overall alternating minimization procedure, these solvers turn out to be too slow for our model.

We now present a simple optimization algorithm for L$_1$-regularized optimization problems. We later show that, despite the simplicity of the algorithm, when the optimal solution is very sparse (as in Equation 3.8-3.9) our method is faster than state-of-the-art solvers for L$_1$-regularized problems.

3.3.1 Computing optimal activations

We first note that since the optimal values for the activation vectors $s^{(i)}$ do not depend on each other, and can be optimized separately, it is sufficient to consider the following optimization problem for a single input x and its activation s:

$$\text{minimize}_s \quad \ell(s) + \beta \|s\|_1 \tag{3.10}$$

where s corresponds to a vector of activations, and $\ell(s)$ is a convex function of s. Specifically, we consider the following function derived from Equations (3.8-3.9):

$$l(s) \triangleq -\log h(x) - s^\top B^\top T(x) + a(Bs) \tag{3.11}$$

Algorithm 2 IRLS-FS algorithm for L$_1$-regularized exponential family problems

```
**Input:** $B \in \mathbb{R}^{d \times n}$, $x \in \mathbb{R}^d$, threshold $\epsilon$.
Initialize $s := \vec{0}$.
**while** decrease in objective value at last step $> \epsilon$ **do**
    Compute diagonal matrix $\Lambda$ with $\Lambda_{ii} = a''\left((Bs)_i\right)$,
    Compute vector $z = \Lambda^{-1}(T(x) - a'(Bs)) + Bs$.
    Initializing feature-sign search at $s$, compute:
        $\hat{s} = \arg\min_{s'} \left\|\Lambda^{1/2}Bs' - \Lambda^{1/2}z\right\|^2 + \beta\|s'\|_1$
    Set $s := (1-t)s + t\hat{s}$, where $t$ is found by backtracking line-search [27] to minimize
    the objective function in Equation (3.8).
**end while**
```

In the case of Gaussian sparse coding, $\ell(s)$ is simply a quadratic function, and the optimization problem is a L$_1$-regularized least squares problem that can be solved efficiently, as shown in the previous chapter. This suggests an iterative algorithm for the general case: at each iteration, we compute a local quadratic approximation $\hat{\ell}(s)$ to the function $\ell(s)$, and optimize the objective function $\hat{\ell}(s) + \beta\|s\|_1$ instead. This method is an instance of a general method called Iteratively Reweighted Least Squares (IRLS) in the literature [60]. Using this insight, Lee et al. [99] proposed the IRLS-LARS algorithm for the case of logistic regression with L$_1$-norm constraints, using Efron et al.'s LARS algorithm in the inner loop to solve the approximated problem. In this chapter, we develop a new algorithm for optimizing smooth convex functions with an L$_1$-norm penalty.

The IRLS formulation can be applied to general L$_1$-regularized optimization problems for which a local quadratic approximation can be efficiently computed. Let's consider a local quadratic quadratic approximation of $l(s)$ at a point s_0. The gradient and Hessian of $l(s)$ at s_0 can be written as follows:

$$\nabla\ell = -B^\top T(x) + B^\top a'(Bs_0), \tag{3.12}$$

$$\nabla^2\ell = B^\top \Lambda B, \tag{3.13}$$

where $\Lambda_{ii} = a''\left((Bs_0)_i\right)$ for a diagonal matrix Λ. Now a quadratic approximation of $l(s)$

at s_0 can be written as:

$$\begin{aligned}
\hat{\ell}(s) &= \frac{1}{2}(s-s_0)^\top \nabla^2 \ell (s-s_0) + \nabla \ell (s-s_0) + l(s_0) \qquad (3.14)\\
&= \frac{1}{2}(s-s_0)^\top B^\top \Lambda B (s-s_0) + (s-s_0)^\top \left(-B^\top T(x) + B^\top a'(Bs_0)\right) + l(s_0)\\
&= \frac{1}{2}(s-s_0)^\top B^\top \Lambda B (s-s_0) + (s-s_0)^\top B^\top \Lambda\, \Lambda^{-1} \left(-T(x) + a'(Bs_0)\right) + l(s_0)\\
&= \frac{1}{2}s^\top B^\top \Lambda B s - s^\top B^\top \Lambda B s_0 + s^\top B^\top \Lambda\, \Lambda^{-1} \left(-T(x) + a'(Bs_0)\right) + constant\\
&= \frac{1}{2}s^\top B^\top \Lambda B s + s^\top B^\top \Lambda\, \left(\Lambda^{-1}(-T(x) + a'(Bs_0)) - Bs_0\right) + constant\\
&= \left\|\Lambda^{1/2} B s - \Lambda^{1/2}\left(\Lambda^{-1}(T(x) - a'(Bs_0)) + Bs_0\right)\right\|^2 + constant\\
&= \left\|\Lambda^{1/2} B s - \Lambda^{1/2} z\right\|^2 + constant,
\end{aligned}$$

where $z = \Lambda^{-1}(T(x) - a'(Bs_0)) + Bs_0$. □

We note that if the objective function $\ell(s)$ is reasonably approximated by a quadratic function, the solutions to the successive quadratic approximations should be close to each other. However, the LARS algorithm used in IRLS-LARS cannot be initialized at an arbitrary point, and thus has to rediscover the solution from scratch while solving each successive approximation. In contrast, the feature-sign search algorithm (originally proposed in the context of Gaussian sparse coding in Chapter 2) can be initialized at an arbitrary point and can thus potentially solve the successive approximations much faster. We propose to use the feature-sign search algorithm to optimize each quadratic approximation.

The final algorithm, which we call IRLS-FS, is described in Algorithm 2. The algorithm is guaranteed to converge to the global optimum in a finite number of iterations. (Proof similar to IRLS-LARS.)

3.4 Computational Efficiency

We compare the IRLS-FS algorithm against state-of-the-art algorithms for optimizing the activations, focusing on the case of binary sparse coding (i.e., $x \in \{0,1\}^d$). This case is especially interesting because this leads to an L_1-regularized logistic regression problem,

Dataset	Small1	Small2	Small3	Med1	Med2	Med3	Large1	Large2	Large3
IRLS-LARS	4.6	4.9	4.3	12.8	12.5	13.2	1131	1270	1214
l1-logreg	18.3	18.9	17.7	181	188	185	3277	3101	3013
CoordDescent	3.6	3.4	5.6	20.7	20.7	31.0	787	653	723
OWL-QN	7.1	7.0	10.3	27.1	31.4	25.6	1018	739	906
SubLBFGS	33.0	22.3	23.1	101	142	57.2	1953	2717	1627
IRLS-FS	**2.5**	**2.3**	**2.2**	**5.3**	**5.5**	**5.4**	**117**	**108**	**109**

Table 3.1: Total running time in seconds for computing activations for 50 input examples for 9 problems (one per column). See text for details.

Dataset	Col	Alon	Duln	Duer	Arr	Mad	Hep	Spf	Prom	Wbc	Ion	Spl	Spc	Spam
Sparsity (%)	0.2	0.5	0.8	1.4	3.5	3.6	26.3	29.5	31.6	40.0	45.5	56.7	63.6	66.7
IRLS-LARS	2.1	3.3	6.2	35.6	2.2	25.6	0.5	5.0	2.1	5.0	3.5	18.3	2.6	57.8
l1-logreg	18.3	16.8	13.6	14.4	34.8	509	1.0	3.0	2.0	3.8	2.7	12.8	2.0	37.1
CoordDescent	83.3	54.1	63.8	129	7.7	101	0.2	**2.0**	0.6	**2.6**	**1.4**	**4.5**	**0.8**	**14.2**
OWL-QN	27.4	29.4	16.9	79.6	7.7	634	**0.1**	3.4	**0.4**	13.4	1.9	7.1	0.9	39.3
SubLBFGS	114	80.8	60.5	311	24.3	261	0.7	9.3	2.7	14.4	4.5	13.4	2.1	43.0
IRLS-FS	**1.9**	**1.9**	**2.5**	**7.1**	**1.5**	**14.0**	0.3	2.3	1.3	2.9	2.0	10.4	1.9	50.8

Table 3.2: Total running time in seconds for 50 trials of learning parameters of various L_1-regularized logistic regression benchmarks (obtained from Lee et al., 2006). The datasets are ordered left-to-right by the increasing fraction of nonzero parameters ("Sparsity (%)") at the optimal solution (e.g., the leftmost problem Col had only 0.2% nonzero parameters at the optimal solution).

where the number of "features" is equal to the number of basis vectors used, but is *independent* of the dimensionality of inputs x in the original problem. This problem is of general interest (e.g., see Ng, 2004), and customized algorithms have also been developed for it.

We consider five recent algorithms: the IRLS-LARS algorithm [99] and the l1-logreg interior point method [78] specifically for logistic regression, the Coordinate Descent method [53] with successive IRLS approximations, and the OWL-QN [5] and SubLBFGS [172] algorithms for L_1-regularized convex optimization problems.[2] IRLS-FS, IRLS-LARS, l1-logreg and CoordDescent were implemented in Matlab, and OWL-QN and SubLBFGS were compiled in C++ with optimization flags.

[2]Baseline algorithms: Lee et al. [99] show that IRLS-LARS outperforms several previous algorithms, including grafting [128], SCGIS [56], GenLasso [139] and Gl1ce [105]. Therefore, we did not compare against these methods.

All algorithms were evaluated on nine L_1-regularized logistic regression problems which arise in the course of solving Equations (3.8-3.9) for unlabeled text documents with binary sparse coding (details in Section 3.5.1). There were three problems for each of three different sizes, and they were labeled Small1 to Small3, Med1 to Med3, or Large1 to Large3 based on the size of the problem. The "Small" problems had 200 basis vectors and input dimension 369, "Med" problems had 600 basis vectors and dimension 369, and "Large" problems had 1000 basis vectors and dimension 3891. The sparsity parameter β was set to produce roughly 20 nonzero activations per example on average.[3] We measured the running time taken by each algorithm to converge within a specified tolerance of the optimal objective value. In detail, since IRLS-LARS solves the dual or Lasso version of our problem (i.e., with a constraint "C" on the L_1 norm of the activations rather than a penalty β), we follow Lee et al.'s method of using the KKT conditions to convert between the constraint value C and the equivalent penalty β for that problem. We ran all algorithms until they reached an objective value of $(1+\epsilon)f^{opt}$ where f^{opt} is the optimal objective value (we used $\epsilon = 10^{-6}$).

Table 3.1 shows the running times computed over 50 trials. IRLS-FS outperforms the other algorithms on this task, showing that it well-suited to exponential family sparse coding. When a large number of basis vectors are used, IRLS-FS can be 5 to 7 times faster than the best baseline algorithm.

This poses the question: can IRLS-FS be a useful algorithm for general L_1-regularized optimization problems (not necessarily ones generated by the sparse coding problem)? We compare the algorithms above on 14 moderate-size benchmark classification datasets, and apply L_1-regularized logistic regression to them. The value of β on each benchmark was picked to optimize the generalization error of the resulting logistic regression classifier; unlike the earlier experiment, β was not set explicitly to obtain sparse solutions. Table 3.2 shows the running time of the algorithms to compute the optimal parameters. IRLS-FS outperforms the other algorithms on 6 out of 14 of these benchmark datasets; more specifically, it performs best when the optimal parameters have a very small number of nonzero values.

[3] In our experiments, such β values produced reasonable basis vectors during basis learning.

3.5 Application to self-taught learning

The model presented in this chapter generalizes Gaussian sparse coding. It is also closely related to exponential family PCA [38], which corresponds to setting the sparsity penalty β to zero, and additionally constraining the basis matrix B to have orthogonal columns. We now show that the exponential family sparse coding model can produce better self-taught learning performance than either of these previous methods.

3.5.1 Text classification

We first apply the exponential family sparse coding algorithms to two self-taught learning problems in text classification: one using binary-valued input vectors, and another using integer-valued (word count) vectors.

We test on five standard webpage classification problems [47], and a newsgroup classification problem (20 newsgroups dataset). We used 470,000 unlabeled news articles (from the Reuters corpus) to learn basis vectors according to the binary and Poisson sparse coding models. The webpage classification problems were created using subcategories of the Arts, Business, Health, Recreation and Sports categories of the DMOZ hierarchy. Each of them consisted of 10 separate binary classification problems over a 500 word vocabulary, with stopword removal and stemming. The newsgroup classification problem consisted of 10 binary classification problems constructed using the 20 newsgroups dataset. We used 600 basis vectors, and picked β to achieve roughly 10% nonzero activations and did not tune these numbers. For learning, we used stochastic updates with mini-batches of 2000 randomly sampled examples, and stopped learning when the objective value did not decrease for 10 consecutive mini-batch iterations. Table 3.3 gives examples of basis vectors obtained from Poisson sparse coding. Many basis vectors appear to encode related words and capture various "topics."

Using the learned basis vectors, we computed features for each classification task using the binary and Poisson sparse coding model. We call our model "ExpSC" and compare against several baselines: the raw words themselves ("Raw"), Gaussian sparse coding ("GSC"), exponential family PCA with the same binary or Poisson exponential family assumption ("ExpPCA"), and Latent Dirichlet Allocation (LDA), a widely-known topic

free	share	subscrib	paint	actor	novel
market	exchange	online	pictur	actress	literari
polici	stock	servic	portrait	film	poet
power	secur	server	museum	comedi	fiction
peopl	commiss	databas	rule	star	univers

Table 3.3: Examples of basis vectors trained for the Business (left 3 columns) and Arts (right 3 columns) problems, using unlabeled Reuters news data with Poisson sparse coding. Each column shows the five word stems that were most highly active (i.e., had the highest weight) for some basis vector.

Features / training set size	4	10	20
Raw	34.3%	23.0%	17.7%
ExpSC	**27.8%**	20.5%	17.4%
ExpSC + Raw	30.0%	**20.4%**	**16.1%**
ExpPCA	29.1%	22.3%	19.3%
ExpPCA + Raw	31.3%	22.7%	17.7%

Table 3.4: Aggregate test error across all text classification problems (webpages and newsgroups), represented using binary vectors. The standard errors were similar across different features, approximately corresponding to 2.5% for 4 training examples, 1.9% for 10 training examples, and 1.3% for 20 training examples.

model for text documents [21]. We also compared Gaussian sparse coding applied to log of raw word counts ("Log-GSC").[4]

All baselines (except the raw features) were trained using the same unlabeled data as our model. For LDA, we tried 20, 50 and 100 topics, and picked the best. We also consider combinations of the raw word features with the other types of features (e.g., "ExpSC+Raw" indicates a combination of the ExpSC features and the raw features). All features were evaluated using standard supervised-learning classifiers over 100 trials each for 4, 10 and 20 training documents. Specifically, We focused on three standard classifiers—SVM, GDA and kernel dependency estimation (KDE)—that performed best for the raw bag-of-words features out of several generic classifiers, including k-NN and decision trees. We report average results of the best performing classifier for each feature. We picked the β value used for computing the features by cross-validation.

[4]We first took log of word counts after adding a small constant, and we normalized the resulting vector (for each example) to zero mean.

Features / training set size	4	10	20
Raw	29.6%	24.3%	18.4%
ExpSC	25.4%	18.9%	15.2%
ExpSC + Raw	**25.0%**	**18.6%**	**14.9%**
GSC	31.2%	26.1%	21.4%
GSC + Raw	26.4%	22.7%	17.6%
Log-Raw	29.9%	23.1%	16.2%
Log-GSC	30.8%	28.6%	22.6%
Log-GSC + Log-Raw	29.7%	27.6%	21.6%
ExpPCA	32.9%	26.2%	24.2%
ExpPCA + Raw	30.4%	23.7%	21.1%
LDA + Raw	33.2%	24.7%	17.6%

Table 3.5: Aggregate test error across all text classification problems (webpages and newsgroups), represented using word count vectors. ExpSC denotes Poisson sparse coding. LDA+Raw performed better than LDA alone, so we show results only for this case. The standard errors were similar across different features, approximately corresponding to 2.8% for 4 training examples, 2.0% for 10 training examples, and 1.4% for 20 training examples.

Table 3.4 and Table 3.5 shows the average test error over all problems for the binary and Poisson case. The exponential family sparse coding features alone frequently outperform the other features, and produce slightly better results when used in combination with the raw features (ExpSC+Raw). The results for Poisson sparse coding are particularly striking, showing 20-30% error reduction over raw counts in some cases. The other three methods for using unlabeled data (GSC, ExpPCA, LDA) perform poorly in many cases.

3.5.2 Robotic perception

We also applied the exponential family sparse coding algorithm to a very different self-taught learning problem: object recognition in 3D range data. The data was collected with a laser range finder (Velodyne lidar) in a parking lot environment. Given a 3D box in this space, the task is to predict whether the box contains a car or not.

A standard, robust representation for such 3D point cloud data is the "spin-image" representation [74]. A detailed description of spin-images is beyond the scope of this chapter; but informally, for our application, a spin-image can be thought of as a sheet spinning about the vertical z vector at a point, accumulating counts of other points in each pixel as it rotates

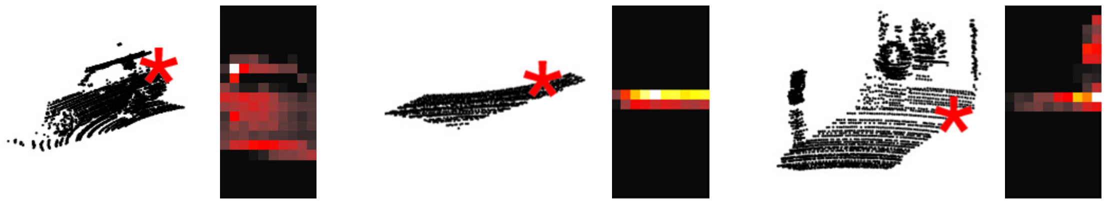


Figure 3.1: Three example pairs of point clouds (a car, a road, and a ground with non-car objects) and the corresponding spin-images. Left of each pair: A view of a point cloud produced by the laser range finder. Right of each pair: A spin-image generated around the point marked with a red star on the left.

Features / training set size	4	10	20
Raw	34.2%	22.4%	18.4%
ExpSC	**23.0%**	**20.0%**	19.3%
ExpSC + Raw	25.0%	20.6%	**16.9%**
GSC	36.8%	31.3%	28.7%
GSC + Raw	36.0%	27.5%	19.5%
Log-Raw	29.9%	22.0%	17.3%
Log-GSC	31.6%	27.8%	27.2%
Log-GSC + Log-Raw	33.2%	24.1%	18.8%
ExpPCA	38.5%	31.4%	23.7%
ExpPCA + Raw	34.6%	22.2%	18.0%

Table 3.6: Test error for various algorithms and training set sizes for classifying spin-images from the laser range finder. The standard errors across different features were similar, corresponding to 3.3% for 4 training examples, 2.1% for 10 training examples, and 1.4% for 20 training examples.

(see Figure 3.1). Spin-images robustly describe the 3D surfaces around a reference point using a 2D array of counts (20x10 array in our case).

Given a large number of unlabeled spin-images (which can be extracted very easily) and a small number of spin-images from regions manually labeled as car or non-car, our task is to predict whether a new spin-image represents a car or non-car region. Since the spin-image is a count-based representation, we apply Poisson sparse coding to this problem. Basis vectors were learned using 31,000 unlabeled spin-images, and manual examination of the result reveals that certain basis vectors capture various 3D features of cars, roads, trees, and other objects.

Table 3.6 shows the test error for varying training set size. When compared with using the raw spin-image alone, the Poisson sparse coding features reduce test error by 30% with 4 training examples, and by 8-10% for the other cases.

3.6 Discussion

Our extension to Gaussian sparse coding is conceptually similar to the extension proposed for PCA by Collins et al. (2001). However, PCA itself defines a linear transform of the features, unlike Gaussian sparse coding, and does not work as well on self-taught learning tasks. It is thus not surprising that the exponential family extension of sparse coding also outperforms exponential family PCA.

Compared to LDA, our model is more general because it can be applied to non-text and non-exchangeable domains. Further, it has been pointed out that when LDA topics are viewed as defining a simplex over word distributions, the exact topics learned by LDA can vary a lot—the only constraint is that the simplex defined by the topics still spans all the training documents [21]. In our view, the additional sparsity constraints in the sparse coding model help reduce this ambiguity, and allow large numbers of "topics" to be learned robustly.

3.7 Summary

In this chapter, we presented a general method for self-taught learning, that extends previous models in a natural way. The extensions can still be solved efficiently using the IRLS-FS algorithm, which we showed to be an efficient algorithm for L_1-regularized learning problems with sparse optimal solutions. We showed that our model achieves better self-taught learning performance than Gaussian sparse coding or exponential family PCA on two very different tasks. We believe our model will extend the applicability of self-taught learning to new, previously difficult problems.

Chapter 4

Sparse Deep Belief Networks

4.1 Introduction

As we have seen in previous chapters, sparse representations are beneficial. Specifically, sparse coding learns useful low-level feature representations for unlabeled data, such as images, sound, and text. Now, suppose that we are interested in learning more complex, higher-level features. For example, we may want to learn object-part features from images rather than just edge detectors. With this goal in mind, it appears to be fairly plausible to attempt to build hierarchical feature representations using sparse coding.

However, it is not straightforward to apply sparse coding recursively to build multiple levels of hierarchy. Although there is some sparse coding-related research that builds hierarchies (e.g., [75, 76, 29]), these algorithms are highly non-trivial (i.e., they are not merely stacking sparse coding layers) and computationally expensive to optimize. In fact, it has been difficult to build hierarchical representations by straightforwardly "stacking" sparse coding layers. We hypothesize that this difficulty may be due to the following factors. First, sparse coding (with real-valued inputs) assumes a non-sparse input distribution, whereas the output distribution of sparse coding is very sparse; therefore, applying another sparse coding on top of the sparse coding output may not satisfy the modeling assumption. Second, the optimization (i.e., inference) in sparse coding is fairly expensive since it involves L_1 regularized optimization.[1] Therefore, we found it quite challenging to develop

[1]We note that there have been attempts to speed up sparse coding by modifying the objective function.

hierarchical sparse coding algorithms based on our experiments.

Faced with such difficulties, we will describe an alternative approach to building sparse hierarchical representations. The basic idea is to build upon so-called "deep learning" algorithms, which have shown promise in building hierarchical representations. Indeed, the last several years have seen significant interest in deep learning algorithms that learn layered, hierarchical representations of high-dimensional data [65, 136, 16, 83]. Deep architectures attempt to learn hierarchical structure and seem promising in learning simple concepts first and then successfully building up more complex concepts by composing the simpler ones together. For example, Hinton et al. [65] proposed an algorithm based on learning individual layers of a hierarchical probabilistic graphical model from the bottom up. Bengio et al. [16] proposed a similarly greedy algorithm based on autoencoders. Ranzato et al. [136] developed an energy-based hierarchical algorithm, using a sequence of sparsified autoencoders/decoders.

In this chapter, we will build upon restricted Boltzmann machines (RBMs) and deep belief networks (DBNs) developed by Hinton et al. [65]. Compared to sparse coding, RBMs can be easily stacked to form a hierarchical representation called DBNs. Further, approximate inference for RBMs and DBNs is computationally efficient because the inference can be performed in a feed-forward way without having to solve any iterative optimization problems. Specifically, we develop a sparse variant of the deep belief networks. In experiments, we train two layers of hidden nodes in the network, and demonstrate that the first layer, similar to previous work on sparse coding and ICA, results in localized, oriented, edge filters, similar to the Gabor functions (known to model V1 cell receptive fields). Furthermore, the second layer in our model encodes correlations of the first layer responses in the data. Specifically, it identifies both colinear ("contour") features as well as corners and junctions. In addition, it turns out that the sparsity regularization proposed in this chapter results in improved (often the best) performance in many machine learning tasks.

For example, Koray et al. [77] proposed an algorithm for solving a variant of sparse coding using the encoder/decoder framework.

4.2 Algorithm

In general, restricted Boltzmann machines and deep belief networks tend to learn distributed, non-sparse representations. In this section, we modify Hinton et al.'s learning algorithm to enable these models to learn sparse representations.

4.2.1 Sparse restricted Boltzmann machines

We begin by describing the restricted Boltzmann machine (RBM) and present a modified version of it. An RBM has a set of hidden units $\mathbf{h}$, a set of visible units $\mathbf{v}$, and symmetric connections weights between these two layers represented by a weight matrix W. The probabilistic semantics for an RBM is defined by its energy function as follows:

$$P(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h})), \tag{4.1}$$

where Z is the partition function. If the visible units are real-valued, we define the energy function as follows:[2]

$$E(\mathbf{v}, \mathbf{h}) = \frac{1}{2\sigma^2} \sum_i v_i^2 - \frac{1}{\sigma^2} \left(\sum_i c_i v_i + \sum_j b_j h_j + \sum_{i,j} v_i w_{ij} h_j \right), \tag{4.2}$$

where σ is a constant, b_j are hidden unit biases, and c_i are visible unit biases. If the visible units are binary-valued, we can similarly define the energy function as follows:

$$E(\mathbf{v}, \mathbf{h}) = -\frac{1}{\sigma^2} \left(\sum_i c_i v_i + \sum_j b_j h_j + \sum_{i,j} v_i w_{ij} h_j \right). \tag{4.3}$$

Under this model, we can easily compute the conditional probability distributions. If

[2]See also [66] for an alternative way of defining energy function for the real-valued visible units.

the visible units are real-valued, the conditional probability can be written as:

$$P(v_i|\mathbf{h}) = \mathcal{N}\left(c_i + \sum_j w_{ij}h_j, \sigma^2\right), \tag{4.4}$$

$$P(h_j = 1|\mathbf{v}) = sigmoid\left(\frac{1}{\sigma^2}(b_j + \sum_i w_{ij}v_i)\right). \tag{4.5}$$

Here, $\mathcal{N}(\cdot)$ is the Gaussian density, and $sigmoid(\cdot)$ is the sigmoid function. Similarly, if the visible units are binary-valued, the conditional probability can be written as:

$$P(v_i|\mathbf{h}) = sigmoid\left(\frac{1}{\sigma^2}(c_i + \sum_j w_{ij}h_j)\right), \tag{4.6}$$

$$P(h_j = 1|\mathbf{v}) = sigmoid\left(\frac{1}{\sigma^2}(b_j + \sum_i w_{ij}v_i)\right). \tag{4.7}$$

Therefore, we can perform efficient block Gibbs sampling by alternately sampling each layer's units (in parallel) given the other layer. We will often refer to a unit's expected value as its *activation*.

For training the parameters of the model, the objective is to maximize the log-likelihood of the data. Informally speaking, the maximum likelihood parameter estimation problem corresponds to learning w_{ij}, c_i and b_j to minimize the energy of states drawn from the data distribution, and raise the energy of states that are improbable given the data. At the same time, we want hidden unit activations to be sparse. However, it is not straightforward to apply "L_1" regularization (as in sparse coding) to enforce sparsity because the RBM representation uses stochastic binary variables. Hence, we will define a regularizer that will make the average hidden variable activation low over the entire training examples. More specifically, we add a regularization term that penalizes any deviation of the expected activation of the hidden units from a (low) fixed level p. Less formally, this regularization ensures that the "firing rate" of the model neurons (corresponding to the latent random variables h_j) are kept at a certain (fairly low) level, so that the activations of the model neurons are sparse. Similar intuition was also used in other models (e.g., see Olshausen and Field [122]). Thus, given a training set $\{\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(m)}\}$ comprising m examples, we

Algorithm 3 Sparse RBM learning algorithm

1. Update the parameters using contrastive divergence learning rule. More specifically,

$$\begin{aligned} w_{ij} &:= w_{ij} + \alpha(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{recon}}) \\ c_i &:= c_i + \alpha(\langle v_i \rangle_{\text{data}} - \langle v_i \rangle_{\text{recon}}) \\ b_j &:= b_j + \alpha(\langle b_j \rangle_{\text{data}} - \langle b_j \rangle_{\text{recon}}), \end{aligned} \tag{4.9}$$

where α is a learning rate, and $\langle \cdot \rangle_{\text{recon}}$ is an expectation over the reconstruction data, estimated using one iteration of Gibbs sampling.

2. Update the parameters using the gradient of the regularization term.

3. Repeat Steps 1 and 2 until convergence.

pose the following optimization problem:

$$\text{minimize}_{\{w_{ij}, c_i, b_j\}} - \sum_{l=1}^{m} \log \left(\sum_h P(\mathbf{v}^{(l)}, \mathbf{h}^{(l)}) \right) + \lambda \sum_{j=1}^{n} \left| p - \frac{1}{m} \sum_{l=1}^{m} \mathbb{E}[h_j^{(l)} | \mathbf{v}^{(l)}] \right|^2 \tag{4.8}$$

where $\mathbb{E}[\cdot]$ is the conditional expectation given the data, λ is a regularization constant, and p is a constant controlling the sparseness of the hidden units h_j. Thus, our objective is the sum of a log-likelihood term and a regularization term (i.e., a tradeoff between "likelihood" and "sparsity"). In principle, we can apply gradient descent to this problem; however, computing the gradient of the log-likelihood term is expensive. Fortunately, the contrastive divergence learning algorithm gives an efficient approximation to the gradient of the log-likelihood [63]. Therefore, in each iteration we can apply the contrastive divergence update rule, followed by one step of gradient descent using the gradient of the regularization term, as summarized in Algorithm 3. In implementation detail, we made one additional change to increase computational efficiency. Note that the regularization term is defined using a sum over the entire training set; if we use stochastic gradient descent or mini-batches (small subsets of the training data) to estimate this term, it results in biased estimates of the gradient. To ameliorate this bias issue, we used mini-batches, but in the gradient step that attempts to minimize the regularization term, we update only the bias terms b_j's (which directly control the degree to which the hidden units are activated, and thus their sparsity), instead of updating all the parameters b_j and w_{ij}'s.

Specifically, the gradient of the sparsity regularization term over the parameters (W and

b) can be written as follows:

$$-\frac{\partial}{\partial w_{ij}}\mathcal{L}_{sparsity} \propto (p - \frac{1}{m}\sum_{l=1}^{m} p_j^{(l)})\frac{1}{m}\sum_{l=1}^{m} p_j^{(l)}(1-p_j^{(l)})v_i^{(l)} \tag{4.10}$$

$$-\frac{\partial}{\partial b_j}\mathcal{L}_{sparsity} \propto (p - \frac{1}{m}\sum_{l=1}^{m} p_j^{(l)})\frac{1}{m}\sum_{l=1}^{m} p_j^{(l)}(1-p_j^{(l)}), \tag{4.11}$$

where $p_j^{(l)} \triangleq sigmoid(\frac{1}{\sigma^2}(\sum_i v_i^{(l)} w_{ij} + b_j))$, and $\mathcal{L}_{sparsity}$ is the regularization penalty term in Equation (4.8). If we approximate the above gradient assuming that $\sum_l p_j^{(l)}(1-p_j^{(l)})$ term is roughly similar over j's, the update rule can be written in a more simple, intuitive way:

$$\Delta b_j \propto -\frac{\partial}{\partial b_j}\mathcal{L}_{sparsity} \approx (p - \frac{1}{m}\sum_{l=1}^{m} p_j^{(l)}) \cdot constant. \tag{4.12}$$

This corresponds to an update rule that increases or decreases the bias of each hidden unit based on the difference between the desired sparsity and the average posterior activation (over training examples) of each hidden unit. In the original publication of this chapter [92], we implemented our algorithm based on Equation (4.11); however, the implementation based on Equation (4.12) also gave qualitatively the same results.

We also note that setting the λ parameter is fairly simple in practice. In our experiments, we set the λ value moderately large to ensure that the average hidden unit activation (over the training examples) is close to the desired sparsity p, yet each individual hidden unit activation varies depending on individual input examples. Since there is a log-likelihood term as an objective, the hidden unit activation is still encouraged to vary depending on the individual training examples.

4.2.2 Learning deep networks using sparse RBM

Once a layer of the network is trained, the parameters w_{ij}, b_j, c_i's are frozen and the hidden unit values given the data are inferred. These inferred values serve as the "data" used to train the next higher layer in the network. Hinton et al. [65] showed that by repeatedly applying such a procedure, one can learn a multi-layered deep belief network. In some cases, this iterative “greedy” algorithm can further be shown to be optimizing a variational bound on the data likelihood, if each layer has at least as many units as the layer below (although in practice this is not necessary to arrive at a desirable solution; see [65] for a detailed discussion). In our experiments using natural images, we learn a network with two hidden layers, with each layer learned using the sparse RBM algorithm described in Section 4.2.1.

4.2.3 Discussion

First, we clarify the similarities and differences between sparse RBM and sparse coding. The sparse RBM and the sparse coding are somewhat similar in that both represent input data using latent variables (i.e., combination of the basis vectors). Furthermore, although the RBM representations are not sparse in general, the sparse RBM can learn very similar representations as sparse coding, as will be shown in this chapter (e.g., representing image patches as a combination of edges). On the other hand, there are a number of differences between sparse RBM and sparse coding. Sparse coding is a directed graphical model that has real-valued latent variables. The sparsity is controlled by L_1 regularization over the coefficients. However, posterior (MAP) inference conditioned on the input data is computationally expensive for sparse coding because it requires solving L_1-regularized optimization problem. In contrast, sparse RBM is an undirected graphical model that has binary-valued latent variables. The sparsity can be implicitly controlled by having a regularization penalty over the average activation of hidden units. The posterior distribution is computationally easy to compute because the hidden units are conditionally independent given input variables.

In terms of usage, sparse coding is better suited for applications that require good reconstruction and a single layer only. Sparse coding is particularly useful when the input data contains noise; for example, sparse coding shows state-of-the-art performance in image denoising [110]. Sparse coding also shows good performance in object recognition, using a single layer representation [131, 167]. In contrast, the RBM is better suited for applications that do not require accurate reconstruction since the RBM has binary latent variables. The RBM can be particularly useful when we want to learn hierarchical representations, and we will show several state-of-the-art applications in the next chapters.

Furthermore, we note that the sparsity regularization described in this chapter can be used for autoencoders as well. The autoencoder is a neural network for learning efficient embeddings, and it encodes input data using the hidden representation (with a single layer or multiple layers) so that the network can reconstruct the original input. In other words, it tries to minimize the reconstruction error between the input and the output of the network. Autoencoders can be trained by backpropagation; therefore, the sparsity regularization can be simply implemented by taking gradient steps via backpropagation followed by updates such as in Equation (4.12). For example, in the case of sigmoid activation function for hidden units, the same sparsity update rule as in Equation (4.12) can be used. For other activation functions (e.g., tanh or softmax), simply replacing "p" with the activation function value works well in practice. We note that sparse autoencoders can be also implemented using energy-based encoder/decoder networks [134, 136].

Empirically, sparse autoencoders learn qualitatively similar features as sparse RBMs and thereby can be used for learning interesting features from unlabeled data [55, 134, 136]. See also [83, 82] for empirical comparisons on deep autoencoders and DBNs. Finally, we emphasize that all of the above-mentioned algorithms are currently under active research, so new connections and insights may be discovered in the near future.

4.3 Visualization

4.3.1 Learning "pen-strokes" from handwritten digits

We applied the sparse RBM algorithm to the MNIST handwritten digit dataset. The dataset was downloaded from `http://yann.lecun.com/exdb/mnist/`. Each pixel was normalized to the unit interval, and we used PCA-whitening to reduce the dimension to 69 principal components for computational efficiency. (However, similar results were obtained without whitening.) We trained a sparse RBM with 69 visible units and 200 hidden units. The learned bases are shown in Figure 4.1. Each basis corresponds to one column of the weight matrix W left-multiplied by the unwhitening matrix. Many bases found by

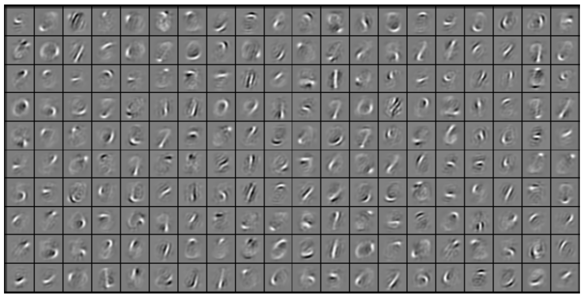


Figure 4.1: Bases learned from MNIST data using sparse RBM

the algorithm roughly represent different pen-strokes of which handwritten digits are comprised. This is consistent with results obtained by applying different algorithms to learn sparse representations of this data set (e.g., [136, 64]).

4.3.2 Learning from natural images

We also applied the algorithm to a training set a set of 14x14 pixel natural image patches, taken from a dataset compiled by van Hateren. The images were obtained from `http://hlab.phys.rug.nl/imlib/index.html` and preprocessed with whitened using $1/f$ whitening (see [122] for details of $1/f$ whitening procedure). We used 100,000 14x14 image patches randomly sampled from an ensemble of 2000 images; each subset of 200 patches was used as a mini-batch. We learned a sparse RBM model with 196 visible units and 400 hidden units. The learned bases are shown in Figure 4.2; they are oriented, Gabor-like bases and resemble the receptive fields of V1 simple cells.

4.3.3 Learning a two-layer model of natural images using sparse RBMs

We further learned a two-layer network by stacking one sparse RBM on top of another (see Section 4.2.2 for details). After learning, the second layer weights were quite sparse—most of the weights were very small, and only a few were either highly positive or highly negative. Positive weights represent excitatory connections between the first layer units and the second layer units, whereas negative elements represent inhibitory connections. By visualizing the second layer bases as shown in Figure 4.3, we observed bases that encoded

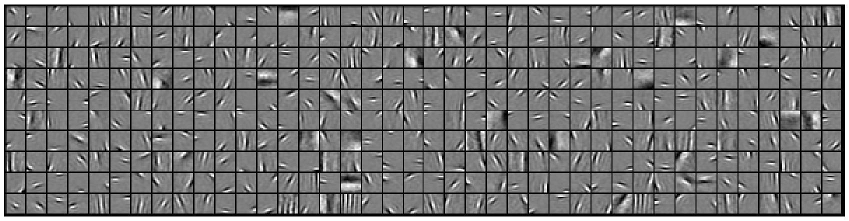


Figure 4.2: 400 first layer bases learned from the van Hateren natural image dataset, using our algorithm.

co-linear first layer bases as well as edge junctions. This shows that by extending the sparse RBM to two layers and using greedy learning, the model is able to learn bases that encode contours, angles, and junctions of edges.

4.4 Experimental results

4.4.1 Biological comparison

We evaluated our sparse DBN model to biological measurements. Specifically, we took Ito & Komatsu [70]'s characterization of V2 neurons in terms of its responses to a large class of angled bar stimuli, and quantitatively measured the degree to which the deep belief network algorithm generated similar responses. In the quantitative comparison, the encoding of these more complex "corner" features using sparse DBN matched well with the results from the Ito & Komatsu's study of biological V2 responses. The detailed results are reported in [92]. This result suggests that our sparse variant of deep belief networks may hold promise for learning higher-order features from natural images.

4.4.2 Machine learning applications

The original publication of sparse DBN work [92] focused on biological comparison. However, since our original publication, many authors have used this sparsity regularization (or its variants) and reported good results in a number of machine learning tasks, as described below.

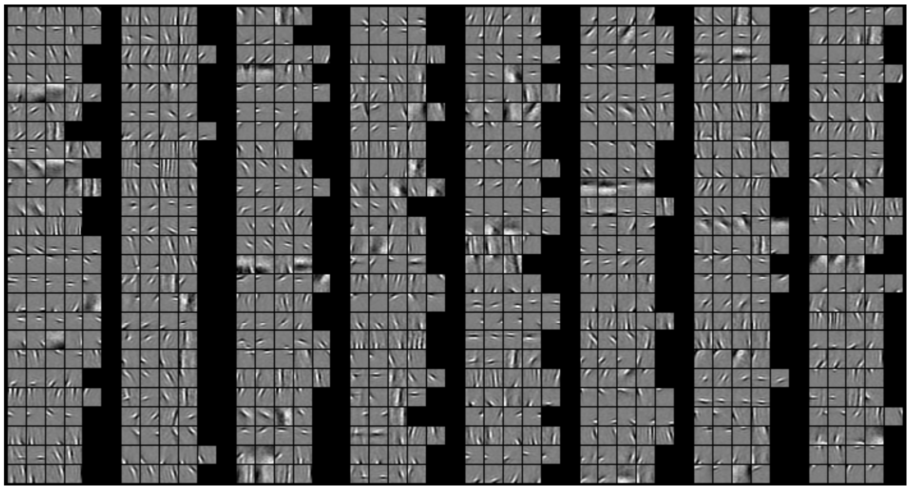


Figure 4.3: Visualization of 200 second layer bases, learned from natural images. Each small group of 3-5 (arranged in a row) images shows one second-layer unit; the leftmost patch in the group is a visualization of the second-layer basis, and is obtained by taking a weighted linear combination of the first layer "V1" bases to which it is connected. The next few patches in the group show the first layer bases that have the strongest weight connection to the second-layer basis.

Larochelle and Bengio (2008) [81] proposed a discriminative version of RBM. In the experiments, their model combined with sparsity regularization gave the best classification performance. In addition, Nair and Hinton [117] used a variant of update rule for sparsity regularization (i.e., the average hidden unit activations were measured via moving averages over the training examples) in training a third-order RBM and reported good results in 3D object recognition. Moreover, conditional RBMs [154] are reported to have good performance when using the sparsity regularization; see also [153] for implementation details.

As we will describe in Chapters 5 and 6, convolutional RBMs [93, 94, 45, 120] have highly overcomplete representations (i.e., more than 20x overcomplete). In such settings, sparsity is not only useful, but also necessary to learn meaningful representations. In other words, without using the sparsity regularization, the learned representations produced much worse results. In our related work, Goodfellow et al. [55] proposed a new evaluation metric for deep networks by measuring invariance. In the experiments, autoencoders with sparsity regularization gave significantly better invariance scores compared to autoencoders without using sparsity. Furthermore, a two-layer convolutional DBN (with sparsity regularization) gave the best invariance score among the models evaluated.

In summary, sparsity regularization in RBMs and DBNs has effect of (1) making the representation sparse and compact, thereby enabling to learn similar bases as those learned from sparse coding; (2) producing bases that are often easier to interpret (i.e., edges in image patches, pen-strokes in handwritten digits, etc.); and (3) leading to improved performance in machine learning tasks (i.e., classification or invariance test).

4.5 Summary

We presented a sparse variant of the deep belief network model. When trained on natural images, this model learns local, oriented, edge filters in the first layer. More interestingly, the second layer captures a variety of both colinear ("contour") features as well as corners and junctions, that in a quantitative comparison to measurements of V2 taken by Ito & Komatsu, appeared to give responses that were similar along several dimensions. This result suggests that sparse deep belief networks can capture interesting high-order features from natural image statistics, and we believe that the proposed method holds promise for learning higher-order features from various data types. Finally, sparsity regularization described in this chapter improves performance of autoencoders, RBMs, and DBNs in many machine learning tasks.

Chapter 5

Convolutional Deep Belief Networks

5.1 Introduction

The visual world can be described at many levels: pixel intensities, edges, object parts, objects, and beyond. The prospect of learning hierarchical models that simultaneously represent multiple levels has recently generated much interest. Ideally, such "deep" representations would learn hierarchies of feature detectors and further be able to combine top-down and bottom-up processing of an image. For instance, lower layers could support object detection by spotting low-level features indicative of object parts. Conversely, information about objects in the higher layers could be highly specific (thus more informative for object recognition) and resolve lower-level ambiguities in the image or infer the locations of hidden object parts. In this chapter, we build upon the deep belief networks because we are interested in learning a generative model of images which can be trained in a purely unsupervised manner.

While DBNs have been successful in controlled domains, scaling them to realistic-sized (e.g., 200x200 pixel) images remains challenging for two reasons. First, images are high-dimensional, so the algorithms must scale gracefully and be computationally tractable even when applied to large images. Second, objects can appear at arbitrary locations in images; thus it is desirable that representations be invariant at least to local translations of the input. We address these issues by incorporating translation invariance. Like [89] and [61], we learn feature detectors which are shared among all locations in an image because a feature detector which captures useful information in one part of an image can pick up the same information elsewhere. Thus, our model can represent large images using only a small number of feature detectors.

This chapter presents the *convolutional deep belief network*, a hierarchical generative model that scales to full-sized images. We also present *probabilistic max-pooling*, a novel technique that allows higher-layer units to cover larger areas of the input in a probabilistically sound way. To the best of our knowledge, ours is the first unsupervised, translation-invariant deep learning model which scales to realistic image sizes and supports full probabilistic inference. The first, second, and third layers of our network learn edge detectors, object parts, and objects respectively. We show that these representations achieve excellent performance on several visual recognition tasks and allow "hidden" object parts to be inferred from high-level object information.

5.2 Algorithm

Both RBMs and DBNs ignore the 2-D structure of images, so weights that detect a given feature must be learned separately for each location. This redundancy makes it difficult to scale these models to full images. One possible way of scaling up is to use massive parallel computation (such as using GPUs), as shown in [132]. However, this method may still suffer from having a huge number of parameters. In this section, we present a new method that scales up DBNs using weight-sharing. Specifically, we introduce our model, the convolutional DBN, where weights are shared among all locations in an image. This model scales well because inference can be done efficiently using convolution.

5.2.1 Notation

For notational convenience, we will make several simplifying assumptions. First, we assume that all inputs to the algorithm are $N_V \times N_V$ images, even though there is no requirement that the inputs be square, equally sized, or even two-dimensional. We also assume that all units are binary-valued, while noting that it is straightforward to extend the formulation to the real-valued visible units (see Section 4.2). We use $*$ to denote convolution,[1] and $\bullet$ to denote element-wise product followed by summation, i.e., $A \bullet B = \mathrm{tr} A^T B$. We place a tilde above an array ($\tilde{A}$) to denote flipping the array horizontally and vertically.

5.2.2 Convolutional RBM

First, we introduce the convolutional RBM (CRBM). Intuitively, the CRBM is similar to the RBM, but the weights between the hidden and visible layers are shared among all locations in an image. The basic CRBM consists of two layers: an input layer V and a hidden layer H (corresponding to the lower two layers in Figure 5.1). The input layer consists of an $N_V \times N_V$ array of binary units. The hidden layer consists of K "groups," where each group is an $N_H \times N_H$ array of binary units, resulting in ${N_H}^2 K$ hidden units. Each of the K groups is associated with a $N_W \times N_W$ filter ($N_W \triangleq N_V - N_H + 1$); the filter weights are shared across all the hidden units within the group. In addition, each hidden group has a bias b_k and all visible units share a single bias c.

We define the energy function $E(\mathbf{v}, \mathbf{h})$ as:

$$\begin{aligned} P(\mathbf{v}, \mathbf{h}) &= \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h})) \\ E(\mathbf{v}, \mathbf{h}) &= -\sum_{k=1}^{K} \sum_{i,j=1}^{N_H} \sum_{r,s=1}^{N_W} h_{ij}^k W_{rs}^k v_{i+r-1,j+s-1} - \sum_{k=1}^{K} b_k \sum_{i,j=1}^{N_H} h_{ij}^k - c \sum_{i,j=1}^{N_V} v_{ij}. \qquad (5.1) \end{aligned}$$

Using the operators defined previously,

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{k=1}^{K} h^k \bullet (\tilde{W}^k * v) - \sum_{k=1}^{K} b_k \sum_{i,j} h_{i,j}^k - c \sum_{i,j} v_{ij}.$$

As with standard RBMs (Section 4.2), we can perform block Gibbs sampling using the

[1]The convolution of an $m \times m$ array with an $n \times n$ array may result in an $(m + n - 1) \times (m + n - 1)$ array or an $(m - n + 1) \times (m - n + 1)$ array. Rather than inventing a cumbersome notation to distinguish between these cases, we let it be determined by context.

following conditional distributions:

$$
\begin{aligned}
P(h_{ij}^k = 1|\mathbf{v}) &= \sigma((\tilde{W}^k * v)_{ij} + b_k) \\
P(v_{ij} = 1|\mathbf{h}) &= \sigma((\sum_k W^k * h^k)_{ij} + c),
\end{aligned}
$$

where σ is the sigmoid function. Gibbs sampling forms the basis of our inference and learning algorithms.

5.2.3 Probabilistic max-pooling

In order to learn high-level representations, we stack CRBMs into a multi-layer architecture analogous to DBNs. This architecture is based on a novel operation that we call *probabilistic max-pooling*.

In general, higher-level feature detectors need information from progressively larger input regions. Existing translation-invariant representations, such as convolutional networks, often involve two kinds of alternating layers: "detection" layers, where responses are computed by convolving a feature detector with the previous layer, and "pooling" layers, which shrink the representation of the detection layers by a constant factor. More specifically, each unit in a pooling layer computes the maximum activation of the units in a small region of the detection layer. Shrinking the representation with max-pooling allows higher-layer representations to be invariant to small translations of the input and reduces the computational burden.

Max-pooling was intended only for deterministic and feed-forward architectures, and it is difficult to perform probabilistic inference (e.g., computing posterior probabilities) since max-pooling is a deterministic operator. In contrast, we are interested in a *generative* model of images which supports full probabilisctic inference; hence, we designed our generative model so that inference involves max-pooling-like behavior.

To simplify the notation, we consider a model with a visible layer V, a detection layer H, and a pooling layer P, as shown in Figure 5.1. The detection and pooling layers both have K groups of units, and each group of the pooling layer has $N_P \times N_P$ binary units. For each $k \in \{1, ..., K\}$, the pooling layer P^k shrinks the representation of the detection

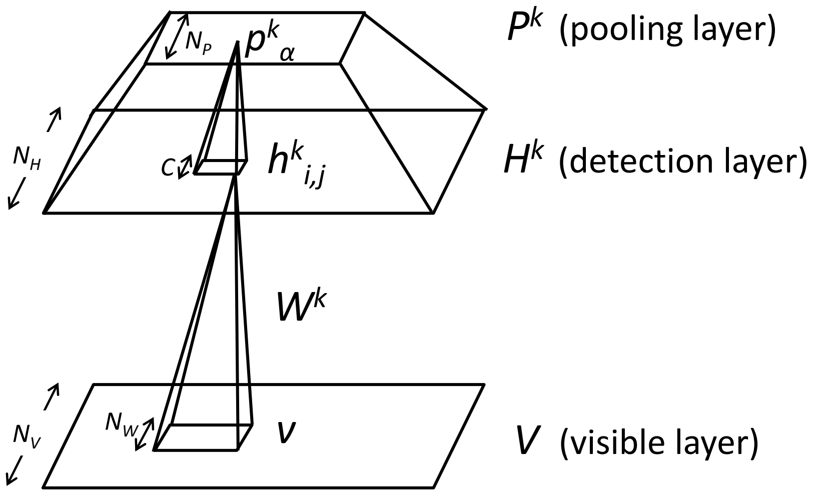


Figure 5.1: Convolutional RBM with probabilistic max-pooling. For simplicity, only group k of the detection layer and the pooling layer are shown. The basic CRBM corresponds to a simplified structure with only visible layer and detection (hidden) layer. See text for details.

layer H^k by a factor of C along each dimension, where C is a small integer such as 2 or 3. In other words, the detection layer H^k is partitioned into blocks of size $C \times C$, and each block α is connected to exactly one binary unit p^k_α in the pooling layer (i.e., $N_P = N_H/C$). Formally, we define $B_\alpha \triangleq \{(i, j) : h_{ij} \text{ belongs to the block } \alpha.\}$.

The detection units in the block B_α and the pooling unit p_α are connected in a single potential which enforces the following constraints: at most one of the detection units may be on, and the pooling unit is on if and only if a detection unit is on. By adding this constraint, we can efficiently sample from the network without explicitly enumerating all 2^{C^2} configurations, as we show later. With the constraint, we can consider these $C^2 + 1$ units as a single random variable which may take on one of $C^2 + 1$ possible values: one value for each of the detection units being on, and one value indicating that all units are off.

We formally define the energy function of this simplified probabilistic max-pooling-CRBM as follows:

$$\begin{aligned} E(\mathbf{v},\mathbf{h}) &= -\sum_k \sum_{i,j} \left(h^k_{i,j} (\tilde{W}^k * v)_{i,j} + b_k h^k_{i,j} \right) - c \sum_{i,j} v_{i,j} \\ \text{subject to} & \quad \sum_{(i,j)\in B_\alpha} h^k_{i,j} \leq 1, \quad \forall k, \alpha. \end{aligned}$$

We now discuss sampling the detection layer H and the pooling layer P given the visible layer V. Group k receives the following bottom-up signal from layer V:

$$I(h^k_{ij}) \triangleq b_k + (\tilde{W}^k * v)_{ij}. \tag{5.2}$$

Now, we sample each block independently as a multinomial function of its inputs. Suppose $h^k_{i,j}$ is a hidden unit contained in block α (i.e., $(i, j) \in B_\alpha$), the increase in energy caused by turning on unit $h^k_{i,j}$ is $-I(h^k_{i,j})$, and the conditional probability is given by:

$$P(h^k_{i,j} = 1|\mathbf{v}) = \frac{\exp(I(h^k_{i,j}))}{1 + \sum_{(i',j')\in B_\alpha} \exp(I(h^k_{i',j'}))} \tag{5.3}$$

$$P(p^k_\alpha = 0|\mathbf{v}) = \frac{1}{1 + \sum_{(i',j')\in B_\alpha} \exp(I(h^k_{i',j'}))}. \tag{5.4}$$

Sampling the visible layer V given the hidden layer H can be performed in the same way as described in Section 5.2.2.

5.2.4 Training via sparsity regularization

Our model is overcomplete in that the size of the representation is much larger than the size of the inputs. In fact, since the first hidden layer of the network contains K groups of units, each roughly the size of the image, it is overcomplete roughly by a factor of K. In general, overcomplete models run the risk of learning trivial solutions, such as feature detectors representing single pixels. One common solution is to force the representation to be "sparse," meaning only a tiny fraction of the units should be active in relation to a given stimulus. In our approach, following Section 4.2.1, we regularize the objective function (log-likelihood) to encourage each hidden unit group to have a mean activation close to a small constant. Specifically, we found that the following simple update (followed by each contrastive divergence update) worked well in practice:

$$\Delta b_k \propto p - \frac{1}{N_H^2} \sum_{i,j} P(h_{ij}^k = 1|\mathbf{v}), \tag{5.5}$$

where p is a target sparsity, and each image is treated as a mini-batch. The learning rate for sparsity update was chosen as a value that makes the hidden group's average activation (over entire training data) close to the target sparsity, while allowing variations depending on specific input images.

5.2.5 Convolutional deep belief network

Finally, we are ready to define the convolutional deep belief network (CDBN), our hierarchical generative model for full-sized images. Analogously to DBNs, this architecture consists of several max-pooling-CRBMs stacked on top of one another. The network defines an energy function by summing together the energy functions for all the individual pairs of layers. Training is accomplished with the same greedy, layer-wise procedure described in Section 4.2: once a given layer is trained, its weights are frozen, and its activations are used as input to the next layer. There is one technical point about learning the biases for each intermediate hidden layer. Specifically, the biases of a given layer are learned twice: once when the layer is treated as the "hidden" layer of the CRBM (using the lower layer as visible units), and once when it is treated as the "visible" layer (using the upper layer as hidden units). We resolved this problem by simply fixing the biases with the learned hidden biases in the former case (i.e., using only the biases learned when treating the given layer as the hidden layer of the CRBM). However, we note that a possibly better solution would be to train jointly all the weights for the entire CDBN, using the greedily trained weights as the initialization.

5.2.6 Hierarchical probabilistic inference

Once the parameters have all been learned, we compute the network's representation of an image by sampling from the joint distribution over all of the hidden layers conditioned on the input image. To sample from this distribution, we use block Gibbs sampling, where each layer's units are sampled in parallel (see Sections 4.2 & 5.2.3).

To illustrate the algorithm, we describe a case with one visible layer V, a detection layer H, a pooling layer P, and another, subsequently-higher detection layer H'. Suppose H' has K' groups of nodes, and there is a set of shared weights $\Gamma = \{\Gamma^{1,1}, \ldots, \Gamma^{K,K'}\}$, where $\Gamma^{k,\ell}$ is a weight matrix connecting pooling unit P^k to detection unit H'^{ℓ}. The definition can be extended to deeper networks in a straightforward way.

Note that an energy function for this sub-network consists of two kinds of potentials: unary terms for each of the groups in the detection layers, and interaction terms between V and H and between P and H':

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}, \mathbf{p}, \mathbf{h}') &= -\sum_k v \bullet (W^k * h^k) - \sum_k b_k \sum_{ij} h^k_{ij} \\ &\quad -\sum_{k,\ell} p^k \bullet (\Gamma^{k\ell} * h'^{\ell}) - \sum_{\ell} b'_{\ell} \sum_{ij} h'^{\ell}_{ij} \end{aligned}$$

To sample the detection layer H and pooling layer P, note that the detection layer H^k receives the following bottom-up signal from layer V:

$$I(h^k_{ij}) \triangleq b_k + (\tilde{W}^k * v)_{ij}, \tag{5.6}$$

and the pooling layer P^k receives the following top-down signal from layer H':

$$I(p^k_{\alpha}) \triangleq \sum_{\ell} (\Gamma^{k\ell} * h'^{\ell})_{\alpha}. \tag{5.7}$$

Now, we sample each of the blocks independently as a multinomial function of their inputs, as in Section 5.2.3. If $(i, j) \in B_\alpha$, the conditional probability is given by:

$$\begin{aligned} P(h^k_{i,j} = 1|\mathbf{v}, \mathbf{h}') &= \frac{\exp(I(h^k_{i,j}) + I(p^k_\alpha))}{1 + \sum_{(i',j') \in B_\alpha} \exp(I(h^k_{i',j'}) + I(p^k_\alpha))} \\ P(p^k_\alpha = 0|\mathbf{v}, \mathbf{h}') &= \frac{1}{1 + \sum_{(i',j') \in B_\alpha} \exp(I(h^k_{i',j'}) + I(p^k_\alpha))}. \end{aligned}$$

As an alternative to block Gibbs sampling, mean-field can be used to approximate the posterior distribution. In all our experiments except for Section 5.3.5, we used the mean-field approximation to estimate the hidden layer activations given the input images.[2]

5.2.7 Discussion

Our model used undirected connections between layers. This approach contrasts with Hinton et al. (2006) [65], which used undirected connections between the top two layers, and top-down directed connections for the layers below. Hinton et al. [65] proposed approximating the posterior distribution using a single bottom-up pass. This feed-forward approach often can effectively estimate the posterior when the image contains no occlusions or ambiguities, but the higher layers cannot help resolve ambiguities in the lower layers. This is due to feed-forward computation, where the lower layer activations are not affected by the higher layer activations. Although Gibbs sampling may more accurately estimate the posterior, applying block Gibbs sampling would be difficult because the nodes in a given layer are not conditionally independent of one another given the layers above and below. In contrast, our treatment using undirected edges enables combining bottom-up and top-down information more efficiently, as shown in Section 5.3.5.

In our approach, probabilistic max-pooling helps to address scalability by shrinking the higher layers. Moreover, weight-sharing (convolutions) further speeds up the algorithm. For example, convolutions between K filters and an input image are more efficient both in memory and time than repeating ${KN_H}^2$ times of innder products between the input image and each of basis vectors (without weight sharing). As a result, inference in a three-layer network (with 200x200 input images) using weight-sharing but without max-pooling was

[2] We found that a small number of mean-field iterations (e.g. five iterations) sufficed.

about 10 times slower. Without weight-sharing, it was more than 100 times slower.

In contemporary work that was done independently of ours, Desjardins and Bengio [45] also applied convolutional weight-sharing to RBMs and experimented on small image patches. Our work, however, developed more sophisticated elements such as probabilistic max-pooling to make the algorithm more scalable.

In another contemporary work, Salakhutdinov and Hinton (2009) [142] proposed an algorithm to train Boltzmann machines with layer-wise connections (i.e., the same topological structure as in deep belief networks, but with undirected connections). They called this model the deep Boltzmann machine (DBM). Specifically, they proposed algorithms for pretraining and fine-tuning DBMs. Our treatment of undirected connections is related with DBMs. However, our model is different from theirs because we apply convolutional structures and incorporate probabilistic max-pooling into the architecture. Although their work is not convolutional and does not scale to as large images as our model, we note that their pretraining algorithm (a modification of contrastive divergence that duplicates the visible units or hidden units when training the RBMs) or fine-tuning algorithm (joint training of all the parameters using a stochastic approximation procedure [170, 174, 157]) can be also applied to our model to improve the training procedure.

5.3 Experimental results

5.3.1 Learning hierarchical representations from natural images

We first tested our model's ability to learn hierarchical representations of natural images. Specifically, we trained a CDBN with two hidden layers from the Kyoto natural image dataset.[3] The first layer consisted of 24 groups (or "bases")[4] of 10x10 pixel filters, while the second layer consisted of 100 bases, each one 10x10 as well. Since the images were real-valued, we used Gaussian visible units for the first-layer CRBM. The pooling ratio C for each layer was 2, so the second-layer bases covered roughly twice as large an area as the first-layer bases. We used 0.003 as the target sparsity for the first layer and 0.005 for

[3] `http://www.cnbc.cmu.edu/cplab/data_kyoto.html`

[4] We will call one hidden group's weights a "basis."

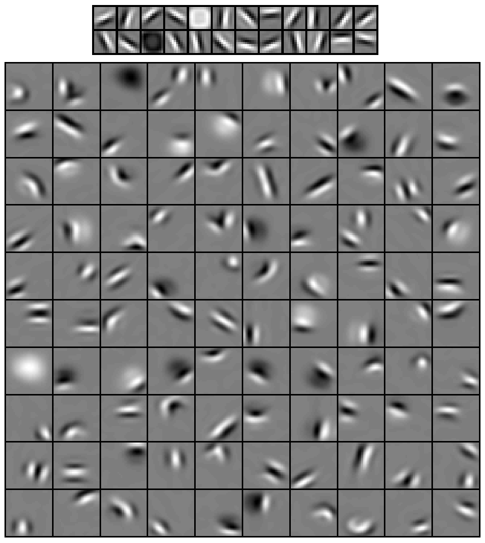


Figure 5.2: The first layer bases (top) and the second layer bases (bottom) learned from natural images. Each second layer basis (filter) was visualized as a weighted linear combination of the first layer bases.

the second layer.

As shown in Figure 5.2 (top), the learned first layer bases are oriented, localized edge filters; this result is consistent with much previous work [122, 14, 136, 125, 64, 160]. We note that sparsity regularization during training was necessary for learning these oriented edge filters; when this term was removed, the algorithm failed to learn oriented edges. The learned second layer bases are shown in Figure 5.2 (bottom), and many of them empirically responded selectively to contours, corners, angles, and surface boundaries in the images. This result is qualitatively consistent with previous work [70, 92, 67].

5.3.2 Self-taught learning for object recognition

In the self-taught learning framework [131], a large amount of unlabeled data can help supervised learning tasks, even when the unlabeled data do not share the same class labels or the same generative distribution with the labeled data. In previous work, sparse coding was used to train single-layer representations from unlabeled data, and the learned representations were used to construct features for supervised learning tasks.

We used a similar procedure to evaluate our two-layer CDBN, described in Section 5.3.1, on the Caltech-101 object classification task. More specifically, given an image from the Caltech-101 dataset [50], we scaled the image so that its longer side was 150 pixels, and computed the activations of the first and second (pooling) layers of our CDBN. We repeated this procedure after reducing the input image by half and concatenated all the activations to construct features. We used an SVM with a spatial pyramid matching kernel for classification, and the parameters of the SVM were cross-validated. We randomly selected 15 or 30 images per class for training test and testing set, and normalized the result such that classification accuracy for each class was equally weighted (following the standard protocol). We report results averaged over 10 random trials, as shown in Table 5.1. First, we observe that combining the first and second layers significantly improves the classification accuracy relative to the first layer alone. Overall, we achieve 57.7% test accuracy using 15 training images per class, and 65.4% test accuracy using 30 training images per class. Our result is competitive with state-of-the-art results using a single type of highly-specialized features, such as SIFT, geometric blur, and shape-context [88, 18, 175]. In addition, recall that the CDBN was trained entirely from natural scenes, which are completely unrelated to the classification task. Hence, the strong performance of these features implies that our CDBN learned a highly general representation of images.

We note that current state-of-the-art methods use multiple kernels (or features) together, instead of using a single type of features. For example, Gehler and Nowozin (2009) [54] reported better performance than ours (77.7% for 30 training images/class), but they combined many state-of-the-art features (or kernels) to improve performance. In another approach, Yu et al. [173] used kernel regularization using a (previously published) state-of-the-art kernel matrix to improve the performance of their convolutional neural network model (achieving 67.4% for 30 training examples/class). However, we expect our features can be also used in both settings to further improve the performance.

5.3.3 Handwritten digit classification

We also evaluated the performance of our model on the MNIST handwritten digit classification task, a widely-used benchmark for testing hierarchical representations. We trained

Training Size	15	30
CDBN (first layer)	53.2%	60.5%
CDBN (first+second layers)	57.7%	65.4%
Raina et al. (2007) [131]	46.6%	-
Ranzato et al. (2007) [135]	-	54.0%
Mutch and Lowe (2006) [116]	51.0%	56.0%
Lazebnik et al. (2006) [88]	54.0%	64.6%
Zhang et al. (2006) [175]	59.0%	66.2%

Table 5.1: Classification accuracy for the Caltech-101 data

Labeled training samples	1,000	2,000	3,000	5,000	60,000
CDBN	2.62%	2.13%	1.91%	1.59%	0.82%
Ranzato et al. (2007) [135]	3.21%	2.53%	-	1.52%	0.64%
Hinton and Salakhutdinov [66]	-	-	-	-	1.20%
Weston et al. (2008) [164]	2.73%	-	1.83%	-	1.50%

Table 5.2: Test error for MNIST dataset

40 first layer bases from MNIST digits, each 12x12 pixels, and 40 second layer bases, each 6x6. The pooling ratio C was 2 for both layers. The first layer bases learned pen-strokes that comprise the digits, and the second layer bases learned bigger digit-parts that combine the pen-strokes. We constructed feature vectors by concatenating the first and second (pooling) layer activations, and used an SVM for classification using these features. For each labeled training set size, we report the test error averaged over 10 randomly chosen training sets, as shown in Table 5.2. For the full training set, we obtained 0.8% test error. Our result is comparable to the state-of-the-art [135].

5.3.4 Unsupervised learning of object parts

We now show that our algorithm can learn hierarchical object-part representations without knowing the position of the objects and the object-parts. Building on the first layer representation learned from natural images, we trained two additional CDBN layers using unlabeled images from single Caltech-101 categories. Training was on up to 100 images, and testing was on different images than the training set. The pooling ratio for the first

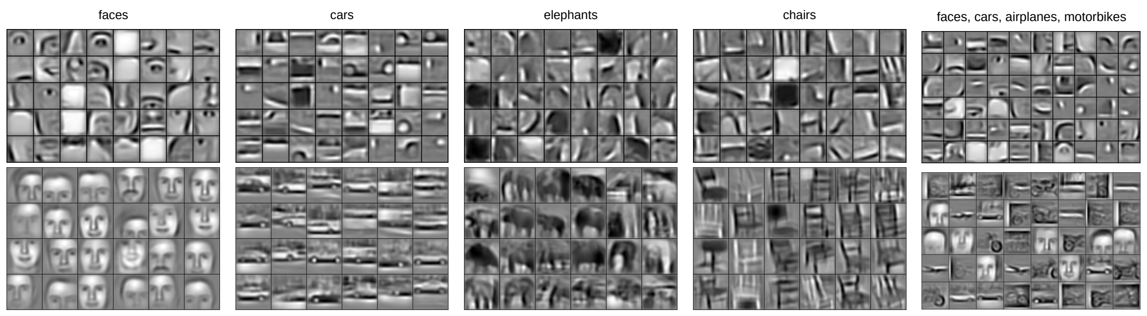
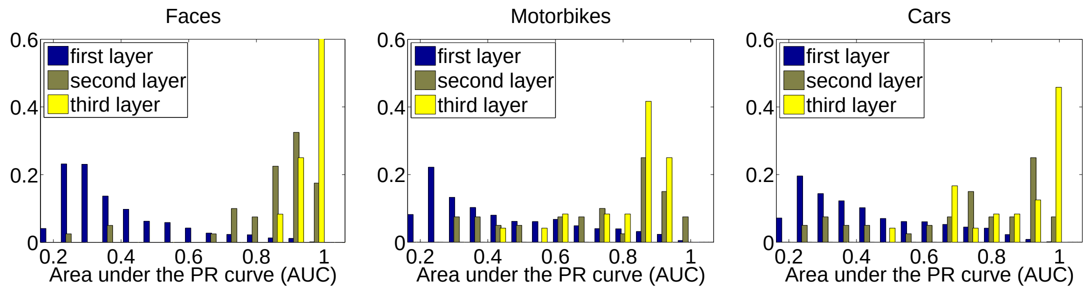


Figure 5.3: Columns 1-4: the second layer bases (top) and the third layer bases (bottom) learned from specific object categories. Column 5: the second layer bases (top) and the third layer bases (bottom) learned from a mixture of four object categories (faces, cars, airplanes, motorbikes).

layer was set as 3. The second layer contained 40 bases, each 10x10, and the third layer contained 24 bases, each 14x14. The pooling ratio in both cases was 2. We used 0.005 as the target sparsity level in both the second and third layers. As shown in Figure 5.3, the second layer learned features that corresponded to object parts, even though the algorithm was not given any labels that specified the locations of either the objects or their parts. The third layer learned to combine the second layer's part representations into more complex, higher-level features. Our model successfully learned hierarchical object-part representations of most of the other Caltech-101 categories as well. We note that some of these categories (such as elephants and chairs) have fairly high intra-class appearance variation, due to deformable shapes or different viewpoints. Despite this variation, our model still learns hierarchical, part-based representations fairly robustly.

Higher layers in the CDBN learn features which are not only higher level, but also more specific to particular object categories. We quantitatively measured the specificity of each layer by determining how indicative each individual feature is of object categories. (This setting contrasts with most work in object classification, which focuses on the informativeness of the entire feature set, rather than individual features.) More specifically, we considered three CDBNs trained on faces, motorbikes, and cars, respectively. For each CDBN, we tested the informativeness of individual features from each layer for distinguishing among these three categories. For each feature, we computed the area under the precision-recall curve (larger means more specific). In detail, for any given image, we computed the layer-wise activations using our algorithm, partitioned the activation into $L \times L$ regions for each group, and computed the $q\%$ highest quantile activation for each region and each group. If the $q\%$ highest quantile activation in region i is γ, we then defined a Bernoulli random variable $X_{i,L,q}$ with probability γ of being 1. To measure the informativeness between a feature and the class label, we computed the mutual information between $X_{i,L,q}$ and the class label. We report results using (L, q) values that maximized the average mutual information (averaging over i). Then for each feature, by comparing its values over positive and negative examples, we obtained the precision-recall curve for each classification problem. As shown in Figure 5.4, the higher-level representations are more selective for the specific object class.

We further tested if the CDBN can learn hierarchical object-part representations when trained on images from several object categories (rather than just one). We trained the second and third layer representations using unlabeled images randomly selected from four object categories (cars, faces, motorbikes, and airplanes). As shown in Figure 5.3 (far right), the second layer learns class-specific and as shared parts, and the third layer learns more object-specific representations. The training examples were unlabeled, so in a sense, the third layer implicitly clusters the images by object category. As before, we quantitatively measured the specificity of each layer's individual features to object categories. Since the training was completely unsupervised, whereas the AUC-PR statistic requires knowing which specific object or object parts the learned bases should represent, we computed the conditional entropy instead. Specifically, we computed the quantile features γ for each layer as previously described, and measured conditional entropy $H(class|\gamma > 0.95)$. Informally speaking, conditional entropy measures the entropy of the posterior over class labels when a feature is active. Since lower conditional entropy corresponds to a more peaked posterior, it indicates greater specificity. As shown in Figure 5.5, the higher-layer features have progressively less conditional entropy, suggesting that they activate more selectively to specific object classes.



Features	Faces	Motorbikes	Cars
First layer	0.39±0.17	0.44±0.21	0.43±0.19
Second layer	0.86±0.13	0.69±0.22	0.72±0.23
Third layer	0.95±0.03	0.81±0.13	0.87±0.15

Figure 5.4: (top) Histogram of the area under the precision-recall curve (AUC-PR) for three classification problems using class-specific object-part representations. (bottom) Average AUC-PR for each classification problem.

5.3.5 Hierarchical probabilistic inference

Lee and Mumford (2003) [100] proposed that the human visual cortex can be modeled conceptually as performing "hierarchical Bayesian inference." For example, if you observe a face image with its left half in dark illumination, you can still recognize the face and further infer the darkened parts by combining the image with your prior knowledge of faces. In this experiment, we show that our model can tractably perform such (approximate) hierarchical probabilistic inference in full-sized images. More specifically, we tested the network's ability to infer the locations of hidden object parts.

To generate examples for evaluation, we used Caltech-101 face images (distinct from the ones the network was trained on). For each image, we simulated an occlusion by zeroing out the left half of the image. We then sampled from the joint posterior over all the hidden layers by performing Gibbs sampling. Figure 5.6 shows a visualization of these samples. To ensure that the filling-in required top-down information, we compare with a "control" condition where only a single upward pass was performed.

In the control (upward-pass only) condition, since there is no evidence from the first

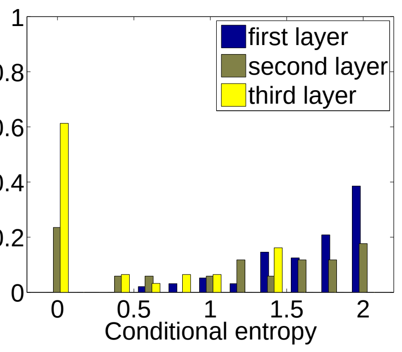


Figure 5.5: Histogram of conditional entropy for the representation learned from the mixture of four object classes.

layer, the second layer does not respond to the left side. However, with full Gibbs sampling, the bottom-up inputs combine with the context provided by the third layer which has detected the object. This combined evidence significantly improves the second layer representation. Selected examples are shown in Figure 5.6. We note that our method may not be competitive to state-of-the-art face completion algorithms using much prior knowledge and heuristics (e.g., using symmetry). However, we find these results promising and view them as a proof of concept for top-down inference.

5.4 Multi-class image segmentation

In previous sections, we have seen that the CDBN can learn high-level features that are useful image classification tasks. In this section, we consider object segmentation tasks and show that the CDBN feature representations can be also useful for object segmentation tasks.[5]

Image segmentation is a task where we want to partition an image into multiple segments (set of pixels). Specifically, we want to classify each pixel in the image into a specific object class or the background. The multi-class image segmentation can be considered as an object recognition task that requires more detailed labels (such as pixel-label correspondences).

[5]This is unpublished joint work with Daphne Koller and Stephen Gould.

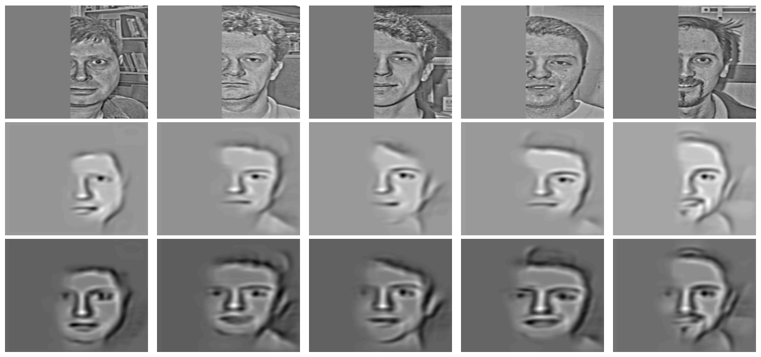


Figure 5.6: Hierarchical probabilistic inference. For each column: (top) input image. (middle) reconstruction from the second layer units after single bottom-up pass, by projecting the second layer activations into the image space. (bottom) reconstruction from the second layer units after 20 iterations of block Gibbs sampling.

Typically, most object segmentation algorithms use low-level features, such as local gradient, texture, and color. Therefore, if we can obtain higher-level features (e.g., object parts), these features can be helpful in image segmentation because they are more informative to class labels. (See [25] for related work using combining high-level and low-level information.) The main idea here is to first learn class-specific, higher-level representation such as object parts in the images and then use the learned features in a state-of-the-art computer vision algorithm. Specifically in our experiments, we use a region-based object segmentation algorithm by Gould et al. (2009) [57].

It is beyond the scope of this thesis to provide full details of the region-based image segmentation algorithm. Briefly speaking, the region-based segmentation algorithm incorporates interactions between many factors, such as interactions between pixels and regions, interactions between pixels, and interactions between regions. Given an input image, the algorithm first over-segments an image into many superpixels (adjacent coherent groups of pixels), computes features for each superpixel, and then tries to assign class labels for each pixel. The algorithm initially may make some mistakes; therefore, the algorithm will try to refine the segmentation via greedy hill-climbing approach based on the energy function. Specifically, the algorithm iteratively proposes a new move and then accepts the move if it improves the energy function.

In our experiments, we used the MSRC-21 dataset, a standard benchmark for image

Method	Test segmentation accuracy
Pixel	75.5%
Region	75.8%
Pixel + CDBN	77.7%
Region + CDBN	78.0%

Table 5.3: Test accuracy on the MSRC-21 segmentation task. Table shows mean pixel accuracy over all semantic classes.

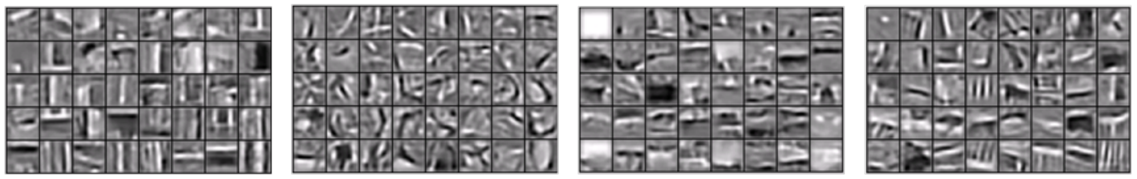


Figure 5.7: Visualization of the learned class-specific second layer bases from MSRC images (books, bicycles, airplanes, and chairs).

segmentation task. Given the 591 images, we randomly split into 296 training and 295 testing images. For each image, we scaled the image so that its longer side was 180 pixels. Then, we trained the second layer CDBN features from the MSRC training images (on top of the first layer bases learned from natural images), with the same setting as in Section 5.3.4. Specifically, we trained 40 bases from each of the seven object classes (airplanes, faces, cars, bicycles, flowers, books, and chairs).[6] Figure 5.7 visualizes the learned object-part representations from MSRC-21 dataset. Note that the MSRC-21 datasets have significantly more variations than Caltech-101 images, in terms of scale, position, shape, and appearance. It appears that CDBNs discover meaningful higher-level features even though no information about bounding boxes of the objects was used for training.

After training the CDBNs, we computed the CDBN features (by rescaling the activations of the second layers to match the input image size) for each image and used these features to augment the original feature set of the Gould et al. implementation. For evaluation metric, we used "mean pixel accuracy" which is an average over the pixel-wise segmentation accuracies over all semantic classes. Here, the pixel-wise segmentation accuracy

[6]This choice was arbitrary since the baseline implementation used in this experiment required saving all the features into a non-compressed file format and thereby required a huge disk space.

is defined as (number of correctly classified pixels)/(number of all pixels) for each class. We measured the test segmentation accuracy in two settings, one with pixel information only and the other with pixel and region information together, using the region-based segmentation implementation. The summary results are shown in Table 5.3. In both settings, augmenting the CDBN features achieves about 10% error reduction over a state-of-the-art algorithm.

5.5 Summary

We presented the convolutional deep belief network, a model for automatically learning hierarchical representations of images from unlabeled data. Our algorithm scales up to realistic image sizes, such as 200x200 pixels. We showed that our CDBN can learn the parts of objects in an unsupervised way. In addition, we showed that our model performs very well in object recognition and multi-class segmentation tasks. We believe our approach holds promise as a scalable algorithm for learning hierarchical representations from high-dimensional, complex data.

Chapter 6

Convolutional DBNs for Audio Classification

6.1 Introduction

Understanding how to recognize complex, high-dimensional audio data is one of the greatest challenges of our time. Previous work [148, 122] has revealed that learning a sparse representation of auditory signals leads to filters that closely correspond to those of neurons in early audio processing in mammals. For example, when sparse coding models were applied to natural sounds or speech, the learned representations (basis vectors) showed a striking resemblance to the cochlear filters in the auditory cortex. In related work, Grosse et al. [61] proposed an efficient sparse coding algorithm for auditory signals and demonstrated its usefulness in audio classification tasks.

However, the proposed methods have been applied to learn relatively shallow, one-layer representations. Learning more complex, higher-level representation is still a non-trivial and challenging problem. Recently, many promising approaches have been proposed to learn the processing steps of the “second stage and beyond” [65, 136, 16, 83, 92]. However, to the best of our knowledge, “deep learning” approaches have not been yet extensively applied to auditory data.

In this chapter, we will apply convolutional deep belief networks to unlabeled auditory data (such as speech and music) and evaluate the learned feature representations on several audio classification tasks. In the case of speech data, we will show that the learned features correspond to phones/phonemes. In addition, our feature representations outperform other baseline features (spectrogram and MFCC) for multiple audio classification tasks. In particular, our method compares favorably to other state-of-the-art algorithms for the speaker identification task. For the phone classification task, MFCC features can be augmented with our features to improve accuracy. We also show for certain tasks that the second-layer features produce higher accuracy than the first-layer features, which justifies the use of deep learning approaches for audio classification. Finally, we show that our features give better performance in comparison to other baseline features for music classification tasks. In our experiments, the learned features often performed much better than other baseline features when there were only a small number of labeled training examples. To the best of our knowledge, we are the first to apply deep learning algorithms to a range of audio classification tasks.

6.2 Algorithm

6.2.1 Convolutional deep belief networks for time-series data

We will follow the CRBM formulation of Chapter 5 and adapt it to a one-dimensional setting. For the purpose of this explanation, we assume that all inputs to the algorithm are single-channel time-series data with N_V frames (an N_V-dimensional vector); however, the formulation can be straightforwardly extended to the case of multiple channels.

Consider the input layer consisting of an n_V-dimensional array of binary units. To construct the hidden layer, consider K n_W-dimensional filter weights W^K (also referred to as "bases" throughout this chapter). The hidden layer consists of K "groups" of n_H-dimensional arrays (where $n_H \triangleq n_V - n_W + 1$) with units in group k sharing the weights W^k. There is also a shared bias b_k for each group, as well as a shared bias c for the visible units. The energy function can then be defined as:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{k=1}^{K}\sum_{j=1}^{N_H}\sum_{r=1}^{N_W} h_j^k W_r^k v_{j+r-1} - \sum_{k=1}^{K} b_k \sum_{j=1}^{N_H} h_j^k - c\sum_{i=1}^{N_V} v_i. \tag{6.1}$$

Similarly, the energy function of CRBM with real-valued visible units can be defined as:

$$E(\mathbf{v},\mathbf{h}) = \frac{1}{2}\sum_{i}^{N_V} v_i^2 - \sum_{k=1}^{K}\sum_{j=1}^{N_H}\sum_{r=1}^{N_W} h_j^k W_r^k v_{j+r-1} - \sum_{k=1}^{K} b_k \sum_{j=1}^{N_H} h_j^k - c\sum_{i=1}^{N_V} v_i. \quad (6.2)$$

The joint and conditional probability distributions are defined as follows:

$$P(\mathbf{v},\mathbf{h}) = \frac{1}{Z}\exp(-E(\mathbf{v},\mathbf{h})) \quad (6.3)$$

$$P(h_j^k = 1|\mathbf{v}) = sigmoid((\tilde{W}^k *_v v)_j + b_k) \quad (6.4)$$

$$P(v_i = 1|\mathbf{h}) = sigmoid(\sum_k (W^k *_f h^k)_i + c) \quad \text{(for binary visible units)} \quad (6.5)$$

$$P(v_i|\mathbf{h}) = Normal(\sum_k (W^k *_f h^k)_i + c, 1) \quad \text{(for real visible units)}, \quad (6.6)$$

where $*_v$ is a "valid" convolution, $*_f$ is a "full" convolution,[1] and $\tilde{W}_j^k \triangleq W_{N_W-j+1}^k$. Since all units in one layer are conditionally independent given the other layer, inference in the network can be efficiently performed using block Gibbs sampling. Further, as described in the previous chapter, we use CRBMs with probabilistic max-pooling as building blocks for convolutional deep belief networks.

To train the convolutional RBMs, we use contrastive divergence to compute the approximate gradient. Since a typical CRBM is highly overcomplete, a sparsity penalty term is added to the log-likelihood objective, as in the previous chapters. Specifically, the training objective can be written as:

$$\text{minimize}_{W,b,c} \qquad \mathcal{L}_{likelihood}(W,b,c) + \mathcal{L}_{sparsity}(W,b,c), \quad (6.7)$$

where $\mathcal{L}_{likelihood}$ is a negative log-likelihood that measures how well the CRBM approximates the input data distribution, and $\mathcal{L}_{sparsity}$ is a penalty term that constrains the hidden units to having sparse average activations. This sparsity regularization can be viewed as limiting the "capacity" of the network, and it often results in more easily interpretable feature representations. Once the parameters for all the layers are trained, we stack the

[1] Given an m-dimensional vector and an n-dimensional kernel (where $m > n$), valid convolution gives a $(m - n + 1)$-dimensional vector, and full convolution gives a $(m + n - 1)$-dimensional vector.

CRBMs to form a convolutional deep belief network. For inference, we use feed-forward approximation.

6.2.2 Application to audio data

To apply CDBNs to audio data, we first convert time-domain signals into spectrograms. Since the dimensionality of the spectrograms is large (e.g., 160 channels), we apply PCA-whitening to the spectrograms and create lower-dimensional representations. Specifically, given a matrix $X \in \mathbb{R}^{d \times m}$, where d is the original dimensionality of the single-frame spectrograms, and m is the number of frames. Here, X is constructed by concatenating the spectrograms over randomly selected training examples. Then, we compute the eigenvalue decomposition of $X^\top X = EDE^\top$, where E is an orthogonal matrix, and D is a (non-negative) diagonal matrix. We define submatrices $\hat{E}$ and $\hat{D}$ by taking columns and digonal entries that correspond to the n_c largest eigenvalues. Based on this eigen-decomposition, we compute the whitened components for each spectrogram frame $x \in \mathbb{R}^d$ as follows:

$$x_{whitened} = (\hat{D} + \epsilon I)^{-1/2} \hat{E}^\top x, \tag{6.8}$$

where ϵ is a small constant to prevent over-amplification of the PCA components with small eigenvalues. (We fixed $\epsilon = 3$ in all of our experiments.) After whitening, we use the whitened spectrogram as input for the CDBN. Thus, the data we feed into the CDBN consists of n_c channels of one-dimensional vectors of length N_V, where n_c is the number of PCA components in our representation. Similarly, the first-layer bases are comprised of n_c channels of one-dimensional filters of length N_W.

6.3 Unsupervised feature learning

6.3.1 Training on unlabeled TIMIT data

We trained the first and second-layer CDBN representations using a large, unlabeled speech dataset. First, we extracted the spectrogram from each utterance of the TIMIT training data [51]. The spectrogram had a 20 ms window size with 10 ms overlaps. The spectrogram was further processed using PCA-whitening (with 80 components) to reduce the dimensionality. Then, we trained 300 first-layer bases with a filter length (n_W) of 6 and a max-pooling ratio (local neighborhood size) of 3. We further trained 300 second-layer bases using the max-pooled first-layer activations as input, again with a filter length of 6 and a max-pooling ratio of 3. We used target sparsity 0.05 in the first layer and 0.02 in the second layer.

6.3.2 Visualization

In this section, we illustrate what the network "learns" through visualization. We visualize the first-layer bases by multiplying the inverse of the PCA-whitening on each first-layer basis (Figure 6.1). Each second-layer basis is visualized as a weighted linear combination of the first-layer bases.

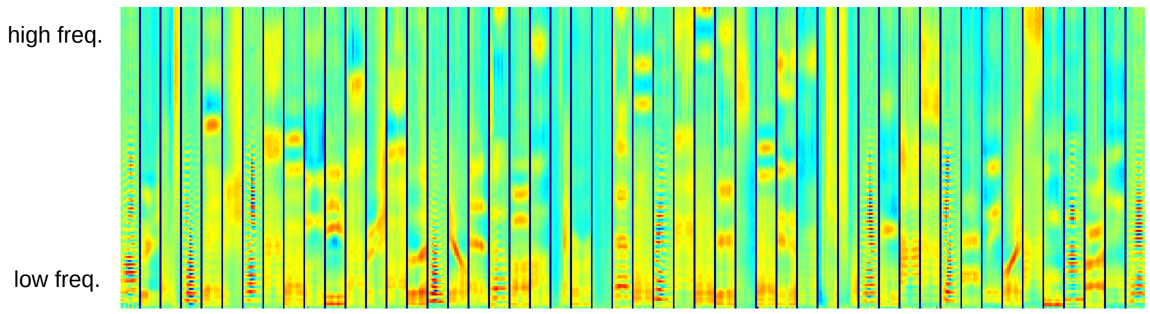


Figure 6.1: Visualization of randomly selected first-layer CDBN bases trained using the TIMIT data. Each column represents a "temporal receptive field" of a first-layer basis in the spectrogram space. The frequency channels are ordered from the lowest frequency (bottom) to the highest frequency (top). All figures in this chapter are best viewed in color.

Phonemes and the CDBN features

In Figure 6.2, we show how our bases relate to phonemes by comparing visualizations of each phoneme with the bases that are most activated by that phoneme.

For each phoneme, we show five spectrograms of sound clips of that phoneme (top five columns in each phoneme group), and the five first-layer bases with the highest average

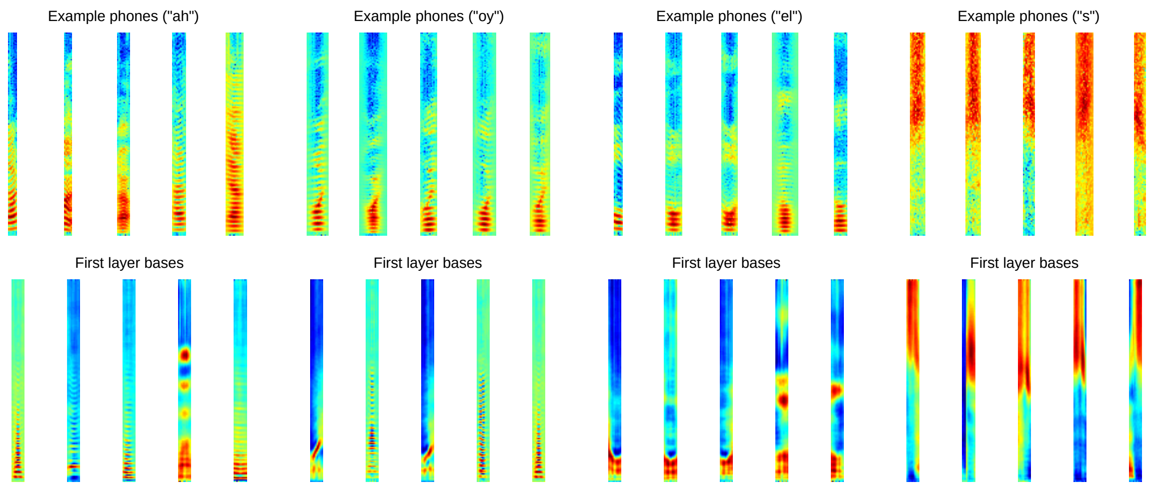


Figure 6.2: Visualization of the four different phonemes and their corresponding first-layer CDBN bases. For each phoneme: (top) the spectrograms of the five randomly selected phones; (bottom) five first-layer bases with the highest average activations on the given phoneme.

activations on the given phoneme (bottom five columns in each phoneme group). Many first-layer bases closely match the shapes of phonemes. There are prominent horizontal bands in the lower frequencies of the first-layer bases that respond most to vowels (e.g., "ah" and "oy"). The bases that respond most to fricatives (e.g., "s") typically take the form of widely distributed areas of energy in the high frequencies of the spectrogram. Both of these patterns reflect the structure of the corresponding phoneme spectrograms.

A closer inspection of the bases provides slight evidence that the first-layer bases also capture more fine-grained details. For example, the first and third "oy" bases reflect the upward-slanting pattern in the phoneme spectrograms. The top "el" bases mirror the intensity patterns of the corresponding phoneme spectrograms: a high intensity region appears in the lowest frequencies, and another region of less intensity appears a bit higher up.

Speaker gender information and the CDBN features

In Figure 6.3, we show an analysis of two-layer CDBN feature representations with respect to the gender classification task (Section 6.4.2). Note that the network was trained on unlabeled data; therefore, no information about the speaker's gender was given during training.

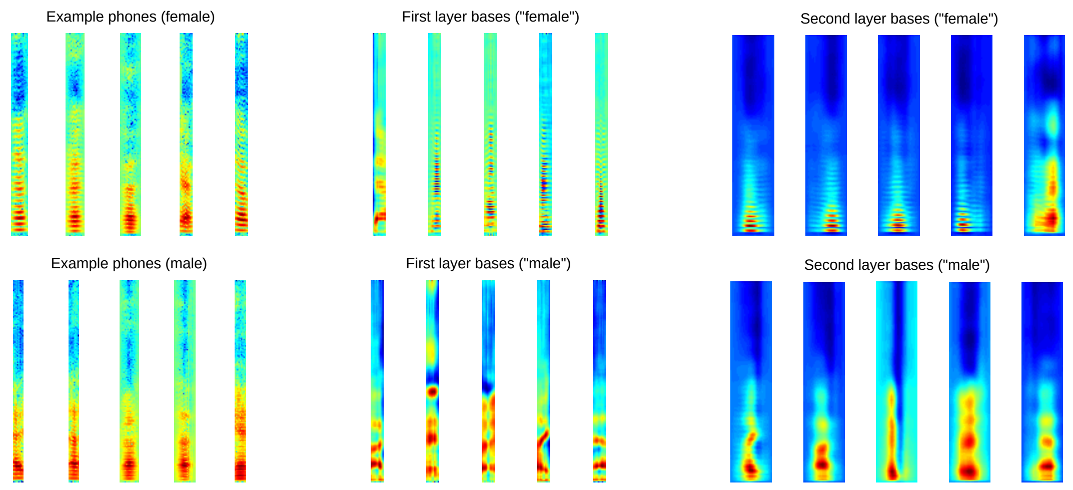


Figure 6.3: (Left) five spectrogram samples of “ae” phoneme from female (top)/male (bottom) speakers. (Middle) Visualization of the five first-layer bases that most differentially activate for female/male speakers. (Right) Visualization of the five second-layer bases that most differentially activate for female/male speakers.

To compare with the CDBN features, we show randomly selected spectrograms of female (top left five columns) and male (bottom left five columns) pronunciations of the “ae” phoneme from the TIMIT dataset. Spectrograms for the female pronunciations are qualitatively distinguishable by a finer horizontal banding pattern in low frequencies, whereas male pronunciations have more blurred patterns. This gender difference in the vowel pronunciation patterns is typical across the TIMIT data.

We also show the bases that are most biased to activate on either male or female speech. The bases that are most active on female speech encode the horizontal band pattern that is prominent in the spectrograms of female pronunciations. On the other hand, the male-biased bases have more blurred patterns, which again visually matches the corresponding spectrograms.

6.4 Application to speech recognition tasks

In this section, we demonstrate that the CDBN feature representations learned from the unlabeled speech corpus can be useful for multiple speech recognition tasks, such as speaker identification, gender classification, and phone classification. In most of our experiments, we followed the self-taught learning framework [131], since one of our main interests was to evaluate the different feature representations given a small number of labeled training examples (as often assumed in self-taught learning or semi-supervised learning settings). More specifically, we trained the CDBN on unlabeled TIMIT data (as described in Section 6.3.1); then we used the CDBN features for classification on labeled training/test data[2] that were randomly selected from the TIMIT corpus. However, in the case of phone classification, we followed the standard protocol (e.g., [37]) instead of the self-taught learning framework to evaluate our algorithm against other methods.

6.4.1 Speaker identification

We evaluated the usefulness of the learned CDBN representations for the speaker identification task. The subset of the TIMIT corpus that we used for speaker identification has 168 speakers and 10 utterances (sentences) per speaker, resulting in a total of 1680 utterances. We performed 168-way classification on this set. For each number of utterances per speaker, we randomly selected training utterances and testing utterances and measured the classification accuracy; we report the results averaged over 10 random trials. That said, there were some exceptions; for the case of eight training utterances, we followed the same procedure as Reynolds (1995) [138]. More specifically, we used eight training utterances (2 sa sentences, 3 si sentences and first 3 sx sentences); the two testing utterances were the remaining 2 sx sentences. We used cross validation to select hyperparameters for classification, except in the case of 1 utterance per speaker, where we used a randomly selected validation sentence per speaker. To construct training and test data for the classification task, we extracted a spectrogram from each utterance in the TIMIT corpus. We

[2]There are two disjoint TIMIT data sets (one with 462 speakers and the other with 168 speakers). We drew unlabeled data from the larger of the two for unsupervised feature learning, and we drew labeled data from the other data set to create our training and test set for the classification tasks.

denote this spectrogram representation as the “RAW” features. We computed the first and second-layer CDBN features using the spectrogram as input. We also computed MFCC features, widely-used standard features for generic speech recognition tasks. As a result, we obtained spectrogram/MFCC/CDBN representations for each utterance with multiple (typically, several hundred) frames. In our experiments, we used simple summary statistics for each channel (such as the average, max, or standard deviation) over all the frames. We evaluated the features using standard supervised classifiers, such as SVM, GDA, and KNN. The choices of summary statistics and hyperparameters for the classifiers were done using cross-validation. We report the average classification accuracy (over 10 random trials) with a varying number of training examples.

Table 6.1 shows the average classification accuracy for each feature representation. Throughout this chapter, we use “CDBN L.1” to denote CDBN layer 1 features, “CDBN L.2” to denote CDBN layer 2 features, and “CDBN L.1+L.2” to denote the concatenation of layer 1 and layer 2 features. The results show that the first and second CDBN representations both outperform baseline features (RAW and MFCC). The numbers compare MFCC and CDBN features with as many of the same factors (such as preprocessing and classification algorithms) as possible. Furthermore, to make a fair comparison between CDBN features and MFCC, we used the best performing implementation[3] among several standard implementations for MFCC. Our results suggest that without special preprocessing or postprocessing (besides the summary statistics needed to reduce the number of features), the CDBN features outperform MFCC features, especially in a setting with a very limited number of labeled examples.

We further experimented to determine if the CDBN features can achieve competitive performance compared to more sophisticated, state-of-the-art methods. For each feature representation, we used the classifier that achieved the highest performance. For example, we evaluated four possible combinations, in other words, either MFCC or CDBN for features and either Gaussian mixture or SVM for the classifier. As a result, MFCC performed the best with Gaussian mixtures, and CDBN features performed the best with SVM. We report the best results for each feature type and describe the corresponding algorithm (for

[3]We used Dan Ellis’ implementation available at: `http://labrosa.ee.columbia.edu/matlab/rastamat`.

#training utterances per speaker	RAW	MFCC	CDBN L.1	CDBN L.2	CDBN L.1+L.2
1	46.7%	54.4%	**74.5%**	62.8%	72.8%
2	43.5%	69.9%	**76.7%**	66.2%	**76.7%**
3	67.9%	76.5%	91.3%	84.3%	**91.8%**
5	80.6%	82.6%	93.7%	89.6%	**93.8%**
8	90.4%	92.0%	**97.9%**	95.2%	97.0%

Table 6.1: Test classification accuracy for speaker identification using summary statistics

each feature type) in detail.

For the MFCC features, we replicated Reynolds (1995)'s method [138]. Specifically, MFCC features (with multiple frames) were computed for each utterance; then a Gaussian mixture model was trained for each speaker (treating each individual MFCC frame as an input example to the GMM. For a given test utterance, a prediction was made by determining the GMM model that had the highest test log-likelihood. For the CDBN features, we used a SVM-based ensemble method, treating each single-frame CDBN features as an individual example. Specifically, we first trained a multi-class linear SVM for these individual frames. For testing, we computed the SVM prediction score (probability output) for each speaker, and then aggregated (i.e., summed over) prediction ouputs from all the frames. Overall, the highest scoring speaker was selected for the final prediction.

As shown in Table 6.2, the CDBN features consistently outperformed MFCC features when the number of training examples was small. We also combined both methods by taking a linear combination of the two classifier outputs (scores for individual speakers before taking the final classification prediction from each algorithm). We fixed the constant for the linear combination across all the numbers of training utterances, and it was selected using cross validation. The resulting combined classifier performed the best, achieving 100% accuracy in the case of 8 training utterances per speaker.

6.4.2 Speaker gender classification

We also evaluated the same CDBN features which were learned using the unlabeled TIMIT data on the gender classification task. We report the classification accuracy for various

#training utterances per speaker	MFCC ([138]'s method)	CDBN	MFCC ([138]) + CDBN
1	40.2%	90.0%	**90.7%**
2	87.9%	97.9%	**98.7%**
3	95.9%	98.7%	**99.2%**
5	99.2%	99.2%	**99.6%**
8	99.7%	99.7%	**100.0%**

Table 6.2: Test classification accuracy for speaker identification using all frames

#training utterances per gender	RAW	MFCC	CDBN L.1	CDBN L.2	CDBN L.1+L.2
1	68.4%	58.5%	78.5%	**85.8%**	83.6%
2	76.7%	78.7%	86.0%	**92.5%**	92.3%
3	79.5%	84.1%	88.9%	**94.2%**	**94.2%**
5	84.4%	86.9%	93.1%	**95.8%**	95.6%
7	89.2%	89.0%	94.2%	**96.6%**	96.5%
10	91.3%	89.8%	94.7%	**96.7%**	96.6%

Table 6.3: Test accuracy for gender classification problem

quantities of training examples (utterances) per gender. For each number of training examples, we randomly sampled training examples and 200 testing examples; we report the test classification accuracy averaged over 20 trials. As in the speaker identification task, we used simple summary statistics (such as the average, max, or standard deviation) to aggregate features over all the frames, and we evaluated the features using standard supervised classifiers. As shown in Table 6.3, both the first and second CDBN features outperformed the baseline features, especially when the number of training examples was small. In addition, the second-layer CDBN features consistently performed better than the first-layer CDBN features. This suggests that the second-layer representation learned more invariant features that are relevant for speaker gender classification, which justifies the use of "deep" architectures.

6.4.3 Phone classification

Finally, we evaluated our learned representation on phone classification tasks. In this experiment, we treated each phone segment as an individual example. For MFCC features,

#training utterances	RAW	MFCC ([37]'s method)	CDBN L.1	MFCC+CDBN L.1 ([37])
100	36.9%	66.6%	53.7%	**67.2%**
200	37.8%	70.3%	56.7%	**71.0%**
500	38.7%	74.1%	59.7%	**75.1%**
1000	39.0%	76.3%	61.6%	**77.1%**
2000	39.2%	78.4%	63.1%	**79.2%**
3696	39.4%	79.6%	64.4%	**80.3%**

Table 6.4: Test accuracy for phone classification problem

we replicated Clarkson and Moreno's method [37]. Specifically, we divided each phone segment into sub-regions and then aggregated (by averaging) within these regions to compute features for an SVM with RBM kernels. For the spectrogram and CDBN features, we simply averaged over the frames (corresponding to a given phone segment) to compute features for an SVM. Then, following the standard protocol (as described in [37]), we report the 39 way phone classification accuracy on the test data (TIMIT core test set) for various numbers of training sentences. For each number of training examples, we report the average classification accuracy over 5 random trials. The summary results are shown in Table 6.4. Here, "MFCC+CDBN L.1 ([37])" denotes Clarkson and Moreno [37]'s SVM-based method using the concatenation of MFCC and the CDBN features as input. In this experiment, the first-layer CDBN features performed better than spectrogram features, but they did not outperform the MFCC features. However, by combining MFCC features and CDBN features, we consistently achieved about 0.7% accuracy improvement over all the numbers of training utterances. In the realm of phone classification, where significant research effort is often needed to achieve even improvements well under one percent, this is a significant improvement [147, 151, 129, 171].

This suggests that the first-layer CDBN features learned informative features for phone classification tasks in an unsupervised way. In contrast to the gender classification task, the second-layer CDBN features did not offer improvement over the first-layer CDBN features. This result is not unexpected considering that the time-scale of most phonemes roughly corresponds to the time-scale of the first-layer CDBN features.

Train examples	RAW	MFCC	CDBN L.1	CDBN L.2	CDBN L.1+L.2
1	51.6%	54.0%	**66.1%**	62.5%	64.3%
2	57.0%	62.1%	**69.7%**	67.9%	69.5%
3	59.7%	65.3%	**70.0%**	66.7%	69.5%
5	65.8%	68.3%	**73.1%**	69.2%	72.7%

Table 6.5: Test accuracy for 5-way music genre classification

6.5 Application to music classification tasks

In this section, we assess the applicability of CDBN features to various music classification tasks.

6.5.1 Music genre classification

For the task of music genre classification, we trained the first and second-layer CDBN representations on an unlabeled collection of music data.[4] First, we computed the spectrogram (20 ms window size with 10 ms overlaps) representation for individual songs. The spectrogram was PCA-whitened (with 80 components) and then fed into the CDBN as input data. (See Section 6.2.2 for details of whitening.) We trained 300 first-layer bases with a filter length of 10 and a max-pooling ratio of 3. In addition, we trained 300 second-layer bases with a filter length of 10 and a max-pooling ratio of 3. As in the speech data, we used target sparse 0.05 in the first layer and 0.02 in the second layer.

We evaluated the learned CDBN representation for the 5-way genre classification tasks. The training and test songs for the classification tasks were randomly sampled from 5 genres (classical, electric, jazz, pop, and rock) and did not overlap with the unlabeled data. We randomly sampled 3-second segments from each song and treated each segment as an individual training or testing example. We report the classification accuracy for various numbers of training examples. For each number of training examples, we averaged over 20 random trials. The results are shown in Table 6.5. In this task, the first-layer CDBN features showed the best overall performance.

[4]Available from `http://ismir2004.ismir.net/ISMIR_Contest.html`.

6.5.2 Music artist classification

Furthermore, we evaluated whether the CDBN features are useful in identifying individual artists. In our experiments, we found that artist identification task was more difficult than the speaker identification task because the local sound patterns can be highly variable even for the same artist. Following the same procedure as in Section 6.5.1, we trained the first and second-layer CDBN representations from an unlabeled collection of classical music data. Some representative bases are shown in Figure 6.4. Then we evaluated the learned CDBN representation for 4-way artist identification tasks. (The four artists were Edward Martin, Vito Paternoster, Jacob Heringman, and Trevor Pinnock.) The disjoint sets of training and test songs for the classification tasks were randomly sampled from the songs of four artists. The unlabeled data and the labeled data did not include the same artists. We randomly sampled 3-second segments from each song and treated each segment as an individual example. We report the classification accuracy for various quantities of training examples. For each number of training examples, we averaged over 20 random trials. The results are shown in Table 6.6. The results show that (a) both the first and second-layer CDBN features performed better than the baseline features, and (b) either using the second-layer features only or combining the first and the second-layer features together yielded the best results. This suggests that the second-layer CDBN representation might have captured somewhat useful, higher-level features than the first-layer CDBN representation.

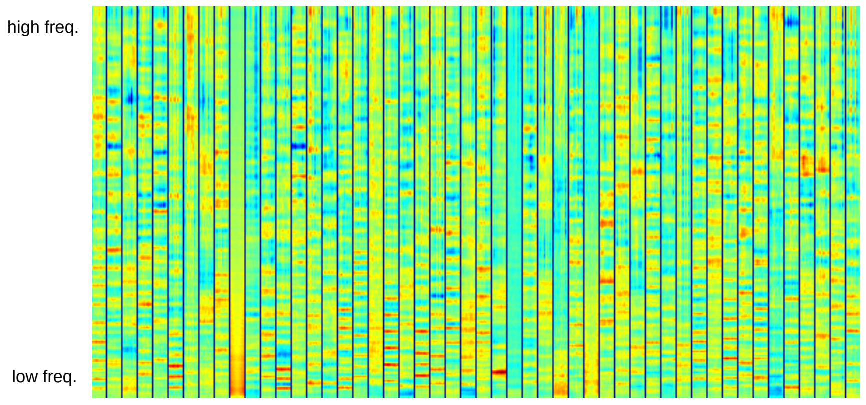


Figure 6.4: Visualization of randomly selected first-layer CDBN bases trained on classical music data.

Train examples	RAW	MFCC	CDBN L.1	CDBN L.2	CDBN L.1+L.2
1	56.0%	63.7%	67.6%	67.7%	**69.2%**
2	69.4%	66.1%	76.1%	74.2%	**76.3%**
3	73.9%	67.9%	78.0%	75.8%	**78.7%**
5	79.4%	71.6%	80.9%	**81.9%**	81.4%

Table 6.6: Test accuracy for 4-way artist identification

6.6 Discussion and summary

Modern speech datasets are much larger than the TIMIT dataset. While the challenge of larger datasets often lies in considering harder tasks, our objective in using the TIMIT data was to restrict the amount of labeled data from which our algorithm had to learn. It remains an interesting problem to apply deep learning to larger datasets and more challenging tasks.

In this chapter, we applied convolutional deep belief networks to audio data and evaluated on various audio classification tasks. By leveraging a large amount of unlabeled data, our learned features often equaled or surpassed MFCC features, which are hand-tailored to audio data. Even when our features did not outperform MFCC, we achieved higher classification accuracy by combining both. Also, our results showed that a single CDBN feature representation can achieve high performance on multiple audio recognition tasks. We believe that our approach hold promise in automatically learning deep feature hierarchies for audio data.

Chapter 7

Conclusion

7.1 Summary of contributions

Efficient sparse coding algorithms

We formulated sparse coding problem as a well-defined objective, proposed efficient algorithms, and showed how it can be applied to machine learning tasks. Specifically, we presented efficient sparse coding algorithms that are based on iteratively solving two convex optimization problems: an L_1-regularized least squares problem and an L_2-constrained least squares problem. We then proposed novel algorithms to solve both of these optimization problems. Our algorithms result in a significant speedup for sparse coding, allowing us to learn larger sparse codes than possible with previously described algorithms. We applied these algorithms to natural images and demonstrate that the inferred sparse codes exhibit end-stopping and non-classical receptive field surround suppression and, therefore, may provide a partial explanation for these two phenomena in V1 neurons. Further, in related work on self-taught learning, we showed that sparse coding can learn succinct, higher-level feature representations (from unlabeled data) that can improve supervised classification tasks.

Extension of sparse coding to a broad range of input data

In addition, we developed generalized sparse coding algorithms that extend the range of data types that sparse coding can be applied. Specifically, drawing on ideas from generalized linear models (GLMs), we presented a generalization of sparse coding to learning with data drawn from any exponential family distribution (such as Bernoulli, Poisson, etc.). This gives a method that is arguably much better suited to model other data types than Gaussian. We presented an algorithm for solving the L_1-regularized optimization problem defined by this model, and showed that it is highly efficient when the optimal solution is sparse. We also showed that the new model results in significantly improved self-taught learning performance when applied to text classification and to robotic perception.

Sparsity regularization for deep belief networks

We developed a sparse variant of the deep belief network model. This is the first work that incorporated sparsity into deep belief networks. Specifically, we developed a form of regularization that encourages sparsity in RBMs and DBNs. This regularization produces similar sparse feature representations that can be obtained from sparse coding, but with an additional advantage that posterior inference is very fast. When trained on natural images, this model learns local, oriented edge-filters in the first layer. More interestingly, the second layer captures a variety of both colinear (or contour) features as well as corners and junctions that appeared to give responses that were similar to biological V2 neurons along several dimensions. This result suggests that sparse deep belief networks can capture interesting high-order features from natural image statistics, and we believe that the proposed method holds promise for modeling higher-order features from various data types. Furthermore, the sparsity regularization can be applied to not only to RBMs and DBNs, but also to autoencoders and neural networks, often improving discriminative power of the learned features in many tasks.

Scaling up for deep belief networks

We developed convolutional deep belief networks, an algorithm that scales up deep belief networks. Our method is the first algorithm that scales up the unsupervised deep learning algorithms to enable learning directly from realistic-sized images (e.g., 200x200 pixels). This model is translation-invariant and supports full probabilistic inference. We also presented probabilistic max-pooling, a novel technique which shrinks the representations of higher layers in a probabilistically sound way. Our experiments show that the algorithm learns useful high-level visual features, such as object parts, from unlabeled images of objects and natural scenes. We demonstrate excellent performance on several visual recognition tasks, such as object recognition and multi-class segmentation. The experimental results also show that our model can perform approximate posterior inference over realistic-sized images.

We applied convolutional deep belief networks for audio recognition and demonstrated excellent performance in speech and music classification tasks. Interestingly, in the case of speech data, we found that the learned features corresponded to phonemes. The learned feature representations showed comparable or superior performance to the MFCC features for multiple speech and music classification tasks. Specifically, the learned features lead to state-of-the-art performance in speaker identification. We also showed that the learned features can be combined with MFCC features to improve upon a state-of-the-art phone classification result.

7.2 Future work and open questions

Learning more invariant feature representations

In this thesis, we showed that incorporating invariance (e.g. translation invariance in convolutional DBNs) allows to learn useful high-level features and achieve good performance in multiple recognition tasks. In fact, our related work [55] show that convolutional DBN shows modest invariance to other types of transformations as well (such as 2D-rotations and 3D-rotations).

It is highly desirable for a good feature representation to be invariant. Invariance for a feature mapping means that: given optimal input for that feature mapping, the feature values are close to the maximal value when the input data is slightly transformed from the optimal input (i.e., the feature values are high in the "vicinity" of optimal inputs), and yet at the same time the feature mapping is also selective to the optimal input (i.e., the feature values are not always high). (See [55] for a more quantitative definition.) For example, let's consider a hypothetical binary classification problem, and we assume that the positive examples are similar to each other (i.e., they are "connected" through some transformations). If we have a perfectly invariant feature that takes the value of 1 for all positive examples and 0 for all negative examples, then we would have solved this classification task by using this single feature! Another example will be visual object recognition, where we want the feature representation selective to class labels, yet be invariant to other confounding factors, such as illumination, viewpoints, translation, scale, and rotation. These examples suggest that making the feature representation invariant will enable machine algorithms to learn from a smaller amount of training data and to achieve good generalization performance.

One question is whether we want to model these invariances explicitly or not. One obvious way of achieving more invariance is to explicitly encode these invariances. For example, in the case of images, we may want to make the feature representation to be invariant to scale and rotation. However, designing feature learning algorithms that explicitly achieve such invariances is not trivial and requires domain-expert knowledge. Further, the invariances in one domain will not be relevant to other domains.

Therefore, an ideal approach will be to learn invariance from data. One promising way is learning invariance from a large amount of unlabeled data, by making the learned feature representations invariant to various transformations (such as from video or similar input pairs). Several promising directions are exploiting temporal coherence or using topographic representations. However, there are many challenges in order for these methods to be applicable to real-world problems (e.g., object detection and scene understanding) because real-world data (e.g., images available from the internet) is very complex and highly-variable. Therefore, achieving more invariant feature representations will be one of the fundamental questions that we need to continuously address.

Multi-modal learning

Another challenges is developing a unified, multi-modal learning framework that uses multiple input domains (e.g., visual inputs and audio inputs together) for learning and inference. For example, when it is ambiguous to recognize phonemes in the speech data, it can be helpful to recognize lip movement and infer about the corresponding phonemes. As another example, when it is difficult to identify a person's face by just using images, it can be useful to also examine the person's speech data. In addition to advancing computer vision and audio scene understanding, this multi-modal learning approach has particular relevance to understanding robotic sensor inputs that do not correspond to the natural human senses, such as thermo-infrared imaging, hyperspectral imaging, sonar, and radar. For these types of input, human intuition is particularly weak, which makes hand-engineering features difficult. It will be much easier to apply a unified learning framework rather than manually engineering features for each sensor type. This multi-modal learning framework will also make it easier to use unlabeled data because the learning algorithm can more robustly infer about the underlying labels.

Large-scale learning

Suppose that we want to learn high-level features from millions of unlabeled images, together with a small number of labeled images. How can we discover hierarchical representations for all object categories including shared representations? This is probably one of the most challenging tasks that can be envisioned in the context of feature learning. Unfortunately, we do not have yet a solution to this problem. For example, there are several limitations of the current convolutional DBN model: (1) it becomes computationally expensive to learn more than several hundred filters in each layer; (2) the algorithm lacks other forms of invariance (such as rotational and scale invariance), and thereby the algorithm works less well for deformable shaped objects. Possible ways of addressing these issues will be (1) using massively parallel computation (such as GPUs) for convolutional DBNs to speed up the learning algorithms (e.g., [132]), (2) learning more invariance from data, and (3) exploiting labeled data (together with unlabeled data) by incorporating the semi-supervised learning framework for convolutional DBNs.

7.3 Closing remarks

This thesis has explored the problems of learning features from unlabeled data. As one solution for this problem, I explored algorithms that exploit sparsity, hierarchy, and convolutional structure, while demonstrating good performance in many AI problems. Given that the quality of features significantly affects the performance of machine learning systems, feature engineering is one of the most fundamental issues in machine learning. I believe that algorithms that can automatically learn good feature representations from unlabeled data will potentially revolutionize machine learning and AI. Such general purpose algorithms will allow machine learning systems to be much more easily applied to problems in vision, audio understanding, text understanding, robotic sensor understanding, and other problems, and to achieve superior performance without the manual feature engineering while using significantly less labeled data.

Bibliography

[1] A. Ahmed, K. Yu, W. Xu, Y. Gong, and E. Xing. Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks. In *Proceedings of the European Conference on Computer Vision*, 2008.

[2] T. Alexandrov, O. Keszöcze, D. A. Lorenz, S. Schiffler, and K. Steinhorst. An active set approach to the elastic-net and its applications in mass spectrometry. In *Proceedings of the Workshop on Signal Processing with Adaptive Sparse Structured Representations*, 2009.

[3] T. Alexandrov, K. Steinhorst, O. Keszoecze, and S. Schiffler. Sparsecodepicking: feature extraction in mass spectrometry using sparse coding algorithms. arXiv:0907.3426, 2009.

[4] R. K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, 2005.

[5] G. Andrew and J. Gao. Scalable training of L_1-regularized log-linear models. In *Proceedings of the International Conference on Machine Learning*, 2007.

[6] A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. *Advances in Neural Information Processing Systems*, 19:41, 2007.

[7] F. Bach. Exploring large feature spaces with hierarchical multiple kernel learning. *CoRR*, abs/0809.1493, 2008.

[8] F. Bach, G. R. G. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the International Conference on Machine Learning*, 2004.

[9] F. Bach, J. Mairal, and J. Ponce. Convex sparse matrix factorizations. *CoRR*, 2008.

[10] M. Banko and E. Brill. Mitigating the paucity-of-data problem: Exploring the effect of training corpus size on classifier performance for natural language processing. In *Proceedings of the International Conference on Human Language Technology Research*, 2001.

[11] M. Banko and E. Brill. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*, 2001.

[12] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[13] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[14] A. J. Bell and T. J. Sejnowski. The 'independent components' of natural scenes are edge filters. *Vision Research*, 37(23):3327–3338, 1997.

[15] S. Belongie, J. Malik, and J. Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *Advances in Neural Information Processing Systems*, pages 831–837, 2001.

[16] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, 2007.

[17] Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. *Large-Scale Kernel Machines*, 2007.

[18] A. C. Berg, T. L. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondence. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[19] M. Berry, M. Browne, A. Langville, V. Pauca, and R. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics & Data Analysis*, 52(1):155–173, 2007.

[20] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1995.

[21] D. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[22] D. M. Blei and J. D. Lafferty. A correlated topic model of science. *Annals*, 1(1):17–35, 2007.

[23] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the International Conference on Machine Learning*, 2001.

[24] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Annual Conference on Computational Learning Theory*, 1998.

[25] E. Borenstein and S. Ullman. Class-specific, top-down segmentation. In *Proceedings of the European Conference on Computer Vision*, 2002.

[26] B. E. Boser, I. M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Annual Conference on Computational Learning Theory*, pages 144–152, 1992.

[27] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[28] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. Large language models in machine translation. In *EMNLP-CoNLL*. ACL, 2007.

[29] C. Cadieu and B. Olshausen. Learning transformational invariants from natural movies. In *Advances in Neural Information Processing Systems 21*, pages 209–216. 2009.

[30] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[31] J. R. Cavanaugh, W. Bair, and J. A. Movshon. Nature and interaction of signals from the receptive field center and surround in macaque V1 neurons. *Journal of Neurophysiology*, 88(5):2530–2546, 2002.

[32] Y. Censor and S. A. Zenios. *Parallel Optimization: Theory, Algorithms and Applications*. Oxford University Press, 1997.

[33] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-supervised learning*. MIT Press, 2006.

[34] C. H. Chen. *Signal processing handbook*. New York: Dekker, 1988.

[35] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1998.

[36] A. Cheriany, J. Andersh, V. Morellas, N. Papanikolopoulos, and B. Mettler. Autonomous altitude estimation of a UAV using a single onboard camera. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[37] P. Clarkson and P. J. Moreno. On the use of support vector machines for phonetic classification. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 585–588, 1999.

[38] M. Collins, S. Dasgupta, and R. E. Schapire. A generalization of principal component analysis to the exponential family. In *Advances in Neural Information Processing Systems*, 2001.

[39] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[40] P. Comon. Independent component analysis, a new concept? *Signal processing*, 36(3):287–314, 1994.

[41] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[42] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[43] T. Dean, G. Corrado, and R. Washington. Recursive sparse, spatiotemporal coding. In *Proceedings of the IEEE International Symposium on Multimedia*, 2009.

[44] J. R. Deller, Jr., J. G. Proakis, and J. H. Hansen. *Discrete Time Processing of Speech Signals*. Macmillan New York, 1993.

[45] G. Desjardins and Y. Bengio. Empirical evaluation of convolutional RBMs for vision. Technical report, University of Montreal, 2008.

[46] S. Dixon. Onset detection revisited. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-06)*, pages 133–137, 2006.

[47] C. Do and A. Y. Ng. Transfer learning for text classification. In *Advances in Neural Information Processing Systems 18*, 2006.

[48] Q. Du, V. Faber, and M. Gunzburger. Centroidal Voronoi tessellations: applications and algorithms. *SIAM review*, 41(4):637–676, 1999.

[49] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32(2):407–499, 2004.

[50] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. In *CVPR Workshop on Generative Model Based Vision*, 2004.

[51] W. Fisher, G. Doddington, and K. Goudie-Marshall. The DARPA speech recognition research database: Specifications and status. In *DARPA Speech Recognition Workshop*, 1986.

[52] Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. Technical report, University of California at Santa Cruz, 1994.

[53] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 2, 2007.

[54] P. Gehler and S. Nowozin. On feature combination for multiclass object classification. In *Proceedings of the International Conference on Computer Vision*, 2009.

[55] I. Goodfellow, Q. Le, A. Saxe, and A. Y. Ng. Measuring invariances in deep networks. In *Advances in Neural Information Processing Systems 22*, pages 646–654. 2009.

[56] J. Goodman. Exponential priors for maximum entropy models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2004.

[57] S. Gould, R. Fulton, and D. Koller. Decomposing a scene into geometric and semantically consistent regions. In *Proceedings of the International Conference on Computer Vision*, 2009.

[58] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008.

[59] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. `http://cvxr.com/cvx`, May 2010.

[60] P. J. Green. Iteratively reweighted least squares for maximum likelihood estimation, and some robust and resistant alternatives. *Journal of the Royal Statistical Society, Series B, Methodological*, 46:149–192, 1984.

[61] R. Grosse, R. Raina, H. Kwong, and A. Y. Ng. Shift-invariant sparse coding for audio classification. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2007.

[62] Z. S. Harris. *Mathematical Structures of Language*. Interscience, 1968.

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

[64] G. E. Hinton, S. Osindero, and K. Bao. Learning causally linked MRFs. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2005.

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

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

[67] A. Hyvarinen, M. Gutmann, and P. O. Hoyer. Statistical model of natural stimuli predicts edge-like pooling of spatial frequency channels in V2. *BMC Neuroscience*, 6:12, 2005.

[68] A. Hyvärinen, P. O. Hoyer, and M. O. Inki. Topographic independent component analysis. *Neural Computation*, 13(7):1527–1558, 2001.

[69] A. Hyvarinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. John Wiley & Sons, 2001.

[70] M. Ito and H. Komatsu. Representation of angles embedded within contour stimuli in area V2 of macaque monkeys. *The Journal of Neuroscience*, 24(13):3313–3324, 2004.

[71] T. Jebara. Multi-task feature and kernel selection for SVMs. In *Proceedings of the International Conference on Machine Learning*, 2004.

[72] R. Jenatton, J.-Y. Audibert, and F. Bach. Structured variable selection with sparsity-inducing norms. Technical report, arXiv:0904.3523, 2009.

[73] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the International Conference on Machine Learning*, 1999.

[74] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):433–449, 1999.

[75] Y. Karklin and M. S. Lewicki. Learning higher-order structures in natural images. *Network: Computation in Neural Systems*, 14:483–499, 2003.

[76] Y. Karklin and M. S. Lewicki. A hierarchical Bayesian model for learning non-linear statistical regularities in non-stationary natural signals. *Neural Computation*, 17(2):397–423, 2005.

[77] K. Kavukcuoglu, M. Ranzato, and Y. LeCun. Fast inference in sparse coding algorithms with applications to object recognition. Technical Report CBLL-TR-2008-12-01, Computational and Biological Learning Lab, Courant Institute, NYU, 2008.

[78] K. Koh, S.-J. Kim, and S. Boyd. An interior-point method for large-scale L_1-regularized logistic regression. *Journal of Machine Learning Research*, 8:1519–1555, 2007.

[79] G. R. G. Lanckriet, N. Cristiani, P. Bartlett, L. El Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[80] J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10:777–801, 2009.

[81] H. Larochelle and Y. Bengio. Classification using discriminative restricted Boltzmann machines. In *Proceedings of the International Conference on Machine Learning*, 2008.

[82] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10:1–40, 2009.

[83] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the International Conference on Machine Learning*, 2007.

[84] D. Lawley and A. Maxwell. Factor analysis as a statistical method. *The Statistician*, 12(3):209–229, 1962.

[85] N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *Advances in Neural Information Processing Systems 16*, 2004.

[86] N. D. Lawrence and M. I. Jordan. Semi-supervised learning via Gaussian processes. *Advances in Neural Information Processing Systems*, 17:753–760, 2005.

[87] S. Lazebnik, C. Schmid, and J. Ponce. Semi-local affine parts for object recognition. In *Proceedings of the British Machine Vision Conference*, 2004.

[88] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[90] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[91] H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems 19*, 2007.

[92] H. Lee, C. Ekanadham, and A. Y. Ng. Sparse deep belief network model for visual area V2. In *Advances in Neural Information Processing Systems 20*, 2008.

[93] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the International Conference on Machine Learning*, 2009.

[94] H. Lee, Y. Largman, P. Pham, and A. Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems 22*, pages 1096–1104. 2009.

[95] H. Lee, R. Raina, A. Teichman, and A. Y. Ng. Exponential family sparse coding with applications to self-taught learning. In *Proceedings of the international Joint Conference on Artifical Intelligence*, pages 1113–1119, 2009.

[96] J. A. Lee and M. Verleysen. *Nonlinear dimensionality reduction*. Springer Verlag, 2007.

[97] L. Lee. Measures of distributional similarity. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 25–32. Association for Computational Linguistics, 1999.

[98] S.-I. Lee, V. Chatalbashev, D. Vickrey, and D. Koller. Learning a meta-level prior for feature relevance from multiple related tasks. In *Proceedings of the International Conference on Machine Learning*, 2007.

[99] S.-I. Lee, H. Lee, P. Abbeel, and A. Y. Ng. Efficient L_1 regularized logistic regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2006.

[100] T. S. Lee and D. Mumford. Hierarchical Bayesian inference in the visual cortex. *Journal of the Optical Society of America A*, 20(7):1434–1448, 2003.

[101] R. G. Leonard and G. Doddington. TIDIGITS speech corpus. *Linguistic Data Consortium, Philadelphia*, 1993.

[102] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.

[103] M. S. Lewicki and T. J. Sejnowski. Learning overcomplete representations. *Neural Computation*, 12(2):337–365, 2000.

[104] D. Lin. Automatic retrieval and clustering of similar words. In *Proceedings of the 17th international conference on Computational linguistics-Volume 2*, pages 768–774. Association for Computational Linguistics, 1998.

[105] J. Lokhorst. *The LASSO and Generalised Linear Models*. Honors Project, Department of Statistics, The University of Adelaide, South Australia, Australia., 1999.

[106] D. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision*, 1999.

[107] 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, pages 281–297, 1967.

[108] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the International Conference on Machine Learning*, pages 689–696, 2009.

[109] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *The Journal of Machine Learning Research*, 11:19–60, 2010.

[110] J. Mairal, M. Elad, G. Sapiro, et al. Sparse representation for color image restoration. *IEEE Transactions on Image Processing*, 17(1):53, 2008.

[111] J. Mairal, M. Leordeanu, F. Bach, M. Hebert, and J. Ponce. Discriminative sparse image models for class-specific edge detection and image interpretation. In *Proceedings of the European Conference on Computer Vision*, 2008.

[112] P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman & Hall, 1989.

[113] P. Mermelstein. Distance measures for speech recognition: Psychological and instrumental. In *Pattern Recognition and Artificial Intelligence*, pages 374–388. 1976.

[114] S. Mika, B. Schölkopf, A. Smola, K. Müller, M. Scholz, and G. Rätsch. Kernel PCA and de-noising in feature spaces. *Advances in Neural Information Processing Systems*, 11(1):536–542, 1999.

[115] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005.

[116] J. Mutch and D. G. Lowe. Multiclass object recognition with sparse, localized features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[117] V. Nair and G. E. Hinton. 3d object recognition with deep belief nets. In *Advances in Neural Information Processing Systems 22*, pages 1339–1347. 2009.

[118] A. Y. Ng. Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In *Proceedings of the International Conference on Machine Learning*, 2004.

[119] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine learning*, 39(2):103–134, 2000.

[120] M. Norouzi, M. Ranjbar, and G. Mori. Stacks of convolutional restricted Boltzmann machines for shift-invariant feature learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[121] B. A. Olshausen. Sparse coding of time-varying natural images. *Journal of Vision*, 2(7):130, 2002.

[122] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

[123] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311–3325, 1997.

[124] B. A. Olshausen and D. J. Field. Sparse coding of sensory inputs. *Current Opinion in Neurobiology*, 14(4):481–487, 2004.

[125] S. Osindero, M. Welling, and G. E. Hinton. Topographic product models applied to natural scene statistics. *Neural Computation*, 18(2):381–344, 2006.

[126] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints), 2009.

[127] P. Pantel and D. Lin. An unsupervised approach to prepositional phrase attachment using contextually similar words. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, 2000.

[128] S. Perkins and J. Theiler. Online feature selection using grafting. In *Proceedings of the International Conference on Machine Learning*, 2003.

[129] S. Petrov, A. Pauls, and D. Klein. Learning structured models for phone recognition. In *EMNLP-CoNLL*, 2007.

[130] R. Raina. *Self-taught Learning*. PhD thesis, Stanford University, 2009.

[131] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng. Self-taught learning: Transfer learning from unlabeled data. In *Proceedings of the International Conference on Machine Learning*, 2007.

[132] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the International Conference on Machine Learning*, 2009.

[133] M. Ranzato. *Unsupervised Learning of Feature Hierarchies*. PhD thesis, New York University, 2009.

[134] M. Ranzato, Y. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. *Advances in Neural Information Processing Systems*, 2007.

[135] M. Ranzato, F.-J. Huang, Y.-L. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[136] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems*, pages 1137–1144, 2006.

[137] M. Ranzato and M. Szummer. Semi-supervised learning of compact document representations with deep networks. In *Proceedings of the International Conference on Machine Learning*, 2008.

[138] D. A. Reynolds. Speaker identification and verification using gaussian mixture speaker models. *Speech Communication*, 17(1-2):91–108, 1995.

[139] V. Roth. The generalized lasso. *IEEE Transactions on Neural Networks*, 15(1):16–28, 2004.

[140] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323, 2000.

[141] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[142] R. Salakhutdinov and G. E. Hinton. Deep Boltzmann Machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2009.

[143] R. Salakhutdinov and G. E. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[144] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. *Advances in Neural Information Processing Systems*, 20:1257–1264, 2008.

[145] M. P. Sceniak, M. J. Hawken, and R. Shapley. Visual spatial characterization of macaque V1 neurons. *The Journal of Neurophysiology*, 85(5):1873–1887, 2001.

[146] E. Scheirer and M. Slaney. Construction and evaluation of a robust multifeature speech/music discriminator. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1997.

[147] F. Sha and L. K. Saul. Large margin gaussian mixture modeling for phonetic classication and recognition. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 2006.

[148] E. C. Smith and M. S. Lewicki. Efficient auditory coding. *Nature*, 439:978–982, 2006.

[149] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1:194–281, 1986.

[150] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1565, 2006.

[151] Y.-H. Sung, C. Boulis, C. Manning, and D. Jurafsky. Regularization, adaptation, and non-independent features improve hidden conditional random fields for phone classification. In *Proceedings of IEEE workshop on Automatic Speech Recognition and Understanding*, 2007.

[152] M. Szummer and T. Jaakkola. Partially labeled classification with Markov random walks. *Advances in Neural Information Processing Systems*, 2:945–952, 2002.

[153] G. W. Taylor and G. E. Hinton. Factored conditional restricted Boltzmann machines for modeling motion style. In *Proceedings of the International Conference on Machine Learning*, pages 1025–1032, 2009.

[154] G. W. Taylor, G. E. Hinton, and S. T. Roweis. Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems 19*, pages 1345–1352. 2007.

[155] J. B. Tenenbaum, V. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319, 2000.

[156] S. Thrun. Is learning the n-th thing any easier than learning the first? *Advances in Neural Information Processing Systems*, pages 640–646, 1996.

[157] T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the International Conference on Machine Learning*, 2008.

[158] D. M. Titterington, A. F. M. Smith, and U. E. Makov. *Statistical Analysis of Finite Mixture Distributions*. John Wiley & Sons, 1985.

[159] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2008.

[160] J. H. van Hateren and A. van der Schaaf. Independent component filters of natural images compared with simple cells in primary visual cortex. *Proceedings of the Royal Society B*, 265:359–366, 1998.

[161] M. Varma and D. Ray. Learning the discriminative power-invariance trade-off. In *Proceedings of the International Conference on Computer Vision*, 2007.

[162] X. Wei and W. B. Croft. LDA-based document models for ad-hoc retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.

[163] J. Weston, R. Collobert, F. Sinz, L. Bottou, and V. Vapnik. Inference with the universum. In *Proceedings of the International Conference on Machine Learning*, 2006.

[164] J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the International Conference on Machine Learning*, 2008.

[165] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, 2003.

[166] J. Yang, J. Wright, T. Huang, and Y. Ma. Image super-resolution via sparse representation. *IEEE Transactions on Image Processing*, 2010. Accepted for publication.

[167] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[168] L. Yang and R. Jin. Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2006.

[169] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 189–196, 1995.

[170] L. Younes. Maximum of likelihood estimation for Gibbsianfields. *Probability Theory and Related Fields*, 82:625–645, 1989.

[171] D. Yu, L. Deng, and A. Acero. Hidden conditional random field with distribution constraints for phone classification. In *Interspeech*, 2009.

[172] J. Yu, S. V. N. Vishwanathan, S. Günter, and N. N. Schraudolph. A quasi-Newton approach to nonsmooth convex optimization. In *Proceedings of the International Conference on Machine Learning*, 2008.

[173] K. Yu, W. Xu, and Y. Gong. Deep learning with kernel regularization for visual recognition. In *Advances in Neural Information Processing Systems*, 2009.

[174] A. L. Yuille. The convergence of contrastive divergences. In *Advances in Neural Information Processing Systems 17*, 2005.

[175] H. Zhang, A. C. Berg, M. Maire, and J. Malik. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[176] X. Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison, 2008.

[177] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the International Conference on Machine Learning*, 2003.

[178] H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286, 2006.