LEARNING DEEP GENERATIVE MODELS

by

Ruslan Salakhutdinov

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Learning Deep Generative Models

Ruslan Salakhutdinov
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2009

Building intelligent systems that are capable of extracting high-level representations from high-dimensional sensory data lies at the core of solving many AI related tasks, including object recognition, speech perception, and language understanding. Theoretical and biological arguments strongly suggest that building such systems requires models with deep architectures that involve many layers of nonlinear processing.

The aim of the thesis is to demonstrate that deep generative models that contain many layers of latent variables and millions of parameters can be learned efficiently, and that the learned high-level feature representations can be successfully applied in a wide spectrum of application domains, including visual object recognition, information retrieval, and classification and regression tasks. In addition, similar methods can be used for nonlinear dimensionality reduction.

The first part of the thesis focuses on analysis and applications of probabilistic generative models called Deep Belief Networks. We show that these deep hierarchical models can learn useful feature representations from a large supply of unlabeled sensory inputs. The learned high-level representations capture a lot of structure in the input data, which is useful for subsequent problem-specific tasks, such as classification, regression or information retrieval, even though these tasks are unknown when the generative model is being trained.

In the second part of the thesis, we introduce a new learning algorithm for a different type of hierarchical probabilistic model, which we call a Deep Boltzmann Machine. Like Deep Belief Networks, Deep Boltzmann Machines have the potential of learning internal representations that become increasingly complex at higher layers, which is a promising way of solving object and speech recognition problems. Unlike Deep Belief Networks and many existing models with deep architectures, the approximate inference procedure, in addition to a fast bottom-up pass, can incorporate top-down feedback. This allows Deep Boltzmann Machines to better propagate uncertainty about ambiguous inputs.

Acknowledgements

First and foremost, I would like to thank my advisor Geoffrey Hinton for being an incredible advisor, an amazing teacher, and providing me with a warm and outstanding intellectual environment at the University of Toronto. I would also like to thank Sam Roweis, my second advisor, and my committee members, Radford Neal and Rich Zemel, for their valuable feedback, support, and guidance.

Many thanks goes to the members of the Toronto Machine Learning group: Andriy Mnih, Iain Murray, Vinod Nair, Ilya Sutskever, and Tijmen Tieleman for many interesting and inspiring discussions.

I also thank my parents and my family for their continuing support. Finally, a special thank you goes to my wife Olga for being supportive and putting up with me.

Contents

Chapter 1

Introduction

Building intelligent systems that have the potential of extracting high-level representations from rich sensory input lies at the core of solving many AI related tasks, including visual object recognition, speech perception, and language understanding. Theoretical and biological arguments strongly suggest that building such systems requires deep architectures – models that are composed of several layers of nonlinear processing.

Many existing machine learning algorithms use "shallow" architectures, including neural networks with only one hidden layer, kernel regression, support vector machines, and many others. Theoretical results show that the internal representations learned by such systems are necessarily simple and are incapable of extracting some types of complex structure from rich sensory input (Bengio and LeCun [2007], Bengio [2009]). Training these systems also requires large amounts of labeled training data. By contrast, it appears that, for example, object recognition in the visual cortex uses many layers of nonlinear processing and requires very little labeled input (Lee et al. [1998]). Therefore developing new and efficient learning algorithms for models with deep architectures, that can also make efficient use of a large supply of unlabeled sensory input, is of crucial importance.

Multilayer neural networks are perhaps the best examples of models with deep architectures. Backpropagation (Rumelhart et al. [1986]) was the first learning algorithm for these deep networks that could learn multiple layers of representation. However, in additional to requiring labeled data, backpropagation does not work well in practice when training models that contain more than a few layers (DeMers and Cottrell [1993], Hecht-Nielsen [1995], Tesauro [1992], Bengio et al. [2007], Larochelle et al. [2009]). In general, since models with deep architectures are composed of several layers of parameterized nonlinear modules, the associated loss functions are almost always non-convex. The presence of many bad local optima or plateaus in the loss function makes deep models far more difficult to optimize. Local gradient-based optimization algorithms, such as backpropagation, that start at some random initial configuration, often get trapped in a poor local optimum, particularly when training models with more than two or three layers. By contrast, models with shallow architectures (e.g. support vector machines) generally use convex loss functions, which typically allows one to carry out parameter optimization efficiently in these models. The appeal of convexity has steered most of machine learning research into developing learning algorithms that can be cast as solving convex optimization problems.

Recently, Hinton et al. [2006] introduced a moderately fast, unsupervised learning algorithm for deep generative models called Deep Belief Networks (DBN's). A key feature of this algorithm is its greedy layer-by-layer training that can be repeated several times in order to efficiently learn a deep, hierarchical probabilistic model. The new learning algorithm has excited many researchers in the machine learning community, primarily because of the following three crucial characteristics:

1. The greedy layer-by-layer learning algorithm can find a good set of model parameters fairly

quickly, even for models that contain many layers of nonlinearities and millions of parameters.

2. The learning algorithm can make efficient use of very large sets of unlabeled data, and so the model can be pretrained in completely unsupervised fashion. The very limited labeled data can then be used to only slightly fine-tune the model for a specific task at hand using standard gradient-based optimization.

3. There is an efficient way of performing approximate inference, which makes the values of the latent variables in the deepest layer easy to infer.

The strategy of layer-wise unsupervised training allows efficient training of deep networks and gives promising results for many challenging learning problems. Many variants of this greedy algorithm have been successfully applied not only for classification tasks (Hinton et al. [2006], Bengio et al. [2007], Larochelle et al. [2009]), but also regression tasks (Salakhutdinov and Hinton [2008]), visual object recognition (Ranzato et al. [2007, 2008], Bengio and LeCun [2007], Ahmed et al. [2008]), dimensionality reduction (Hinton and Salakhutdinov [2006], Salakhutdinov and Hinton [2007b]), information retrieval (Ranzato and Szummer [2008], Torralba et al. [2008], Salakhutdinov and Hinton [2007a]), modeling image patches (Osindero and Hinton [2008]), extracting optical flow (Memisevic and Hinton [2007]), and robotics (Hadsell et al. [2008]). Research on models with deep architectures is still at an early stage. Much of the current thesis will focus on analysis and applications, as well as developing new learning algorithms for deep hierarchical generative models.

The thesis has two main parts, which can be read almost independently. In the first part, we will primarily concentrate on analysis and applications of Deep Belief Networks. First, we will address the question of how well Deep Belief Networks perform in various applications, including dimensionality reduction, information retrieval, regression and classification tasks, particularly when dealing with a large supply of high-dimensional, richly structured unlabeled input and very limited amount of labeled training data. Second, we will address the problem of assessing generalization performance of Deep Belief Networks as density models, which will allow us to not only compare DBN's to other probabilistic models, but also perform model selection and complexity control.

In the second part of the thesis, we will introduce a new learning algorithm for a different type of hierarchical probabilistic model, which we call a Deep Boltzmann Machine (DBM). Deep Boltzmann Machines, like Deep Belief Networks, have the potential of learning internal representations that become progressively complex at higher layers. High-level representations can be built from a large supply of unlabeled sensory inputs and the very limited labeled data can then be used to only slightly adjust the model for a problem-specific task. Second, unlike Deep Belief Networks and many existing models with deep architectures (Larochelle et al. [2009], Bengio and LeCun [2007], Ahmed et al. [2008]), the approximate inference procedure, in addition to a bottom-up pass, can incorporate top-down feedback, allowing Deep Boltzmann Machines to better propagate uncertainty about ambiguous inputs. We will show that DBM's can learn good generative models and perform well on handwritten digit and visual object recognition tasks.

1.1 Contributions of This Thesis

The most significant research contributions in this thesis are:

1. We show how the feature representations that a Deep Belief Network extracts from a large supply of unlabeled data can be used to learn a good covariance kernel for a Gaussian process. If the input data is high-dimensional and highly-structured, a Gaussian kernel applied to the top layer of extracted features in the DBN works much better than a similar kernel applied to the raw input,

especially if the DBN is fine-tuned by backpropagating gradients obtained from the Gaussian process.

2. We introduce an efficient way of initializing the weights of deep autoencoders based on the greedy learning algorithm for Deep Belief Networks. This allows deep autoencoder networks to learn low-dimensional codes that work much better than principal component analysis as a tool to reduce the dimensionality of data.

3. We demonstrate how deep autoencoders can learn to map documents into "semantic" binary codes. By using learned binary codes as memory addresses, we can learn a *Semantic Address Space*, so a document can be mapped to a memory address in such a way that a small hamming-ball around that memory address contains semantically similar documents. We call this model "Semantic Hashing" and show that it allows us to perform very fast and accurate information retrieval.

4. We show how to efficiently pretrain and fine-tune a deep nonlinear transformation from the input space to a low-dimensional feature space in which K-nearest neighbour classification performs well.

5. We show how a Monte Carlo based algorithm, Annealed Importance Sampling, combined with approximate inference, can be used to estimate a lower bound on the log-probability that a Deep Belief Network with multiple hidden layers assigns to the test data. This allows us to directly assess generalization performance of Deep Belief Networks as density models.

6. Finally, we introduce a new learning algorithm for Boltzmann machines that combines variational techniques and Markov chain Monte Carlo. The new algorithm readily extends to learning Boltzmann machines with real-valued, count, or tabular data. We further introduce a modified greedy layer-by-layer pretraining algorithm that will allow us to quickly find a good set of model parameters for Deep Boltzmann Machines.

1.2 Summary of Remaining Chapters

Chapter 2: Deep Belief Networks. In this chapter we provide a brief technical overview of Restricted Boltzmann Machines (RBM's), that form component modules of Deep Belief Networks, as well as generalizations of RBM's to modeling real-valued and count data. We then review the greedy learning algorithm for Deep Belief Networks.

Chapter 3: Learning Feature Hierarchies with Deep Belief Networks. This chapter presents several ideas based on greedily learning a hierarchy of features from high-dimensional, highly-structured sensory input. We first show how unlabeled data and a Deep Belief Network can be used learn a good covariance kernel for a Gaussian process. We then show how the greedy learning algorithm can be used to make nonlinear autoencoders work considerably better than widely used methods, such as principal component analysis and singular value decomposition. We also demonstrate that these deep autoencoders can be used to discover binary "semantic" codes that allow fast and accurate information retrieval. Finally, we show how to pretrain and fine-tune a deep nonlinear network to learn a similarity metric over the input space that facilitates nearest-neighbor classification. Some of this material appeared in Hinton and Salakhutdinov [2006], Salakhutdinov and Hinton [2007a,b, 2008, 2009b, 2010], and Goldberger, Roweis, Hinton and Salakhutdinov [2004].

Chapter 4: Evaluating Deep Belief Networks as Density Models. In this chapter we show how a Monte Carlo method, Annealed Importance Sampling (AIS), can be used to efficiently estimate the partition function of an RBM. We further show how an AIS estimator, along with approximate inference, can be used to estimate a lower bound on the log-probability that a Deep Belief Network assigns to the test data. Some of this material appeared in Salakhutdinov [2008] and Salakhutdinov and Murray [2008].

Chapter 5: Deep Boltzmann Machines. This chapter presents a new learning algorithms for a different type of hierarchical probabilistic model: a Deep Boltzmann Machine (DBM). Approximate inference can be performed using variational approaches, such as mean-field. Learning can then be carried out by applying a stochastic approximation procedure that uses Markov chain Monte Carlo (MCMC) to approximate a model's expected sufficient statistics, which is needed for maximum likelihood learning. The MCMC based approximation procedure provides nice asymptotic convergence guarantees and belongs to the general class of approximation algorithms of Robbins–Monro type. We show that this unusual combination of variational methods and MCMC is essential for creating a fast learning algorithm for Deep Boltzmann Machines. Some of this material appeared in Salakhutdinov [2008, 2010] and Salakhutdinov and Hinton [2009a].

Chapter 6: Conclusions. In this chapter we provide a brief summary of our contributions and discuss possible future research directions.

Chapter 2

Deep Belief Networks

Deep Belief Networks (DBN's) are probabilistic generative models that contain many layers of hidden variables, in which each layer captures high-order correlations between the activities of hidden features in the layer below. The top two layers of the DBN form an undirected bipartite graph with the lower layers forming a directed sigmoid belief network, as shown in Fig. 2.3. Hinton et al. [2006] introduced a fast, unsupervised learning algorithm for these deep networks, which we review in this chapter. A key feature of this algorithm is its greedy layer-by-layer training that can be repeated several times to learn a deep, hierarchical model. The learning procedure also provides an efficient way of performing approximate inference, which only requires a single bottom-up pass to infer the values of the top-level hidden variables.

The main building block of a DBN is a bipartite undirected graphical model called the Restricted Boltzmann Machine (RBM). RBM's, and their generalizations to exponential family models (Welling et al. [2005]), have been successfully applied in collaborative filtering (Salakhutdinov et al. [2007]), information and image retrieval (Gehler et al. [2006]), and time series modeling (Taylor et al. [2006], Sutskever and Hinton [2006]). In this chapter we provide a brief technical overview of RBM's, generalizations of RBM's to modeling real-valued and count data, and the greedy learning algorithm for Deep Belief Networks.

2.1 Restricted Boltzmann Machines

A Restricted Boltzmann Machine is a particular type of Markov random field that has a two-layer architecture (Smolensky [1986]), in which the visible, binary stochastic units $\mathbf{v} \in \{0,1\}^D$ are connected to hidden binary stochastic units $\mathbf{h} \in \{0,1\}^F$, as shown in Fig. 2.1. The energy of the state $\{\mathbf{v}, \mathbf{h}\}$ is:

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}; \theta) &= -\mathbf{v}^\top W \mathbf{h} - \mathbf{b}^\top \mathbf{v} - \mathbf{a}^\top \mathbf{h} \\ &= -\sum_{i=1}^{D}\sum_{j=1}^{F} W_{ij} v_i h_j - \sum_{i=1}^{D} b_i v_i - \sum_{j=1}^{F} a_j h_j, \end{aligned} \tag{2.1}$$

where $\theta = \{W, \mathbf{b}, \mathbf{a}\}$ are the model parameters: W_{ij} represents the symmetric interaction term between visible unit i and hidden unit j; b_i and a_j are bias terms. The joint distribution over the visible and hidden units is defined by:

$$P(\mathbf{v}, \mathbf{h}; \theta) = \frac{1}{\mathcal{Z}(\theta)} \exp\left(-E(\mathbf{v}, \mathbf{h}; \theta)\right), \tag{2.2}$$

$$\mathcal{Z}(\theta) = \sum_{\mathbf{v}}\sum_{\mathbf{h}} \exp\left(-E(\mathbf{v}, \mathbf{h}; \theta)\right). \tag{2.3}$$

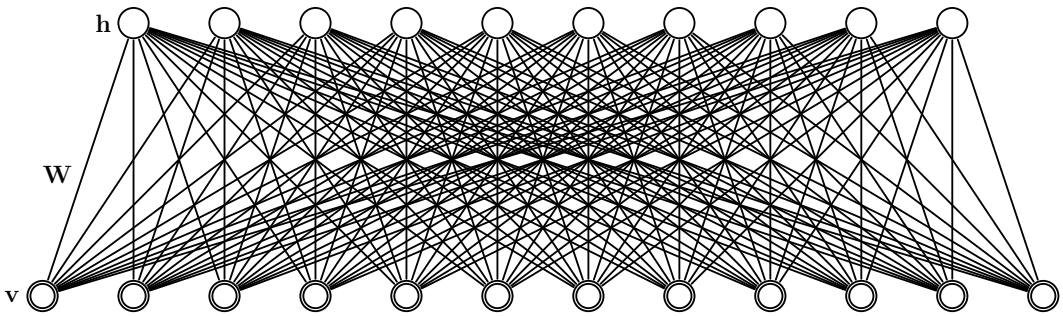

Figure 2.1: Restricted Boltzmann Machine. The top layer represents a vector of stochastic binary units $\mathbf{h}$ and the bottom layer represents a vector of stochastic binary visible variables $\mathbf{v}$.

$\mathcal{Z}(\theta)$ is known as the partition function or normalizing constant. The probability that the model assigns to a visible vector $\mathbf{v}$ is:

$$P(\mathbf{v};\theta) \;=\; \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{v},\mathbf{h};\theta)\right). \tag{2.4}$$

Due to the special bipartite structure of RBM's, the hidden units can be explicitly marginalized out:

$$\begin{aligned} P(\mathbf{v};\theta) \;&=\; \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left(\mathbf{v}^\top W\mathbf{h} + \mathbf{b}^\top\mathbf{v} + \mathbf{a}^\top\mathbf{h}\right) \\ &=\; \frac{1}{\mathcal{Z}(\theta)} \exp(\mathbf{b}^\top\mathbf{v}) \prod_{j=1}^{F} \sum_{h_j\in\{0,1\}} \exp\left(a_j h_j + \sum_{i=1}^{D} W_{ij} v_i h_j\right) \\ &=\; \frac{1}{\mathcal{Z}(\theta)} \exp(\mathbf{b}^\top\mathbf{v}) \prod_{j=1}^{F} \left(1 + \exp\left(a_j + \sum_{i=1}^{D} W_{ij} v_i\right)\right). \end{aligned} \tag{2.5}$$

The conditional distributions over hidden units $\mathbf{h}$ and visible vector $\mathbf{v}$ can be easily derived from Eq. 2.2 and are given by logistic functions:

$$P(\mathbf{h}|\mathbf{v};\theta) \;=\; \prod_j p(h_j|\mathbf{v}), \quad P(\mathbf{v}|\mathbf{h};\theta) \;=\; \prod_i p(v_i|\mathbf{h}), \tag{2.6}$$

$$p(h_j = 1|\mathbf{v}) \;=\; g\left(\sum_i W_{ij} v_i + a_j\right), \tag{2.7}$$

$$p(v_i = 1|\mathbf{h}) \;=\; g\left(\sum_j W_{ij} h_j + b_i\right), \tag{2.8}$$

where $g(x) = 1/(1+\exp(-x))$ is the logistic function. The derivative of the log-likelihood with respect to the model parameters θ can be obtained from Eq. 2.4:

$$\frac{\partial \log P(\mathbf{v};\theta)}{\partial W} \;=\; \mathrm{E}_{P_{\mathrm{data}}}[\mathbf{v}\mathbf{h}^\top] - \mathrm{E}_{P_{\mathrm{Model}}}[\mathbf{v}\mathbf{h}^\top], \tag{2.9}$$

$$\frac{\partial \log P(\mathbf{v};\theta)}{\partial \mathbf{a}} \;=\; \mathrm{E}_{P_{\mathrm{data}}}[\mathbf{h}] - \mathrm{E}_{P_{\mathrm{Model}}}[\mathbf{h}], \tag{2.10}$$

$$\frac{\partial \log P(\mathbf{v};\theta)}{\partial \mathbf{b}} \;=\; \mathrm{E}_{P_{\mathrm{data}}}[\mathbf{v}] - \mathrm{E}_{P_{\mathrm{Model}}}[\mathbf{v}]. \tag{2.11}$$

$\mathrm{E}_{P_{\mathrm{data}}}[\cdot]$ denotes an expectation with respect to the data distribution $P_{\mathrm{data}}(\mathbf{h}, \mathbf{v}; \theta) = P(\mathbf{h}|\mathbf{v}; \theta)P_{\mathrm{data}}(\mathbf{v})$, with $P_{\mathrm{data}}(\mathbf{v}) = \frac{1}{N}\sum_n \delta(\mathbf{v} - \mathbf{v}_n)$ representing the empirical distribution, and $\mathrm{E}_{P_{\mathrm{Model}}}[\cdot]$ is an expectation with respect to the distribution defined by the model, as in Eq. 2.2. Exact maximum likelihood learning in this model is intractable because exact computation of the expectation $\mathrm{E}_{P_{\mathrm{Model}}}[\cdot]$ takes time that is exponential in $\min\{D, F\}$, i.e the number of visible or hidden units. In practice, learning is done by following an approximation to the gradient of a different objective function, called the "Contrastive Divergence" (CD) (Hinton [2002]):

$$\Delta W \;=\; \alpha\left(\mathrm{E}_{P_{\mathrm{data}}}[\mathbf{v}\mathbf{h}^\top] - \mathrm{E}_{P_T}[\mathbf{v}\mathbf{h}^\top]\right), \tag{2.12}$$

where α is the learning rate and P_T represents a distribution defined by running a Gibbs chain, initialized at the data, for T full steps. The special bipartite structure of RBM's allows for quite an efficient Gibbs sampler that alternates between sampling the states of the hidden units independently given the states of the visible units, and vise versa (see Eq. 2.6). Setting $T = \infty$ recovers maximum likelihood learning. In many application domains, however, the CD learning with $T = 1$ (or CD1) has been shown to work quite well (Hinton [2002], Welling et al. [2005], Larochelle et al. [2009]).

2.2 A Greedy Learning Algorithm for Deep Belief Networks

The ideas underlying the greedy learning algorithm for DBN's are actually rather simple. Consider learning a DBN with two layers of hidden units $\{\mathbf{h}^1, \mathbf{h}^2\}$. We will also assume that the number of the 2^{nd} layer hidden units is the same as the number of visible units (see Fig. 2.2, right panel). The top two layers of the DBN form an undirected bipartite graph (an RBM) and the lower layers form a directed sigmoid belief network. The joint distribution over $\mathbf{v}$, $\mathbf{h}^1$, and $\mathbf{h}^2$ defined by this model takes the following form[1]:

$$P(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2; \theta) \;=\; P(\mathbf{v}|\mathbf{h}^1; W^1)P(\mathbf{h}^1, \mathbf{h}^2; W^2), \tag{2.13}$$

where $\theta = \{W^1, W^2\}$ are the model parameters, $P(\mathbf{v}|\mathbf{h}^1; W^1)$ is the directed sigmoid belief network, and $P(\mathbf{h}^1, \mathbf{h}^2; W^2)$ is the joint distribution defined by the second layer RBM:

$$P(\mathbf{v}|\mathbf{h}^1; W^1) \;=\; \prod_i p(v_i|\mathbf{h}^1; W^1), \qquad p(v_i = 1|\mathbf{h}^1; W^1) = g\left(\sum_j W^1_{ij} h^1_j\right), \tag{2.14}$$

$$P(\mathbf{h}^1, \mathbf{h}^2; W^2) \;=\; \frac{1}{Z(W^2)} \exp\left({\mathbf{h}^1}^\top W^2 \mathbf{h}^2\right). \tag{2.15}$$

The greedy strategy relies on the following key observation. Consider a two-hidden-layer DBN with tied parameters $W^2 = {W^1}^\top$. Then this DBN's joint distribution $P(\mathbf{v}, \mathbf{h}^1; \theta) = \sum_{\mathbf{h}^2} P(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2; \theta)$ is identical to the RBM's joint distribution $P(\mathbf{v}, \mathbf{h}^1; W^1)$. Indeed, it is easy to see from Fig. 2.2 that both $P(\mathbf{h}^1; W^1)$ and $P(\mathbf{v}|\mathbf{h}^1; W^1)$ are the same for both models. To be more precise, using

[1] We will omit the bias terms for clarity of presentation.

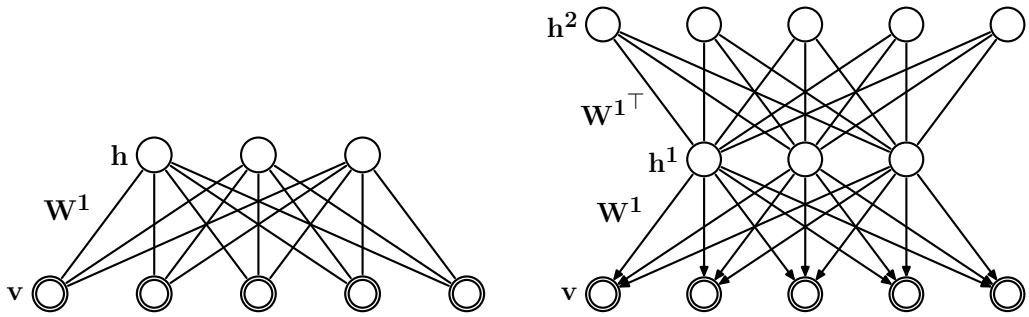

Figure 2.2: **Left:** Restricted Boltzmann Machine. **Right:** A two-hidden-layer Deep Belief Network with tied weights $W^2 = W^{1\top}$. The joint distribution $P(\mathbf{v}, \mathbf{h}^1; W^1)$ defined by this DBN is identical to the joint distribution $P(\mathbf{v}, \mathbf{h}^1; W^1)$ defined by an RBM.

Eqs. 2.13, 2.14, 2.15 and the fact that $W^2 = W^{1\top}$, we obtain the DBN's joint distribution:

$$\begin{aligned}
P(\mathbf{v}, \mathbf{h}^1; \theta) &= P(\mathbf{v}|\mathbf{h}^1; W^1) \times \sum_{\mathbf{h}^2} P(\mathbf{h}^1, \mathbf{h}^2; W^2) \\
&= \prod_i p(v_i|\mathbf{h}^1; W^1) \times \frac{1}{\mathcal{Z}(W^2)} \prod_i \left(1 + \exp\left(\sum_j W^2_{ji} h^1_j\right)\right) \\
&= \prod_i \frac{\exp\left(v_i \sum_j W^1_{ij} h^1_j\right)}{1 + \exp\left(\sum_j W^1_{ij} h^1_j\right)} \times \frac{1}{\mathcal{Z}(W^2)} \prod_i \left(1 + \exp\left(\sum_j W^2_{ji} h^1_j\right)\right) \\
&= \frac{1}{\mathcal{Z}(W^1)} \prod_i \left(\exp\left(v_i \sum_j W^1_{ij} h^1_j\right)\right) \qquad \left[\text{since } W^2_{ji} = W^1_{ij}, \quad \mathcal{Z}(W^1) = \mathcal{Z}(W^2)\right] \\
&= \frac{1}{\mathcal{Z}(W^1)} \exp\left(\sum_{ij} W^1_{ij} v_i h^1_j\right), \qquad (2.16)
\end{aligned}$$

which is identical to the joint distribution defined by an RBM (Eq. 2.2).

The greedy learning algorithm uses a stack of RBM's and proceeds as follows. We first train the bottom RBM with parameters W^1, as described in section 2.1. We then initialize the 2^{nd} layer weights to $W^2 = W^{1\top}$, which ensures that the two-hidden-layer DBN is at least as good as our original RBM. We can now improve the DBN's fit to the training data by untying and refining W^2.

For any approximating distribution $Q(\mathbf{h}^1|\mathbf{v})$, the log-likelihood of the two-hidden-layer DBN model has the following variational lower bound, where the states $\mathbf{h}^2$ are analytically summed out:

$$\begin{aligned}
\log P(\mathbf{v}; \theta) &\geq \sum_{\mathbf{h}^1} Q(\mathbf{h}^1|\mathbf{v}) \Big[\log P(\mathbf{v}, \mathbf{h}^1; \theta)\Big] + \mathcal{H}(Q(\mathbf{h}^1|\mathbf{v})) \\
&= \sum_{\mathbf{h}^1} Q(\mathbf{h}^1|\mathbf{v}) \Big[\log P(\mathbf{h}^1; W^2) \; + \log P(\mathbf{v}|\mathbf{h}^1; W^1)\Big] + \mathcal{H}(Q(\mathbf{h}^1|\mathbf{v})), \qquad (2.17)
\end{aligned}$$

where $\mathcal{H}(\cdot)$ is the entropy functional. We set $Q(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v}; W^1)$ defined by the bottom RBM (Eq. 2.6). Initially, when $W^2 = W^{1\top}$, Q is the DBN's true factorial posterior over $\mathbf{h}^1$, in which case the bound is tight. The strategy of greedy learning algorithm is to freeze the parameter vector W^1 and

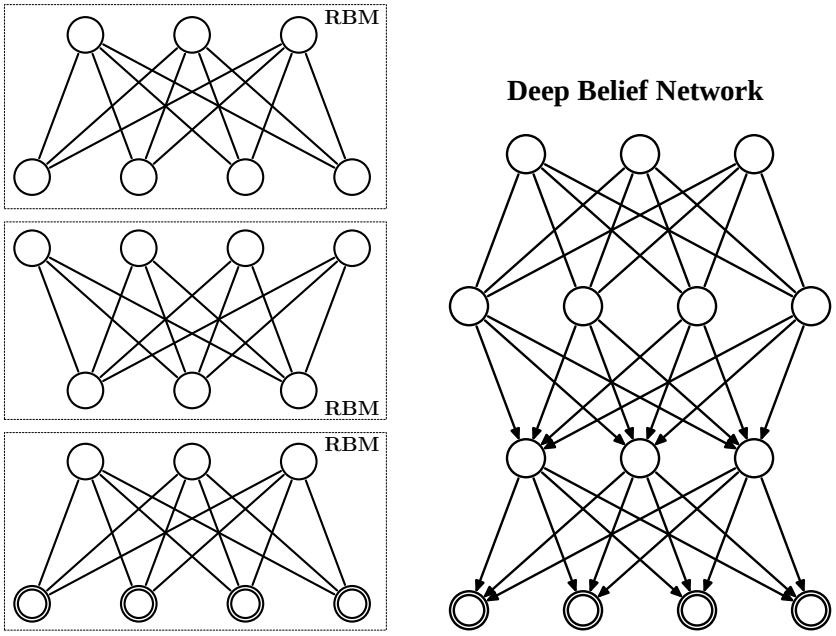

Figure 2.3: **Left:** Greedy learning a stack of RBM's in which the samples from the lower-level RBM are used as the data for training the next RBM. **Right:** The corresponding Deep Belief Network.

Algorithm 1 Recursive Greedy Learning Procedure for the DBN.

1: Fit parameters W^1 of the 1^{st} layer RBM to data.
2: Freeze the parameter vector W^1 and use samples $\mathbf{h}^1$ from $Q(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v}, W^1)$ as the data for training the next layer of binary features with an RBM.
3: Freeze the parameters W^2 that define the 2^{nd} layer of features and use the samples $\mathbf{h}^2$ from $Q(\mathbf{h}^2|\mathbf{h}^1) = P(\mathbf{h}^2|\mathbf{h}^1, W^2)$ as the data for training the 3^{rd} layer of binary features.
4: Proceed recursively for the next layers.

attempt to learn a better model for $P(\mathbf{h}^1; W^2)$ by maximizing the variational lower bound of Eq. 2.17 with respect to W^2. Maximizing this bound with frozen W^1 amounts to maximizing:

$$\sum_{\mathbf{h}^1} Q(\mathbf{h}^1|\mathbf{v}) \log P(\mathbf{h}^1; W^2), \tag{2.18}$$

which is equivalent to maximum likelihood training of the 2^{nd} layer RBM with vectors $\mathbf{h}^1$ drawn from $Q(\mathbf{h}^1|\mathbf{v})$ as data. When presented with a dataset of N training input vectors, the 2^{nd} layer RBM, $P(\mathbf{h}^1; W^2)$, will learn a better model of the "aggregated" posterior over $\mathbf{h}^1$, which is simply the mixture of factorial posteriors for all the training cases: $\frac{1}{N}\sum_n P(\mathbf{h}^1|\mathbf{v}_n; W^1)$. Note that any increase in the variational lower bound, as a result of changing W^2, will result in an increase of the DBN's data likelihood[2]

This idea can be extended to training the 3^{rd} layer RBM on vectors $\mathbf{h}^2$ drawn from the second RBM. By initializing $W^3 = W^{2\top}$, we are guaranteed to improve the lower bound on the log-likelihood,

[2]Improving the variational bound by changing W^2 from the value it initially had when the second hidden layer was created increases the log-likelihood because the bound is initially tight. Further changes to W^2 that increase the variational bound further are not guaranteed to increase the log-likelihood further, but they are guaranteed to keep it above the value it had when W^2 was created. When learning deeper layers, the variational bound does not start off being tight so even the initial improvement in the bound when the deepest weights are first modified is not guaranteed to increase the log-likelihood.

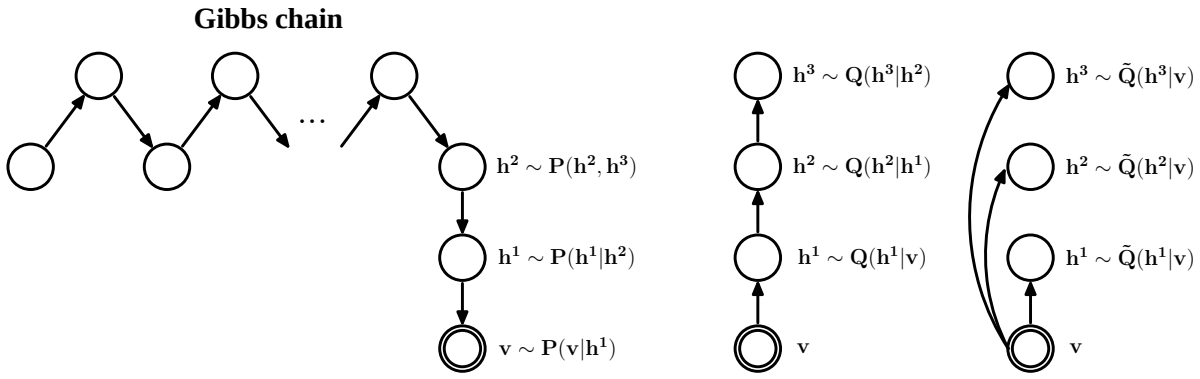

Figure 2.4: **Left:** Generating a sample from the Deep Belief Network. **Right:** Generating a sample from approximate posterior $Q(\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3|\mathbf{v})$ vs. generating a sample from fully factorized approximate posterior $\tilde{Q}(\mathbf{h}^1|\mathbf{v})\tilde{Q}(\mathbf{h}^2|\mathbf{v})\tilde{Q}(\mathbf{h}^3|\mathbf{v})$.

Algorithm 2 Modified Recursive Greedy Learning Procedure for the DBN.

1: Fit parameters W^1 of the 1^{st} layer RBM to data.
2: Freeze the parameter vector W^1 and use samples $\mathbf{h}^1$ from $\tilde{Q}(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v}, W^1)$ as the data for training the next layer of binary features with an RBM.
3: Freeze the parameters W^2 that define the 2^{nd} layer of features and use the samples $\mathbf{h}^2$ from $\tilde{Q}(\mathbf{h}^2|\mathbf{v})$ as the data for training the 3^{rd} layer of binary features.
4: Proceed recursively for the next layers.

although changing W^3 to improve the bound can decrease the actual log-likelihood. This greedy, layer-by-layer training can be repeated several times to learn a deep, hierarchical model. The procedure is summarized in Algorithm 1.

After training a DBN with L layers, the model's joint distribution and its approximate posterior distribution Q are given by:

$$\begin{aligned} P(\mathbf{v}, \mathbf{h}^1, ..., \mathbf{h}^L) &= P(\mathbf{v}|\mathbf{h}^1)...P(\mathbf{h}^{L-2}|\mathbf{h}^{L-1})P(\mathbf{h}^{L-1}, \mathbf{h}^L), \\ Q(\mathbf{h}^1, ..., \mathbf{h}^L|\mathbf{v}) &= Q(\mathbf{h}^1|\mathbf{v})Q(\mathbf{h}^2|\mathbf{h}^1)...Q(\mathbf{h}^L|\mathbf{h}^{L-1}). \end{aligned}$$

To generate an approximate sample from the Deep Belief Network, we can run a prolonged alternating Gibbs sampler (Eq. 2.6) to generate an approximate sample $\mathbf{h}^{L-1}$ from $P(\mathbf{h}^{L-1}, \mathbf{h}^L)$, defined by the top-level RBM, followed by a "top-down" pass through the sigmoid belief network by stochastically activating each lower layer in turn (see Fig. 2.4, left panel). To get an exact sample from the approximate posterior distribution Q, we can simply perform a "bottom-up" pass by stochastically activating each higher layer in turn. The marginal distribution of the top-level hidden units of our approximate posterior $Q(\mathbf{h}^L|\mathbf{v})$ will be non-factorial and, in general, could be multimodal. However, for many practical applications (e.g. information retrieval) having an explicit form for $Q(\mathbf{h}^L|\mathbf{v})$, which allows efficient approximate inference, can be of crucial importance. One possible alternative is to choose the following fully factorized approximating distribution $\tilde{Q}$:

$$\tilde{Q}(\mathbf{h}^1, ..., \mathbf{h}^L|\mathbf{v}) = \prod_{l=1}^{L} \tilde{Q}(\mathbf{h}^l|\mathbf{v}), \tag{2.19}$$

where we define:

$$\tilde{Q}(\mathbf{h}^1|\mathbf{v}) = \prod_j q(h_j^1|\mathbf{v}), \quad q(h_j^1 = 1|\mathbf{v}) = g\left(\sum_i W_{ij}^1 v_i + a_j^1\right), \quad \text{and} \tag{2.20}$$

$$\tilde{Q}(\mathbf{h}^l|\mathbf{v}) = \prod_j q(h_j^l|\mathbf{v}), \quad q(h_j^l = 1|\mathbf{v}) = g\left(\sum_i W_{ij}^l q(h_i^{l-1} = 1|\mathbf{v}) + a_j^l\right), \tag{2.21}$$

where $g(x) = 1/(1 + \exp(-x))$ and $l = 2, .., L$. The factorial posterior $\tilde{Q}(\mathbf{h}^L|\mathbf{v})$ can be obtained by simply replacing the stochastic hidden units in bottom layers with real-valued probabilities, and then performing a single deterministic bottom-up pass to compute $q(h_j^L = 1|\mathbf{v})$. This fully factorized approximation also suggests a modified greedy learning algorithm, summarized in Algorithm 2. In this algorithm the samples, used for training higher-level RBM's, are instead sampled from a fully factorized approximate posterior $\tilde{Q}$. It is important to observe that the modified algorithm *does not guarantee* to improve the lower bound on the log-probability of the training data. Nonetheless, this is the actual algorithm commonly used in practice (Taylor et al. [2006], Hinton and Salakhutdinov [2006], Torralba et al. [2008], Bengio [2009]), and we will use it in the next chapter of this thesis. The modified algorithm performs well, particularly when a fully factorized $\tilde{Q}$ is used to perform approximate inference in the final model. Details of Matlab implementation of the modified greedy learning algorithm can be found in Appendix A.

In practice, however, many of the assumptions that we have to make in order to guarantee the improvement of the lower bound on the data likelihood are violated. In particular, the assumption that learning higher-level RBM's can be carried out using maximum likelihood (see Eq. 2.18) is clearly violated. Furthermore, when adding a new layer l, we typically do not initialize $W^l = W^{l-1\top}$, which would force the number of hidden units of the new RBM to be the same as the number of the visible units of the lower level RBM[3]. In chapter 4 of this thesis, we will address a problem of evaluating generalization performance of Deep Belief Networks as density models, which will allow us to do model selection and complexity control.

2.3 Generalizing RBM's to Modeling Real-valued and Count Data

Welling et al. [2005] introduced a class of two-layer undirected graphical models that generalize RBM's to exponential family distributions. In the remaining part of this section, we will review two specific models: Gaussian RBM and Replicated Softmax model (Salakhutdinov and Hinton [2010]). These models will allow us to model real-valued data (e.g. image patches) and count data (e.g. word-count vectors of documents), when learning DBN's. Other extensions include exponential or truncated exponential RBM's (Bengio et al. [2007]), and Poisson RBM's (Gehler et al. [2006])

Gaussian RBM's

Consider modeling visible real-valued units $\mathbf{v} \in \mathbb{R}^D$ and let $\mathbf{h} \in \{0, 1\}^F$ be binary stochastic hidden units. The energy of the state $\{\mathbf{v}, \mathbf{h}\}$ of the Gaussian RBM is defined as follows:

$$E(\mathbf{v}, \mathbf{h}; \theta) = \sum_{i=1}^{D} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{D}\sum_{j=1}^{F} W_{ij} h_j \frac{v_i}{\sigma_i} - \sum_{j=1}^{F} a_j h_j, \tag{2.22}$$

[3]Although if the number of hidden units per layer does not decrease, it easy to show (Hinton et al. [2006]) that adding each new layer guarantees to increase a lower bound on the data likelihood, provided higher-level RBM's are trained by maximum likelihood.

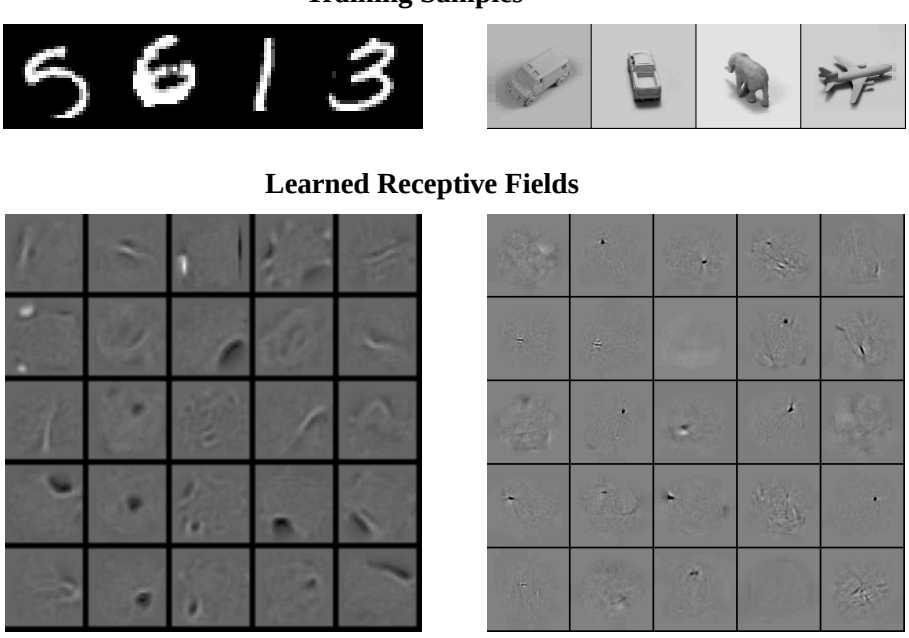

Figure 2.5: Random subsets of the learned receptive fields. **Left:** The binary RBM trained on the MNIST dataset (resolution is 28×28). **Right:** The Gaussian RBM trained on the NORB dataset (resolution is 96×96). Each square displays the incoming weights from all the visible units into one hidden unit. White encodes a positive weight and black encodes a negative weight on the scale of -3 to 3.

where $\theta = \{W, \mathbf{a}, \mathbf{b}, \sigma^2\}$ are the model parameters. The marginal distribution over the visible vector $\mathbf{v}$ takes form:

$$P(\mathbf{v};\theta) \;=\; \sum_{\mathbf{h}} \frac{\exp\left(-E(\mathbf{v},\mathbf{h};\theta)\right)}{\int_{\mathbf{v}'}\sum_{\mathbf{h}}\exp\left(-E(\mathbf{v},\mathbf{h};\theta)\right)d\mathbf{v}'}. \tag{2.23}$$

From Eq. 2.22, it is straightforward to derive the following conditional distributions:

$$p(v_i = x|\mathbf{h}) \;=\; \frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(-\frac{\left(x - b_i - \sigma_i\sum_j h_j W_{ij}\right)^2}{2\sigma_i^2}\right), \tag{2.24}$$

$$p(h_j = 1|\mathbf{v}) \;=\; g\left(b_j + \sum_i W_{ij}\frac{v_i}{\sigma_i}\right), \tag{2.25}$$

where $g(x) = 1/(1+\exp(-x))$ is the logistic function. Observe that conditioned on the states of the hidden units (Eq. 2.24), each visible unit is modeled by a Gaussian distribution, whose mean is shifted by the weighted combination of the hidden unit activations. The derivative of the log-likelihood with respect to W takes form:

$$\frac{\partial \log P(\mathbf{v};\theta)}{\partial W_{ij}} \;=\; \mathrm{E}_{P_{\mathrm{data}}}\left[\frac{1}{\sigma_i}v_i h_j\right] - \mathrm{E}_{P_{\mathrm{Model}}}\left[\frac{1}{\sigma_i}v_i h_j\right].$$

As described in section 2.1, learning of the model parameters, including the variance σ^2, can be carried out using Contrastive Divergence. In practice, however, instead of learning σ^2, one would typically use a fixed, predetermined value for σ^2 (Nair and Hinton [2009], Hinton and Salakhutdinov [2006]).

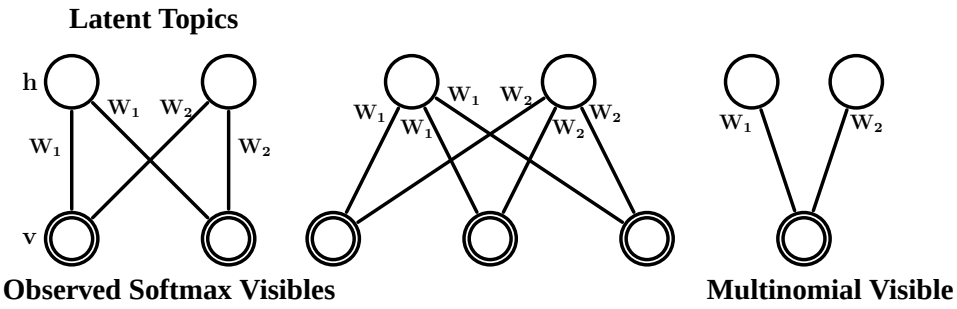

Figure 2.6: The Replicated Softmax model. The top layer represents a vector $\mathbf{h}$ of stochastic, binary topic features and the bottom layer consists of softmax visible units, $\mathbf{v}$. All visible units share the same set of weights, connecting them to the binary hidden units. **Left and Middle:** Two members of a Replicated Softmax family for documents containing two and three words. **Right:** A different interpretation of the Replicated Softmax model, in which M softmax units with identical weights are replaced by a single multinomial unit which is sampled M times.

To see what a single RBM module can learn, Fig. 2.5 shows a random subset of parameters W, also known as receptive fields, learned by a standard binary and a Gaussian RBM using CD1. Observe that both RBM's learn highly localized receptive fields.

Modeling Word Counts with a Family of Replicated Softmax Models

Consider an undirected graphical model that consists of one visible layer and one hidden layer as shown in Fig. 2.6. This model is a type of Restricted Boltzmann Machine in which the visible units that are usually binary have been replaced by "softmax" units that can have one of a number of different states. Let $\mathbf{v} \in \{1, ..., K\}^D$ be a vector of visible units that takes on values in some discrete alphabet, and let $\mathbf{h} \in \{0, 1\}^F$ be binary stochastic hidden topic features. Let $\mathbf{V}$ be a $K \times D$ observed indicator matrix with $v_i^k = 1$ if visible unit i takes on value k. The energy of the state $\{\mathbf{V}, \mathbf{h}\}$ is defined as follows:

$$E(\mathbf{V}, \mathbf{h}) = -\sum_{i=1}^{D}\sum_{j=1}^{F}\sum_{k=1}^{K} W_{ij}^k h_j v_i^k - \sum_{i=1}^{D}\sum_{k=1}^{K} v_i^k b_i^k - \sum_{j=1}^{F} h_j b_j, \tag{2.26}$$

where W_{ij}^k is a symmetric interaction term between visible unit i that takes on value k, and hidden unit j, b_i^k is the bias of unit i that takes on value k, and a_j is the bias of hidden unit j. The conditional distributions are given by softmax and logistic functions:

$$p(v_i^k = 1|\mathbf{h}) = \frac{\exp\left(b_i^k + \sum_{j=1}^{F} h_j W_{ij}^k\right)}{\sum_{q=1}^{K} \exp\left(b_i^q + \sum_{j=1}^{F} h_j W_{ij}^q\right)} \tag{2.27}$$

$$p(h_j = 1|\mathbf{V}) = g\left(a_j + \sum_{i=1}^{D}\sum_{k=1}^{K} v_i^k W_{ij}^k\right). \tag{2.28}$$

Now suppose that for each document we create a separate RBM with as many softmax units as there are words in the document. Assuming we can ignore the order of the words, all of these softmax units can share the same set of weights. Consider a document that contains M words. In this case, we define the energy of the state $\{\mathbf{V}, \mathbf{h}\}$ to be:

$$E(\mathbf{V}, \mathbf{h}) = -\sum_{j=1}^{F}\sum_{k=1}^{K} W_j^k h_j \hat{v}^k - \sum_{k=1}^{K} \hat{v}^k b^k - M\sum_{j=1}^{F} h_j b_j, \tag{2.29}$$

where $\hat{v}^k = \sum_{i=1}^{M} v_i^k$ denotes the count for the k^{th} word. The bias terms of the hidden units are scaled up by the length of the document. This scaling is crucial and allows hidden units to behave sensibly when dealing with documents of different lengths. We also note that using M softmax units with identical weights is equivalent to having one multinomial unit which is sampled M times, as shown in Fig. 2.6. The derivative of the log-likelihood with respect to parameters W takes form:

$$\frac{\partial \log P(\mathbf{V};\theta)}{\partial W_j^k} = \mathrm{E}_{P_{\mathrm{data}}}\left[\hat{v}^k h_j\right] - \mathrm{E}_{P_{\mathrm{Model}}}\left[\hat{v}^k h_j\right].$$

The weights can now be shared by the whole family of different RBM's that are created for documents of different lengths. We call this the "Replicated Softmax" model]. Learning can be performed using Contrastive Divergence.

Chapter 3

Learning Feature Hierarchies with Deep Belief Networks

This chapter presents several ideas based on greedily learning a hierarchy of features from high-dimensional, richly structured sensory input. Through extensive empirical evaluations we will attempt to address the question of how well Deep Belief Networks perform in various application domains. All of the presented ideas will exploit the following two key properties of DBN's. First, they can be learned efficiently from large amounts of unlabeled data. Second, they can be discriminatively fine-tuned using the standard backpropagation algorithm.

In section 3.1 we show how a Deep Belief Network can be used to extract useful feature representations that would allow us to learn a good covariance kernel for a Gaussian process. In particular, if the input data is high-dimensional and highly-structured, a Gaussian kernel applied to the top layer of extracted features in the DBN works much better than a similar kernel applied to the raw input. In sections 3.2 and 3.3 we show how the greedy learning algorithm can be used to make nonlinear autoencoders work considerably better compared to widely used methods, such as principal component analysis (PCA) and singular value decomposition (SVD). We then demonstrate that these deep autoencoders can be used to discover binary "semantic" codes that allow fast and accurate information retrieval. Finally, in section 3.4 we show how the DBN framework, using partially labeled data, can also be used to efficiently learn a nonlinear transformation from the input space to a low-dimensional feature space in which K-nearest neighbour classification performs well.

3.1 Learning Features for Discrimination and Regression

Many real-world applications are characterized by high-dimensional, highly-structured data with a large supply of unlabeled data and a very limited amount of labeled data. Applications such as information retrieval and machine vision are examples where unlabeled data is readily available. Many models, including logistic regression, Gaussian processes, and Support Vector Machines, are discriminative models by nature, and within the standard regression or classification scenario, unlabeled data is of no use. Given a set of *i.i.d.* labeled input vectors $\mathbf{X}_l = \{\mathbf{x}_n\}_{n=1}^N$ and their associated target labels $\{y_n\}_{n=1}^N \in \mathbb{R}$ for regression or $\{y_n\}_{n=1}^N \in \{-1, 1\}$ for classification, discriminative methods model $p(y_n|\mathbf{x}_n)$ directly. Unless some assumptions are made about the underlying distribution of the input data $\mathbf{X} = [\mathbf{X}_l, \mathbf{X}_u]$, unlabeled data, $\mathbf{X}_u$, cannot be used. Many researchers have tried to use unlabeled data by incorporating a model of $P(\mathbf{X})$. For classification tasks, Lawrence and Schölkopf [2001] model $P(\mathbf{X})$ as a mixture $\sum_{y_n} p(x_n|y_n)p(y_n)$ and then infer $p(y_n|x_n)$, Seeger [2001] attempts to learn a covariance kernel for a Gaussian process based on $P(\mathbf{X})$, and Lawrence and Jordan [2004] assume that

the decision boundaries should occur in regions where the data density, $P(\mathbf{X})$, is low. When faced with high-dimensional, highly-structured data, however, none of the existing approaches have proved to be particularly successful.

To make use of unlabeled data, we propose to first learn a DBN model of $P(\mathbf{X})$ in an entirely unsupervised way using the fast, greedy learning algorithm introduced in section 2.2. We then use this deep generative model to initialize a multilayer, nonlinear mapping $F(\mathbf{x}; W)$, parameterized by W, with $F : \mathbf{X} \rightarrow \mathbf{Z}$ mapping the input vectors in $\mathbf{X}$ into a feature space $\mathbf{Z}$. The top-level features produced by this mapping typically allow for a rather accurate reconstruction of the input and tend to capture a lot of the higher-order structure in the input data. We can now fit a discriminative model to the labeled data using the top-level features of the DBN model as inputs. Performance can be further improved by using backpropagation through the DBN to discriminatively fine-tune the model parameters.

While greedily pretrained DBN's can be used to provide input vectors for many discriminative methods, including logistic regression, SVM's (Vapnik [1998], Lauer et al. [2007]), and kernel regression (Benedetti [1977]), in this section we will concentrate on using a Deep Belief Network to learn a covariance kernel for a Gaussian process. In particular, we show that the parameters W of the covariance kernel can be fine-tuned using the labeled data by maximizing the log probability of the labels with respect to W.

3.1.1 Gaussian Processes for Regression and Binary Classification

Gaussian processes (GP's) are a widely used method for Bayesian nonlinear non-parametric regression and classification (Rasmussen and Williams [2006], Seeger [2004], Neal [1997], Rasmussen [1996]). GP's are based on defining a covariance function that encodes prior knowledge of the smoothness of the underlying process that is being modeled. Because of their flexibility and computational simplicity, GP's have been successfully used in many areas of machine learning.

Let us consider the following regression task. We are given a dataset of N $i.i.d.$ labeled input vectors $\mathbf{X}_l = \{\mathbf{x}_n\}_{n=1}^N$ and their corresponding real-valued targets $\mathbf{y} = \{y_n\}_{n=1}^N$. We are interested in the following probabilistic regression model:

$$y_n \;=\; f(\mathbf{x}_n) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2), \tag{3.1}$$

where $\mathcal{N}(\mu, \sigma^2)$ denotes a Gaussian distribution with mean μ and variance σ^2. A Gaussian process regression places a zero-mean GP prior over the underlying latent function $\mathbf{f}$ we are modeling, so that a-priori $\mathbf{f}|\mathbf{X}_l \sim \mathcal{N}(0, K)$, where $\mathbf{f} = [f(\mathbf{x}_1), ..., f(\mathbf{x}_n)]^T$ and K is the covariance matrix, whose entries are specified by the covariance function $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. The covariance function encodes our prior notion of the smoothness of $\mathbf{f}$, or the prior assumption that if two input vectors are similar according to some distance measure, their labels should be highly correlated. In this work we will use the spherical Gaussian kernel, parameterized by $\{\alpha, \beta\}$:

$$K_{ij} \;=\; \alpha \exp\left(-\frac{1}{2\beta}(\mathbf{x}_i - \mathbf{x}_j)^\top (\mathbf{x}_i - \mathbf{x}_j)\right). \tag{3.2}$$

Integrating out the function values $\mathbf{f}$, the marginal log-likelihood takes form:

$$L \;=\; \log P(\mathbf{y}|\mathbf{X}_l; \theta) \;=\; -\frac{N}{2}\log 2\pi - \frac{1}{2}\log|K + \sigma^2 I| - \frac{1}{2}\mathbf{y}^\top (K + \sigma^2 I)^{-1}\mathbf{y}, \tag{3.3}$$

which can then be maximized with respect to the parameters $\theta = \{\alpha, \beta, \sigma\}$. Given a new test point $\mathbf{x}^*$, a prediction is obtained by conditioning on the observed data and θ. The distribution of the predicted value y^* at $\mathbf{x}^*$ takes the form:

$$y^*|\mathbf{x}^*, \mathbf{X}_l, \mathbf{y}; \theta \;\sim\; \mathcal{N}\left(\mathbf{k}^{*\top}(K + \sigma^2 I)^{-1}\mathbf{y}, k^{**} - \mathbf{k}^{*\top}(K + \sigma^2 I)^{-1}\mathbf{k}^* + \sigma^2\right), \tag{3.4}$$

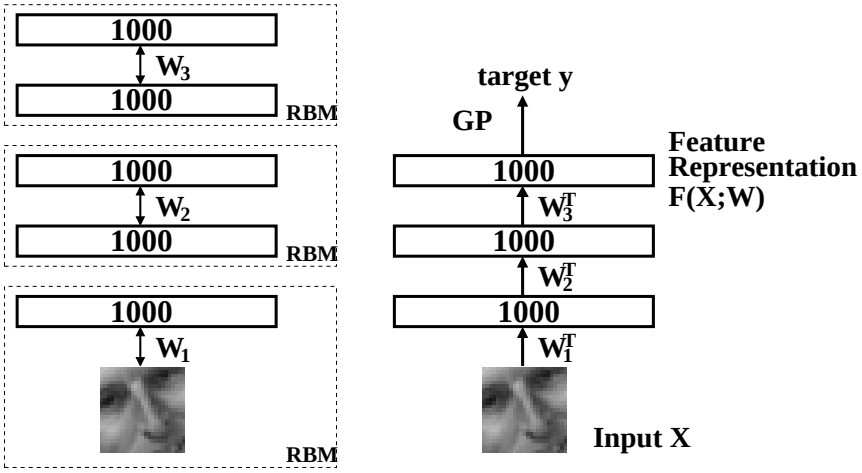

Figure 3.1: **Left:** Pretraining consists of learning a stack of RBM's. **Right:** After pretraining, the RBM's are used to initialize a covariance function of the Gaussian process, which is then fine-tuned by backpropagation.

where $k^{**} = K(\mathbf{x}^*, \mathbf{x}^*)$ and $\mathbf{k}^* = K(\mathbf{x}^*, \mathbf{X}_l)$ is the $N \times 1$ vector of the covariances evaluated between all training and a test point.

For a binary classification task, we similarly place a zero mean GP prior over the values of an underlying latent function, $\mathbf{f}$, which are then passed through the logistic function $g(x) = 1/(1+\exp(-x))$ to define a prior $p(y_n = 1|\mathbf{x}_n) = g(f(\mathbf{x}_n))$. Given a new test point $\mathbf{x}^*$, inference is done by first obtaining the distribution over $f^* = f(\mathbf{x}^*)$:

$$p(f^*|x^*, \mathbf{X}_l, \mathbf{y}; \theta) \;=\; \int p(f^*|\mathbf{x}^*, \mathbf{X}_l, \mathbf{f}; \theta) P(\mathbf{f}|\mathbf{X}_l, \mathbf{y}; \theta) d\mathbf{f}, \tag{3.5}$$

which is then used to produce a probabilistic prediction:

$$p(y^* = 1|\mathbf{x}^*, \mathbf{X}_l, \mathbf{y}; \theta) \;=\; \int g(f^*) p(f^*|\mathbf{x}^*, \mathbf{X}_l, \mathbf{y}; \theta) df^*. \tag{3.6}$$

The non-Gaussian likelihood makes the integral in Eq. 3.5 analytically intractable. In our experiments, we approximate the non-Gaussian posterior $P(\mathbf{f}|\mathbf{X}_l, \mathbf{y}; \theta)$ with a Gaussian one using expectation propagation (Minka [2001]). For more thorough reviews and implementation details refer to Rasmussen and Williams [2006], Seeger [2004], and Neal [1997].

3.1.2 Learning the Covariance Function for a Gaussian Process

Using the layer-by-layer learning algorithm of section 2.2, we first learn a stack of RBM's. After learning is complete, the stochastic activities of the binary units in each layer are replaced by deterministic, real-valued probabilities and the DBN is used to initialize a multilayer, nonlinear mapping $F(\mathbf{x}; W)$ as shown in Fig. 3.1. This learning is treated as a *pretraining* stage that captures a lot of the higher-order structure in the input data and is used to define a Gaussian covariance function, parameterized by $\{\alpha, \beta\}$ and W:

$$K_{ij} \;=\; \alpha \exp\left(-\frac{1}{2\beta}\big(F(\mathbf{x}_i; W) - F(\mathbf{x}_j; W)\big)^\top \big(F(\mathbf{x}_i; W) - F(\mathbf{x}_j; W)\big)\right). \tag{3.7}$$

The covariance kernel is initialized in an entirely unsupervised way. We can now maximize the marginal log-likelihood of Eq. 3.3 with respect to the parameters of the covariance kernel $\{\alpha, \beta, W\}$ and observation noise σ^2, using the labeled training data (Rasmussen and Williams [2006], Lawrence [2004]).

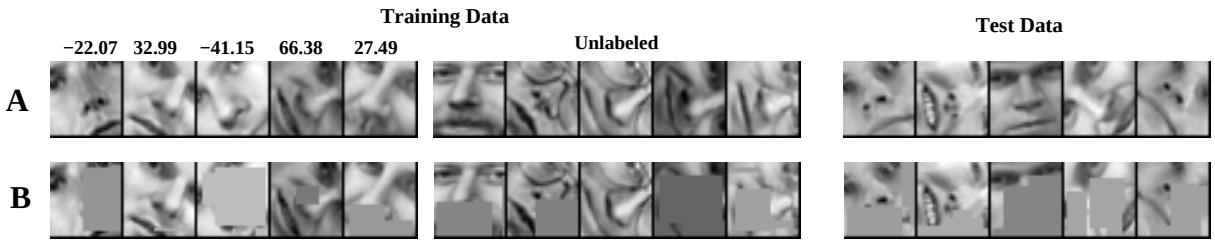

Figure 3.2: **Top A:** Randomly sampled examples of the training and test data. **Bottom B:** The same sample of the training and test images but with rectangular occlusions.

The partial derivatives of the marginal log-likelihood with respect to the parameters takes the form:

$$\frac{\partial L}{\partial \theta_i} = \frac{1}{2}\mathrm{tr}\left(\left(K_y^{-1}\mathbf{y}\mathbf{y}^\top K_y^{-1} - K_y^{-1}\right)\frac{\partial K_y}{\partial \theta_i}\right), \tag{3.8}$$

where $K_y = K + \sigma^2 I$ and $\theta = \{\alpha, \beta, W, \sigma^2\}$. Using the chain rule, the gradient $\partial K_y / \partial W$ is computed using standard backpropagation algorithm. It is necessary to compute the inverse of K_y, so each gradient evaluation has $O(N^3)$ complexity where N is the number of the labeled training cases. However, when learning the stack of Restricted Boltzmann Machines that are composed to form the initial DBN, each gradient evaluation scales linearly in time and space with the number of unlabeled training cases. Therefore the pretraining stage can make efficient use of very large sets of unlabeled data to build high-level features. The small amount of labeled data can then be used to only slightly refine those features.

3.1.3 Experimental results

We present several experimental results on three publicly available datasets: the MNIST dataset, the Olivetti face dataset, and the Reuters (RCV1-v2) dataset. Our first task is to extract the orientation of a face from a gray-level image of a large patch of the face. The second task is to discriminate between images of odd digits and images of even digits. The third task is to discriminate between two different classes of newswire story based on the vector of word counts in each story.

In all of experiments, when training higher-level RBM's, the visible units were set to the activation probabilities of the hidden units in the lower-level RBM, but the hidden units of every RBM had stochastic binary values. For the fine-tuning stage, we used the method of conjugate gradients. Details of pretraining and fine-tuning, along with the detailed description of the used datasets, can be found in Appendix A.

Extracting the Orientation of a Face Patch

The Olivetti face dataset contains ten 64×64 images of each of forty different people. We constructed a dataset of 13,000 25×25 images by rotating (-90° to $+90^\circ$), cropping, and subsampling the original 400 images. The intensities in the cropped images were normalized to have zero mean and the entire dataset was then scaled by a single number to make the average pixel variance be 1. The dataset was then subdivided into 12,000 training images, which contained the first 30 people, and 1,000 test images, which contained the remaining 10 people. 1,000 randomly sampled face patches from the training set were assigned an orientation label. The remaining 11,000 training images were used as unlabeled data. We also made a more difficult version of the task by occluding part of each face patch with randomly chosen rectangles. Figure 3.2 shows randomly sampled examples from the training and test data.

For training on the Olivetti face patches we used the 784-1000-1000-1000 architecture shown in Fig. 3.1. When pretraining the first layer, the real-valued pixel intensities were modeled by a Gaussian

	Training labels	**GPstandard** Sph.	ARD	**GP-DBNgreedy** Sph.	ARD	**GP-DBNfine** Sph.	ARD	**GPpca** Sph.	ARD
A	100	22.24	28.57	17.94	18.37	15.28	**15.01**	18.13 (10)	16.47 (10)
	500	17.25	18.16	12.71	8.96	7.25	**6.84**	14.75 (20)	10.53 (80)
	1000	16.33	16.36	11.22	8.77	6.42	**6.31**	14.86 (20)	10.00 (160)
B	100	26.94	28.32	23.15	19.42	19.75	**18.59**	25.91 (10)	19.27 (20)
	500	20.20	21.06	15.16	11.01	10.56	**10.12**	17.67 (10)	14.11 (20)
	1000	19.20	17.98	14.15	10.43	**9.13**	9.23	16.26 (10)	11.55 (80)

Table 3.1: Performance results on the face-orientation regression task. The root mean squared error (RMSE) on the test set is shown for each method using a spherical Gaussian kernel and a Gaussian kernel with ARD hyper-parameters. **By row:** A) Non-occluded face data, B) Occluded face data. For the GPpca model, the number of principal components that performs best on the test data is shown in parenthesis.

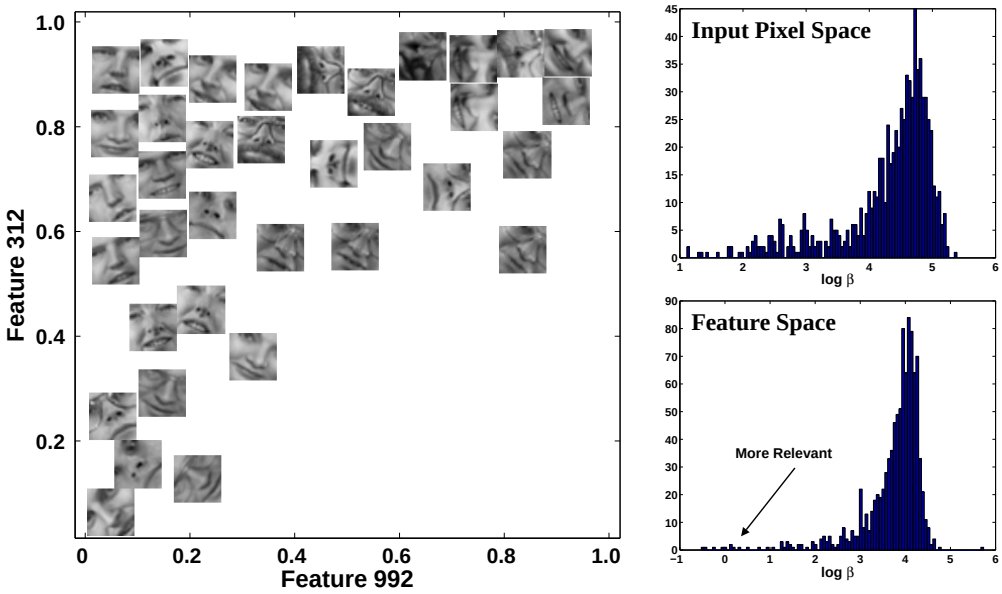

Figure 3.3: **Left:** A scatter plot of the two most relevant features, with each point replaced by the corresponding input test image. For better visualization, overlapped images are not shown. **Right:** The histogram plots of the learned ARD hyper-parameters $\log \beta$.

RBM with unit variance. The remaining RBM's in the stack used logistic units. The entire training set of 12,000 unlabeled images was used for greedy, layer-by-layer training of a Deep Belief Network. The model contained about 2.8 million parameters, which may seem excessive for 12,000 training cases. However, each training case involves modeling 625 real-valued pixels rather than just a single real-valued target label.

After the DBN has been pretrained on the unlabeled data, a GP model was fitted to the labeled data using the top-level features of the DBN model as inputs. We call this model **GP-DBNgreedy**. GP-DBNgreedy can be further fine-tuned by maximizing the marginal log probability of the labels with respect to W using backpropagation algorithm. We call this model **GP-DBNfine**. For comparison, we fitted a GP model that used the pixel intensities of only the labeled images as its inputs. We call this model **GPstandard**. We also used PCA[1] to reduce the dimensionality of the labeled images and fitted several different GP models using the projections onto the first m principal components as the input.

[1]Principal components were extracted using all 12,000 training cases.

Train labels	GPstandard Sph.	GPstandard ARD	GP-DBNgreedy Sph.	GP-DBNgreedy ARD	GP-DBNfine Sph.	GP-DBNfine ARD	GPpca Sph.	GPpca ARD
100	0.0884	0.1087	0.0528	0.0597	**0.0501**	0.0599	0.0785 (10)	0.0920 (10)
500	0.0222	0.0541	0.0100	0.0161	**0.0055**	0.0104	0.0160 (40)	0.0235 (20)
1000	0.0129	0.0385	0.0058	0.0059	**0.0050**	0.0100	0.0091 (40)	0.0127 (40)

Table 3.2: Performance results on discriminating odd vs. even digits classification task, using the area under the ROC (AUROC) metric. For each method we show 1-AUROC on the test set. An AUROC of 0.5 corresponds to the classifier that makes random predictions. All methods were tried using both a spherical Gaussian kernel and a Gaussian kernel with ARD hyper-parameters. For the GPpca model, the number of principal components that performs best on the test data is shown in parenthesis.

Since we are only interested in a lower bound on the error of this model, we simply use the value of m that performs best on the *test* data. We call this model **GPpca**. Table 3.1 shows the root mean squared error (RMSE) of the predicted face orientations using all four types of GP models with varying amounts of labeled data. The results show that both GP-DBNgreedy and GP-DBNfine significantly outperform regular GP and GPpca models. Indeed, GP-DBNfine with only 100 labeled training cases outperforms GPstandard with 1000 labeled training cases.

To test the robustness of our approach to noise in the input we took the same dataset and created artificial rectangular occlusions (see Fig. 3.2, panel B). The number of rectangles per image was drawn from a Poisson with $\lambda = 2$. The top-left location, length and width of each rectangle was sampled from a uniform [0,25]. The pixel intensity of each occluding rectangle was set to the mean pixel intensity of the entire image. Table 3.1 shows that the performance of all models degrades, but their relative performances remain the same. GP-DBNfine on occluded data is still much better than GPstandard on non-occluded data.

We have also experimented with using a Gaussian kernel with ARD hyper-parameters (Rasmussen and Williams [2006]), which is a common practice when the input vectors are high-dimensional:

$$K_{ij} \;=\; \alpha \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{D}(\mathbf{x}_i - \mathbf{x}_j)\right), \tag{3.9}$$

where $\mathbf{D}$ is the diagonal matrix with $\mathbf{D}_{ii} = 1/\beta_i$, so that the covariance function has a separate length-scale parameter for each dimension. ARD hyper-parameters were optimized by maximizing the marginal log-likelihood of Eq. 3.3. Table 3.1 shows that ARD hyper-parameters do not improve GPstandard, but they do slightly improve GP-DBNfine and they strongly improve GP-DBNgreedy and GPpca when there are 500 or 1000 labeled training cases.

The histogram plot of $\log \beta$ in Fig. 3.3 reveals that there are a few extracted features that are very relevant (small β) to our prediction task. The same figure, left panel, shows a scatter plot of the two most relevant features of GP-DBNgreedy model, with each point replaced by the corresponding input test image. Clearly, these two features carry a lot of information about the orientation of the face. We suspect that the GP-DBNfine model does not benefit as much from the ARD hyper-parameters because the fine-tuning stage is already capable of turning down the activities of irrelevant top-level features.

Discriminating between Images of Odd and Even Digits

The MNIST digit dataset contains 60,000 training and 10,000 test 28×28 images of ten handwritten digits (0 to 9). 1000 randomly sampled training images were categorized into an even or an odd class. The remaining 59,000 training images were used as unlabeled data. As in the previous experiment, we used the 784-1000-1000-1000 architecture with the entire training set of 60,000 unlabeled digits used

Number of labeled cases (50% in each class)	**GPstandard**	**GP-DBNgreedy**	**GP-DBNfine**
100	0.1295	0.1180	**0.0995**
500	0.0875	0.0793	**0.0609**
1000	0.0645	0.0580	**0.0458**

Table 3.3: Performance results using the area under the ROC (AUROC) metric on the text classification task. For each method we show 1-AUROC on the test set.

for greedily pretraining the DBN model. Table 3.2 shows the area under the ROC curve for discriminating between odd and even digits. GP-DBNfine and GP-DBNgreedy perform considerably better than GPstandard both with and without ARD hyper-parameters.

Classifying News Stories

The Reuters RCV1-v2 dataset is an archive of 804,414 newswire stories. The corpus covers four major groups: Corporate/Industrial, Economics, Government/Social, and Markets. The data was randomly split into 802,414 training and 2000 test articles. The test set contains 500 articles of each major group. The available data was already in a convenient, preprocessed format, where common stopwords were removed and all the remaining words were stemmed. We only made use of the 2000 most frequently used word stems in the training data. As a result, each document was represented as a vector containing 2000 word counts. No other preprocessing was done.

For the text classification task we used a 2000-1000-1000-1000 architecture. The entire unlabeled training set of 802,414 articles was used for learning a multilayer generative model of the text documents. The bottom layer of the DBN was trained using a Replicated Softmax model (see section 2.3). Table 3.3 shows the area under the ROC curve for classifying documents belonging to the Corporate/Industrial vs. Economics groups. As expected, GP-DBNfine and GP-DBNgreedy work better than GPstandard. The results of binary discrimination between other pairs of document classes are very similar to the results presented in table 3.3. Our experiments using a Gaussian kernel with ARD hyper-parameters did not show any significant improvements. Examining the histograms of the length-scale parameters β, we found that most of the input word-counts as well as most of the extracted features were relevant to the classification task.

3.1.4 Discussion

We have shown how to greedily pretrain and discriminatively fine-tune a covariance kernel for a Gaussian process. For high-dimensional, highly-structured input, this is a very effective way to make use of large unlabeled datasets, especially when labeled training data is scarce. The performance of pretrained and fine-tuned GP models further reveals that the learned high-level feature representations capture a lot of structure in the unlabeled input data, which is useful for subsequent classification or regression tasks, even though these tasks are unknown when the deep generative model is being trained.

The same framework can also be used to discover useful low-dimensional representations of high-dimensional data, which can be used for exploratory data analysis, preprocessing, and data visualization. We explore this idea in the next section.

3.2 Nonlinear Dimensionality Reduction

Scientists working with large amounts of high-dimensional data are constantly facing the problem of dimensionality reduction: how to discover low-dimensional structure from high-dimensional observa-

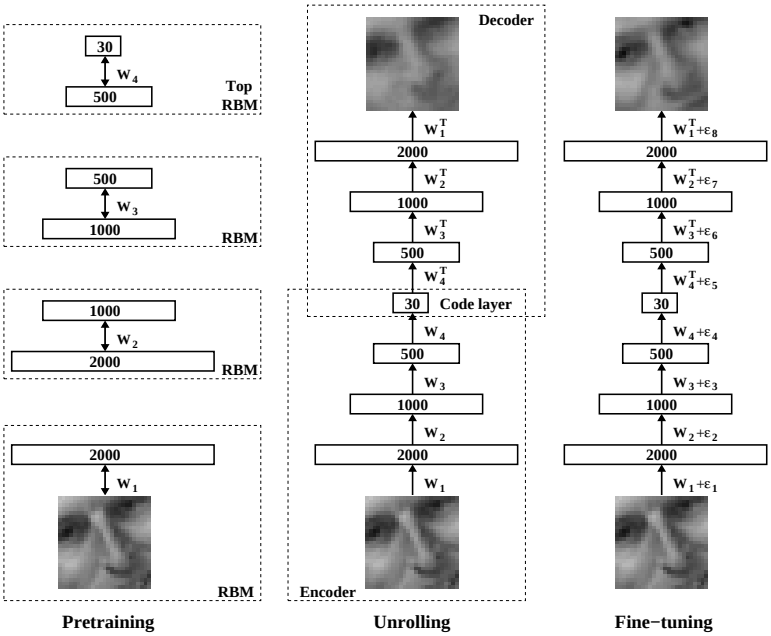

Figure 3.4: Pretraining consists of learning a stack of Restricted Boltzmann Machines each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBM's are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

tions. There exist a variety of dimensionality reduction techniques, which can be broadly classified into: linear methods, such as principal component analysis (PCA), nonlinear mappings, such as autoencoders (Plaut and Hinton [1987], DeMers and Cottrell [1993]), and proximity based methods, such as Local Linear Embedding (Roweis and Saul [2000]).

Most of the existing algorithms suffer from various drawbacks. If the data lie on an embedded low-dimensional nonlinear manifold, then linear methods, even though computationally efficient, cannot recover this structure as well as their nonlinear counterparts. Proximity based methods are more powerful, but their computational cost scales quadratically with the number of observations, so they generally cannot be applied to very large high-dimensional datasets. Nonlinear mapping algorithms, such as autoencoders, are generally painfully slow to train, and are prone to getting stuck in local minima.

3.2.1 Pretraining Autoencoders

The standard way to train autoencoders is to use backpropagation to reduce the reconstruction error. As we show, it is generally very difficult to optimize nonlinear autoencoders that have multiple hidden layers with hundreds of thousands of parameters (DeMers and Cottrell [1993], Hecht-Nielsen [1995], Larochelle et al. [2009]). This is perhaps the main reason why this potentially powerful dimensionality reduction algorithm has not found its applications in practice. Instead, we will use the greedy learning algorithm to pretrain autoencoders, by learning a stack of RBM's, as shown in Fig. 3.4. The key idea is that the greedy learning algorithm can quickly find parameters that already produce a good data reconstruction model.

After the pretraining stage, which is similar to the construction defined in subsection 3.1.2, the stochastic activities of the binary features in each layer are replaced by deterministic, real-valued probabilities and the model is "unrolled" to produce encoder and decoder networks. Initially both the encoder and decoder networks share the same set of weights. The global fine-tuning stage then slightly refines the weights for optimal reconstruction by using backpropagation of error derivatives through the whole

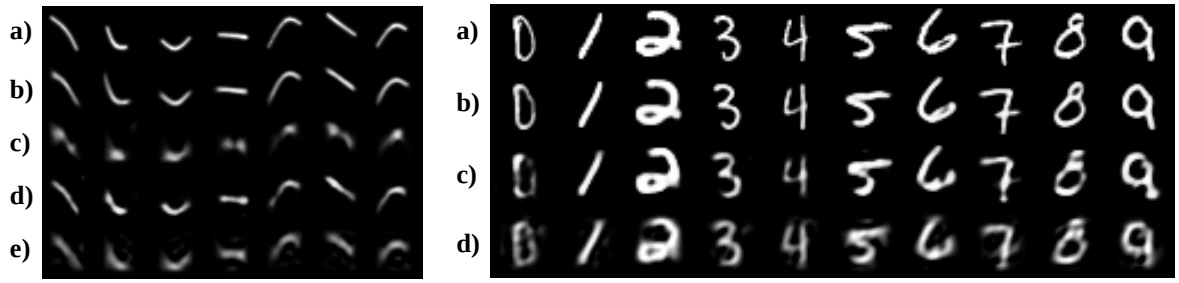

Figure 3.5: **Left Panel (by row): a)** Random samples of curves from the test dataset; **b)** reconstructions produced by the 6-dimensional deep autoencoder; **c)** reconstructions by "logistic PCA" using 6 components; **d)** reconstructions by logistic and **e)** standard PCA using 18 components. The average squared error per image for the last four rows is 1.44, 7.64, 2.45, 5.90. **Right panel (by row): a)** A random MNIST test image from each class; **b)** reconstructions by the 30-dimensional autoencoder; **c)** reconstructions by 30-dimensional logistic PCA and d) standard PCA. The average squared errors for the last three rows are 3.00, 8.01, and 13.87.

autoencoder.

3.2.2 Experimental results

In all our experiments, when pretraining deep autoencoders, the top level RBM was a Gaussian RBM, as described in section 2.3, but with visible and hidden units switched. So the hidden units of the top level RBM were modeled by a unit variance Gaussian distribution, whose mean was determined by the weighted combination of the binary visible units. This allowed the low-dimensional codes to make good use of continuous variables and facilitated comparisons with PCA. Description of the datasets, along with details of software that we used for pretraining and fine-tuning deep autoencoders, can be found in Appendix A.

Synthetic curves dataset

To evaluate the two-stage learning procedure, we first used a synthetic curves dataset that contains 28×28 images of "curves", generated from three randomly chosen 2-dimensional points. For this dataset, the true intrinsic dimensionality of data is six, but the relationship between the pixel intensities and the six numbers used to generate them is highly nonlinear. The pixel intensities were normalized to lie in the interval $[0, 1]$ (see Fig. 3.5, left panel). The intensities had a tendency to mostly take on extreme values, and therefore were modeled in the first layer by a standard binary RBM. During the fine-tuning stage, we minimized the cross-entropy error:

$$E = -\sum_i p_i \log \hat{p}_i - \sum_i (1 - p_i) \log(1 - \hat{p}_i), \tag{3.10}$$

where p_i is the intensity of pixel i and $\hat{p}_i$ is the intensity of its reconstruction.

We used a deep autoencoder that consisted of an encoder with layers of size (28×28)-400-200-100-50-25-6 and a symmetric decoder. This is probably much deeper than is necessary for this task, but one of the points of this experiment was to demonstrate that we could train very deep networks. The 6 units in the code layer were linear and all the other units were logistic. The autoencoder was trained on 20,000 images and tested on 10,000 new images. Figure 3.5 shows that the autoencoder was able to discover the nonlinear mapping between 784 pixel image and 6 real numbers that allow almost perfect reconstruction. PCA, on the other hand, gives considerably worse results. Figure 3.5 also compares

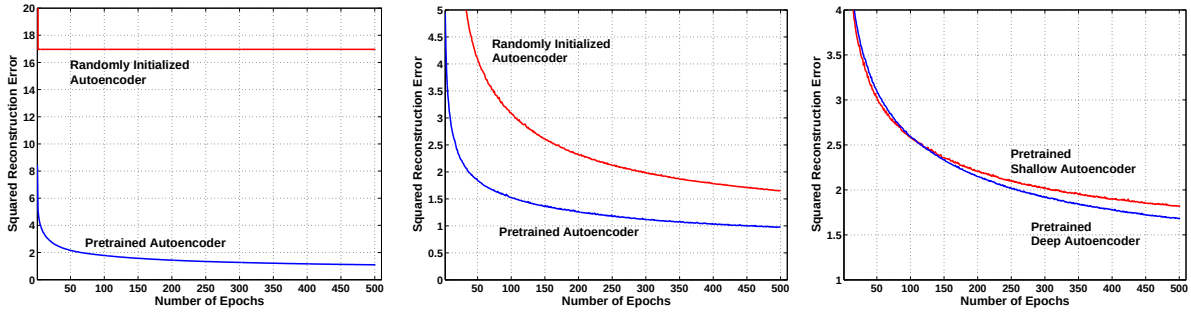

Figure 3.6: The average squared reconstruction error per test image during fine-tuning on the curves training data. **Left:** The deep 784-400-200-100-50-25-6 autoencoder makes rapid progress after pretraining but no progress without pretraining. **Middle:** A shallow 784-532-6 autoencoder can learn without pretraining but pretraining makes the fine-tuning much faster. **Right:** A 784-100-50-25-6 autoencoder performs slightly better than a shallower 784-108-6 autoencoder that has about the same number of parameters. Both autoencoders were pretrained.

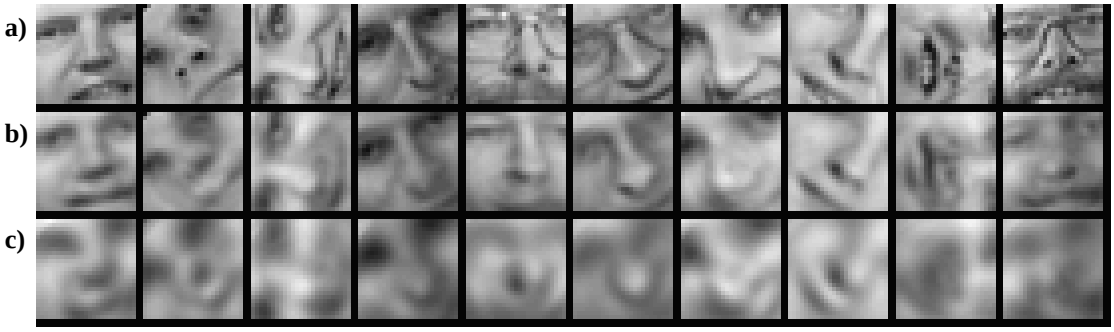

Figure 3.7: **By row: a)** Random samples from the test dataset; **b)** reconstructions by the 30-dimensional autoencoder; and **c)** reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.

our deep nonlinear autoencoder to a shallow linear autoencoder, which we call logistic PCA, where the linear code units are directly connected to both the inputs and the logistic output units.

Figure 3.6, left panel, shows performance of pretrained and randomly initialized deep autoencoders on the curves dataset. Note that without pretraining, the deep autoencoder gets stuck at poor local optimum. It always reconstructs the average of the training data, even after prolonged fine-tuning. Shallow autoencoders can learn without pretraining, but pretraining greatly reduces the total training time. Figure 3.6, right panel, further reveals that when the number of parameters is the same, a deep autoencoder produces a slightly lower reconstruction error on test data than a shallow autoencoder.

MNIST and Olivetti datasets

We used a (28×28)-1000-500-250-30 autoencoder to extract 30-dimensional codes of the handwritten digits in the MNIST training set. Similar to the curves dataset, all units were logistic except for the 30 linear units in the code layer. After fine-tuning on all 60,000 training images using cross-entropy error, the autoencoder was tested on 10,000 new images. Figure 3.5 shows that the deep autoencoder captures the structure of the data much better than logistic PCA, which, in turn, is much better than standard PCA. Figure 3.8 also shows that a two-dimensional autoencoder produces a much better visualization of the data compared to the first two principal components of PCA.

We then used a (25×25)-2000-1000-500-30 autoencoder with linear input units to discover 30-

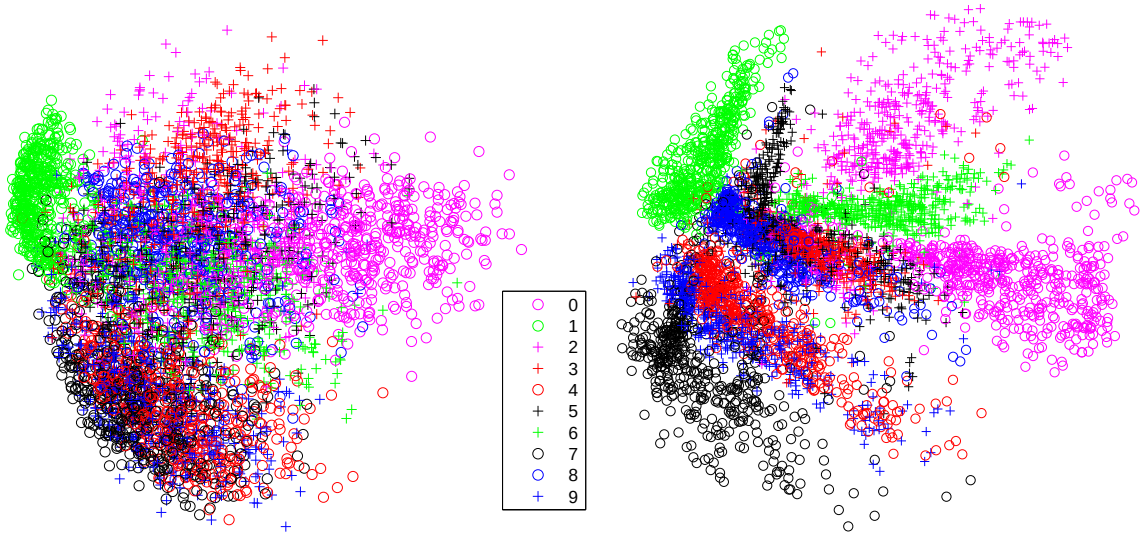

Figure 3.8: **Left**: The 2-D codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. **Right**: The 2-D codes discovered by a 784-1000-500-250-2 autoencoder.

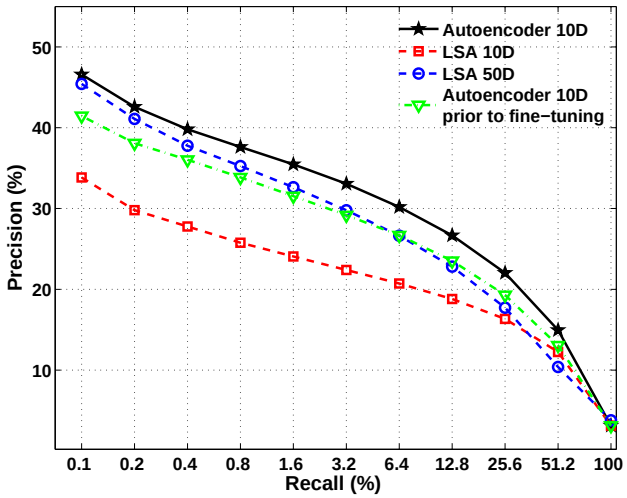

Figure 3.9: Precision-Recall curves for the Reuters RCV1-v2 dataset, when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries.

dimensional codes for grey-level image patches that were derived from the Olivetti face dataset. A dataset of 165,600 25×25 images was created by randomly rotating ($-90°$ to $+90°$), scaling (1.4 to 1.8), cropping, and subsampling the original 400 images. The dataset was then subdivided into 124,200 training images, which contained the first thirty people, and 41,400 test images, which contained the remaining ten people. Figure 3.5, bottom panel, shows that the the 30-dimensional autoencoder produces much better reconstructions than 30-dimensional PCA.

Reuters Corpus

We can also use autoencoders to discover low-dimensional codes that would allow for fast document retrieval. We performed a set of experiments on Reuters RCV1-v2 dataset that contains 804,414 newswire

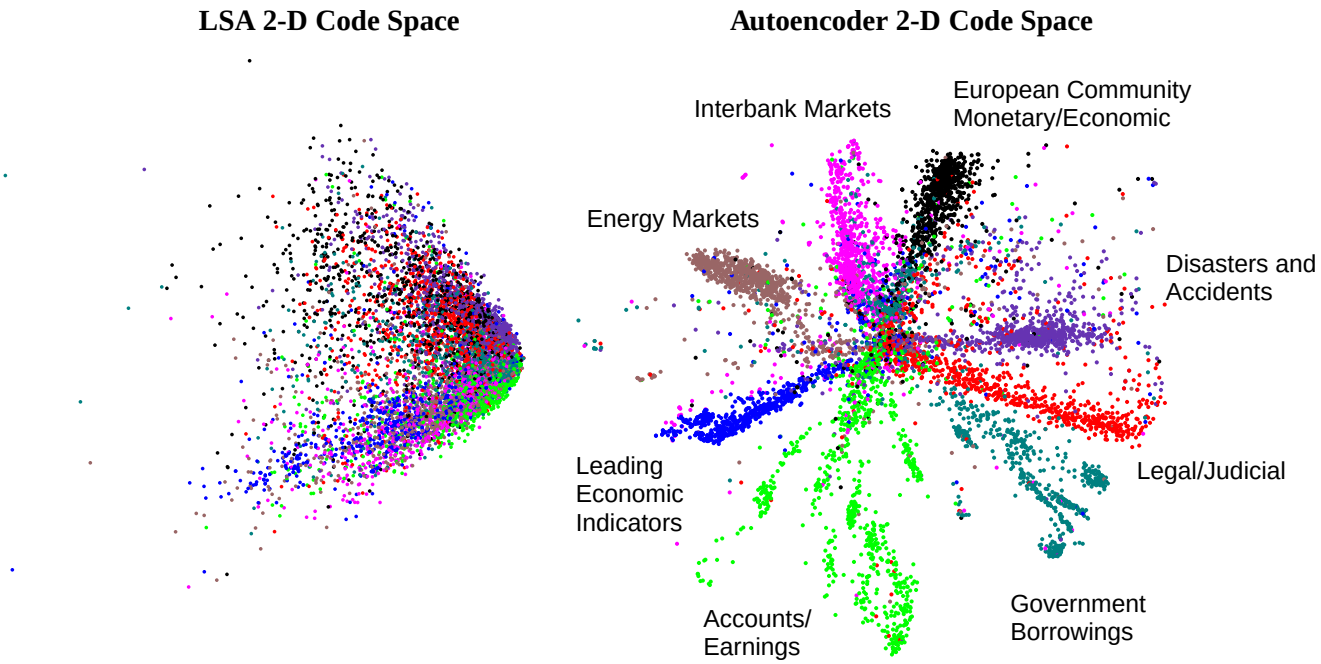

Figure 3.10: **Left**: The codes produced by the 2-dimensional LSA. **Right**: The codes produced by a 2000-500-250-125-2 autoencoder.

stories, manually categorized into 103 topics. The data was split randomly into 402,212 training and 402,212 test documents. We trained a 2000-500-250-125-10 autoencoder, where each document was represented as a vector containing 2000 most frequently used words in the training dataset. When pre-training the first layer, the word-count vectors were modelled by the Replicated Softmax model. For the fine-tuning stage, we divided the count vector by the number of words, so that it represented a probability distribution across words, and then used the cross-entropy error function with a softmax at the output layer. Figure 3.9 shows that when we use the cosine of the angle between two codes to measure similarity, the autoencoder significantly outperforms Latent Semantics Analysis (LSA) (Deerwester et al. [1990]), a well-known document retrieval method based on singular value decomposition. Even prior to fine-tuning, the autoencoder outperforms LSA. Figure 3.10 further shows that the two-dimensional codes produced an autoencoder are much better organized, compared to the codes produced by LSA.

3.2.3 Discussion

Together with pretraining, nonlinear autoencoders can be very effective for nonlinear dimensionality reduction. Unlike non-parametric methods, such as LLE (Roweis and Saul [2000]), ISOMAP (Tenenbaum et al. [2000]), or t-SNE (van der Maaten and Hinton [2009]), autoencoders provide mappings in both directions between the data and code spaces. They can also be applied to very large datasets, because both the pretraining and fine-tuning scale linearly in time and space with the number of training cases. In the next section we show how a network with multiple layers and with hundreds of thousands of parameters can be trained to discover "semantic" binary codes, so that similar input vectors will have similar binary codewords. This, in turn, will allow us to very quickly and accurately retrieve a set of documents that are similar to a given query document.

3.3 Document Retrieval

One of the most popular and widely used algorithms for retrieving documents that are similar to a query document is TF-IDF (Salton and Buckley [1988], Salton [1991]), which measures the similarity between documents by comparing their word-count vectors. The similarity metric weights each word by both its frequency in the query document (Term Frequency) and the logarithm of the reciprocal of its frequency in the whole set of documents (Inverse Document Frequency). TF-IDF, however, has several major limitations: it computes document similarity directly in the word-count space, and it assumes that the counts of different words provide independent evidence of similarity.

To remedy these drawbacks, numerous models for capturing low-dimensional, latent representations have been proposed and successfully applied in the domain of information retrieval. A simple and widely-used method is Latent Semantic Analysis (LSA) (Deerwester et al. [1990]), which extracts low-dimensional semantic structure using SVD decomposition to get a low-rank approximation of the word-document co-occurrence matrix. This allows document retrieval to be based on "semantic" content rather than just on individually weighted words. LSA, however, is very restricted in the types of semantic content it can capture because it is a linear method, so it can only capture pairwise correlations between words. A probabilistic version of LSA (pLSA) was introduced by Hofmann [1999], using the assumption that each word is modeled as a sample from a document-specific multinomial mixture of word distributions. A proper generative model at the level of documents, Latent Dirichlet Allocation, was introduced by Blei et al. [2003].

These probabilistic models can be viewed as graphical models in which hidden topic variables have directed connections to variables that represent word-counts. Their major drawback is that exact inference is intractable due to explaining away, so they have to resort to slow or inaccurate approximations to compute the posterior distribution over topics. This makes it difficult to fit the models to data. Also, as Welling et al. [2005] point out, fast inference is important for information retrieval. To achieve this, they introduce a class of two-layer undirected graphical models that generalize Restricted Boltzmann Machines (RBM's) to exponential family distributions. This allows them to model non-binary data and to use non-binary latent variables. Maximum likelihood learning is intractable in these models, but learning can still be performed by using Contrastive Divergence (Hinton [2002]). Several further developments of these undirected models (Gehler et al. [2006], Xing et al. [2005]) show that they are competitive in terms of retrieval accuracy with their directed counterparts.

All of the above models, however, have several important limitations. First, there are limitations on the types of structure that can be represented efficiently by a single layer of hidden variables. We have already seen in section 3.2 that a network with many hidden layers and with hundreds of thousands of parameters can discover latent representations that work much better for information retrieval. Second, many of the existing retrieval algorithms are based on computing a similarity measure between a query document and other documents in the collection. The similarity is computed either directly in the word space or in a low-dimensional latent space. Typically, the larger the size of document collection, the longer it will take to search for relevant documents.

3.3.1 Semantic Hashing

Using the two-stage learning procedure, introduced in section 3.2, we can build an autoencoder that learns to map documents into "semantic" binary codes. We call this model Semantic Hashing. By using learned binary codes as memory addresses, we can effectively learn a *semantic address space*, so a document can be mapped to a memory address in such a way that a small hamming-ball around that memory address contains semantically similar documents, as shown in Fig. 3.11, left panel. This representation will allow us to retrieve a short-list of semantically similar documents on very large

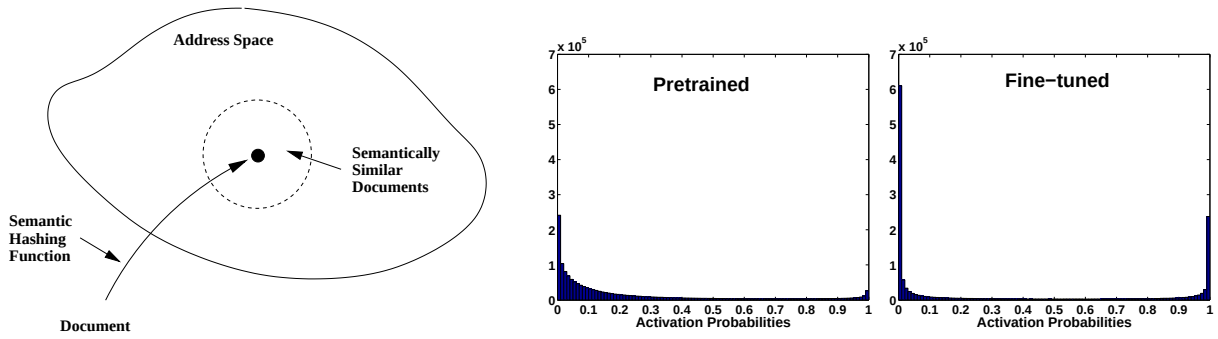

Figure 3.11: **Left**: A schematic representation of Semantic Hashing. **Right**: The distribution of the activities of the 128 code units on the 20-newsgroups training data before and after fine-tuning with backpropagation and deterministic noise.

document sets in time independent of the number of documents. The short-list can then be given to a slower but more precise retrieval method, such as TF-IDF.

Making the codes binary

During the pretraining stage, the bottom layer RBM is trained using a Replicated Softmax model, while the remaining RBM's in the stack use logistic units. During the fine-tuning, we want to find codes that are good at reconstructing the data but are as close to binary as possible. To make the codes binary, we add Gaussian noise to the bottom-up input received by each code unit[2]. Let us assume that the output of the decoder network is not very sensitive to small changes in the output of a code unit. Then the best way to communicate information in the presence of added noise is to make the bottom-up input received by a code unit be either very large and negative or very large and positive. Figure 3.11, right panel, shows that this is what the fine-tuning does.

To prevent the added Gaussian noise from messing up the conjugate gradient fine-tuning, we used "deterministic noise" with mean zero and standard deviation 4, chosen by cross-validation. For each training case, the sampled noise values are fixed in advance and do not change during training. After fine-tuning, the codes were thresholded to produce binary code vectors. The asymmetry between 0 and 1 in the energy function of an RBM causes the unthresholded codes to have many more values near 0 than near 1, so we used a threshold of 0.1. We also experimented with various values for the noise variance and the threshold. Our results are fairly robust to variations in these parameters and also to variations in the number of layers and the number of units in each layer.

3.3.2 Experimental Results

To evaluate performance of our model on an information retrieval task we use Precision-Recall curves, where we define:

$$\text{Recall} = \frac{\text{Number of retrieved relevant documents}}{\text{Total number of all relevant documents}}$$

$$\text{Precision} = \frac{\text{Number of retrieved relevant documents}}{\text{Total number of retrieved documents}}$$

[2]We tried other ways of encouraging the code units to be binary, such as penalizing the entropy $-p \log p - (1-p) \log (1-p)$ for each code unit, but Gaussian noise worked better.

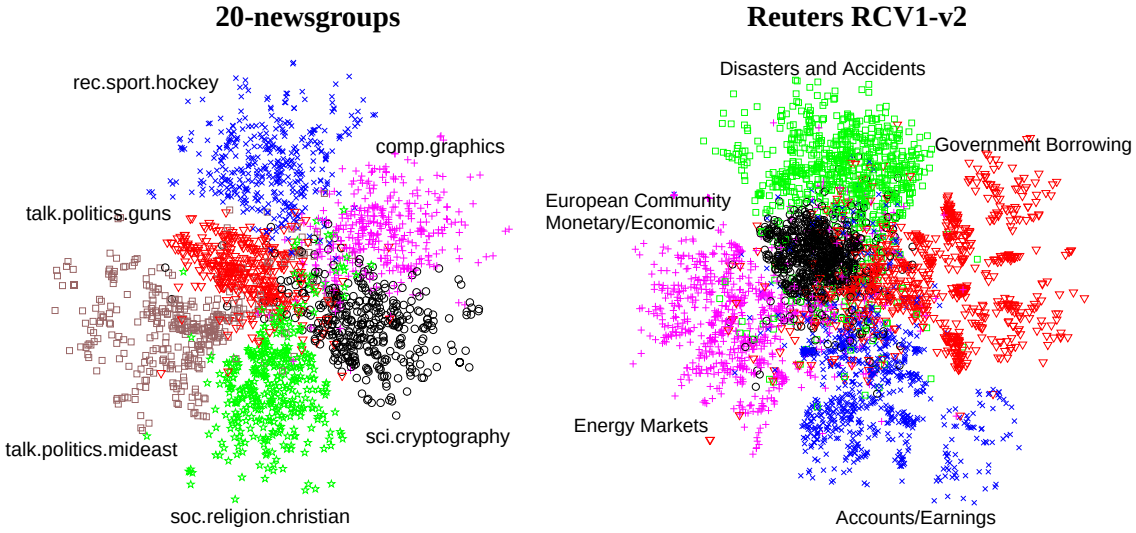

Figure 3.12: A 2-dimensional embedding of the 128-bit codes using stochastic neighbor embedding for the 20-newsgroups data (left panel) and the Reuters RCV1-v2 corpus (right panel). See in color for better visualization.

To decide whether a retrieved document is relevant to the query document, we simply look to see if they have the same class label. This is the only time that the class labels are used. Results of Gehler et al. [2006] show that pLSA and LDA models do not generally outperform LSA and TF-IDF. Therefore for comparison we only used LSA and TF-IDF as benchmark methods. For LSA each word count, c_i, was replaced by $\log(1 + c_i)$ before the SVD decomposition, which slightly improved performance. For both these methods we used the cosine of the angle between two vectors as a measure of their similarity.

Description of the Text Corpora

We present experimental results for document retrieval on two text datasets: 20-newsgroups and Reuters RCV1-v2. The 20-newsgroups corpus contains 18,845 postings taken from the Usenet newsgroup collection. The corpus is partitioned fairly evenly into 20 different newsgroups, each corresponding to a separate topic. The data was split by date into 11,314 training and 7,531 test articles, so the training and test sets were separated in time. The training set was further randomly split into 8,314 training and 3,000 validation documents. We only considered the 2000 most frequently used words in the training dataset. As a result, each posting was represented as a vector containing 2000 word counts.

The Reuters RCV1-v2 corpus, which we used in two previous sections, is an archive of 804,414 newswire stories that have been manually categorized into 103 topics. Sample topics are displayed in Fig. 3.12. The data was randomly split into 402,207 training and 402,207 test articles. The training set was further randomly split into 302,207 training and 100,000 validation documents. We only used the 2000 most frequently used words in the training dataset. More detailed description of these two document corpora can be found in Appendix A.

Results using 128-bit codes

For both datasets we used a 2000-500-500-128 architecture. To see whether the learned 128-bit codes preserve class information, we used stochastic neighbor embedding (Hinton and Roweis [2002]) to visualize the 128-bit codes of all the documents from 5 and 6 separate classes. Figure 3.12 shows that for both datasets the 128-bit codes preserve the class structure of the documents.

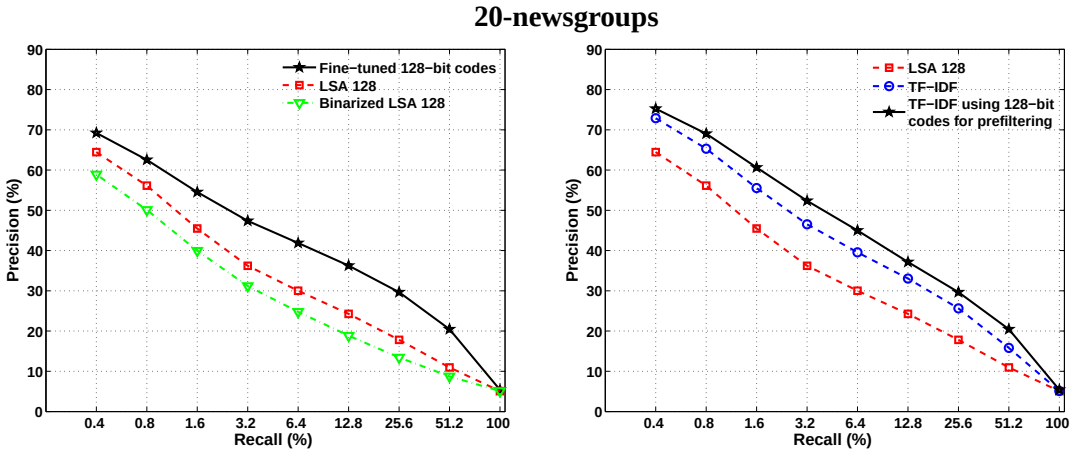

Figure 3.13: Precision-Recall curves for the 20-newsgroups dataset, when a query document from the test set is used to retrieve other test set documents, averaged over all 7,531 possible queries.

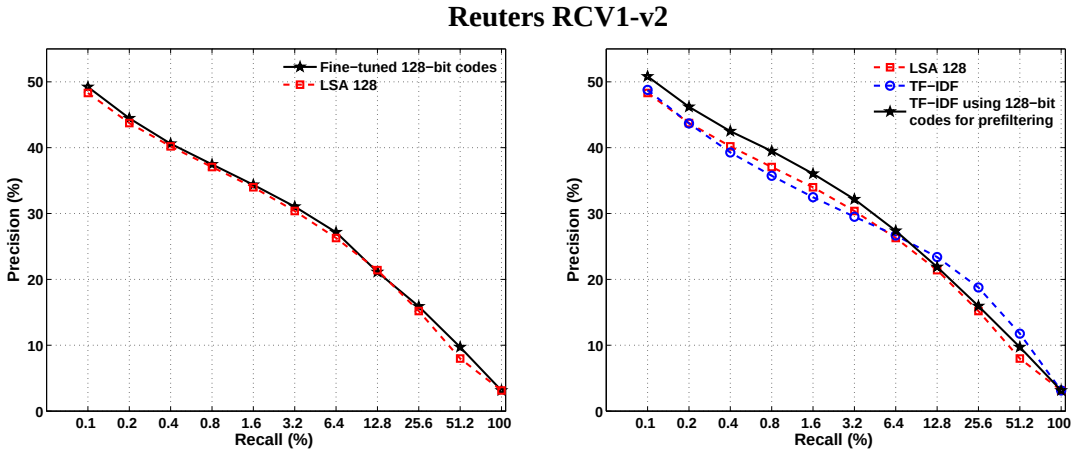

Figure 3.14: Precision-Recall curves for the Reuters RCV1-v2 dataset, when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries.

In addition to requiring very little memory, binary codes allow very fast search because fast bit counting routines[3] can be used to compute the Hamming distance between two binary codes. On a 3GHz Intel Xeon running C, for example, it only takes 3.6 milliseconds to search through 1 million documents using 128-bit codes. The same search takes 72 milliseconds for 128-dimensional LSA.

Figures 3.13 and 3.14, left panels, show that the learned 128-bit codes are better at document retrieval than the 128 real-values produced by LSA. We also tried thresholding the 128 real-values produced by LSA to get binary codes. The thresholds were set so that each of the 128 components was a 0 for half of the training set and a 1 for the other half. The results of Fig. 3.13 reveal that binarizing LSA significantly reduces its performance. This is hardly surprising since LSA has not been optimized to make the binary codes perform well.

TF-IDF was slightly more accurate than our 128-bit codes when retrieving the top few documents in either dataset. If, however, we use the 128-bit codes to preselect the top 100 documents for the 20-newsgroups data or the top 1000 for the Reuters data, and then re-rank these preselected documents using TF-IDF, we get better accuracy than running TF-IDF alone on the whole document set (see

[3]Code is available at http://www-db.stanford.edu/∼manku/bitcount/bitcount.html

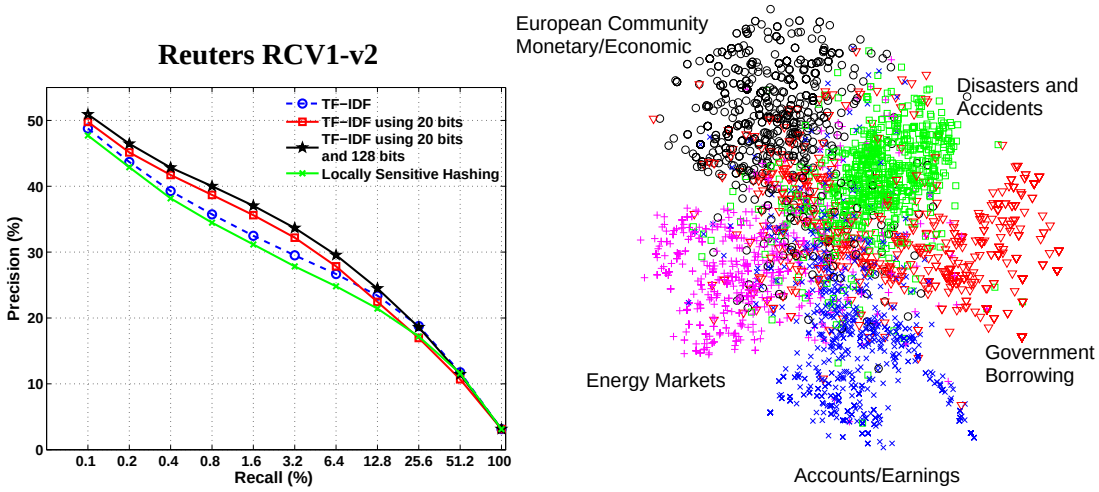

Figure 3.15: **Left:** Precision-Recall curves for the Reuters RCV1-v2 dataset, when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. **Right:** A 2-dimensional embedding of the 20-bit codes using stochastic neighbor embedding for the Reuters RCV1-v2 corpus.

Figs. 3.13 and 3.14, right panels). This means that some documents that TF-IDF would have considered a very good match to the query document have been correctly eliminated by using the 128-bit codes as a filter.

Results using 20-bit codes

Using 20-bit codes, we also checked whether our learning procedure could discover a way to model similarity of count-vectors by similarity of 20-bit codewords that was good enough to allow high precision and retrieval for our set of 402,207 test documents. After learning to assign 20-bit addresses to the training documents, we compute the 20-bit address of each test document and place a pointer to the document at its address.[4]

For the 402,207 test documents, a 20-bit address space gives a density of about 0.4 documents per address. For a given query document, we compute its 20-bit address and then retrieve all of the documents stored in a hamming ball of radius 4 (about $6196 \times 0.4 \approxeq 2500$ documents) without performing any search at all. Figure 3.15 shows that neither precision nor recall is lost by restricting TF-IDF to this fixed, preselected set.

Using a simple implementation of Semantic Hashing in C, it takes about 0.5 milliseconds to create the short-list of about 3000 semantically similar documents (hamming-ball of radius 4) and about 0.01 seconds to retrieve the top few matches from that short-list using TF-IDF. Locality Sensitive Hashing (LSH) (Datar et al. [2004], Andoni and Indyk [2006]) takes about 0.5 seconds to perform the same search using E^2LSH 0.1 software, provided by Alexandr Andoni and Piotr Indyk. Also, LSH is an approximation to nearest-neighbor matching in the word-count space, so it cannot be expected to perform better than TF-IDF and it generally performs slightly worse. Figure 3.15 shows that using Semantic Hashing as a filter, in addition to being much faster, achieves higher accuracy than either LSH or TF-IDF applied to the whole document set.

We can also use a two-stage filtering procedure by first retrieving documents using 20-bit addresses in a hamming ball of larger radius 6 (about 24317 documents), filter these down to 1000 using 128-

[4]We actually start with a pointer to *null* at all addresses and then replace it by a one-dimensional array that contains pointers to all the documents that have that address.

bit codes, and then apply TF-IDF. This two-stage method achieves higher precision and better recall compared to a single stage Semantic Hashing as shown in Fig. 3.15.

Semantic Hashing for Large Document Collections

For a billion documents, a 30-bit address space gives a density of about 1 document per address and Semantic Hashing only requires a few Gigabytes of memory. Using a hamming-ball of radius 5 around the address of a query document, we can create a long "shortlist" of about 175,000 similar documents with no search at all. The items in the long shortlist could be further filtered using, for example, 128-bit binary codes produced by an autoencoder. Using stochastic gradient descent, scaling up the learning to a billion training cases would not be particularly difficult. Indeed, Semantic Hashing has already been scaled up and successfully applied to large image retrieval tasks that use 13 million images downloaded from the web (Torralba et al. [2008]).

3.3.3 Discussion

By treating the learned binary codes as memory addresses, we can find semantically similar documents in a time that is independent of the size of the document collection. Our results show that using using Semantic Hashing as a filter for TF-IDF, we can achieve higher precision and recall than TF-IDF or Locality Sensitive Hashing applied to the whole document collection.

One additional way to improve model performance is to regularize the code space using label information when it is available. This can be accomplished by forcing the codes for data points of the same label to lie close to each other in the code space, which leads to the idea of supervised learning of nonlinear mappings.

3.4 Learning Nonlinear Mappings that Preserve Class Neighbourhood Structure

Learning a similarity measure or distance metric over the input space $\mathbf{X}$ is an important task in machine learning. A good similarity measure can provide insight into how high-dimensional data is organized and it can significantly improve the performance of algorithms like K-nearest neighbours (KNN) that are based on computing distances (Cover and Hart [1967]).

For any given distance metric D (e.g. Euclidean) we can measure similarity between two input vectors $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$ by computing $D[F(\mathbf{x}_i; W), F(\mathbf{x}_j; W)]$, where $F(\mathbf{x}; W)$ is a function $F : \mathbf{X} \rightarrow \mathbf{Z}$ mapping the input vectors in $\mathbf{X}$ into a feature space $\mathbf{Z}$. As noted by Globerson and Roweis [2005], learning a similarity measure is closely related to the problem of feature extraction, since for any fixed D, any feature extraction algorithm can be thought of as learning a similarity metric. Previous work studied the case when D is Euclidean distance and $F(\mathbf{x}; W)$ is a simple linear projection $F(\mathbf{x}; W) = W\mathbf{x}$. The Euclidean distance in the feature space is then the Mahalanobis distance in the input space:

$$D[F(\mathbf{x}_i), F(\mathbf{x}_j)] \;=\; (\mathbf{x}_i - \mathbf{x}_j)^\top W^\top W\,(\mathbf{x}_i - \mathbf{x}_j)\,. \tag{3.11}$$

Linear discriminant analysis (LDA) learns the matrix W that minimizes the ratio of within-class distances to between-class distances. Goldberger et al. [2004] learned the linear transformation that optimized the performance of KNN in the resulting feature space. This differs from LDA because it allows two members of the same class to be far apart in the feature space as long as each member of the class is close to K other class members. Globerson and Roweis [2005] learned the matrix W such that the input vectors from the same class mapped to a tight cluster. They showed that their method

approximates the local covariance structure of the data and is not based on a Gaussian assumption as opposed to LDA that uses the global covariance structure. Weinberger et al. [2005] also learned W with the twin goals of making the K-nearest neighbours belong to the same class and making examples from different classes be separated by a large margin. They succeeded in achieving a test error rate of 1.3% on the MNIST dataset.

A linear transformation has a limited number of parameters and it cannot model higher-order correlations between the original data dimensions. As shown in sections 3.2 and 3.3, using a nonlinear transformation function $F(\mathbf{x}; W)$, we can discover low-dimensional representations that work much better than existing linear methods provided the dataset is large enough to allow the parameters to be estimated. Using greedy unsupervised learning algorithm, we can train a multilayer, nonlinear encoder network that transforms the input data vector $\mathbf{x}$ into a low-dimensional feature representation $F(\mathbf{x}; W)$. After the initial pretraining, the parameters can be fine-tuned by performing gradient descent in the Neighbourhood Component Analysis (NCA) objective function, introduced by Goldberger et al. [2004]. The learning results in a nonlinear transformation of the input space that has been optimized to make KNN perform well in the low-dimensional feature space.

3.4.1 Learning Nonlinear NCA

We are given a set of N labeled training cases $(\mathbf{x}_i, c_i)$, $i = 1, 2, ..., N$, where $\mathbf{x}_i \in \mathbb{R}^D$, and $c_i \in \{1, 2, ..., C\}$. For each training vector $\mathbf{x}_i$, define the probability that point i selects one of its neighbours j in the transformed feature space as:

$$p_{ij} = \frac{\exp\left(-d_{ij}^2\right)}{\sum_{z \neq i} \exp\left(-d_{iz}^2\right)}, \qquad p_{ii} = 0. \tag{3.12}$$

We focus on the Euclidean distance metric:

$$d_{ij} = \parallel F(\mathbf{x}_i; W) - F(\mathbf{x}_j; W) \parallel,$$

and $F(\cdot; W)$ is a multilayer neural network parameterized by the weight vector W. In the NCA model, the probability that point i belongs to class k depends on the relative proximity of all other data points that belong to class k:

$$p(c_i = k) = \sum_{j: c_j = k} p_{ij}. \tag{3.13}$$

The NCA objective is to maximize the expected number of correctly classified points on the training data:

$$O_{\text{NCA}} = \sum_{i=1}^{N} \sum_{j: c_i = c_j} p_{ij}. \tag{3.14}$$

One could alternatively maximize the sum of the log probabilities of correct classification:

$$O_{\text{LG}} = \sum_{i=1}^{N} \log \left(\sum_{j: c_i = c_j} p_{ij} \right). \tag{3.15}$$

When $F(\mathbf{x}; W) = W\mathbf{x}$ is constrained to be a linear transformation, we get linear NCA (Goldberger et al. [2004]). When $F(\mathbf{x}; W)$ is defined by a multilayer, nonlinear neural network, we can explore a

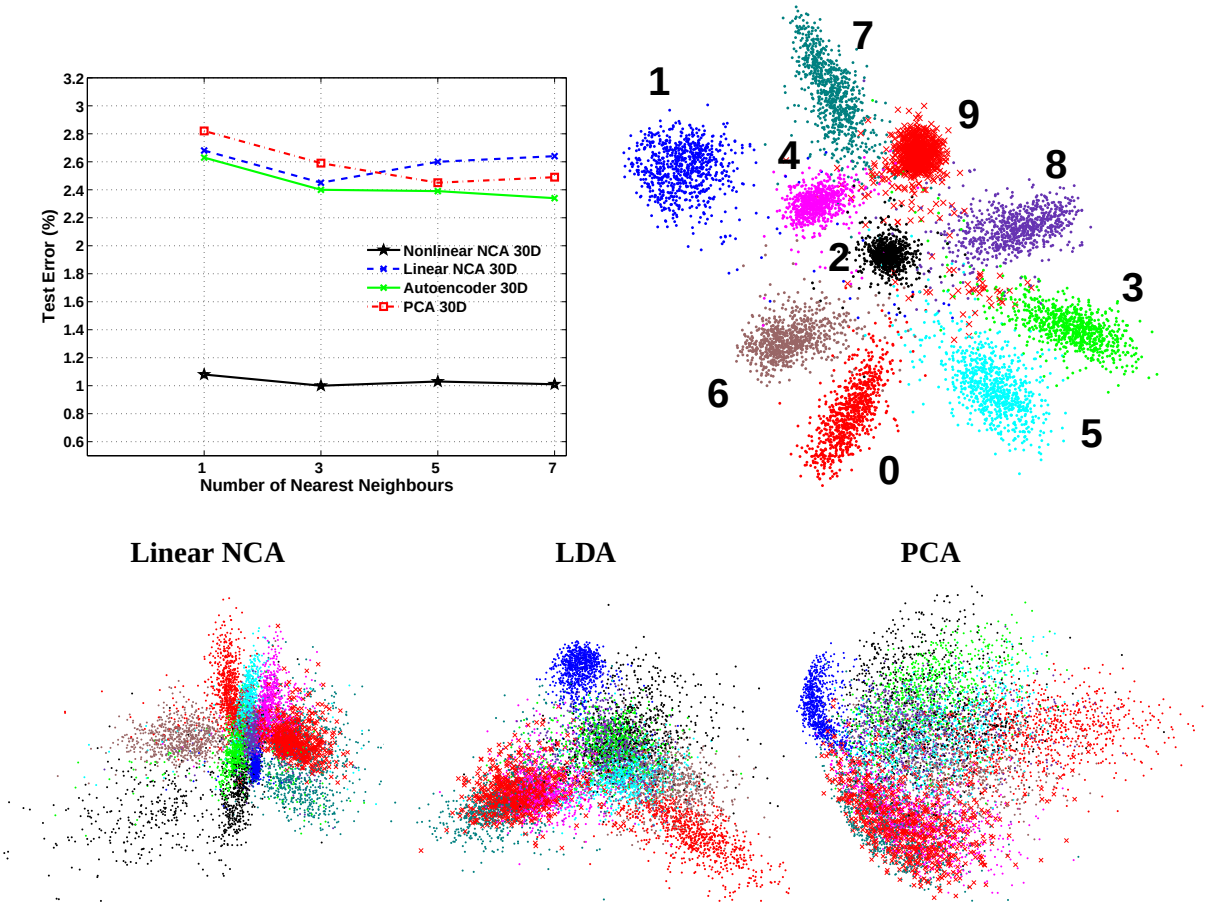

Figure 3.16: **Top right:** The 2-dimensional codes produced by nonlinear NCA on the MNIST test data using a 784-500-500-2000-2 encoder. **Bottom:** The 2-dimensional codes produced by linear NCA, Linear Discriminant Analysis, and PCA.

much richer class of transformations by backpropagating the derivatives of the objective functions in Eq. 3.14 or 3.15 with respect to parameter vector W through the layers of the encoder network. In our experiments, the NCA objective O_{NCA} of Eq. 3.14 worked slightly better than O_{LG}. We suspect that this is because O_{NCA} is more robust to handling outliers. O_{ML}, on the other hand, would strongly penalize configurations where a point in the feature space does not lie close to any other member of its class.

Denote $\mathbf{d}_{ij} = F(\mathbf{x}_i; W) - F(\mathbf{x}_j; W)$, then the derivatives of O_{NCA} with respect to parameter vector W for the i^{th} training case are:

$$\frac{\partial O_{\mathrm{NCA}}}{\partial W} = \frac{\partial O_{\mathrm{NCA}}}{\partial F(\mathbf{x}_i; W)} \frac{\partial F(\mathbf{x}_i; W)}{\partial W},$$

where

$$\frac{\partial O_{\mathrm{NCA}}}{\partial F(\mathbf{x}_i; W)} = -2 \left[\sum_{j: c_i = c_j} p_{ij} \left(\mathbf{d}_{ij} - \sum_{z \neq i} p_{iz} \mathbf{d}_{iz} \right) \right] + 2 \left[\sum_{j: c_j = c_i} p_{ji} \mathbf{d}_{ji} - \sum_{z \neq i} \left(\sum_{q: c_z = c_q} p_{zq} \right) p_{zi} \mathbf{d}_{zi} \right],$$

and $\frac{\partial F(\mathbf{x}_i; W)}{\partial W}$ is computed using standard backpropagation.

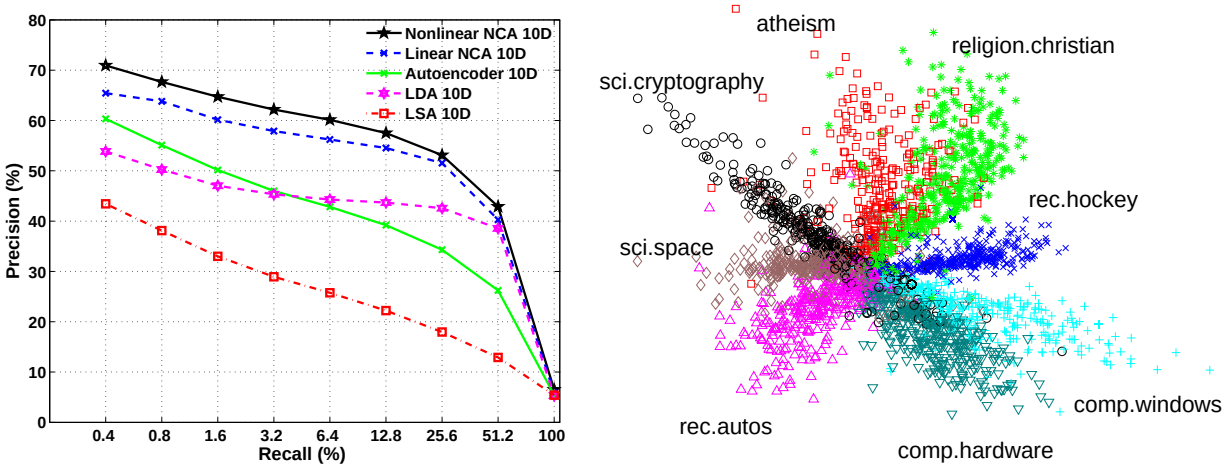

Figure 3.17: **Left:** Precision-Recall curves for the 20-newsgroups dataset when a query document from the test set is used to retrieve other test set documents, averaged over all 7,531 possible queries. **Right:** The 2-dimensional codes produced by nonlinear NCA on test dataset using 2000-500-500-2 encoder.

3.4.2 Experimental Results

In this section we present experimental results for the MNIST and 20-newsgroup datasets. For the MNIST dataset, we pretrained a 28×28-500-500-2000-30 architecture. The 30 code units were linear and the remaining hidden units were logistic. Figure 3.16 shows that Nonlinear NCA, after 50 epochs of fine-tuning, achieves an error rate of 1.08%, 1.00%, 1.03%, and 1.01% using 1, 3, 5, and 7 nearest neighbours. This is compared to the best reported error rates (without using any domain-specific knowledge) of 1.6% for randomly initialized backpropagation and 1.4% for Support Vector Machines (Decoste and Schölkopf [2002]). Linear methods such as linear NCA or PCA are much worse than nonlinear NCA. Figure 3.16, right panel, shows the 2-dimensional codes produced by nonlinear NCA compared to linear NCA, Linear Discriminant Analysis, and PCA.

For the 20-newsgroups dataset we used 2000-500-500-10 architecture. The 10 code units were linear and the remaining hidden units were logistic. Figure 3.17 shows that Nonlinear NCA significantly outperforms other standard methods that use labeled data such as NCA and LDA, and other methods that do not use labeled data such as Latent Semantic Analysis (LSA), and a deep autoencoder of the same architecture. Clearly, additional labeled data can improve model performance.

3.4.3 Regularized Nonlinear NCA

The NCA objective, which encourages codes to lie close to other codes belonging to the same class, can be combined with the autoencoder objective function (see Fig. 3.18, left panel) to maximize:

$$C \;=\; \lambda O_{\mathrm{NCA}} + (1-\lambda)(-\mathrm{E}), \tag{3.16}$$

where O_{NCA} is defined in Eq. 3.14, E is the reconstruction error, and λ is a trade-off parameter. When the derivative of the reconstruction error E is backpropagated through the autoencoder, it is combined, at the code level, with the derivatives of O_{NCA}.

This setting is particularly useful for semi-supervised learning tasks. Consider having a set of N_l labeled training data $(\mathbf{x}_l, c_l)$, where as before $\mathbf{x}_l \in \mathbb{R}^D$, and $c_l \in \{1, 2, ..., C\}$, and a set of N_u unlabeled training data $\mathbf{x}_u$. Let $N = N_l + N_u$. The overall objective to maximize can be written as:

$$O \;=\; \lambda \frac{1}{N_l} \sum_{l=1}^{N_l} \sum_{j|c_l=c_j} p_{lj} + (1-\lambda)\frac{1}{N}\sum_{n=1}^{N}(-\mathrm{E}^n)\,, \tag{3.17}$$

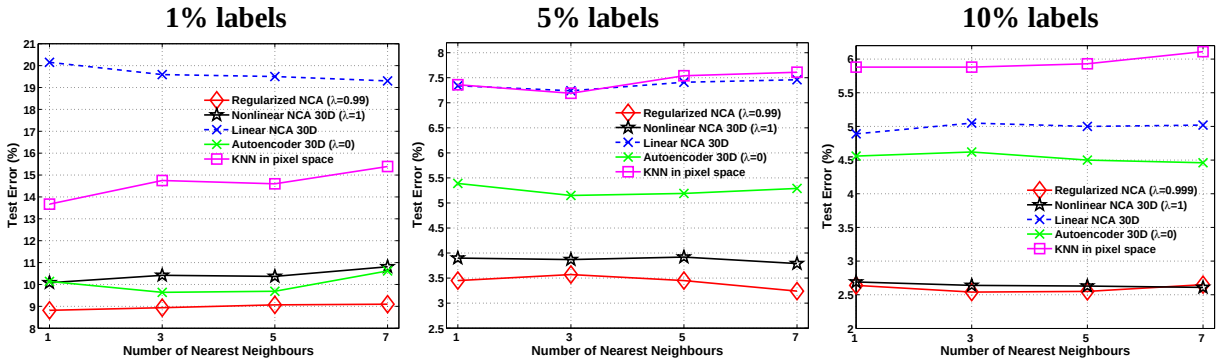

Figure 3.18: KNN on the MNIST test set when only a small fraction of class labels is available. Linear NCA and KNN in pixel space do not take advantage of the unlabeled data.

where E^n is the reconstruction error for the input data vector $\mathbf{x}^n$. For the MNIST dataset we use the cross-entropy error:

$$\mathrm{E}^p = -\sum_i x_i^n \log \hat{x}_i^n - \sum_i (1 - x_i^n) \log(1 - \hat{x}_i^n), \tag{3.18}$$

where $x_i^n \in [0, 1]$ is the intensity of pixel i for the training example n, and $\hat{x}_i^n$ is the intensity of its reconstruction.

When the number of labeled examples is small, regularized nonlinear NCA performs better than nonlinear NCA ($\lambda = 1$), which uses the unlabeled data for pretraining but ignores it during the fine-tuning. It also performs better than an autoencoder ($\lambda = 0$), which ignores the labeled set. To test the effect of the regularization when most of the data is unlabeled, we randomly sampled 1%, 5% and 10% of the handwritten digits in each class and treated them as labeled data. The remaining digits were treated as unlabeled data. Figure 3.18 reveals that regularized nonlinear NCA ($\lambda = 0.99$) outperforms both nonlinear NCA ($\lambda = 1$) and an autoencoder ($\lambda = 0$). The parameter λ was selected using cross-validation from among the values $\{0.5, 0.9, 0.99, 0.999\}$. Even when the entire training set is labeled, regularized NCA still performs slightly better. In Chapter 5 we will show that other deep models can outperform regularized nonlinear NCA, even when the number of labeled examples is small.

Splitting codes into class-relevant and class-irrelevant parts

To allow accurate reconstruction of a digit image, the code must contain information about aspects of the image such as its orientation, slant, size and stroke thickness that are not relevant to its classification. These irrelevant aspects necessarily contribute to the Euclidean distance between codes and harm classification. To reduce this unwanted effect, we used 50-dimensional codes but only used the first 30 dimensions in the NCA objective function. The remaining 20 dimensions were free to code all the aspects of an image that do not affect its class label but are important for its reconstruction.

Figure 3.18, right panel, shows how the reconstruction is affected by changing the activity level of a single code unit. Changing a unit among the first 30 changes the class, while changing a unit among the last 20 does not. With $\lambda = 0.99$ the split codes achieve an error rate of 1.00% 0.97% 0.98% 0.97% using 1, 3, 5, and 7 nearest neighbours. We also computed the 3NN error rate on the test set using only the last 20 code units. It was 4.3%, clearly indicating that the class-relevant information is concentrated in the first 30 units.

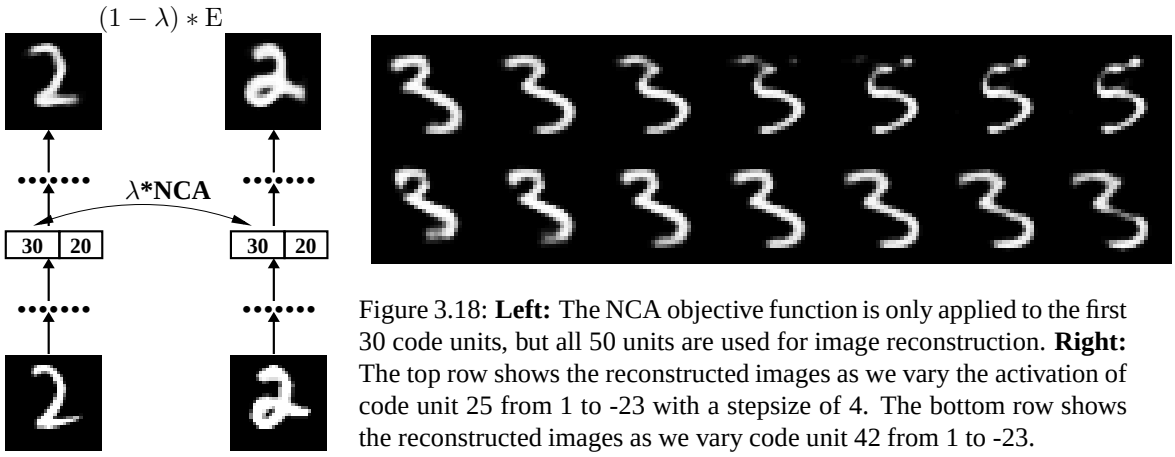

Figure 3.18: **Left:** The NCA objective function is only applied to the first 30 code units, but all 50 units are used for image reconstruction. **Right:** The top row shows the reconstructed images as we vary the activation of code unit 25 from 1 to -23 with a stepsize of 4. The bottom row shows the reconstructed images as we vary code unit 42 from 1 to -23.

3.4.4 Discussion

Learning a similarity measure over the input space by greedily pretraining and then fine-tuning a deep nonlinear encoder network can greatly facilitate nearest-neighbor classification. Using the reconstruction error as a regularizer and split codes to suppress the influence of class-irrelevant information in the input, nonlinear NCA achieves an error rate of 1.00% on a widely used version of the MNIST handwritten digit recognition task that does not use any domain-specific knowledge. Furthermore, similar to learning a covariance kernel for a Gaussian process, the regularized version of nonlinear NCA can make good use of large amounts of unlabeled data, so the classification accuracy is high even when the amount of labeled training data is very limited.

Chapter 4

Evaluating Deep Belief Networks as Density Models

Deep Belief Networks are generative models that contain many layers of hidden variables. Efficient greedy algorithms for learning and approximate inference have allowed these models to be applied successfully in many application domains, as we discussed in chapter 3. The main building block of a DBN is a bipartite undirected graphical model called a Restricted Boltzmann Machine (RBM). Due to the presence of the partition function exact maximum likelihood learning in RBM's is intractable, and model selection and complexity control are difficult. In this chapter we show that a Monte Carlo based method, Annealed Importance Sampling (AIS), can be used to efficiently estimate the partition function of an RBM. We further show how an AIS estimator, along with approximate inference, can be used to estimate a lower bound on the log-probability that a DBN model with multiple hidden layers assigns to the *test data*. This allows us to directly assess generalization performance of Deep Belief Networks as density models.

4.1 Introduction

Deep Belief Networks (DBN's), reviewed in chapter 2, are probabilistic generative models that contain several layers of latent variables. The greedy learning algorithm for DBN's proceeds by learning a stack of undirected graphical models, called Restricted Boltzmann Machines (RBM's). A key feature of RBM's is that inference in these models is easy. An unfortunate limitation is that the probability of data under the model is known only up to a computationally intractable normalizing constant – the partition function. A good estimate of the partition function would allow us to assess generalization performance of RBM's and DBN's as density models. Indeed, assessing the generalization performance plays an important role in model selection and complexity control. For many specific tasks, such as information retrieval or object recognition, performance of RBM's and DBN's can be directly evaluated (Nair and Hinton [2009], Bengio et al. [2007], Salakhutdinov and Hinton [2009b]). More broadly, however, the model's generalization capability can be evaluated by computing the probability that the model assigns to the previously unseen input vectors, which is independent of any specific application.

There has been extensive research on obtaining deterministic approximations (Yedidia et al. [2005]) or deterministic upper bounds (Wainwright et al. [2005], Globerson and Jaakkola [2007]) on the log-partition function of arbitrary discrete Markov random fields (MRF's). These variational methods rely critically on an ability to approximate the entropy of the undirected graphical model. However, for densely connected MRF's, such as RBM's, these methods are unlikely to perform well (Salakhutdinov [2008]). There have also been many developments in the use of Monte Carlo methods for estimating

the partition function, including Annealed Importance Sampling (AIS) (Neal [2001]), Nested Sampling (Skilling [2004]), and many others (see e.g. Neal [1993]). In this chapter we show how one such method, AIS, by taking advantage of the bipartite structure of an RBM, can be used to efficiently estimate its partition function. We further show that this estimator, along with approximate inference, can be used to estimate a lower bound on the log-probability that a DBN model with multiple hidden layers assigns to training or test data. This result will allow us to assess the performance of DBN's as generative models and to compare them to other probabilistic models, such as plain mixture models.

4.2 Estimating Partition Functions

Suppose we have two distributions defined on some space $\mathcal{X}$ with probability density functions $P_A(\mathbf{x}) = P_A^*(\mathbf{x})/\mathcal{Z}_A$ and $P_B(\mathbf{x}) = P_B^*(\mathbf{x})/\mathcal{Z}_B$, where $P^*(\cdot)$ denotes the unnormalized probability density. Let Ω_A and Ω_B be the support sets of P_A and P_B respectively. One way of estimating the ratio of normalizing constants is to use a simple importance sampling (SIS) method. We use the following identity, assuming that $\Omega_B \subseteq \Omega_A$, i.e. $P_A(\mathbf{v}) \neq 0$ whenever $P_B(\mathbf{v}) \neq 0$:

$$\frac{\mathcal{Z}_B}{\mathcal{Z}_A} = \frac{\int P_B^*(\mathbf{x})\mathrm{d}\mathbf{x}}{\mathcal{Z}_A} = \int \frac{P_B^*(\mathbf{x})}{P_A^*(\mathbf{x})} P_A(\mathbf{x})\mathrm{d}\mathbf{x} = \mathrm{E}_{P_A}\left[\frac{P_B^*(\mathbf{x})}{P_A^*(\mathbf{x})}\right].$$

Assuming we can draw independent samples from P_A, an unbiased estimate of the ratio of partition functions can be obtained by using a simple Monte Carlo approximation:

$$\frac{\mathcal{Z}_B}{\mathcal{Z}_A} \approx \frac{1}{M}\sum_{i=1}^{M} \frac{P_B^*(\mathbf{x}^{(i)})}{P_A^*(\mathbf{x}^{(i)})} \equiv \frac{1}{M}\sum_{i=1}^{M} w^{(i)} = \hat{r}_{\mathrm{SIS}}, \tag{4.1}$$

where $\mathbf{x}^{(i)} \sim P_A$. If we choose $P_A(\mathbf{x})$ to be a tractable distribution for which we can compute $\mathcal{Z}_A$ analytically, we obtain an unbiased estimate of the partition function $\mathcal{Z}_B$. However, if P_A and P_B are not close enough, the estimator $\hat{r}_{\mathrm{SIS}}$ will be very poor. In high-dimensional spaces, the variance of an estimator $\hat{r}_{\mathrm{SIS}}$ will be very large, or possibly infinite (see MacKay [2003], chapter 29), unless P_A is a near-perfect approximation to P_B.

4.2.1 Annealed Importance Sampling (AIS)

Suppose that we can define a sequence of intermediate probability distributions: $P_0, ..., P_K$, with $P_0 = P_A$ and $P_K = P_B$, which satisfy the following conditions:

C1 $P_k(\mathbf{x}) \neq 0$ whenever $P_{k+1}(\mathbf{x}) \neq 0$.

C2 We must be able to easily evaluate the unnormalized probability $P_k^*(\mathbf{x})$, $\forall \mathbf{x} \in \mathcal{X}$, $k = 0, ..., K$.

C3 For each $k = 1, ..., K-1$, we must be able to draw a sample $\mathbf{x}'$ given $\mathbf{x}$ using a Markov chain transition operator $T_k(\mathbf{x}' \leftarrow \mathbf{x})$ that leaves $P_k(\mathbf{x})$ invariant:

$$\int T_k(\mathbf{x}' \leftarrow \mathbf{x}) P_k(\mathbf{x})\mathrm{d}\mathbf{x} = P_k(\mathbf{x}'). \tag{4.2}$$

C4 We must be able to draw (preferably independent) samples from P_A.

Algorithm 3 Annealed Importance Sampling (AIS) run.

1: Select β_k with $0 = \beta_0 < \beta_1 < ... < \beta_K = 1$.
2: Sample $\mathbf{x}_1$ from $P_A = P_0$.
3: **for** $k = 1 : K - 1$ **do**
4: Sample $\mathbf{x}_{k+1}$ given $\mathbf{x}_k$ using $T_k(\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k)$.
5: **end for**
6: Set $w_{\text{AIS}} = \prod_{k=1}^{K} P_k^*(\mathbf{x}_k)/P_{k-1}^*(\mathbf{x}_k)$.

The transition operators $T_k(\mathbf{x}' \leftarrow \mathbf{x})$ represent the probability or probability density of transitioning from state $\mathbf{x}$ to $\mathbf{x}'$. Constructing a suitable sequence of intermediate probability distributions will depend on the problem. One general way to define this sequence is to set:

$$P_k(\mathbf{x}) \propto P_A^*(\mathbf{x})^{1-\beta_k} P_B^*(\mathbf{x})^{\beta_k}, \tag{4.3}$$

with $0 = \beta_0 < \beta_1 < ... < \beta_K = 1$ chosen by the user.

A single run of the AIS procedure is summarized in Algorithm 3. Note that there is no need to compute the normalizing constants of any intermediate distributions. After performing M runs of AIS, the importance weights $w_{\text{AIS}}^{(i)}$ can be substituted into Eq. 4.1 to obtain an estimate of the ratio of partition functions:

after performed M time of AIS:

$$\frac{\mathcal{Z}_B}{\mathcal{Z}_A} \approx \frac{1}{M} \sum_{i=1}^{M} w_{\text{AIS}}^{(i)} = \hat{r}_{\text{AIS}}. \tag{4.4}$$

It is shown in (Neal [2001, 2005]) that for sufficiently large number of intermediate distributions K, the variance of $\hat{r}_{\text{AIS}}$ will be proportional to $1/MK$. Provided K is kept large, the total amount of computation can be split in any way between the number of intermediate distributions K and the number of annealing runs M without adversely affecting the accuracy of the estimator. If samples drawn from p_A are independent, the AIS runs can be used to obtain the variance of the estimate $\hat{r}_{\text{AIS}}$:

$$\text{Var}\left(\hat{r}_{\text{AIS}}\right) = \frac{1}{M} \text{Var}\left(w_{\text{AIS}}^{(i)}\right) \approx \frac{\hat{s}^2}{M} = \hat{\sigma}^2, \tag{4.5}$$

where $\hat{s}^2$ is estimated simply from the sample variance of the importance weights.

The intuition behind AIS is the following. Consider the following identity:

$$\frac{\mathcal{Z}_K}{\mathcal{Z}_0} = \frac{\mathcal{Z}_1}{\mathcal{Z}_0} \frac{\mathcal{Z}_2}{\mathcal{Z}_1} ... \frac{\mathcal{Z}_K}{\mathcal{Z}_{K-1}}. \tag{4.6}$$

Provided the two intermediate distributions P_k and P_{k+1} are close enough, a simple importance sampler can be used to estimate each ratio $\mathcal{Z}_{k+1}/\mathcal{Z}_k$:

$$\frac{\mathcal{Z}_{k+1}}{\mathcal{Z}_k} \approx \frac{1}{M} \sum_{i=1}^{M} \frac{P_{k+1}^*(\mathbf{x}^{(i)})}{P_k^*(\mathbf{x}^{(i)})}, \qquad \text{where } \mathbf{x}^{(i)} \sim P_k.$$

These ratios can then be used to estimate $\frac{\mathcal{Z}_K}{\mathcal{Z}_0} = \prod_{k=0}^{K-1} \frac{\mathcal{Z}_{k+1}}{\mathcal{Z}_k}$. The problem with this approach is that, except for P_0, we typically cannot easily draw exact samples from intermediate distributions P_k. We could resort to Markov chain methods, but then it is hard to determine when the Markov chain has converged to the desired distribution.

A remarkable fact shown by Neal [2001], Jarzynski [1997] is that the estimate of $\mathcal{Z}_K/\mathcal{Z}_0$ will be exactly unbiased if each ratio $\mathcal{Z}_{k+1}/\mathcal{Z}_k$ is estimated using $M = 1$, and a new sample is obtained by using

Markov chain starting at the previous sample. The proof of this fact relies on the observation that the AIS procedure is just a simple importance sampling defined on the extended state space $X=(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_K)$. Indeed, AIS starts by first sampling from distribution $P_0(\mathbf{x})$ and then applying a series of transition operators $T_1, T_2, \ldots, T_{K-1}$ that "move" the sample through the intermediate distributions $P_k(\mathbf{x})$ towards the target distribution $P_K(\mathbf{x})$. The probability of the resultant state sequence X is given by:

$$\mathcal{Q}(X) \;=\; P_0(\mathbf{x}_1) \prod_{k=1}^{K-1} T_k(\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k). \tag{4.7}$$

We can view $\mathcal{Q}(\mathbf{X})$ as a proposal distribution for the target distribution $\mathcal{P}(\mathbf{X})$ on the extended space $\mathbf{X}$. This target distribution is defined by the reverse AIS procedure:

$$\mathcal{P}(X) \;=\; P_K(\mathbf{x}_K) \prod_{k=1}^{K-1} \widetilde{T}_k(\mathbf{x}_k \leftarrow \mathbf{x}_{k+1}), \tag{4.8}$$

where $\widetilde{T}_k$ are the reverse transition operators:

$$\widetilde{T}_k(\mathbf{x}' \leftarrow \mathbf{x}) \;=\; T_k(\mathbf{x} \leftarrow \mathbf{x}') \frac{P_k(\mathbf{x}')}{P_k(\mathbf{x})}. \tag{4.9}$$

If T_k is reversible then $\widetilde{T}_k$ is the same as T_k. Due to invariance of P_k with respect to T_k (Eq. 4.2), the reverse transition operators are valid transition probabilities, which ensures that the marginal distribution over $\mathbf{x}_K$ in Eq. 4.8 is the correct distribution of interest $P_K(\mathbf{x}_K)$. Denoting the unnormalized proposal and target distributions as $\mathcal{Q}^*(X) = Z_0 \mathcal{Q}(X)$ and $\mathcal{P}^*(X) = Z_K \mathcal{P}(X)$, the importance weight can be found using Eq. 4.1:

$$\begin{aligned} w \;&=\; \frac{\mathcal{P}^*(X)}{\mathcal{Q}^*(X)} = \frac{\mathcal{Z}_K \mathcal{P}(X)}{\mathcal{Z}_0 \mathcal{Q}(X)} = \frac{\mathcal{Z}_K P_K(\mathbf{x}_K) \prod_{i=1}^{K-1} \widetilde{T}_k(\mathbf{x}_k \leftarrow \mathbf{x}_{k+1})}{\mathcal{Z}_0 P_0(\mathbf{x}_1) \prod_{k=1}^{K-1} T_k(\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k)} \\ &=\; \frac{P_K^*(x_K)}{P_0^*(\mathbf{x}_1)} \prod_{k=1}^{K-1} \frac{P_k^*(\mathbf{x}_k)}{P_k^*(\mathbf{x}_{k+1})} = \prod_{k=1}^{K} \frac{P_k^*(\mathbf{x}_k)}{P_{k-1}^*(\mathbf{x}_k)}, \end{aligned} \tag{4.10}$$

which are the weights provided by the AIS algorithm. Observe that the Markov transition operators do not necessarily need to be ergodic. In particular, if we were to choose dumb transition operators that do nothing, $T_k(\mathbf{x}' \leftarrow \mathbf{x}) = \delta(\mathbf{x}' - \mathbf{x})$ for all k, we recover the original simple importance sampling procedure.

4.2.2 Ratios of Partition Functions of two RBM's

As discussed in section 2.1, a Restricted Boltzmann Machine has a two-layer architecture in which the visible, binary stochastic units $\mathbf{v} \in \{0,1\}^D$ are connected to hidden binary stochastic units $\mathbf{h} \in \{0,1\}^F$. The energy of the state $\{\mathbf{v}, \mathbf{h}\}$ is:

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}; \theta) \;&=\; -\mathbf{v}^\top W \mathbf{h} - \mathbf{b}^\top \mathbf{v} - \mathbf{a}^\top \mathbf{h} \\ &=\; -\sum_{i=1}^{D} \sum_{j=1}^{F} W_{ij} v_i h_j - \sum_{i=1}^{D} b_i v_i - \sum_{j=1}^{F} a_j h_j, \end{aligned} \tag{4.11}$$

where $\theta = \{W, \mathbf{b}, \mathbf{a}\}$ are the model parameters. The probability that the model assigns to a visible vector $\mathbf{v}$ is:

$$P(\mathbf{v};\theta) = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{v},\mathbf{h};\theta)\right)$$
$$= \frac{1}{\mathcal{Z}(\theta)} \exp(\mathbf{b}^\top \mathbf{v}) \prod_{j=1}^{F} \left(1 + \exp\left(a_j + \sum_{i=1}^{D} W_{ij} v_i\right)\right). \tag{4.12}$$

Suppose we have two RBM's with parameter values $\theta_A = \{W^A, \mathbf{b}^A, \mathbf{a}^A\}$ and $\theta_B = \{W^B, \mathbf{b}^B, \mathbf{a}^B\}$ that define probability distributions P_A and P_B over $\mathcal{V} \in \{0,1\}^D$. Each RBM can have a different number of hidden units $\mathbf{h}^A \in \{0,1\}^{F_A}$ and $\mathbf{h}^B \in \{0,1\}^{F_B}$. Using Eq. 4.12, we could define generic AIS intermediate distributions on the state space $\mathbf{x} = \{\mathbf{v}\}$, as defined in Eq. 4.3. However, sampling from these intermediate distributions would be slower than sampling from an RBM. Instead, we can make use of the bipartite structure of an RBM by introducing the following sequence of distributions for $k = 0, ..., K$:

$$P_k(\mathbf{v}) = \frac{P_k^*(\mathbf{v})}{\mathcal{Z}_k} = \frac{1}{\mathcal{Z}_k} \sum_{\mathbf{h}} \exp\left(-E_k(\mathbf{v},\mathbf{h})\right), \tag{4.13}$$

where the energy function is given by:

$$E_k(\mathbf{v},\mathbf{h}) = (1-\beta_k) E(\mathbf{v},\mathbf{h}^A;\theta_A) + \beta_k E(\mathbf{v},\mathbf{h}^B;\theta_B), \tag{4.14}$$

with $0 = \beta_0 < \beta_1 < ... < \beta_K = 1$ and $\mathbf{h} = \{\mathbf{h}^A, \mathbf{h}^B\}$. For $i = 0$, we have $\beta_0 = 0$ and so $P_0 = P_A$. Similarly, for $i = K$, we have $P_K = P_B$. For the intermediate values of k, we will have some interpolation between P_A and P_B.

Let us now define a Markov chain transition operator $T_k(\mathbf{v}' \leftarrow \mathbf{v})$ that leaves $P_k(\mathbf{v})$ invariant. Using Eqs. 4.13, 4.14, it is straightforward to derive a Gibbs sampler. The conditional distributions are given by logistic functions:

$$p(h_j^A = 1|\mathbf{v}) = g\left((1-\beta_k)\left(\sum_i W_{ij}^A v_i + a_j^A\right)\right), \tag{4.15}$$

$$p(h_j^B = 1|\mathbf{v}) = g\left(\beta_k\left(\sum_i W_{ij}^B v_i + a_j^B\right)\right), \tag{4.16}$$

$$p(v_i' = 1|\mathbf{h}) = g\left((1-\beta_k)\left(\sum_j W_{ij}^A h_j^A + b_i^A\right) + \beta_k\left(\sum_j W_{ij}^B h_j^B + b_i^B\right)\right), \tag{4.17}$$

where $g(x) = 1/(1+\exp(-x))$. Given $\mathbf{v}$, Eqs. 4.15, 4.16 are used to stochastically activate hidden units $\mathbf{h}^A$ and $\mathbf{h}^B$. Eq. 4.17 is then used to draw a new sample $\mathbf{v}'$ as shown in Fig. 4.1, left panel. Due to the special structure of RBM's, the cost of summing out $\mathbf{h}$ is linear in the number of hidden units. We can therefore easily evaluate:

$$P_k^*(\mathbf{v}) = \sum_{\mathbf{h}^A,\mathbf{h}^B} e^{(1-\beta_k)E(\mathbf{v},\mathbf{h}^A;\theta_A) + \beta_k E(\mathbf{v},\mathbf{h}^B;\theta_B)}$$
$$= e^{(1-\beta_k)\sum_i b_i^A v_i} \left[\prod_{j=1}^{F_A}\left(1 + e^{(1-\beta_k)\left(\sum_i W_{ij}^A v_i + a_j^A\right)}\right)\right] \times e^{\beta_k \sum_i b_i^B v_i} \left[\prod_{j=1}^{F_B}\left(1 + e^{\beta_k\left(\sum_i W_{ij}^B v_i + a_j^B\right)}\right)\right].$$

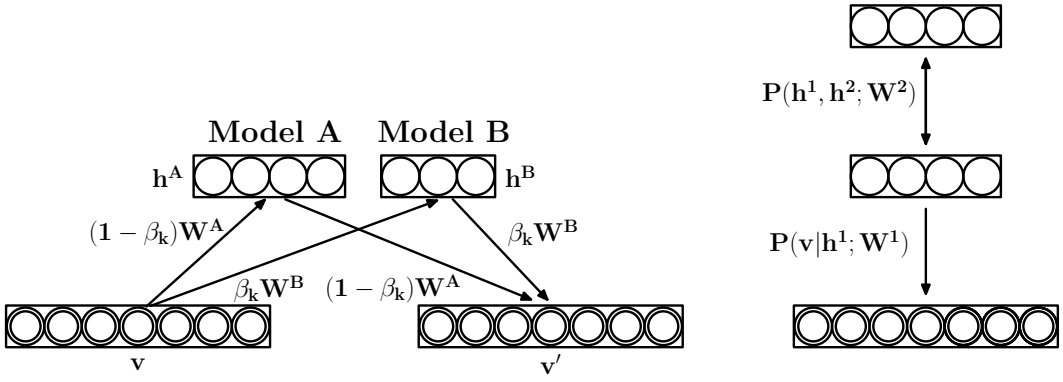

Figure 4.1: **Left:** The Gibbs transition operator $T_k(\mathbf{v}' \leftarrow \mathbf{v})$ leaves $P_k(\mathbf{v})$ invariant when estimating the ratio of partition functions $\mathcal{Z}_B/\mathcal{Z}_A$. **Right:** Two-hidden-layer Deep Belief Network as a generative model.

We will assume that the parameter values of each RBM satisfy $|\theta| < \infty$, in which case $P(\mathbf{v}) > 0$ for all $\mathbf{v} \in \mathcal{V}$. This will ensure that condition C1 of the AIS procedure is always satisfied. We have already shown that conditions C2 and C3 are satisfied. For condition C4, we can run a Gibbs sampler (Eqs. 2.7, 2.8) to generate samples from P_A. These sample points will not be independent, but the AIS estimator will still converge to the correct value, provided our Markov chain is ergodic (Neal [2001]). However, assessing the accuracy of this estimator can be difficult, as it depends on both the variance of the importance weights and on autocorrelations in the Gibbs sampler.

4.2.3 Estimating Partition Functions of RBM's

In the previous section we showed that we can use AIS to obtain an estimate of $\mathcal{Z}_B/\mathcal{Z}_A$. Consider an RBM with parameter vector $\theta_A = \{0, \mathbf{b}^A, \mathbf{a}^A\}$, i.e. an RBM with a zero weight matrix. From Eq. 4.12, we know:

$$\mathcal{Z}_A \;=\; \prod_j \left(1 + e^{a_j}\right) \prod_i \left(1 + e^{b_i}\right). \tag{4.18}$$

Moreover,

$$P_A(\mathbf{v}) \;=\; \prod_i p_A(v_i) = \prod_i 1/(1 + e^{-b_i}),$$

so we can draw exact independent samples from this "base-rate" RBM. AIS in this case allows us to obtain an *unbiased* estimate of the partition function $\mathcal{Z}_B$. This approach closely resembles simulated annealing (Kirkpatrick et al. [1983], Bertsimas and Tsitsiklis [1993]), since the intermediate distributions of Eq. 4.13 take form:

$$P_k(\mathbf{v}) \;=\; \frac{\exp\left((1-\beta_k)\left(\mathbf{v}^T\mathbf{b}^A\right)\right)}{\mathcal{Z}_k} \sum_{\mathbf{h}^B} \exp\left(-\beta_k E\left(\mathbf{v}, \mathbf{h}^B; \theta_B\right)\right).$$

We gradually change β_k (the inverse temperature) from 0 to 1, annealing from a simple "base-rate" model to the final complex model.

4.3 Estimating Lower Bounds for DBN's

Let us consider the Deep Belief Network with two layers of hidden features shown in Fig. 4.1, right panel. The model's joint distribution is:

$$P(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2) \;=\; P(\mathbf{v}|\mathbf{h}^1)\, P(\mathbf{h}^2, \mathbf{h}^1), \tag{4.19}$$

where $P(\mathbf{v}|\mathbf{h}^1)$ is the sigmoid belief network (Eq. 2.8), and $P(\mathbf{h}^1, \mathbf{h}^2)$ is the joint distribution defined by the second layer RBM. Note that $P(\mathbf{v}|\mathbf{h}^1)$ is normalized.

By explicitly summing out $\mathbf{h}^2$, we can easily evaluate an unnormalized probability $P^*(\mathbf{v}, \mathbf{h}^1) = \mathcal{Z}P(\mathbf{v}, \mathbf{h}^1)$. Using the approximating factorial distribution Q of Eq. 2.19, which we get as a byproduct of the greedy learning procedure, and the variational lower bound of Eq. 2.17, we obtain:

$$\log \sum_{\mathbf{h}^1} P(\mathbf{v}, \mathbf{h}^1) \geq \sum_{\mathbf{h}^1} Q(\mathbf{h}^1|\mathbf{v}) \log P^*(\mathbf{v}, \mathbf{h}^1) - \log \mathcal{Z} + \mathcal{H}(Q(\mathbf{h}^1|\mathbf{v})) = B(\mathbf{v}). \quad (4.20)$$

The entropy term $\mathcal{H}(\cdot)$ can be computed analytically, since Q is factorial. The partition function $\mathcal{Z}$ is estimated by running AIS on the top-level RBM. And the expectation term can be estimated by a simple Monte Carlo approximation:

$$\sum_{\mathbf{h}^1} Q(\mathbf{h}^1|\mathbf{v}) \log P^*(\mathbf{v}, \mathbf{h}^1) \approx \frac{1}{M} \sum_{i=1}^{M} \log P^*(\mathbf{v}, \mathbf{h}^{1(i)}), \quad (4.21)$$

where $\mathbf{h}^{1(i)} \sim Q(\mathbf{h}^1|\mathbf{v})$. The variance of this Monte Carlo estimator will be proportional to $1/M$ provided the variance of $\log P^*(\mathbf{v}, \mathbf{h}^{1(i)})$ is finite. In general, we will be interested in calculating the lower bound averaged over the test set containing N_t samples:

$$\begin{aligned} \frac{1}{N_t} \sum_{n=1}^{N_t} B(\mathbf{v}^n) &\approx \frac{1}{N_t} \sum_{n=1}^{N_t} \left[\frac{1}{M} \sum_{i=1}^{M} \log p^*(\mathbf{v}^n, \mathbf{h}^{1(i)}) + \mathcal{H}(Q(\mathbf{h}^1|\mathbf{v}^n)) \right] - \log \hat{\mathcal{Z}} \\ &= \hat{r}_B - \log \hat{\mathcal{Z}} = \hat{r}_{\text{Bound}}. \end{aligned} \quad (4.22)$$

In this case the variance of the estimator induced by the Monte Carlo approximation will asymptotically scale as $1/(N_t M)$. We will show in the experimental results section that the value of M can be small provided N_t is large. The error of the overall estimator $\hat{r}_{\text{Bound}}$ in Eq. 4.22 will be mostly dominated by the error in the estimate of $\log \mathcal{Z}$.

Estimating this lower bound for Deep Belief Networks with more layers is now straightforward. Consider a DBN with L hidden layers. The model's joint distribution and its approximate factorial posterior distribution Q are given by (see section 2.2):

$$\begin{aligned} P(\mathbf{v}, \mathbf{h}^1, ..., \mathbf{h}^L) &= P(\mathbf{v}|\mathbf{h}^1)...P(\mathbf{h}^{L-2}|\mathbf{h}^{L-1})P(\mathbf{h}^{L-1}, \mathbf{h}^L), \\ Q(\mathbf{h}^1, ..., \mathbf{h}^L|\mathbf{v}) &= Q(\mathbf{h}^1|\mathbf{v})Q(\mathbf{h}^2|\mathbf{v})...Q(\mathbf{h}^L|\mathbf{v}). \end{aligned}$$

The estimate of the lower bound can now be obtained by using Eqs. 4.20, 4.22. Note that most of the computation resources will be spent on estimating the partition function $\mathcal{Z}$ of the top level RBM.

4.4 Experimental Results

In our experiments we used the MNIST digit dataset, which contains 60,000 training and 10,000 test images of ten handwritten digits (0 to 9), with 28×28 pixels. The dataset was binarized: each pixel value was stochastically set to 1 in proportion to its pixel intensity. Samples from the training set are shown in Fig. 4.2, middle left panel. Annealed importance sampling requires us to specify β_k that define a sequence of intermediate distributions. In all of our experiments this sequence was chosen by quickly running a few preliminary experiments and picking the spacing of β_k so as to minimize the variance of the final importance weights. For the base-rate model the biases of the hidden units $\mathbf{a}^A$ were set to zero (see Eq. 4.18) and the biases of the visible units $\mathbf{b}^A$ were set by maximum likelihood, then smoothed to ensure that $P(\mathbf{v}) > 0, \forall\, \mathbf{v} \in \mathcal{V}$. In all experiments we obtained unbiased estimates of $\hat{\mathcal{Z}}$ and its standard deviation $\hat{\sigma}$ using Eqs. 4.4, 4.5. We also use natural logarithms, providing values in nats. Details of the Matlab code, used in experiments, can be found in Appendix A.

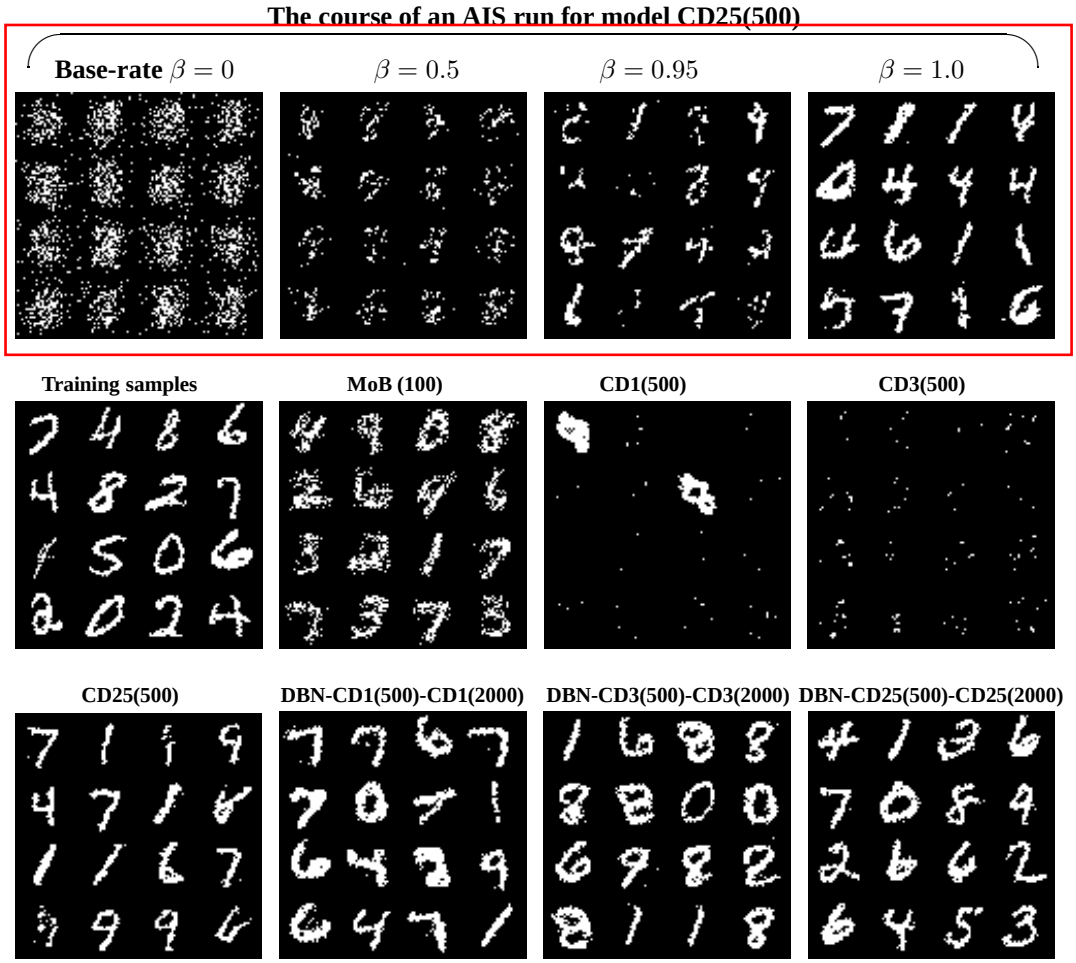

Figure 4.2: **Top row:** displays the course of 16 AIS runs for CD25(500) model by starting from a simple base-rate model and annealing to the final complex model. **Middle row:** First two panels show random samples from the training set and a mixture of Bernoullis model with 100 components. The last 2 panels display random samples generated from two RBM's. **Bottom row:** Random samples generated from an RBM and three DBN models.

4.4.1 Estimating partition functions of RBM's

In our first experiment we trained three RBM's on the MNIST digits. The first two RBM's had 25 hidden units and were learned using Contrastive Divergence (subsection 2.1) with T=1 and T=3 respectively. We call these models CD1(25) and CD3(25). The third RBM had 20 hidden units and was learned using CD with T=1. For all three models we can calculate the exact value of the partition function simply by summing out the 784 visible units for each configuration of the hiddens. For all three models we used 500 β_k spaced uniformly from 0 to 0.5, 4,000 β_k spaced uniformly from 0.5 to 0.9, and 10,000 β_k spaced uniformly from 0.9 to 1.0, with a total of 14,500 intermediate distributions.

Table 4.1 shows that AIS can rather quickly provide an accurate estimate of the partition function. For all three models, using only 10 AIS runs, we were able to obtain good estimates of partition functions in just 20 seconds on a Pentium Xeon 3.00GHz machine. For model CD1(25), however, the variance of the estimator was high, even with 100 AIS runs. Figure 4.3, top row, further reveals that as the number of annealing runs is increased, AIS can almost exactly recover the true value of the partition function across all three models.

We also directly estimated the ratio of normalizing constants of two RBM's that have different numbers of hidden units: CD1(20) and CD1(25). This estimator could be used to do complexity control. In detail, using 100 AIS runs with uniform spacing of 10,000 β_k, we obtained $\log \hat{r}_{\text{AIS}} =$

Table 4.1: Results of estimating log-partition functions of RBM's along with the estimates of the average training and test log-probabilities. For all models we used 14,500 intermediate distributions and 100 AIS runs.

	True	Estimates			Time	Avg. Test log-prob.		Avg. Train log-prob.	
	$\log \mathcal{Z}$	$\log \hat{\mathcal{Z}}$	$\log(\hat{\mathcal{Z}} \pm \hat{\sigma})$	$\log(\hat{\mathcal{Z}} \pm 3\hat{\sigma})$	(mins)	true	estimate	true	estimate
CD1(25)	255.41	256.52	$255.00, 257.10$	$0.0000, 257.73$	3.3	-151.57	-152.68	-152.35	-153.46
CD3(25)	307.47	307.63	$307.44, 307.79$	$306.91, 308.05$	3.3	-143.03	-143.20	-143.94	-144.11
CD1(20)	279.59	279.57	$279.43, 279.68$	$279.12, 279.87$	3.1	-164.52	-164.50	-164.89	-164.87
CD1(500)	—	350.15	$350.04, 350.25$	$349.77, 350.42$	10.4	—	-125.53	—	-122.86
CD3(500)	—	280.09	$279.99, 280.17$	$279.76, 280.33$	10.4	—	-105.50	—	-102.81
CD25(500)	—	451.28	$451.19, 451.37$	$450.97, 451.52$	10.4	—	-86.34	—	-83.10

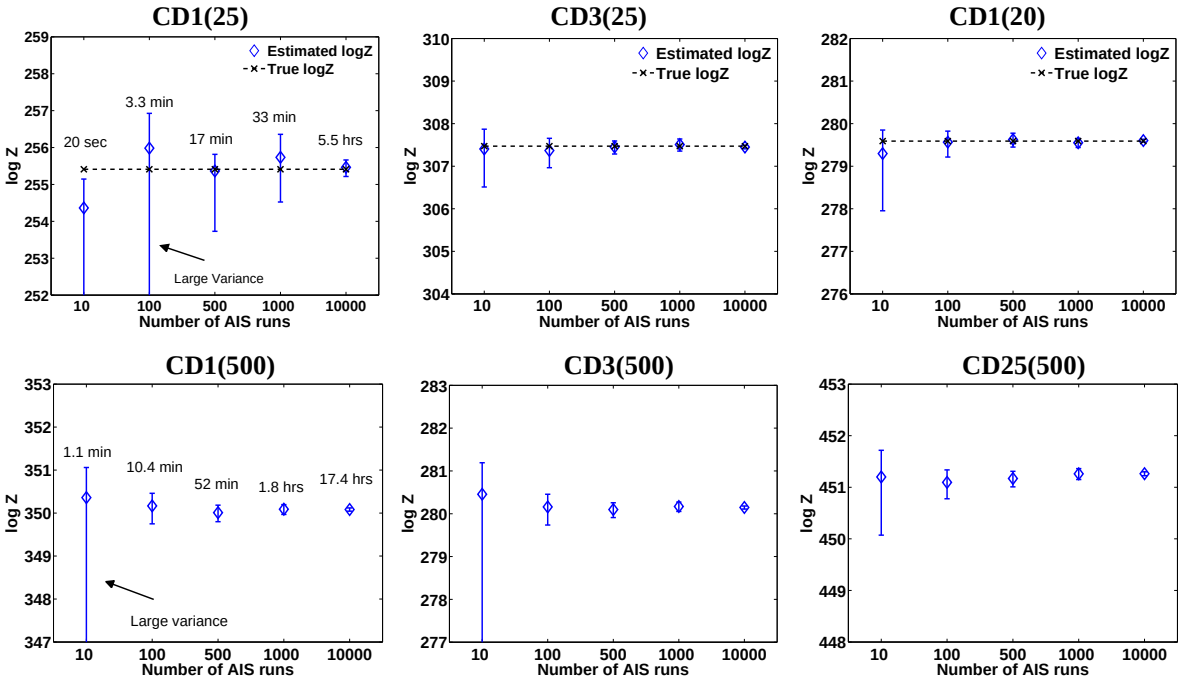

Figure 4.3: Estimates of the log-partition functions $\log \hat{\mathcal{Z}}$ as we increase the number of annealing runs. The error bars show $\log(\hat{\mathcal{Z}} \pm 3\hat{\sigma})$.

$\log(\mathcal{Z}_{\text{CD1(20)}}/\mathcal{Z}_{\text{CD1(25)}}) = -24.49$ with an error estimate $\log(\hat{r}_{\text{AIS}} \pm 3\hat{\sigma}) = (-24.19, -24.93)$. Each sample from CD1(25) was generated by starting a Markov chain at the previous sample and running it for 10,000 steps. Compared to the true value of -24.18, this result suggests that our estimates may have a small systematic error due to the Markov chain failing to visit some modes.

Our second experiment consisted of training two more realistic models: CD1(500) and CD3(500). We used exactly the same spacing of β_k as before and exactly the same base-rate model. Results are shown in table 4.1, bottom row. For each model we were able to get what appears to be a rather accurate estimate of $\mathcal{Z}$. Of course, we are relying on an empirical estimate of AIS's accuracy, which could potentially be misleading. Nonetheless, Fig. 4.3, bottom panel, shows that as we increase the number of annealing runs, the value of the estimator does not oscillate drastically.

While performing these tests, we observed that Contrastive Divergence learning with T=3 results in a considerably better generative model than CD learning with T=1: the difference of 20 nats is striking! Clearly, the widely used practice of CD1 learning is a rather poor "substitute" for maximum likelihood

Table 4.2: Results of estimating lower bounds $\hat{r}_{\text{Bound}}$ (Eq. 4.22) on the average training and test log-probabilities for DBN's. On average, the total error of the estimator is about $\pm$ 2 nats.

	Estimates		Avg. bound on log-prob.			
	$\log \hat{\mathcal{Z}}$	$\log(\hat{\mathcal{Z}} \pm 3\hat{\sigma})$	Test	$\pm$ 3std	Train	$\pm$ 3std.
DBN-CD1(500)-CD1(2000)	277.33	275.90, 277.90	-100.64	± 0.77	-97.67	± 0.30
DBN-CD3(500)-CD3(2000)	229.92	229.01, 230.23	-98.29	± 0.75	-94.86	± 0.29
DBN-CD25(500)-CD25(2000)	466.70	465.86, 467.35	-86.22	± 0.67	-82.47	± 0.25

learning. Inspired by this result, we trained a model by starting with T=1, and gradually increasing T to 25 during the course of CD learning, as suggested by Carreira-Perpinan and Hinton [2005]. We call this model CD25(500). Training this model was computationally much more demanding. However, the estimate of the average test log-probability for this model was about -86, which is 39 and 19 nats better than the CD1(500) and CD3(500) models respectively. Figure 4.2 shows samples generated from all three models by randomly initializing binary states of the visible units and running the Gibbs sampler for 100,000 steps. Certainly, samples generated by CD25(500) look much more like the real handwritten digits, than either CD1(500) or CD3(500).

Using 10,000 β_k and 100 annealing runs, we also obtained an estimate of the log ratio of two partition functions $\hat{r}_{\text{AIS}} = \log \mathcal{Z}_{\text{CD25(500)}}/\mathcal{Z}_{\text{CD3(500)}} = 169.96$. The estimates of the individual log-partition functions were $\log \hat{\mathcal{Z}}_{\text{CD25(500)}} = 451.28$ and $\log \hat{\mathcal{Z}}_{\text{CD3(500)}} = 280.09$, in which case the log ratio is $451.28 - 280.09 = 171.19$. This is in agreement (to within three standard deviations) with the direct estimate of the ratio, $\hat{r}_{\text{AIS}} = 169.96$.

For a simple comparison we also trained several mixture of Bernoullis models (see Fig. 4.2, middle left panel) with 10, 100, and 500 components. The corresponding average test log-probabilities were -168.95, -142.63, and -137.64. The data generated from the mixture model with 100 components looks better than either CD1(500) or CD3(500), although our quantitative results reveal this is due to overfitting. Restricted Boltzmann Machines give much higher density to test data.

4.4.2 Estimating lower bounds for DBN's

We trained three Deep Belief Networks with two hidden layers. The first model, which we call DBN-CD1(500)-CD1(2000), was greedily learned by freezing the parameter vector of the CD1(500) model and fitting the 2^{nd} layer RBM with 2000 hidden units using CD with T=1. Similarly, the other two models, DBN-CD3(500)-CD3(2000) and DBN-CD25(500)-CD25(2000), added 2000 hidden units on top of CD3(500) and CD25(500), and were learned using CD with T=3 and T=25 respectively. Training the DBN's took roughly three times longer than the RBM's.

Table 4.2 shows the results. We used 15,000 intermediate distributions and 500 annealing runs to estimate the partition function of the 2^{nd} layer RBM, which took 2.3 hours. We further used M=5 samples from the approximating distribution $Q(\mathbf{h}|\mathbf{v})$ to do a simple Monte Carlo approximation of Eq. 4.21. Setting M=100 did not make much difference. Table 4.2 also reports the empirical error in the estimate of the lower bound $\hat{r}_{\text{Bound}}$. From Eq. 4.22, we have $\text{Var}(\hat{r}_{\text{Bound}}) = \text{Var}(\hat{r}_B) + \text{Var}(\log \hat{\mathcal{Z}})$, both of which are shown in table 4.2. Note that the two-hidden-layer Deep Belief Networks, DBN-CD1(500)-CD1(2000) and DBN-CD3(500)-CD3(2000), significantly outperform their single layer counterparts: CD1(500) and CD3(500). Adding a second layer for those two models improves model performance by at least 25 and 7 nats. Figure 4.2 also shows the dramatic difference between samples generated by the single layer RBM's and corresponding two-hidden-layer DBN's.

Table 4.3: Results of estimating lower bounds on the average test log-probabilities for DBN's. On average, the total error of the estimator is about $\pm$ 2 nats.

	No 2^{nd}	2^{nd}layer					
1^{st}layer	layer	CD1(500)	CD3(500)	CD25(500)	CD1(2000)	CD3(2000)	CD25(2000)
DBN-CD1(500)-	-125.53	-101.40	-97.54	-89.52	-100.64	-94.44	-87.13
DBN-CD3(500)-	-105.50	-100.27	-95.21	-88.38	-101.62	-98.29	-88.92
DBN-CD25(500)-	-86.34	-101.55	-98.38	-86.90	-98.57	-96.97	-86.22
DBN-CD1(2000)-	-122.84	-134.81	-119.45	-90.50	-116.44	-100.08	-92.53
DBN-CD3(2000)-	-100.92	-112.81	-107.49	-89.48	-106.67	-98.73	-91.25
DBN-CD25(2000)-	-86.26	-121.83	-112.64	-89.96	-102.24	-97.87	-88.29

Surprisingly, the greedy learning of DBN's does not appear to suffer severely from overfitting. For single layer models, the difference between the estimates of training and test log-probabilities was about 3 nats. For DBN's, the corresponding difference in the estimates of the lower bounds was about 4 nats, even though adding a second layer introduced over twice as many (or one million) new parameters.

The result of our experiments for DBN-CD25(500)-CD25(2000), however, was very different. For this model, on the test data we obtained $\hat{r}_{\text{Bound}} = -86.22$. This is comparable to the estimate of -86.34 for the average test log-probability of the CD25(500) model. Clearly, we cannot confidently assert that the DBN is a better generative model compared to the carefully trained single layer RBM. This peculiar result also supports previous claims that if the first level RBM already models data well, adding extra layers will not help (LeRoux and Bengio [2008], Hinton et al. [2006]).

To estimate how loose the variational bound is, we randomly sampled 50 test cases, 5 of each class, and ran AIS for each test case to estimate the true test log probability. Computationally, this is equivalent to estimating 50 additional partition functions. Our estimate of the variational bound was 87.05 per test case. The estimate of the true test log probability was 85.20, showing that the bound is actually rather tight.

To further examine the effect that CD learning has on pretraining a stack of RBM's, we trained additional 36 models. Table 4.3 shows results of estimating the lower bound on the average test log-probabilities. These results support our two previous observations. First, when training a lower-level RBM with CD1 or CD3, adding an extra layer generally improves model performance. Second, when a lower-level RBM already models data well, adding an extra layer does not help. Performance of models DBN-CD1(2000)-CD1(500), DBN-CD3(2000)-CD3(500), and DBN-CD25(2000)-CD25(500) further reveals that decreasing the number of hidden units per layer can actually hurt model performance.

As an additional test, instead of randomly initializing parameters of the 2^{nd} layer RBM, we initialized it by using the same parameters as the 1^{st} layer RBM but with hidden and visible units switched, so that the 2^{nd} and 3^{rd} layers contained 500 and 784 hidden units. This initialization ensures that the distribution over the visible units $\mathbf{v}$ defined by the two-hidden-layer DBN is *exactly the same* as the distribution over $\mathbf{v}$ defined by the 1^{st} layer RBM (see section 2.2). Therefore, after learning parameters of the 2^{nd} layer RBM, the lower bound on the training data log-likelihood should improve. After carefully training the second level RBM, our estimate of the lower bound on the test log-probability was -85.97. Once again, we cannot confidently claim that adding an extra layer in this case yields better generalization.

4.5 Discussion

As we discussed in chapter 2, under some strong assumptions, each additional layer of a DBN increases a lower bound on the log-probability of the *training* data, provided the number of hidden units per layer

does not decrease. However, assessing generalization performance of these generative models is quite difficult, since computing the exact probability of a test vector requires enumeration over an exponential number of terms. In this chapter we developed an Annealed Importance Sampling procedure that takes advantage of the bipartite structure of the RBM. This can provide a good estimate of the partition function in a reasonable amount of computer time. Furthermore, we showed that this estimator, along with approximate inference, can be used to obtain an estimate of the lower bound on the log-probability of the *test* data. This allowed us to obtain some quantitative evaluation of the generalization performance of these deep hierarchical models.

There are some disadvantages to using AIS. There is a need to specify the β_k that define a sequence of intermediate distributions. The number and the spacing of β_k will be problem dependent and will affect the variance of the estimator. We also have to rely on the empirical estimate of AIS accuracy, which could potentially be very misleading (Neal [2001, 2005]). Even though AIS provides an unbiased estimator of $\mathcal{Z}$, it may often give large underestimates and occasionally give even larger overestimates. So in practice, it is more likely to underestimate the true value of the partition function, which will result in an overestimate of the log-probability. But these drawbacks should not result in disfavoring the use of AIS for RBM's and DBN's: it is much better to have a slightly unreliable estimate than no estimate at all, or an extremely indirect estimate, such as discriminative performance (Hinton et al. [2006], Bengio et al. [2007]). We also find Annealed Importance Sampling and other stochastic methods attractive because they can just as easily be applied to undirected graphical models that generalize RBM's and DBN's to exponential family distributions. This will allow future application to models of real-valued data, such as image patches (Osindero and Hinton [2008]), or count data (Gehler et al. [2006]).

Another alternative would be to employ deterministic approximations (Yedidia et al. [2005]) or deterministic upper bounds (Wainwright et al. [2005]) on the log-partition function. However, for densely connected MRF's, we would not expect these methods to work well. My own findings (Salakhutdinov [2008]) show that these methods provide quite inaccurate estimates of (or very loose upper bounds on) the partition function, even for small RBM's when *trained on real data* that has many modes with similar probabilities.

Chapter 5

Deep Boltzmann Machines

In this chapter we present a new learning algorithm for a different type of hierarchical probabilistic model: a Deep Boltzmann Machine (DBM). Unlike Deep Belief Networks, a DBM is a type of Markov random field, or undirected graphical model, where all connections between layers are undirected. Deep Boltzmann Machines are interesting for several reasons. First, like Deep Belief Networks, DBM's have the potential of learning internal representations that become increasingly complex at higher layers, which is a promising way of solving object and speech recognition problems. High-level representations can be built from a large supply of unlabeled sensory inputs and the very limited labeled data can then be used to only slightly fine-tune the model for a specific task at hand. Second, unlike Deep Belief Networks and many other models with deep architectures (Ranzato et al. [2007], Vincent et al. [2008], Serre et al. [2007]), the approximate inference procedure, in addition to a bottom-up pass, can incorporate top-down feedback, allowing Deep Boltzmann Machines to better propagate uncertainty about ambiguous inputs. This is perhaps the most important distinguishing characteristic of this model. Finally, in the presence of enormous amounts of sensory data, the entire model can be trained online, processing one example at a time.

5.1 Introduction

The original learning algorithm for Boltzmann machines, introduced by Hinton and Sejnowski [1983], used randomly initialized Markov chains in order to perform approximate inference to estimate the model's expected sufficient statistics. This learning procedure, however, was too slow to be practical. There have been many attempts in developing efficient learning and inference algorithms for Boltzmann machines (see Welling and Hinton [2002], Welling and Teh [2003], Zhu and Liu [2002] and references therein), but none of them have proven to be useful for large-scale problems in machine learning.

In this chapter we present an efficient learning procedure for fully general Boltzmann machines. Approximate inference can be performed using variational approaches, such as mean-field. Learning can then be carried out by applying a stochastic approximation procedure that uses Markov chain Monte Carlo (MCMC) to approximate a model's expected sufficient statistics. The MCMC based approximation procedure provides nice asymptotic convergence guarantees and belongs to the general class of approximation algorithms of Robbins–Monro type (Robbins and Monro [1951], Younes [1989]). This unusual combination of variational methods and MCMC is essential for creating a fast learning algorithm for general Boltzmann machines, or, more generally, undirected graphical models in the exponential family (Wainwright and Jordan [2003]). If the connections between hidden units are restricted in such a way that the hidden units form multiple layers, it is possible to modify the greedy learning algorithm for Restricted Boltzmann Machines so that they can be used to initialize the parameters of a

Deep Boltzmann Machine before applying our new learning procedure. Finally, we present results on the MNIST and NORB datasets showing that Boltzmann machines learn good generative models and perform well on handwritten digit and visual object recognition tasks.

5.2 Boltzmann Machines (BM's)

A Boltzmann machine is a network of symmetrically coupled stochastic binary units. It contains a set of visible units $\mathbf{v} \in \{0,1\}^D$, and a set of hidden units $\mathbf{h} \in \{0,1\}^F$ (see Fig. 5.1, left panel), that model complicated, higher-order correlations between the visible units. The energy of the state $\{\mathbf{v}, \mathbf{h}\}$ is defined as:

$$E(\mathbf{v}, \mathbf{h}; \theta) \;=\; -\frac{1}{2}\mathbf{v}^\top L\mathbf{v} - \frac{1}{2}\mathbf{h}^\top J\mathbf{h} - \mathbf{v}^\top W\mathbf{h}, \tag{5.1}$$

where $\theta = \{W, L, J\}$ are the model parameters[1]: W, L, J represent visible-to-hidden, visible-to-visible, and hidden-to-hidden symmetric interaction terms. The diagonal elements of L and J are set to 0. The probability that the model assigns to a visible vector $\mathbf{v}$ is:

$$P(\mathbf{v}; \theta) \;=\; \frac{P^*(\mathbf{v}; \theta)}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{v}, \mathbf{h}; \theta)\right), \tag{5.2}$$

$$\mathcal{Z}(\theta) \;=\; \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{v}, \mathbf{h}; \theta)\right), \tag{5.3}$$

where P^* denotes unnormalized probability, and $\mathcal{Z}(\theta)$ is the partition function. The *conditional* distributions over hidden and visible units are given by:

$$p(h_j = 1 | \mathbf{v}, \mathbf{h}_{-j}) \;=\; g\left(\sum_i W_{ij} v_i + \sum_{m \neq j} J_{jm} h_j\right), \tag{5.4}$$

$$p(v_i = 1 | \mathbf{h}, \mathbf{v}_{-i}) \;=\; g\left(\sum_j W_{ij} h_j + \sum_{k \neq i} L_{ik} v_j\right), \tag{5.5}$$

where $g(x) = 1/(1 + \exp(-x))$ is the logistic function and $\mathbf{x}_{-i}$ denotes a vector $\mathbf{x}$ but with x_i omitted. The parameter updates, originally derived by Hinton and Sejnowski [1983], that are needed to perform gradient ascent in the log-likelihood can be obtained from Eq. 5.2:

$$\begin{aligned}
\Delta W \;&=\; \alpha\left(\mathrm{E}_{P_{\mathrm{data}}}[\mathbf{v}\mathbf{h}^\top] - \mathrm{E}_{P_{\mathrm{model}}}[\mathbf{v}\mathbf{h}^\top]\right), \\
\Delta L \;&=\; \alpha\left(\mathrm{E}_{P_{\mathrm{data}}}[\mathbf{v}\mathbf{v}^\top] - \mathrm{E}_{P_{\mathrm{model}}}[\mathbf{v}\mathbf{v}^\top]\right), \\
\Delta J \;&=\; \alpha\left(\mathrm{E}_{P_{\mathrm{data}}}[\mathbf{h}\mathbf{h}^\top] - \mathrm{E}_{P_{\mathrm{model}}}[\mathbf{h}\mathbf{h}^\top]\right),
\end{aligned}$$

where α is a learning rate. $\mathrm{E}_{P_{\mathrm{data}}}[\cdot]$ denotes an expectation with respect to the completed data distribution $P_{\mathrm{data}}(\mathbf{h}, \mathbf{v}; \theta) = P(\mathbf{h}|\mathbf{v}; \theta) P_{\mathrm{data}}(\mathbf{v})$, with $P_{\mathrm{data}}(\mathbf{v}) = \frac{1}{N}\sum_n \delta(\mathbf{v} - \mathbf{v}_n)$ representing the empirical distribution, and $\mathrm{E}_{P_{\mathrm{model}}}[\cdot]$ is an expectation with respect to the distribution defined by the model (Eq. 5.2). We will sometimes refer to $\mathrm{E}_{P_{\mathrm{data}}}[\cdot]$ as the *data-dependent expectation*, and $\mathrm{E}_{P_{\mathrm{model}}}[\cdot]$ as the *model's expectation*.

[1] We have omitted the bias terms for clarity of presentation

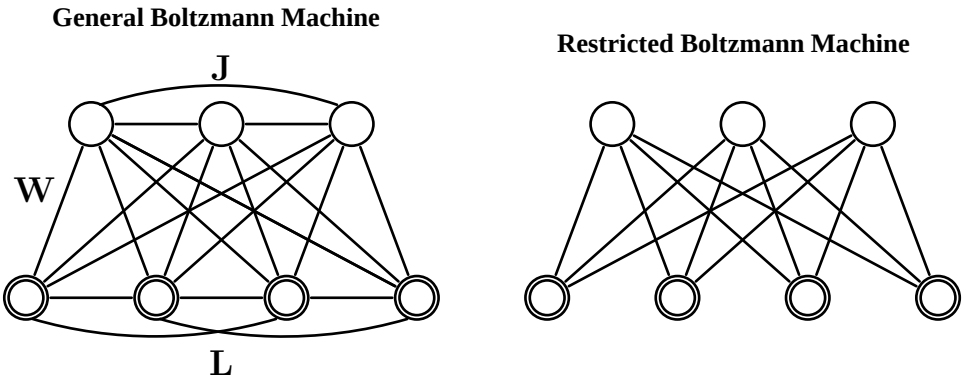

Figure 5.1: **Left:** A general Boltzmann machine. The top layer represents a vector of stochastic binary "hidden" features and the bottom layer represents a vector of stochastic binary "visible" variables. **Right:** A Restricted Boltzmann Machine with no hidden-to-hidden and no visible-to-visible connections.

Exact maximum likelihood learning in this model is intractable. The exact computation of the data-dependent expectation takes time that is exponential in the number of hidden units, whereas the exact computation of the model's expectation takes time that is exponential in the number of hidden and visible units. Hinton and Sejnowski [1983] proposed an algorithm that uses Gibbs sampling (Geman and Geman [1984]) to approximate both expectations. For each iteration of learning, a separate Markov chain is run for every training data vector to approximate $\mathrm{E}_{P_{\mathrm{data}}}[\cdot]$, and an additional chain is run to approximate $\mathrm{E}_{P_{\mathrm{model}}}[\cdot]$. The main problem with this learning algorithm is the time required to approach the stationary distribution, especially when estimating the model's expectations, since the Gibbs chain may need to explore a highly multimodal energy landscape. This is typical when modeling real-world distributions, such as datasets of images, in which almost all of the possible images have extremely low probability, but there are many very different images that occur with quite similar probabilities.

Setting both J=0 and L=0 recovers the Restricted Boltzmann Machine (RBM) model (see Fig. 5.1, right panel). Although exact maximum likelihood learning in RBM's is still intractable, learning can be carried out efficiently using Contrastive Divergence (CD) (Hinton [2002]). It was further observed (Welling and Hinton [2002], Hinton [2002]) that for Contrastive Divergence to perform well, it is important to obtain exact samples from the conditional distribution $P(\mathbf{h}|\mathbf{v};\theta)$, which is intractable when learning full Boltzmann machines.

5.2.1 A Stochastic Approximation Procedure for Estimating the Model's Expectations

Instead of using CD learning, it is possible to make use of a stochastic approximation procedure (SAP) that uses MCMC methods to stochastically approximate the model's expectations (Younes [1989, 2000], Neal [1992], Yuille [2004], Tieleman [2008]). SAP belongs to the general class of well-studied stochastic approximation algorithms of the Robbins–Monro type (Younes [1989], Robbins and Monro [1951]). To be more precise, let us consider the following canonical form of the exponential family associated with the sufficient statistics vector Φ:

$$P(\mathbf{x};\theta) \;=\; \frac{1}{\mathcal{Z}(\theta)} \exp\left(\theta^{\top}\Phi(\mathbf{x})\right). \tag{5.6}$$

The derivative of the log-likelihood for an observation $\mathbf{x}_0$ with respect to parameter vector θ is:

$$\frac{\partial \log P(\mathbf{x}_0;\theta)}{\partial\theta} \;=\; \Phi(\mathbf{x}_0) - \mathrm{E}_{P_{\mathrm{model}}}[\Phi(\mathbf{x})]. \tag{5.7}$$

The idea behind learning parameter vector θ using SAP is straightforward. Let θ^t and $\mathbf{x}^t$ be the current parameters and the state. Then $\mathbf{x}^t$ and θ^t are updated sequentially as follows:

Algorithm 4 Stochastic Approximation Algorithm.

1: Randomly initialize θ^0 and M sample particles $\{\tilde{\mathbf{x}}^{0,1}, ..., \tilde{\mathbf{x}}^{0,M}\}$.
2: **for** $t = 0 : T$ (number of iterations) **do**
3: **for** $i = 1 : M$ (number of parallel Markov chains) **do**
4: Sample $\tilde{\mathbf{x}}^{t+1,i}$ given $\tilde{\mathbf{x}}^{t,i}$ using transition operator $T_{\theta^t}(\tilde{\mathbf{x}}^{t+1,i} \leftarrow \tilde{\mathbf{x}}^{t,i})$.
5: **end for**
6: Update: $\theta^{t+1} = \theta^t + \alpha_t \left[\Phi(\mathbf{x}_0) - \frac{1}{M}\sum_{m=1}^{M} \Phi(\tilde{\mathbf{x}}^{t+1,m})\right]$.
7: Decrease α_t.
8: **end for**

- Given $\mathbf{x}^t$, a new state $\mathbf{x}^{t+1}$ is sampled from the transition operator $T_{\theta^t}(\mathbf{x}^{t+1} \leftarrow \mathbf{x}^t)$ that leaves $P(\cdot; \theta^t)$ invariant.
- A new parameter θ^{t+1} is then obtained by replacing the intractable model's expectation $\mathrm{E}_{P_{\mathrm{model}}}[\Phi(\mathbf{x})]$ with $\Phi(\mathbf{x}^t)$

In practice, we typically maintain a set of M sample points $X^t = \{\tilde{\mathbf{x}}^{t,1}, ..., \tilde{\mathbf{x}}^{t,M}\}$, which we will often refer to as sample particles. In this case, the intractable model's expectation is replaced by the sample average $1/M \sum_{m=1}^{M} \Phi(\tilde{\mathbf{x}}^{t+1,m})$. The procedure is summarized in Algorithm 4.

The proof of convergence of these algorithms relies on the following basic decomposition. First, the gradient of the log-likelihood function takes the form:

$$S(\theta) \;=\; \frac{\partial \log P(\mathbf{x}; \theta)}{\partial \theta} \;=\; \Phi(\mathbf{x}_0) - \mathrm{E}_{P_{\mathrm{model}}}[\Phi(\mathbf{x})]. \tag{5.8}$$

The parameter update rule then takes the following form:

$$\begin{aligned} \theta^{t+1} \;&=\; \theta^t + \alpha_t \left[\Phi(\mathbf{x}_0) - \frac{1}{M}\sum_{m=1}^{M} \Phi(\tilde{\mathbf{x}}^{t+1,m})\right] && (5.9) \\ &=\; \theta^t + \alpha_t S(\theta^t) + \alpha_t \left[\mathrm{E}_{P_{\mathrm{model}}}[\Phi(\mathbf{x})] - \frac{1}{M}\sum_{m=1}^{M} \Phi(\tilde{\mathbf{x}}^{t+1,m})\right] \\ &=\; \theta^t + \alpha_t S(\theta^t) + \alpha_t \epsilon_{t+1}. && (5.10) \end{aligned}$$

The first term is the discretization of the ordinary differential equation $\dot{\theta} = S(\theta)$. The algorithm is therefore a perturbation of this discretization with the noise term ϵ. The proof then proceeds by showing that the noise term is not too large.

Precise sufficient conditions that ensure almost sure convergence to an asymptotically stable point of $\dot{\theta} = S(\theta)$ are given in Younes [1989, 2000], Yuille [2004]. One necessary condition requires the learning rate to decrease with time, so that $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$. This condition can, for example. be satisfied simply by setting $\alpha_t = 1/(t_0 + t)$. Other conditions ensure that the speed of convergence of the Markov chain, governed by the transition operator T_θ, does not decrease too fast as θ tends to infinity, and that the noise term ϵ in the update of Eq. 5.9 is bounded. Typically, in practice, the sequence $|\theta^t|$ is bounded, and the Markov chain, governed by the transition kernel T_θ, is ergodic. Together with the condition on the learning rate, this ensures almost sure convergence of SAP to an asymptotically stable point of $\dot{\theta} = S(\theta)$.

The intuition behind why this procedure works is the following. As the learning rate becomes sufficiently small compared with the mixing rate of the Markov chain, this "persistent" chain will always

stay very close to the stationary distribution, even if it is only run for a few MCMC steps per parameter update. Samples from the persistent chain will be highly correlated for successive parameter updates. However, if the learning rate is sufficiently small, the chain will mix before the parameters have changed enough to significantly alter the value of the estimator. This technique for learning Boltzmann machines was used by Neal [1992]. When applied to learning RBM's, Tieleman [2008] further shows that this stochastic approximation algorithm, also termed Persistent Contrastive Divergence, performs quite well compared to Contrastive Divergence learning.

5.2.2 A Variational Approach to Estimating the Data-Dependent Expectations

In variational learning (Hinton and Zemel [1994], Neal and Hinton [1998], Jordan et al. [1999]), the true posterior distribution over latent variables $P(\mathbf{h}|\mathbf{v};\theta)$ for each training vector $\mathbf{v}$, is replaced by an approximate posterior $Q(\mathbf{h}|\mathbf{v};\mu)$ and the parameters are updated to maximize the variational lower bound on the log-likelihood:

$$\begin{aligned}\log P(\mathbf{v};\theta) \;&\geq\; \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{v};\mu)\log P(\mathbf{v},\mathbf{h};\theta) + \mathcal{H}(Q) \\ &=\; \log P(\mathbf{v};\theta) - \mathrm{KL}\left[Q(\mathbf{h}|\mathbf{v};\mu)||P(\mathbf{h}|\mathbf{v};\theta)\right], \end{aligned} \tag{5.11}$$

where $\mathcal{H}(\cdot)$ is the entropy functional. Variational learning has the nice property that in addition to trying to maximize the log-likelihood of the training data, it tries to find parameters that minimize the Kullback-Leibler divergences between the approximating and true posteriors. Using a naive mean-field approach, we choose a fully factorized distribution in order to approximate the true posterior: $Q(\mathbf{h};\mu) = \prod_{j=1}^{F} q(h_i)$, with $q(h_i = 1) = \mu_i$ and F is the number of hidden units. The lower bound on the log-probability of the data takes the following form:

$$\begin{aligned}\log P(\mathbf{v};\theta) \;\geq\; \frac{1}{2}\sum_{i,k} L_{ik}v_i v_k + \frac{1}{2}\sum_{j,m} J_{jm}\mu_j\mu_m + \sum_{i,j} W_{ij}v_i\mu_j - \log\mathcal{Z}(\theta) \\ + \sum_j \left[\mu_j\log\mu_j + (1-\mu_j)\log(1-\mu_j)\right].\end{aligned}$$

The learning proceeds by first maximizing this lower bound with respect to the variational parameters μ for fixed θ, which results in the mean-field fixed-point equations:

$$\mu_j \leftarrow g\left(\sum_i W_{ij}v_i + \sum_{m\neq j} J_{mj}\mu_m\right). \tag{5.12}$$

This is followed by applying SAP to update the model parameters θ. We emphasize that variational approximations cannot be used for approximating the expectations with respect to the model distribution in the Boltzmann machine learning rule, as attempted in Galland [1991], because the minus sign would cause variational learning to adjust the parameters so as to *maximize* the divergence between the approximating and true distributions. If, however, a Markov chain is used to estimate the model's expectations, variational learning can be applied for estimating the data-dependent expectations.

The choice of naive mean-field was deliberate. First, the convergence is usually fast, which greatly facilitates learning. Second, for applications such as the interpretation of images or speech, we expect the posterior over hidden states *given the data* to have a single mode, so simple and fast variational approximations such as mean-field should be adequate. Indeed, making the true posterior unimodal by sacrificing some log-likelihood could be advantageous for a system that will use the posterior to

Algorithm 5 Boltzmann Machine Learning Procedure.

Given: a training set of N binary data vectors $\{\mathbf{v}\}_{n=1}^{N}$, and M, the number of samples.
Randomly initialize parameter vector θ^0 and M samples: $\{\tilde{\mathbf{v}}^{0,1}, \tilde{\mathbf{h}}^{0,1}\}, ..., \{\tilde{\mathbf{v}}^{0,M}, \tilde{\mathbf{h}}^{0,M}\}$.
for $t = 0$ to T (number of iterations) **do**

    // Variational Inference:
    for each training example $\mathbf{v}^n$, $n = 1$ to N **do**
        Randomly initialize μ and run mean-field updates until convergence:
            $\mu_j \leftarrow g\left(\sum_i W_{ij} v_i + \sum_{m \neq j} J_{mj} \mu_m\right)$.
        Set $\mu^n = \mu$.
    end for

    // Stochastic Approximation:
    for each sample $m = 1$ to M **do**
        Sample $(\tilde{\mathbf{v}}^{t+1,m}, \tilde{\mathbf{h}}^{t+1,m})$ given $(\tilde{\mathbf{v}}^{t,m}, \tilde{\mathbf{h}}^{t,m})$ by running a Gibbs sampler (Eqs. 5.4, 5.5).
    end for

    // Parameter Update:
    $W^{t+1} = W^t + \alpha_t \left(\frac{1}{N} \sum_{n=1}^{N} \mathbf{v}^n (\mu^n)^\top - \frac{1}{M} \sum_{m=1}^{M} \tilde{\mathbf{v}}^{t+1,m} (\tilde{\mathbf{h}}^{t+1,m})^\top \right)$.
    $J^{t+1} = J^t + \alpha_t \left(\frac{1}{N} \sum_{n=1}^{N} \mu^n (\mu^n)^\top - \frac{1}{M} \sum_{m=1}^{M} \tilde{\mathbf{h}}^{t+1,m} (\tilde{\mathbf{h}}^{t+1,m})^\top \right)$.
    $L^{t+1} = L^t + \alpha_t \left(\frac{1}{N} \sum_{n=1}^{N} \mathbf{v}^n (\mathbf{v}^n)^\top - \frac{1}{M} \sum_{m=1}^{M} \tilde{\mathbf{v}}^{t+1,m} (\tilde{\mathbf{v}}^{t+1,m})^\top \right)$.
    Decrease α_t.
end for

control its actions. Having multiple alternative representations of the same sensory input increases the likelihood, but makes it far more difficult to associate an appropriate action with that sensory input. The mean-field inference helps to eliminate this problem. During learning, if the posterior given a training input vector is multimodal, the mean-field inference will lock onto exactly one mode, and learning will make that mode more probable. Our learning algorithm will therefore tend to find regions in the parameter space in which the true posterior is unimodal.

5.3 Deep Boltzmann Machines (DBM's)

In general, we will rarely be interested in learning a complex, fully connected Boltzmann machine. Instead, we will focus on learning Deep Boltzmann Machines (see Fig. 5.2, right panel). Unlike Deep Belief Networks, a Deep Boltzmann Machine is a Markov random field, where all connections between layers are undirected.

Consider a three-hidden-layer Boltzmann machine, as shown in Fig. 5.2, right panel, with no within-layer connections. The energy of the state $\{\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3\}$ is defined as:

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3; \theta) = -\mathbf{v}^\top W^1 \mathbf{h}^1 - \mathbf{h}^{1\top} W^2 \mathbf{h}^2, -\mathbf{h}^{2\top} W^3 \mathbf{h}^3, \tag{5.13}$$

where $\theta = \{W^1, W^2, W^3\}$ are the model parameters, representing visible-to-hidden and hidden-to-

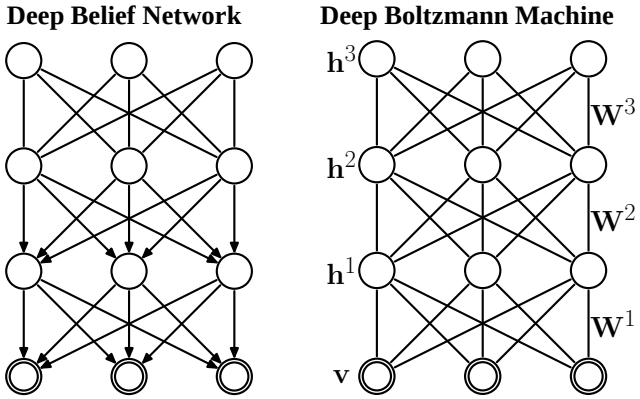

Figure 5.2: **Left:** Deep Belief Network, with the top two layers forming an undirected graph and the remaining layers form a belief net with directed, top-down connections **Right:** Deep Boltzmann machine, with both visible-to-hidden and hidden-to-hidden connections but with no within-layer connections.

hidden symmetric interaction terms. The probability that the model assigns to a visible vector $\mathbf{v}$ is:

$$P(\mathbf{v};\theta) \;=\; \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}^1,\mathbf{h}^2,\mathbf{h}^3} \exp\left(-E(\mathbf{v},\mathbf{h}^1,\mathbf{h}^2,\mathbf{h}^3;\theta)\right). \tag{5.14}$$

The conditional distributions over the visible and the three sets of hidden units are given by logistic functions:

$$p(h^1_j = 1|\mathbf{v},\mathbf{h}^2) \;=\; g\left(\sum_i W^1_{ij} v_i + \sum_m W^2_{jm} h^2_m\right), \tag{5.15}$$

$$p(h^2_m = 1|\mathbf{h}^1,\mathbf{h}^3) \;=\; g\left(\sum_j W^2_{jm} h^1_j + \sum_l W^3_{ml} h^3_l\right), \tag{5.16}$$

$$p(h^3_l = 1|\mathbf{h}^2) \;=\; g\left(\sum_m W^3_{ml} h^2_m\right), \tag{5.17}$$

$$p(v_i = 1|\mathbf{h}^1) \;=\; g\left(\sum_j W^1_{ij} h^1_j\right). \tag{5.18}$$

For approximate maximum likelihood learning, we could still apply the learning procedure for general Boltzmann machines described above, but it would be rather slow, particularly when the hidden units form layers that become increasingly remote from the visible units. There is, however, a fast way to initialize the model parameters to good values, which we describe in the next section.

5.3.1 Greedy Layerwise Pretraining of DBM's

In chapter 2 we reviewed a greedy, layer-by-layer unsupervised learning algorithm that consists of learning a stack of RBM's one layer at a time. After greedy learning, the whole stack can be viewed as a single probabilistic model called a Deep Belief Network. Surprisingly, this composite model is *not* a Deep Boltzmann Machine. The top two layers form a Restricted Boltzmann Machine, but the lower layers form a *directed* sigmoid belief network (see Fig. 5.2, left panel).

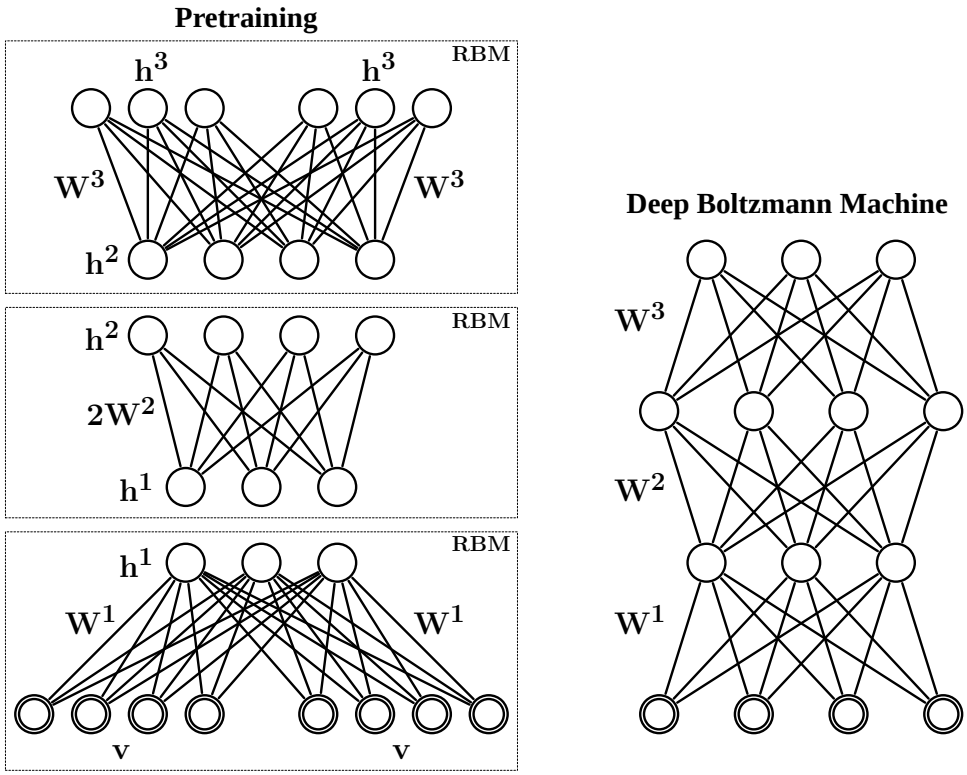

Figure 5.3: Pretraining a DBM with three hidden layers consists of learning a stack of RBM's that are then composed to create a Deep Boltzmann Machine. The first and last RBM's in the stack need to be modified by copying the visible or hidden units.

After learning the first RBM in the stack, the generative model can be written as:

$$P(\mathbf{v};\theta) \;=\; \sum_{\mathbf{h}^1} P(\mathbf{h}^1;W^1)P(\mathbf{v}|\mathbf{h}^1;W^1), \tag{5.19}$$

where $P(\mathbf{h}^1;W^1) = \sum_{\mathbf{v}} P(\mathbf{h}^1,\mathbf{v};W^1)$ is an implicit prior over $\mathbf{h}^1$. The second RBM in the stack attempts to learn a better model for $P(\mathbf{h}^1;W^2)$ by maximizing the variational lower bound (see Eq. 2.17) with respect to W^2. If initialized correctly, the 2^{nd} layer RBM $P(\mathbf{h}^1;W^2)$ will become a better model of the aggregated posterior over $\mathbf{h}^1$, which is simply the mixture of factorial posteriors for all the training cases: $\frac{1}{N}\sum_n P(\mathbf{h}^1|\mathbf{v}_n;W^1)$ (see section 2.2). Since the 2^{nd} layer RBM replaces $P(\mathbf{h}^1;W^1)$ by a better model, inferring $P(\mathbf{h}^1;W^1,W^2)$ would be possible by taking a geometric average of the two models of $\mathbf{h}^1$, which could be approximated by using $1/2W^1$ bottom-up and $1/2W^2$ top-down. But using W^1 bottom-up and W^2 top-down would effectively double the total input into the hidden units $\mathbf{h}^1$, which may cause saturation.

To initialize model parameters of a DBM, we propose greedy, layer-by-layer pretraining heuristic by learning a stack of RBM's, but with a small change that is introduced to eliminate the doubling effect. For the lower-level RBM, to compensate for the lack of top-down input into $\mathbf{h}^1$, we double the input and tie the visible-to-hidden weights, as shown in Fig. 5.3, left panel. In this modified RBM with tied parameters, the conditional distributions over the hidden and visible states are defined as:

$$p(h^1_j = 1|\mathbf{v}) \;=\; g\left(2\sum_i W^1_{ij}v_i\right), \tag{5.20}$$

$$p(v_i = 1|\mathbf{h}^1) \;=\; g\left(\sum_j W^1_{ij}h^1_j\right). \tag{5.21}$$

Algorithm 6 Greedy Pretraining Algorithm for a Deep Boltzmann Machine with L-layers.

1: Make two copies of the visible vector and tie the visible-to-hidden weights W^1. Fit W^1 of the 1^{st} layer RBM to data.
2: Freeze W^1 that defines the 1^{st} layer of features, and use samples $\mathbf{h}^l$ from $P(\mathbf{h}^1|\mathbf{v}, 2W^1)$ (Eq. 5.20) as the data for training the next layer RBM with weight vector $2W^2$.
3: Freeze W^2 that defines the 2^{nd} layer of features and use the samples $\mathbf{h}^2$ from $P(\mathbf{h}^2|\mathbf{h}^1, 2W^2)$ as the data for training the 3^{rd} layer RBM with weight vector $2W^3$.
4: Proceed recursively for the next layers $L-1$.
5: When learning the top-level RBM, double the number of hidden units and tie the visible-to-hidden weights W^L.
6: Use the weights $\{W^1, W^2, ..., W^L\}$ to compose a Deep Boltzmann Machine.

Contrastive Divergence learning works well and the modified RBM is good at reconstructing its training data. Conversely, for the top-level RBM, to compensate for the lack of bottom-up input into $\mathbf{h}^2$, we double the number of hidden units. The conditional distributions for this model take the form:

$$p(h_m^2 = 1|\mathbf{h}^3) \;=\; g\left(\sum_l W_{ml}^3 h_l^{3_{(a)}} + \sum_l W_{ml}^3 h_l^{3_{(b)}}\right) \tag{5.22}$$

$$p(h_l^3 = 1|\mathbf{h}^2) \;=\; g\left(\sum_m W_{ml}^3 h_m^2\right). \tag{5.23}$$

For the intermediate RBM we simply double the weights. The conditional distributions take the form:

$$p(h_j^1 = 1|\mathbf{h}^2) \;=\; g\left(2\sum_m W_{jm}^2 h_m^2\right) \tag{5.24}$$

$$p(h_m^2 = 1|\mathbf{h}^1) \;=\; g\left(2\sum_j W_{jm}^2 h_j^1\right). \tag{5.25}$$

When these three modules are composed to form a single system, the total input coming into the first and second hidden layers is halved, which leads to the following conditional distribution over $\mathbf{h}^1$ and $\mathbf{h}^2$:

$$p(h_j^1 = 1|\mathbf{v}, \mathbf{h}^2) \;=\; g\left(\sum_i W_{ij}^1 v_i + \sum_m W_{jm}^2 h_m^2\right), \tag{5.26}$$

$$p(h_m^2 = 1|\mathbf{h}^1, \mathbf{h}^3) \;=\; g\left(\sum_j W_{jm}^2 h_j^1 + \sum_l W_{ml}^3 h_l^3\right). \tag{5.27}$$

The conditional distributions over $\mathbf{v}$ and $\mathbf{h}^3$ remain the same as defined by Eqs. 5.21, 5.23.

Observe that the conditional distributions defined by the composed model are exactly the same conditional distributions defined by the DBM (Eqs. 5.15, 5.16, 5.17, 5.18). Therefore greedily pretraining the stack of modified RBM's leads to an undirected model with symmetric weights – a Deep Boltzmann Machine. We note that the modification only needs to be used for the first and the last RBM's in the stack. For all the intermediate RBM's we simply halve their weights in both directions when composing them to form a Deep Boltzmann Machine.

Greedily pretraining the weights of a DBM in this way serves two purposes. First, it initializes the weights to reasonable values. Second, it ensures that there is a fast way of performing approximate inference by a single upward pass through the stack of modified RBM's. Given an input vector, each layer of hidden units can be activated in a single deterministic bottom-up pass by doubling the bottom-up input to compensate for the lack of top-down feedback, except for the very top layer, which does not have a top-down input. This fast approximate inference is used to initialize the mean-field, which then converges much faster than the mean-field with random initialization.

5.3.2 Evaluating DBM's

In chapter 4 we showed that a Monte Carlo based method, Annealed Importance Sampling (AIS), can be used to efficiently estimate the partition function of an RBM. In this section we show how AIS can be used to estimate the partition functions of Deep Boltzmann Machines. Together with variational inference this will allow us obtain good estimates of the lower bound on the log-probability of the train and test data.

Using the special layer-by-layer structure of DBM's, we can derive an efficient AIS scheme for estimating the model's partition function. Let us consider a three-hidden-layer Boltzmann machine (see Fig. 5.3, right panel) whose energy is defined as:

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3; \theta) \;=\; -\mathbf{v}^\top W^1 \mathbf{h}^1 - \mathbf{h}^{1\top} W^2 \mathbf{h}^2 - \mathbf{h}^{2\top} W^3 \mathbf{h}^3. \tag{5.28}$$

By explicitly summing out the 1^{st} and the 3^{rd} layer hidden units $\{\mathbf{h}^1, \mathbf{h}^3\}$, we can easily evaluate an unnormalized probability $P^*(\mathbf{v}, \mathbf{h}^2; \theta)$. We can therefore run AIS on a much smaller state space $\mathbf{x} = \{\mathbf{v}, \mathbf{h}^2\}$ with $\mathbf{h}^1$ and $\mathbf{h}^3$ analytically summed out. The sequence of intermediate distributions, parameterized by β, is defined as follows:

$$\begin{aligned} P_k(\mathbf{v}, \mathbf{h}^2; \theta) \;&=\; \sum_{\mathbf{h}^1, \mathbf{h}^3} P_k(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3; \theta) \\ &=\; \frac{1}{\mathcal{Z}_k} \prod_j \left(1 + e^{\beta_k \left(\sum_i v_i W^1_{ij} + \sum_m h^2_m W^2_{jm}\right)}\right) \prod_l \left(1 + e^{\beta_k \left(\sum_m h^2_m W^3_{ml}\right)}\right). \end{aligned}$$

We gradually change β_k (the inverse temperature) from 0 to 1, annealing from a simple "uniform" model to the final complex model. Using Eqs. 5.15, 5.16, 5.17, 5.18, it is straightforward to derive a Gibbs transition operator that leaves $P_k(\mathbf{v}, \mathbf{h}^2; \theta)$ invariant:

$$p(h^1_j = 1 | \mathbf{v}, \mathbf{h}^2) \;=\; g\left(\beta_k \left(\sum_i W^1_{ij} v_i + \sum_m W^2_{jm} h^2_m\right)\right), \tag{5.29}$$

$$p(h^2_m = 1 | \mathbf{h}^1, \mathbf{h}^3) \;=\; g\left(\beta_k \left(\sum_j W^2_{jm} h^1_j + \sum_l W^3_{ml} h^3_l\right)\right), \tag{5.30}$$

$$p(h^3_l = 1 | \mathbf{h}^2) \;=\; g\left(\beta_k \sum_m W^3_{ml} h^2_m\right), \tag{5.31}$$

$$p(v_i = 1 | \mathbf{h}^1) \;=\; g\left(\beta_k \sum_j W^1_{ij} h^1_j\right). \tag{5.32}$$

Once we obtain an estimate of the global partition function $\hat{\mathcal{Z}}$, we can estimate, for a given test case

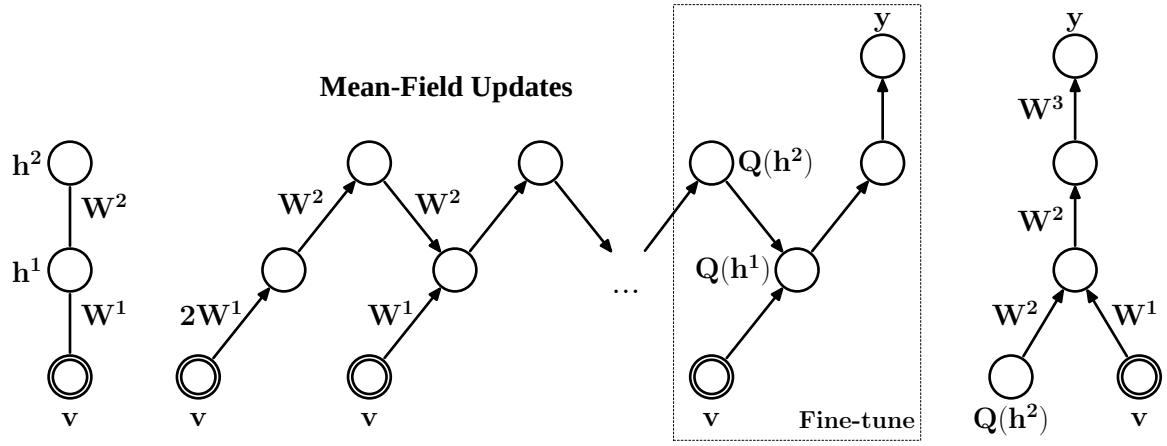

Figure 5.4: **Left:** A two-hidden-layer Boltzmann machine. **Right:** After learning, DBM is used to initialize a multilayer neural network. The marginals of approximate posterior $q(h_j^2 = 1|\mathbf{v})$ are used as additional inputs. The network is fine-tuned by backpropagation.

$\mathbf{v}^*$, the variational lower bound of Eq. 5.11:

$$\begin{aligned}\log P(\mathbf{v}^*;\theta) \;&\geq\; -\sum_{\mathbf{h}} Q(\mathbf{h};\mu)E(\mathbf{v}^*,\mathbf{h};\theta) + \mathcal{H}(Q) - \log \mathcal{Z}(\theta) \\ &\approx\; -\sum_{\mathbf{h}} Q(\mathbf{h};\mu)E(\mathbf{v}^*,\mathbf{h};\theta) + \mathcal{H}(Q) - \log \hat{\mathcal{Z}},\end{aligned}$$

where we defined $\mathbf{h} = \{\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3\}$. For each test vector under consideration, this lower bound is maximized with respect to the variational parameters μ using the mean-field update equations.

Furthermore, by explicitly summing out the states of the hidden units $\{\mathbf{h}^2, \mathbf{h}^3\}$, we can obtain a tighter variational lower bound on the log-probability of the test data. Of course, we can also adopt AIS to estimate $P^*(\mathbf{v}) = \sum_{\mathbf{h}^1,\mathbf{h}^2,\mathbf{h}^3} P^*(\mathbf{v},\mathbf{h}^1,\mathbf{h}^2,\mathbf{h}^3)$, and together with an estimate of the global partition function we can actually estimate the true log-probability of the test data. This however, would be computationally very expensive, since we would need to perform a separate AIS run for each test case. As an alternative, we could adopt a variation of the Chib-style estimator, proposed by Murray and Salakhutdinov [2009]. In the case of Deep Boltzmann Machines, where the posterior over the hidden units tends to be unimodal, their proposed Chib-style estimator can provide good estimates of $\log P^*(\mathbf{v})$ in a reasonable amount of computer time.

When learning a Deep Boltzmann Machine with more than two hidden layers, and no within-layer connections, we can explicitly sum out either odd or even layers. This will result in a better estimate of the model's partition function and tighter lower bounds on the log-probability of the test data.

5.3.3 Discriminative Fine-tuning of DBM's

After learning, the stochastic activities of the binary features in each layer can be replaced by deterministic, real-valued probabilities, and a Deep Boltzmann Machine can be used to initialize a multilayer neural network in the following way. For each input vector $\mathbf{v}$, the mean-field inference is used to obtain an approximate posterior distribution $Q(\mathbf{h}^2|\mathbf{v})$. The marginals $q(h_j^2 = 1|\mathbf{v})$ of this approximate posterior, together with the data, are used to create an "augmented" input for this deep multilayer neural network as shown in Fig. 5.4. Standard backpropagation of error derivatives can then be used to discriminatively fine-tune the model.

The unusual representation of the input is a by-product of converting a DBM into a deterministic neural network. In general, the gradient-based fine-tuning may choose to ignore $Q(\mathbf{h}^2|\mathbf{v})$, i.e. drive

the 1^{st} layer connections W^2 to zero, which will result in a standard neural network. Conversely, the network may choose to ignore the input by driving the 1^{st} layer weights W^1 to zero, and make its predictions based on only the approximate posterior. However, the network typically makes use of the entire augmented input for making predictions.

5.4 Experimental Results

In our experiments we used the MNIST and NORB datasets. To speed-up learning, we subdivided datasets into mini-batches, each containing 100 cases, and updated the weights after each mini-batch. The number of sample particles, used for approximating the model's expected sufficient statistics, was also set to 100. For the stochastic approximation algorithm, we always used 5 Gibbs updates. Each model was trained using 300,000 weight updates. The initial learning rate was set 0.005 and was decreased as 10/(2000+t). For discriminative fine-tuning of DBM's we used the method of conjugate gradients. Details of pretraining and fine-tuning, along with details of Matlab code that we used for learning and fine-tuning Deep Boltzmann Machines, can be found in Appendix A.

MNIST

The MNIST digit dataset contains 60,000 training and 10,000 test images of ten handwritten digits (0 to 9), with 28×28 pixels. In our first experiment we trained a fully connected "flat" BM on the MNIST dataset. The model had 500 hidden units and 784 visible units. To estimate the model's partition function we used 20,000 β_k spaced uniformly from 0 to 1. Results are shown in table 5.1. The lower bound on the average test log-probability was -84.67 per test case, which is slightly better compared to the lower bound of -85.97, achieved by a carefully trained two-hidden-layer Deep Belief Network (see section 4.4).

In our second experiment, we trained two Deep Boltzmann Machines: one with two hidden layers (500 and 1000 hidden units), and the other with three hidden layers (500,500, and 1000 hidden units), as shown in Fig. 5.6. To estimate the model's partition function, we also used 20,000 intermediate distributions spaced uniformly from 0 to 1. Table 5.1 shows that the estimates of the lower bound on the average test log-probability were -84.62 and -85.18 for the 2- and 3-layer Boltzmann machines respectively.

Observe that the two DBM's that contain over 0.9 and 1.15 million parameters do not appear to suffer much from overfitting. The difference between the estimates of the training and test log-probabilities was about 1 nat. Figure 5.5 further shows samples generated from all three models by randomly initializing all binary states and running the Gibbs sampler for 100,000 steps. Certainly, all samples look like the real handwritten digits. We also emphasize that without greedy pretraining, we could not successfully learn good DBM models of MNIST digits.

To estimate how loose the variational bound is, we randomly sampled 100 test cases, 10 of each class, and ran AIS to estimate the true test log-probability[2] for the 2-layer Boltzmann machine. The estimate of the variational bound was -83.35 per test case, whereas the estimate of the true test log-probability was -82.86. The difference of about 0.5 nats shows that the bound is rather tight.

Finally, after discriminative fine-tuning, the two-hidden-layer BM achieves an error rate of 0.95% on the full MNIST test set. This is, to our knowledge, the best published result on the permutation-invariant version of the MNIST task. The 3-layer BM gives a slightly worse error rate of 1.01%. The flat BM, on the other hand, gives considerably worse error rate of 1.27%. This is compared to 1.4% achieved by SVM's (Decoste and Schölkopf [2002]), 1.6% achieved by randomly initialized backprop, 1.2%

[2]Note that computationally, this is equivalent to estimating 100 partition functions.

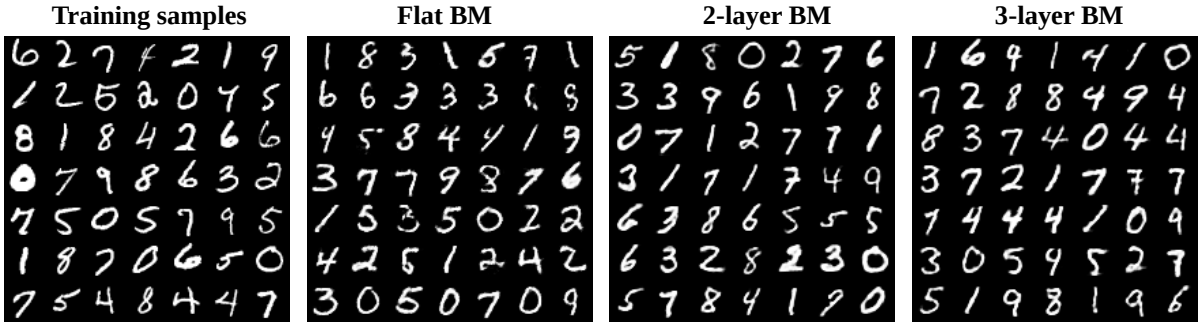

Figure 5.5: Random samples from the training set, and samples generated from three Boltzmann machines by running the Gibbs sampler for 100,000 steps. The images shown are the *probabilities* of the binary visible units given the binary states of the hidden units

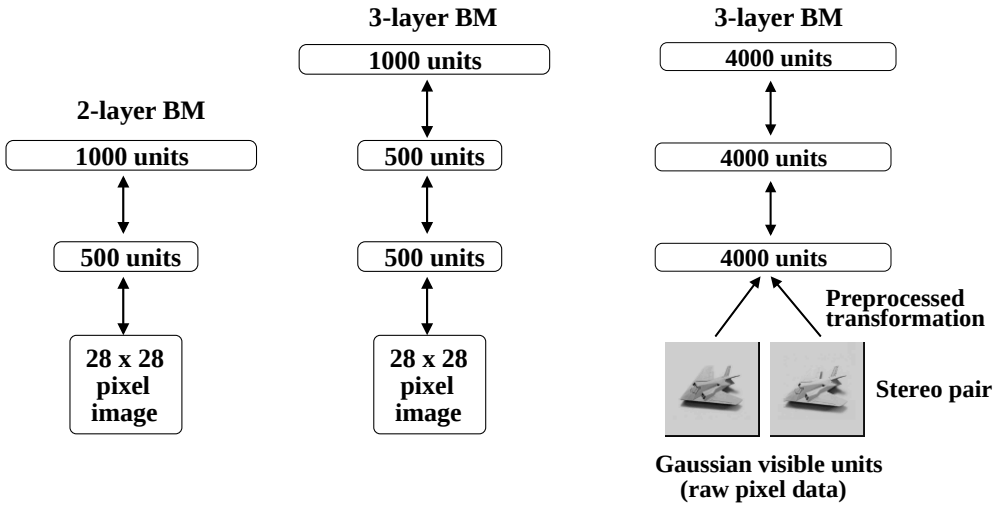

Figure 5.6: **Left:** The architectures of two Deep Boltzmann Machines used in MNIST experiments. **Right:** The architecture of Deep Boltzmann Machine used in NORB experiments.

achieved by the Deep Belief Network, described in (Hinton et al. [2006], Hinton and Salakhutdinov [2006]), and 0.97% obtained by using a combination of discriminative and generative fine-tuning on the same DBN (Hinton [2007]).

To test discriminative performance of DBM's when the number of labeled examples is small, we randomly sampled 1%, 5%, and 10% of the handwritten digits in each class and treated them as labeled data. Table 5.2 shows that after discriminative fine-tuning, a two-hidden-layer BM achieves error rates of 4.82%, 2.72%, and 2.46%. Deep Boltzmann Machines clearly outperform regularized nonlinear NCA, discussed in section 3.4, as well as linear NCA, an autoencoder, and K-nearest neighbours, particularly when the number of labeled examples is only 600.

NORB

Results on MNIST show that Deep Boltzmann Machines can significantly outperform many other models on the well-studied but relatively simple task of handwritten digit recognition. In this section we present results on NORB, which is considerably more difficult dataset than MNIST. NORB (LeCun et al. [2004]) contains images of 50 different 3D toy objects with 10 objects in each of five generic classes: cars, trucks, planes, animals, and humans. Each object is captured from different viewpoints

Table 5.1: Results of estimating partition functions of BM models, along with the estimates of lower bound on the average training and test log-probabilities. For all BM's we used 20,000 intermediate distributions. Results were averaged over 100 AIS runs.

	Estimates		Avg. log-prob.	
	$\log \hat{\mathcal{Z}}$	$\log(\hat{\mathcal{Z}} \pm \hat{\sigma})$	Test	Train
Flat BM	198.29	$198.17, 198.40$	-84.67	-84.35
2-layer BM	356.18	$356.06, 356.29$	-84.62	-83.61
3-layer BM	456.57	$456.34, 456.75$	-85.10	-84.49

Table 5.2: Classification error rates on MNIST test set when only a small fraction of labeled data is available.

	Two-Layer DBM	Regularized Nonlinear NCA	Linear NCA	Autoencoder	KNN
1% (600)	4.82%	8.81%	19.37%	9.62%	13.74%
5% (3000)	2.72%	3.24%	7.23%	5.18%	7.19%
10% (6000)	2.46%	2.58%	4.89%	4.46%	5.87%
100% (60000)	0.95%	1.00%	2.45%	2.41%	3.09%

and under various lighting conditions. The training set contains 24,300 stereo image pairs of 25 objects, 5 per class, while the test set contains 24,300 stereo pairs of the remaining, different 25 objects. The goal is to classify each previously unseen object into its generic class. From the training data, 4,300 were set aside for validation.

Each image has 96×96 pixels with integer greyscale values in the range [0,255]. To speed-up experiments, we reduced the dimensionality by using a foveal representation of each image in a stereo pair. The central 64×64 portion of an image is kept at its original resolution. The remaining 16 pixel-wide ring around it is compressed by replacing non-overlapping square blocks of pixels in the ring with a single scalar given by the average pixel-value of a block. We split the ring into four smaller ones: the outermost ring consists of 8×8 blocks, followed by a ring of 4×4 blocks, and finally two innermost rings of 2×2 blocks. The resulting dimensionality of each training vector, representing a stereo pair, was $2 \times 4488 = 8976$. A random sample from the training data used in our experiments is shown in Fig. 5.7, left panel[3].

To model raw pixel data, we use an RBM with Gaussian visible and binary hidden units. Gaussian RBM's have been previously successfully applied for modeling greyscale images, such as images of faces (see subsection 3.2.1). However, learning an RBM with Gaussian units can be slow, particularly when the input dimensionality is quite large. Here we follow the approach of Nair and Hinton [2009] by first learning a Gaussian RBM and then treating the the activities of its hidden layer as "preprocessed" data. Effectively, the learned low-level RBM acts as a preprocessor that converts greyscale pixels into a binary representation, which we then use for learning a Deep Boltzmann Machine.

The number of hidden units for the preprocessing RBM was set to 4000 and the model was trained using Contrastive Divergence learning for 500 epochs. We then trained a two-hidden-layer DBM with each layer containing 4000 hidden units, as shown in Fig. 5.6, right panel. Note that the entire model was trained in a completely unsupervised way. After the subsequent discriminative fine-tuning, the "unrolled" DBM achieves a misclassification error rate of 10.8% on the full test set. This is compared

[3]We thank Vinod Nair for sharing his code for blurring and translating NORB images.

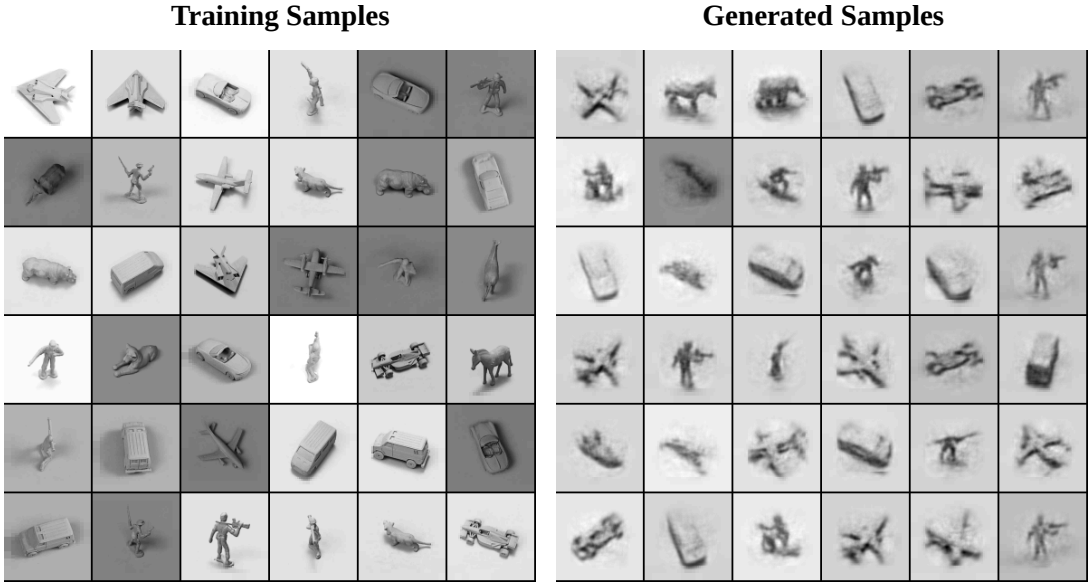

Figure 5.7: Random samples from the training set, and samples generated from a three-hidden-layer Deep Boltzmann Machine by running the Gibbs sampler for 10,000 steps.

to 11.6% achieved by SVM's (Bengio and LeCun [2007]), 22.5% achieved by logistic regression, and 18.4% achieved by the K-nearest neighbours (LeCun et al. [2004]).

To show that DBM's can benefit from additional *unlabeled* training data, we augmented the training data with additional unlabeled data by applying simple pixel translations, creating a total of 1,166,400 training instances. After learning a good generative model, the discriminative fine-tuning (using only the 24,300 labeled training examples without any translation) reduces the misclassification error down to 7.2%. Figure 5.7 shows samples generated from the model by running prolonged Gibbs sampling. Note that the model was able to capture a lot of regularities in this high-dimensional, richly structured data, including different object classes, various viewpoints and lighting conditions.

Surprisingly, even though the Deep Boltzmann Machine contains about 68 million parameters, it significantly outperforms many of the competing models. Clearly, unsupervised learning helps generalization because it ensures that most of the information in the model parameters comes from modeling the input data. The very limited information in the labels is used only to slightly adjust the layers of features already discovered by the Deep Boltzmann Machine.

5.5 Discussion

We have presented a new learning algorithm for training Deep Boltzmann Machines that combines variational learning and MCMC and showed that it can be used to successfully learn good generative models of MNIST digits and NORB 3D objects. The new algorithm readily extends to learning Boltzmann machines with real-valued, count, or tabular data. We further introduced a greedy layer-by-layer learning algorithm that can be used to quickly initialize the parameters of DBM's to sensible values. This greedy initialization strategy allowed us to successfully learn a good generative model of NORB 3D objects, even though the model contained about 68 million parameters. We also showed how Annealed Importance Sampling, along with variational inference, can be used to estimate a variational lower bound on the log-probability that a Boltzmann machine with multiple hidden layers assigns to test data. This allowed us to directly assess the performance of Deep Boltzmann Machines as generative models of data.

Finally, we showed that the discriminatively fine-tuned Deep Boltzmann Machines perform well on the MNIST digit and NORB 3D object recognition tasks.

Chapter 6

Conclusions

6.1 Summary of Contributions

The aim of the thesis was to demonstrate that learning deep generative models that contain many layers of latent variables and millions of parameters can be carried out efficiently, and that the learned high-level feature representations can be successfully applied in a wide spectrum of application domains, including visual object recognition, information retrieval, classification and regression tasks, as well as nonlinear dimensionality reduction. Many of the ideas presented in this thesis are based on the following three crucial principles behind learning deep generative models: First, multiple layers of representation can be greedily learned one layer at a time. Second, the greedy learning is carried out in a completely unsupervised way. Third, a separate fine-tuning stage can be used to further improve either generative or discriminative performance of the final model.

The first part of the thesis focused on analysis and applications of a particular family of deep generative models, called Deep Belief Networks (DBN's), and their building modules Restricted Boltzmann Machines (RBM's). In chapter 2 we provided a detailed overview of RBM's and DBN's, along with the greedy learning algorithms for DBN's. In chapter 3 we discussed various applications of Deep Belief Networks. In particular, we first showed that these deep hierarchical models can be used to learn useful feature representations from large amounts of high-dimensional, highly-structured unlabeled input data. The learned high-level representations capture a lot of structure in the unlabeled input, which is useful for subsequent discrimination or regression tasks, even though these tasks are unknown when the deep model is being trained. We then demonstrated how the greedy learning of multiple layers of representation can be used to initialize deep nonlinear autoencoders. This allowed deep autoencoders to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data. We further explored the idea of using the deep autoencoders to learn "semantic" binary codes that allowed us to perform very fast and accurate information retrieval. Finally, we discussed how the unsupervised greedy learning algorithm can be used to pretrain and fine-tune a deep encoder network in order to learn a similarity metric over the input space, which greatly facilitates nearest-neighbor classification.

Chapter 4 focused on evaluating generalization performance of Deep Belief Networks as density models. Indeed, assessing the generalization performance of DBN's plays an important role in model selection and controlling model complexity. For many specific tasks, such as information retrieval or classification, performance of DBN's can be directly evaluated, as we demonstrated in chapter 3. More broadly, however, the ability of DBN's to generalize can be evaluated by computing the probability of held-out input vectors, which is independent of any specific application. Computing this probability exactly is intractable, since it requires enumeration over an exponential number of terms. In chapter 4

we showed how a Monte Carlo based method, Annealed Importance Sampling, along with approximate inference, can be used to estimate a lower bound on the log-probability that a DBN model with multiple hidden layers assigns to the test data. This allowed us to measure the generalization performance of Deep Belief Networks as density models and to compare them to other probabilistic models, such as plain mixture models.

In the second part of the thesis we developed a new learning algorithm for a different type of hierarchical probabilistic model called Deep Boltzmann Machine (DBM). Like Deep Belief Networks, DBM's contain many layers of latent variables. High-level representations can be built from large amounts of unlabeled sensory inputs and the limited labeled data can then be used to only slightly adjust the model parameters for a specific task at hand. Unlike existing models with deep architectures, the approximate inference procedure, in addition to a bottom-up pass, can incorporate top-down feedback, which allows Deep Boltzmann Machines to better propagate uncertainty about ambiguous inputs.

Approximate inference in DBM's can be performed using variational approaches, such as mean-field. Learning can then be carried out by applying a stochastic approximation procedure that uses Markov chain Monte Carlo (MCMC) to approximate a model's expected sufficient statistics, which is needed for maximum likelihood learning. This unusual combination of variational methods and MCMC is essential for creating a fast learning algorithm for Deep Boltzmann Machines. The new algorithm readily extends to learning Boltzmann machines with real-valued, count, or tabular data. Finally, results on the MNIST and NORB datasets show that Deep Boltzmann Machines can learn good generative models and perform well on handwritten digit and visual object recognition tasks. In fact, we found that after discriminative fine-tuning, a two-hidden-layer Deep Boltzmann Machine produces the best published result on the permutation-invariant version of the MNIST task. It significantly outperforms logistic regression and support vector machines, and is similar to the best published result for Deep Belief Networks.

6.2 Future Directions

There are several potential extensions and applications of the ideas presented in this thesis, particularly related to learning Deep Boltzmann Machines.

Better Learning Algorithms for Deep Boltzmann Machines. The success of the Boltzmann machine learning algorithm heavily relies on the ability of the Markov chain to explore the highly multi-modal energy landscape. Particularly towards the end of learning, as the learning rate becomes small, the Markov chain used to approximate model's expected sufficient statistics tends to mix very poorly. Hence the need for Markov chain sampling methods that can better explore distributions with many isolated modes (Salakhutdinov [2010]). Indeed, the transition operators used in the stochastic approximation algorithm do not necessarily need to be simple Gibbs or Metropolis-Hastings updates. Other valid MCMC operators, such as those based on tempered transitions (Neal [1996]) or parallel tempering (Geyer [1991], Swendsen and Wang [1986], Earl and Deem [2005]), may significantly improve model performance.

Semi-Supervised Learning with DBM's. In many practical learning domains, there is a large supply of high-dimensional unlabeled data and very limited labeled data. Applications such as information retrieval and machine vision are examples where large amounts of unlabeled data is readily available. In chapter 5, we only considered unsupervised learning of Deep Boltzmann Machines, followed by deterministic discriminative fine-tuning. However, the general Boltzmann machine framework should allow us to readily extend the proposed learning algorithm to semi-supervised setting. Treating units with missing labels as "additional hidden units", variational inference can

be used to effectively "fill in" the missing label information. Learning can then proceed as if there were no missing labels. Similar reasoning can be applied to learning Boltzmann machines when parts of input vectors are missing (at random). Although our initial experiments seem to be encouraging, more theoretical analysis and empirical work needs to be done to determine the effectiveness of such a semi-supervised learning procedure.

Extracting Structure from Temporal Data using DBM's. Modeling complex nonlinear dynamics of high-dimensional time series data, such as video sequences, is an active area of research in machine learning. Many of the existing time series models, such as linear dynamical systems (LDS), switching LDS, hidden Markov models (HMM's), factorial HMM's, and product of HMM's, have been widely used in practice. However, these models are very limited in the types of structure they can model. We believe that multiple layers of distributed representation should work better for modeling temporal structure. One could therefore build a stack of Deep Boltzmann Machines linked together through time. These models could potentially not only model nonlinear dynamics, but also make multimodal predictions and handle missing inputs.

Large Scale Object Recognition and Information Retrieval. As we have stated before, both Deep Boltzmann Machines and Deep Belief Networks have the potential to learn layers of feature detectors that become progressively more complex, which is believed to be a promising way to solve object recognition problems. However, at present, most of the existing object recognition systems achieve state-of-the-art results using shallow architectures or deep hand coded methods like SIFT (Lowe [1999]), and include many hand-crafted features, which requires considerable human input and parameter tweaking. It is therefore necessary to apply and evaluate predictive performance of DBM's and DBN's on large scale object recognition tasks, including, for example, benchmark datasets such as the PASCAL dataset. Similarly, given the recent success of Deep Belief Networks on image and text retrieval tasks (Torralba et al. [2008], Salakhutdinov and Hinton [2007a]), it will be beneficial to also evaluate the retrieval accuracy of Deep Boltzmann Machines.

We have outlined several potential research directions. However, research on deep learning is very new and there are many broad open questions to consider. To name a few: Can we develop better optimization or approximation techniques that would allow us to learn deep models more efficiently without significant human intervention? Can we develop computer systems that are more adaptive, and capable of extracting distributed representations that can better generalize to unknown future tasks. How can we make deep models be more robust to dealing with highly ambiguous or missing sensory inputs? We believe that answering many of those questions will allow us to build more intelligent machines.

Bibliography

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

A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468. IEEE Computer Society, 2006.

J. Benedetti. On the nonparametric estimation of regression functions. *Journal of the Royal Statistical Society series B*, 39:248–253, 1977.

Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2009.

Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*. MIT Press, 2007.

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *Advances in Neural Information Processing Systems*, pages 153–160. MIT Press, 2007.

D. Bertsimas and J. Tsitsiklis. Simulated annealing. *STATSCI: Statistical Science: A Review Journal of the Institute of Mathematical Statistics*, 8, 1993.

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

M. A. Carreira-Perpinan and G. E. Hinton. On contrastive divergence learning. In *10th Int. Workshop on Artificial Intelligence and Statistics (AISTATS'2005)*, 2005.

T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, IT-13(1):21–7, January 1967.

M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *COMPGEOM: Annual ACM Symposium on Computational Geometry*, 2004.

D. Decoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 46(1/3): 161, 2002.

S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

D. DeMers and G. W. Cottrell. Nonlinear dimensionality reduction. In Cowan Hanson and Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 580–587, San Mateo, CA, 1993. Morgan Kaufmann.

D. J. Earl and M. W. Deem. Parallel tempering: Theory, applications, and new perspectives. *Phys. Chem.*, 7(3910), 2005.

C. Galland. Learning in deterministic Boltzmann machine networks. In *PhD Thesis*, 1991.

P. Gehler, A. Holub, and M. Welling. The Rate Adapting Poisson (RAP) model for information retrieval and object recognition. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

C. J. Geyer. Markov chain Monte Carlo maximum likelihood. In *Computing Science and Statistics*, pages 156–163, 1991.

A. Globerson and T. S. Jaakkola. Approximate inference using conditional entropy decompositions. In *11th International Workshop on AI and Statistics (AISTATS'2007)*, 2007.

A. Globerson and S. T. Roweis. Metric learning by collapsing classes. In *Advances in Neural Information Processing Systems*, 2005.

J. Goldberger, S. T. Roweis, G. E. Hinton, and R. R. Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems*, 2004.

R. Hadsell, A. Erkan, P. Sermanet, M. Scoffier, U. Muller, and Y. LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *IROS*, pages 628–633. IEEE, 2008.

R. Hecht-Nielsen. Replicator neural networks for universal optimal source coding. *Science*, 269:1860–1863, 1995.

G. E. Hinton. To recognize shapes, first learn to generate images. *Computational Neuroscience: Theoretical Insights into Brain Function.*, 2007.

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1711–1800, 2002.

G. E. Hinton and V. Nair. Inferring motor programs from images of handwritten digits. In *Advances in Neural Information Processing Systems*. MIT Press, 2006.

G. E. Hinton and S. T. Roweis. Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems*, pages 833–840. MIT Press, 2002.

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

G. E. Hinton and T. Sejnowski. Optimal perceptual inference. In *IEEE conference on Computer Vision and Pattern Recognition*, 1983.

G. E. Hinton and R. S. Zemel. Autoencoders, minimum description length and Helmholtz free energy. In *Advances in Neural Information Processing Systems*, volume 6, pages 3–10, 1994.

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

T. Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the 15th Conference on Uncertainty in AI*, pages 289–296, San Fransisco, California, 1999. Morgan Kaufmann.

C. Jarzynski. A nonequilibrium equality for free energy differences. *Physical Review Letters*, 78: 2690–2693, 1997.

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. In *Machine Learning*, volume 37, pages 183–233, 1999.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220: 671–680, 1983.

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.

F. Lauer, C. Y. Suen, and G. Bloch. A trainable feature extractor for handwritten digit recognition. *Pattern Recognition*, 40(6):1816–1824, 2007.

N. D. Lawrence. Gaussian process models for visualisation of high dimensional data. In *Advances in Neural Information Processing Systems*, pages 329–336. MIT Press, 2004.

N. D. Lawrence and M. I. Jordan. Semi-supervised learning via Gaussian processes. In *Advances in Neural Information Processing Systems*, 2004.

N. D. Lawrence and B. Schölkopf. Estimating a kernel Fisher discriminant in the presence of label noise. In *Proc. 18th International Conf. on Machine Learning*, pages 306–313. Morgan Kaufmann, San Francisco, CA, 2001.

Y. LeCun, F. J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR (2)*, pages 97–104, 2004.

T. S. Lee, D. Mumford, R. Romero, and V. Lamme. The role of the primary visual cortex in higher level vision. *Vision research*, 38:2429–2454, 1998.

N. LeRoux and Y. Bengio. Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation*, 2008.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999.

D. J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, September 2003.

R. Memisevic and G. E. Hinton. Unsupervised learning of image transformation. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

T. P. Minka. Expectation propagation for approximate Bayesian inference. In Jack Breese and Daphne Koller, editors, *UAI*, pages 362–369, San Francisco, CA, 2001. Morgan Kaufmann Publishers.

I. Murray and R. R. Salakhutdinov. Evaluating probabilities under high-dimensional latent variable models. In *Advances in Neural Information Processing Systems*, volume 21, 2009.

V. Nair and G. E. Hinton. Implicit mixtures of restricted Boltzmann machines. In *Advances in Neural Information Processing Systems*, volume 21, 2009.

R. M. Neal. Estimating ratios of normalizing constants using linked importance sampling. Technical Report 0511, Department of Statistics, University of Toronto, 2005.

R. M. Neal. Annealed importance sampling. *Statistics and Computing*, 11:125–139, 2001.

R. M. Neal. Sampling from multimodal distributions using tempered transitions. *Statistics and Computing*, 6:353–366, 1996.

R. M. Neal. Connectionist learning of belief networks. *Artif. Intell*, 56(1):71–113, 1992.

R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, September 1993.

R. M. Neal. Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. Technical Report no. 9702. Department of Statistics, University of Toronto, 1997.

R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Press, 1998.

S. Osindero and G. E. Hinton. Modeling image patches with a directed hierarchy of Markov random fields. In *Advances in Neural Information Processing Systems*, Cambridge, MA, 2008. MIT Press.

D. Plaut and G. E. Hinton. Learning sets of filters using back-propagation. *Computer Speech and Language*, 2:35–61, 1987.

M. A. Ranzato and M. Szummer. Semi-supervised learning of compact document representations with deep networks. In *Proceedings of the International Conference on Machine Learning*, volume 25, pages 792 – 799, 2008.

M. A. Ranzato, F. Huang, Y. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

M. A. Ranzato, Y. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. *Advances in Neural Information Processing Systems*, 2008.

C. E. Rasmussen. Evaluation of Gaussian processes and other methods for non-linear regression. In *PhD Thesis*, 1996.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Stat.*, 22:400–407, 1951.

S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

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

R. R. Salakhutdinov. Learning and evaluating Boltzmann machines. Technical Report UTML TR 2008-002, Department of Computer Science, University of Toronto, 2008.

R. R. Salakhutdinov. Learning in markov random fields using tempered transitions. In *Advances in Neural Information Processing Systems*, volume 22, 2010.

R. R. Salakhutdinov and G. E. Hinton. Using deep belief nets to learn covariance kernels for Gaussian processes. In *Advances in Neural Information Processing Systems*, volume 20, 2008.

R. R. Salakhutdinov and G. E. Hinton. Replicated softmax: an undirected topic model. In *Advances in Neural Information Processing Systems*, volume 22, 2010.

R. R. Salakhutdinov and G. E. Hinton. Semantic Hashing. In *SIGIR workshop on Information Retrieval and applications of Graphical Models*, 2007a.

R. R. Salakhutdinov and G. E. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 11, 2007b.

R. R. Salakhutdinov and G. E. Hinton. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 12, 2009a.

R. R. Salakhutdinov and G. E. Hinton. Semantic Hashing. *International Journal of Approximate Reasoning*, 50:969–978, 2009b.

R. R. Salakhutdinov and I. Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the International Conference on Machine Learning*, volume 25, pages 872 – 879, 2008.

R. R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted Boltzmann machines for collaborative filtering. In Zoubin Ghahramani, editor, *Proceedings of the International Conference on Machine Learning*, volume 24, pages 791–798. ACM, 2007.

G. Salton. Developments in automatic text retrieval. *Science*, 253, 1991.

G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

F. Samaria and F. Fallside. Face identification and feature extraction using hidden markov models. In G. Vernazza, editor, *Image Processing: Theory and Applications*, pages 295–298. Elsevier, June 1993.

M. W. Seeger. Covariance kernels from Bayesian generative models. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems*, pages 905–912. MIT Press, 2001.

M. W. Seeger. Gaussian processes for machine learning. *International Journal Neural Syst*, 14(2): 69–106, 2004.

T. Serre, A. Oliva, and T. A. Poggio. A feedforward architecture accounts for rapid categorization. *Proceedings of the National Academy of Sciences*, 104:6424–6429, 2007.

J. Skilling. Nested sampling. *Bayesian inference and maximum entropy methods in science and engineering, AIP Conference Proceedings*, 735:395–405, 2004.

P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge, 1986.

I. Sutskever and G. E. Hinton. Learning multilevel distributed representations for high-dimensional sequences. Technical Report UTML TR 2006-003, Dept. of Computer Science, University of Toronto, 2006.

R. H. Swendsen and J. S. Wang. Replica Monte Carlo simulation of spin-glasses. *Physical Review Letters*, 57(21):2607–2609, 1986.

G. Taylor, G. E. Hinton, and S. T. Roweis. Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems*. MIT Press, 2006.

J. A. Tenenbaum, V. J. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2008)*. ACM, 2008.

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*, 2008.

L. van der Maaten and G. E. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2009.

V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Proceedings of the Twenty-Fifth International Conference*, volume 307, pages 1096–1103, 2008.

M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. Technical report, Department of Statistics, University of California, Berkeley, 2003.

M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335, 2005.

K. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems*, 2005.

M. Welling and G. E. Hinton. A new learning algorithm for mean field Boltzmann machines. *Lecture Notes in Computer Science*, 2415, 2002.

M. Welling and Y. W. Teh. Approximate inference in Boltzmann machines. *Artificial Intelligence*, 143 (1):19–50, 2003.

M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems*, pages 1481–1488, Cambridge, MA, 2005. MIT Press.

E. Xing, R. Yan, and A. Hauptmann. Mining associated text and images with dual-wing harmoniums. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-2005)*, 2005.

J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.

L. Younes. On the convergence of Markovian stochastic algorithms with rapidly decreasing ergodicity rates, March 17 2000.

L. Younes. Parameter inference for imperfectly observed Gibbsian fields. *Probability Theory Rel. Fields*, 82:625–645, 1989.

A. L. Yuille. The convergence of contrastive divergences. In *Advances in Neural Information Processing Systems*, 2004.

S. Zhu and X. Liu. Learning in Gibbsian fields: How accurate and how fast can it be. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(7):1001–1006, 2002.

Appendix A

Appendix

A.1 Details of the Datasets

Synthetic Curves Dataset The synthetic curves dataset contains 28×28 images of "curves" that are generated from three randomly chosen 2-dimensional points (see Fig. A.1, left panel). For this dataset, the true intrinsic dimensionality is known, and the relationship between the pixel intensities and the six numbers used to generate them is highly non-linear. The training and test data contained 20,000 and 10,000 images respectively. To generate the synthetic curves we constrained the x coordinate of each point to be at least 2 greater than the x coordinate of the previous point. We also constrained all coordinates to lie in the range $[2, 26]$. The three points define a cubic spline which is "inked" to produce the 28×28 pixel images. The details of the inking procedure are described in Hinton and Nair [2006] and the Matlab code for generating synthetic curves is available at http://www.cs.toronto.edu/∼hinton.

MNIST Dataset The MNIST digit data set contains 60,000 training and 10,000 test 28×28 images of ten handwritten digits (0 to 9). Out of 60,000 training images, 10,000 were used for validation. The original pixel intensities were normalized to lie in the interval $[0, 1]$. The normalized pixel intensities tend to take on extreme values, and were therefore modeled much better by a standard binary RBM. Random samples from the training set are shown in Fig. A.1, right panel. The dataset is available at http://yann.lecun.com/exdb/mnist/index.html.

Olivetti Face Dataset The Olivetti face dataset (Samaria and Fallside [1993]) from which we obtained the face patches contains ten 64×64 images of each of forty different people, shown in Fig. A.2. For the experiments used in section 3.1, we constructed a dataset of 13,000 25×25 images by rotating (-90° to $+90^\circ$), cropping, and subsampling the original 400 images. The dataset was then subdivided into 12,000 training images, which contained the first 30 people, and 1,000 test images, which contained the remaining 10 people. For the experiments used in section 3.2, we constructed a much larger dataset of 165,600 25×25 images by rotating (-90° to $+90^\circ$), cropping, and subsampling as well as scaling (1.4 to 1.8) the original 400 images. The dataset was then subdivided into 124,200 training images, which contained the first 30 people, and 41,400 test images, which contained the remaining 10 people. In all of our experiments, the intensities in the cropped images were shifted so that every pixel had zero mean and the entire dataset was then scaled by a single number to make the average pixel variance be 1. The Olivetti face dataset is available at http://www.cs.toronto.edu/ roweis/data.html.

20-Newsgroups Dataset The 20-newsgroups corpus contains 18,845 postings taken from the Usenet newsgroup collection. The corpus is partitioned fairly evenly into 20 different newsgroups, each corre-

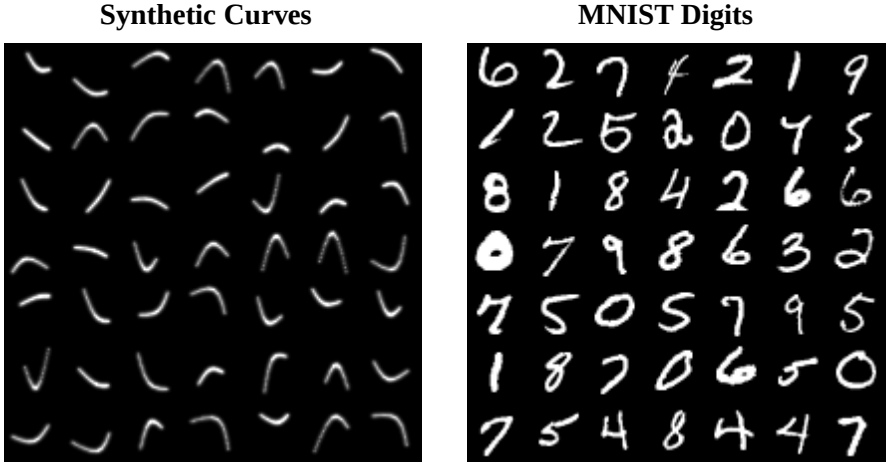

Figure A.1: **Left:** Random training samples from the synthetic curves dataset. **Right:** Random training samples from the MNIST dataset.

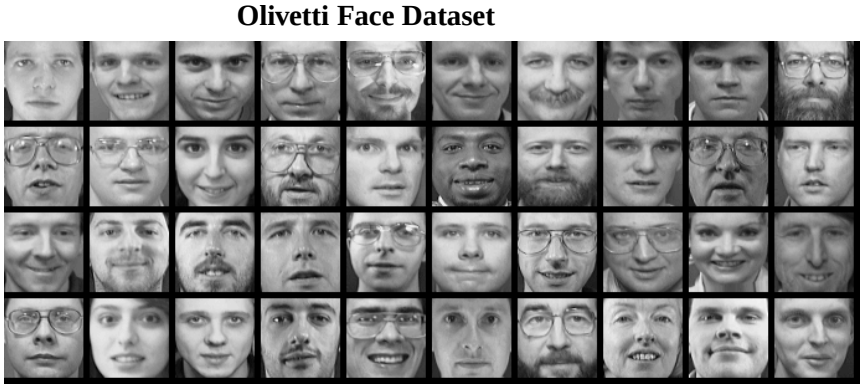

Figure A.2: The Olivetti face dataset showing a random sample of each of forty different people.

sponding to a separate topic. The data was split by date into 11,314 training and 7,531 test articles, so the training and test sets were separated in time. The training set was further randomly split into 8,314 training and 3,000 validation documents. Newsgroups such as soc.religion.christian and talk.religion.misc are very closely related to each other, while newsgroups such as comp.graphics and rec.sport.hockey are very different. We further preprocessed the data by removing common stopwords, stemming, and then only considering the 2000 most frequently used words in the training dataset. As a result, each posting was represented as a vector containing 2000 word counts. No other preprocessing was done. The dataset is available at http://people.csail.mit.edu/jrennie/20Newsgroups (20news-bydate.tar.gz). It has already been organized by date.

Reuters Corpus Volume I (RCV1-v2) Reuters Corpus Volume I (RCV1-v2) (Lewis et al. [2004]) is an archive of 804,414 newswire stories that have been manually categorized into 103 topics. The corpus covers four major groups: corporate/industrial, economics, government/social, and markets. Sample topics include Energy Markets, Accounts/Earnings, Government Borrowings, Disasters and Accidents, Interbank Markets, Legal/Judicial, Production/Services, etc. The topic classes form a tree which is typically of depth 3. For this dataset, we define the relevance of one document to another to be the fraction of the topic labels that agree on the two paths from the root to the two documents. The data was randomly split into 402,207 training and 402,207 test articles. The training set was further randomly split into 302,207 training and 100,000 validation documents. The available data was already in the

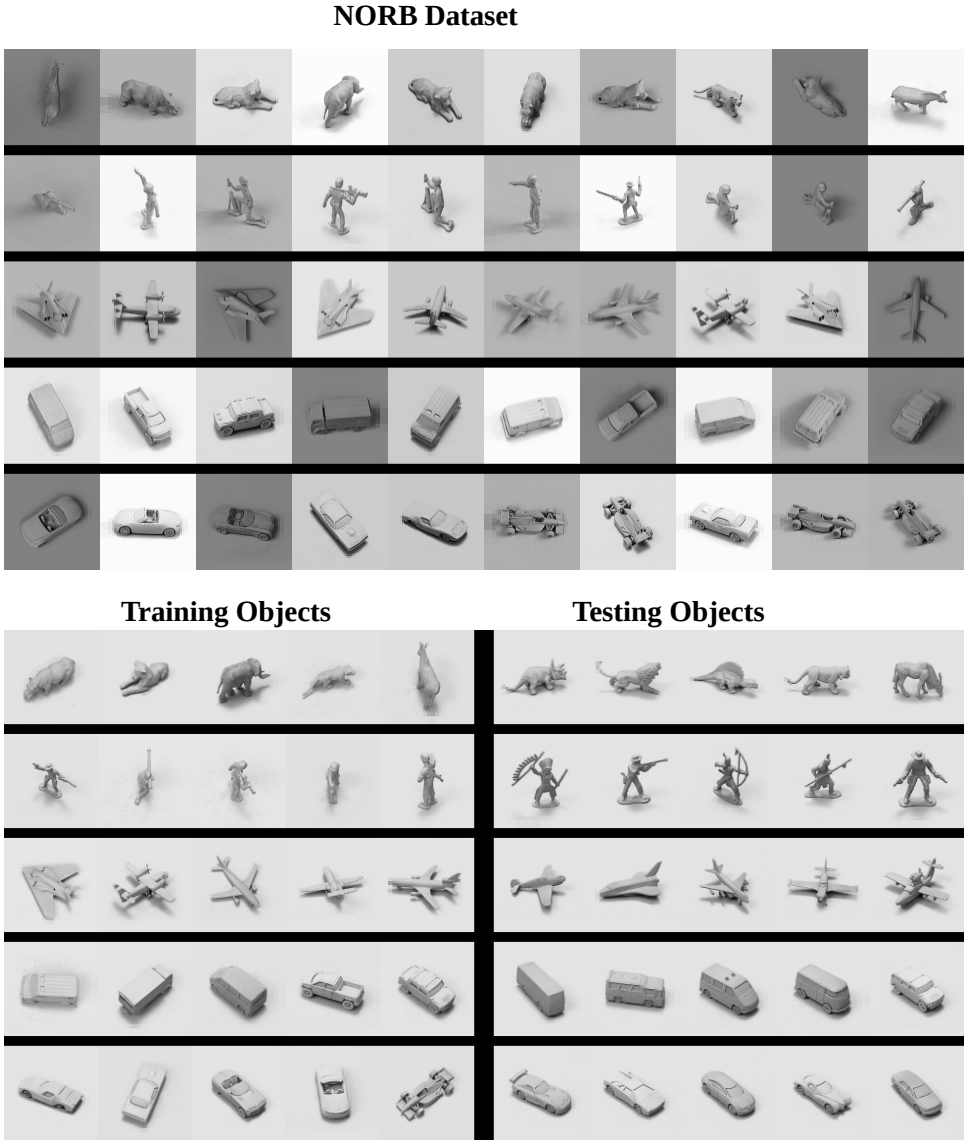

Figure A.3: **Top:** Random training samples from each of five generic object categories. **Bottom:** 25 training vs. 25 testing objects. Note that the training objects are quite different from the testing objects.

preprocessed format, where common stopwords were removed and all documents were stemmed. We again only consider the 2000 most frequently used words in the training dataset. The dataset is available at http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm.

NORB Dataset NORB (LeCun et al. [2004]) contains images of 50 different 3D toy objects with 10 objects in each of five generic classes: cars, trucks, planes, animals, and humans, shown in Fig. A.3, top panel. Each object is captured from 162 viewpoints (9 elevations, 18 azimuth) and under 6 lighting conditions. The training set contains 24,300 stereo image pairs of 25 objects, 5 per class, while the test set contains 24,300 stereo pairs of the remaining, different 25 objects. Figure A.3, bottom panel, shows 25 training and 25 test objects. The goal is to classify each previously unseen object into its generic class. From the training data, 4,300 were set aside for validation. The dataset is available at http://www.cs.nyu.edu/∼ylclab/data/norb-v1.0/

A.2 Details of Training

Details of the pretraining: To speed up the pretraining of each RBM, we subdivided all datasets into mini-batches, each containing 100 data vectors and updated the weights after each mini-batch. For datasets that are not divisible by the size of a minibatch, the remaining data vectors were included in the last minibatch. For all datasets, each hidden layer was pretrained for 50 passes (epochs) through the entire training set. The weights were updated using a learning rate of 0.1, momentum of 0.9, and a weight decay of $0.0002\times$weight$\times$learning rate. The weights were initialized with small random values sampled from a zero-mean normal distribution with standard deviation 0.01. When modeling real-valued Gaussian visible units, e.g. in the case of modeling Olivetti face patches, pretraining the first layer of features typically requires a much smaller learning rate to avoid oscillations. The learning rate was set to 0.001 and pretraining proceeded for 200 epochs.

During the pretraining stage, the visible units of standard binary RBM had real-valued activities, which were in the range $[0, 1]$. When training higher-level RBM's, the visible units were set to the activation *probabilities* of the hidden units in the lower-level RBM, but the hidden units of every RBM had stochastic binary values.

Details of the fine-tuning: For the fine-tuning, we used the method of conjugate gradients on larger minibatches containing 1000 data vectors. We used Carl Rasmussen's "minimize" code, available at http://www.kyb.tuebingen.mpg.de/bs/people/carl/code/minimize. Three line searches were performed for each mini-batch in each epoch. To determine an adequate number of epochs and to check for overfitting, we fine-tuned each model on a fraction of the training data and tested its performance on the remaining validation set. We then repeated the fine-tuning on the entire training set.

For many of the experiments presented in this thesis, we tried various values of the learning rate, momentum, and weight-decay parameters and we also tried training the RBM's for more epochs. We did not observe any significant differences in the final results after the fine-tuning. This suggests that the precise weights found by the greedy pretraining do not matter as long as it finds a good region from which to start the fine-tuning.

A.3 Details of Matlab code

Deep Autoencoders. The Matlab code for pretraining a stack of RBM's using Contrastive Divergence (CD1) and fine-tuning deep autoencoders is available at http://www.cs.toronto.edu/$\sim$rsalakhu/software/.

Estimating Partition Functions of RBM's. The Matlab code for estimating partition functions of RBM's using Annealed Importance Sampling is available at http://www.cs.toronto.edu/$\sim$rsalakhu/software/.

Pretraining and Learning Deep Boltzmann Machines. The Matlab code includes: pretraining a modified stack of RBM's, the new learning algorithm that combines mean-field inference along with stochastic approximation algorithm, and discriminative fine-tuning of Deep Boltzmann Machines. The code is available at: http://www.cs.toronto.edu/$\sim$rsalakhu/software/.