Inductive Principles for Learning Restricted Boltzmann Machines

by

Kevin Swersky

BSc. Computing Science, University of Alberta, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

August 2010

Abstract

We explore the training and usage of the Restricted Boltzmann Machine for unsupervised feature extraction. We investigate the many different aspects involved in their training, and by applying the concept of iterate averaging we show that it is possible to greatly improve on state of the art algorithms. We also derive estimators based on the principles of pseudo-likelihood, ratio matching, and score matching, and we test them empirically against contrastive divergence, and stochastic maximum likelihood (also known as persistent contrastive divergence). Our results show that ratio matching and score matching are promising approaches to learning Restricted Boltzmann Machines. By applying score matching to the Restricted Boltzmann Machine, we show that training an auto-encoder neural network with a particular kind of regularization function is asymptotically consistent. Finally, we discuss the concept of deep learning and its relationship to training Restricted Boltzmann Machines, and briefly explore the impact of fine-tuning on the parameters and performance of a deep belief network.

Table of Contents

Abstract . . . . . **ii**

Table of Contents . . . . . **iii**

List of Tables . . . . . **vi**

List of Figures . . . . . **vii**

Acknowledgments . . . . . **x**

1 Introduction **1**

1.1 Restricted Boltzmann Machines 2

1.2 Representing an RBM using Free Energy 3

1.3 Using RBMs for Classification 6

1.4 Gaussian Units 7

1.5 Thesis Contributions 9

2 Inductive Principles for Parameter Estimation **11**

2.1 Parameter Estimation 11

2.2 Maximum Likelihood 13

2.3 Stochastic Estimation Principles 15

2.3.1 Markov Chain Monte Carlo Maximum Likelihood Estimation 15

2.3.2 Contrastive Divergence 16

2.3.3 Stochastic Maximum Likelihood 18

2.4 Deterministic Estimation Principles 20

2.4.1 Maximum Pseudo-Likelihood 20
2.4.2 Ratio Matching 22
2.4.3 Score Matching 26
2.4.4 Generalized Score Matching 29
2.4.5 Alternatives 31

3 Connections Between Auto-Encoders and Gaussian Restricted Boltzmann Machines 32
3.1 Learning to Predict 33
3.2 Multilayer Perceptrons 35
3.3 Auto-Encoders 36
3.4 A Closer Look at Score Matching for the Gaussian-Binary RBM . 39
3.5 Deep Learning 42

4 Training Restricted Boltzmann Machines 45
4.1 Experiments Training RBMs on MNIST 45
4.1.1 Iterate Averaging 46
4.1.2 Learning Rate, Momentum, and Weight Decay 47
4.1.3 Mini-Batches 49
4.1.4 Binary v.s. Soft Data 50
4.1.5 Continuation Methods 52
4.1.6 Fast Weights and Averaging 54
4.2 Discussion 60

5 Experiments using Inductive Principles for Binary Data 61
5.1 Experiments with Simulated Data 61
5.1.1 Experiment Setup 62
5.1.2 Results 64
5.2 Experiments with Real Data 65
5.2.1 Experiment Setup 65
5.2.2 Classification 67
5.2.3 Likelihood 67
5.2.4 Denoising 68
5.2.5 Novelty Detection 69

5.2.6 Visualization of Learned Features 71
5.3 Computation Time 72
5.4 Discussion 74

6 Experiments using Inductive Principles for Continuous Data 75
6.1 Experiment Setup 75
6.2 Classification 76
6.3 Denoising 76
6.4 Discussion 79

7 Deep Learning 80
7.1 Comparing Algorithms for Training Deep Belief Networks 80
7.2 The Effects of Fine-Tuning 81
7.3 Greedy Fine-Tuning 84

8 Conclusion and Future Work 86

Bibliography 89

List of Tables

7.1 Test Error Rates based on greedily training a DBN with CD and SML, followed by supervised fine-tuning. 81
7.2 The effect of features learned by a stack of RBMs, and by fine-tuning a DBN 81
7.3 Error rates for a DBN of various sizes trained greedily with an RBM followed immediately by greedy fine-tuning. 85

List of Figures

1.1 An RBM with visible units v and hidden units h. The parameters W correspond to the strengths of the edge connections. 2

2.1 The Markov-chain Monte Carlo Maximum Likelihood algorithm. 16
2.2 The contrastive divergence algorithm with one-step sampling (CD-1). 18
2.3 The stochastic maximum likelihood algorithm. 19
2.4 Comparison of the link functions $g(u)$ between ratio matching and pseudo-likelihood. 25

3.1 A multilayer perceptron with M hidden layers, inputs v, and outputs $\widehat{y}$. 37
3.2 Greedy training of a deep belief network, where the weights being learned are shown in blue. Notice that in 3.2b, the weights in the first layer are frozen. The edges become directed and the first layer hidden units become deterministic, serving as the visible units for the next layer RBM. Also note that we can repeat this process to train as many layers as desired. 44

4.1 The contrastive divergence algorithm with one-step sampling using momentum. 48
4.2 Test error vs iteration for training an RBM using stochastic approximation with different mini-batch sizes. 51
4.3 Receptive fields from RBMs trained on MNIST with various weight decay settings. 53
4.4 Test Error vs Iteration for various settings of the weight decay parameter λ, as well as an RBM trained by annealing λ. 54

4.5 The stochastic maximum likelihood algorithm using fast weights. 56
4.6 The stochastic maximum likelihood algorithm using iterate averaging. 57
4.7 A comparison of fast weights and averaging for training an RBM using SML. 59

5.1 An algorithm to sample data from an RBM with a small number of hidden units. 62
5.2 10×10 digits from a downsized version of MNIST, and synthetic digits generated from an RBM with 100 visible units and 15 hidden units trained using maximum likelihood. 63
5.3 Test set KL vs dataset size for an RBM trained on synthetic data using various inductive principles. 64
5.4 Examples from the Caltech-101 Silhouettes dataset 66
5.5 Test set classification error from various RBM training methods using an SVM classifier on hidden activations. 68
5.6 Test set negative log-likelihood from various RBM training methods estimated with AIS. 69
5.7 Examples from Caltech Silhouettes after being corrupted with noise, then repaired by an RBM trained with CD. 70
5.8 Test set denoising mean squared error as a function of the percent of bits corrupted by noise. 71
5.9 Relative test set free energy as a function of the percent of bits corrupted by noise. 72
5.10 Randomly selected receptive fields and visible biases learned using various training methods on the MNIST dataset. The top left cell in each figure corresponds to the visible bias vector. 73

6.1 Receptive fields from Gaussian-binary RBMs trained on greyscale CIFAR-10 using various training methods. 77

6.2 Test set classification error on greyscale CIFAR-10 from various Gaussian-binary RBM training methods using an SVM classifier on hidden activations. The method labelled “RAW” represents the application of the SVM classifier to the raw image pixels. 78
6.3 Test set denoising mean squared error as a function of the variance of zero-mean corrupting additive Gaussian noise. 78

7.1 Comparison of the first layer weights of a DBN before and after performing fine-tuning with backpropagation. 82
7.2 Comparison of 2500 Data points from MNIST in the 2-dimensional hidden layer of a deep auto-encoder before and after fine-tuning with backpropagation. Each point represents a different example, and each class is represented with a different colour/marker combination. 83

Acknowledgments

This thesis would not be possible without the help and support of many wonderful people, and to them I would like to extend my utmost gratitude. First and foremost: my supervisor Nando de Freitas. He devoted countless hours to myself and to my work, and provided excellent guidance while giving me the freedom to explore my own ideas. His continuing support goes well beyond the call of duty; I truly could not ask for more in a supervisor. Benjamin Marlin for his brilliance, persistence, and patience. I hope he had as much fun working with me as I did with him. Bo Chen, Matt Hoffmann, David Duveneaud, Nimalan Mahendran, and Mohammad Emtiyaz Khan for always listening when I had a new idea, and for sharing their own ideas with me. Mark Schmidt for always somehow having the right intuition, or piece of software, to solve whatever problem I brought him, and professors Kevin Murphy and David Lowe for always being available for some advice or assistance. I would also like to acknowledge Nando's excellent students, many fellow students in LCI, and the crazy members of the UBC sushi group for all of the great times we had. Lastly, I would like to thank my family: I could not have done this without their tireless love and support.

Chapter 1

Introduction

The RBM is fast becoming a workhorse of probabilistic modeling.
— Anonymous AISTATS Reviewer 3 (2010)

Unsupervised Learning is a branch of Machine Learning that deals with automatically discovering interesting patterns from unlabelled data. Specifically, given a dataset $\mathbf{v_N} = \{v_1, v_2, ...v_N\}$ where each data point v_n is a D dimensional vector, we wish to train a model that will learn structure in the data that may not be readily apparent from the data itself. For example, we may wish to cluster the data into several possible categories, or we may wish to create a probabilistic model $p(v_1, ..., v_N|W)$ with parameters W that can be used to generate new, previously unseen examples. One of the most useful applications of Unsupervised Learning is the generation of K-dimensional feature vectors $\{h_1, h_2, ..h_N\}$ which can be used in place of the data to improve classifiers, denoise images, group data into semantically related clusters, and much more. In this thesis we will focus on the task of learning features from data and in particular we will focus on an increasingly popular model used for this purpose called the Restricted Boltzmann Machine.

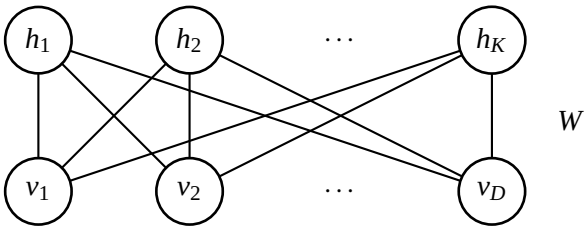


Figure 1.1: An RBM with visible units v and hidden units h. The parameters W correspond to the strengths of the edge connections.

1.1 Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM), shown in Figure 1.1, is a bipartite probabilistic graphical model corresponding to the following distribution:

$$p(v,h|W) = \frac{1}{Z(W)} \exp\left(-E(v,h,W)\right),$$

where $v \in V$ denotes the observation nodes, $h \in H$ the latent random variables and $W \in \mathbb{R}^{D\times K}$ the parameters governing the interactions between the D visible units and the K hidden units. The term $E(v,h,W)$ is called the energy function and $Z(W)$ is a normalizing term given by:

$$Z(W) = \sum_{v' \in V} \sum_{h' \in H} p(v',h'|W).$$

That is, $Z(W)$ involves a combinatorial summation over all possible visible and hidden states.

In the binary case where $V = \{0,1\}^D$ and $H = \{0,1\}^K$ the energy function can be expressed as:

$$E(v,h,W) = -\sum_{i=1}^{D}\sum_{j=1}^{K} v_i W_{ij} h_j - \sum_{i=1}^{D} v_i c_i - \sum_{j=1}^{K} h_j b_j.$$

In this case, the computation of $Z(W)$ involves $2^D \times 2^K$ terms.

Here we have added bias terms for the visible units and the hidden units, which are parameterized by c and b respectively. We can absorb these terms into W by simply adding a visible unit and a hidden unit which are always set to 1.

In general, RBMs can be used to model many types of data, and a procedure for deriving RBMs for general exponential family models can be found in [75]. For simplicity, we restrict our current description to the binary case, but will discuss Gaussian extensions in section Section 1.4.

RBMs have the property that given the hidden units, all of the visible units become independent and given the visible units, all of the hidden units become independent. Thus we can sample from the conditionals $p(v_i|h)$ and $p(h_j|v)$, $i \in \{1,...,D\}$, $j \in \{1,...,K\}$ exactly since they are logistic functions taking the form:

$$p(v_i = 1|h,W) = sigm\left(\sum_{j=1}^{K} W_{ij}h_j\right) \tag{1.1}$$

$$p(h_j = 1|v,W) = sigm\left(\sum_{i=1}^{D} W_{ij}v_i\right), \tag{1.2}$$

where $sigm(a) = \frac{1}{1+exp(-a)}$.

The ability to quickly infer $p(h_j = 1|v,W)$ is one of the most appealing features of the RBM. First, we will show that if we treat the expected value of the hidden units $E[h_j|v,W] = p(h_j = 1|v,W)$ as features, then using these features in place of the data itself yields significant improvements on a number of tasks. Secondly, these features can be used as a basis from which to learn even more features. This principle is called deep learning and it can produce a final set of features that yield state of the art performance in a number of tasks such as object recognition [65], document classification [58], semantic hashing [64], and more.

1.2 Representing an RBM using Free Energy

Due to the fact that the hidden units are independent given the visible units, we can easily marginalize out h to equivalently express the RBM in terms of the visible units only:

$$
\begin{aligned}
p(v|W) &\propto \sum_h exp(-E(v,h,W)) \qquad (1.3)\\
&= \sum_h exp(\sum_{i=1}^{D}\sum_{j=1}^{K} v_i W_{ij} h_j)\\
&= \sum_h \prod_{j=1}^{K} exp(\sum_{i=1}^{D} v_i W_{ij} h_j)\\
&= \prod_{j=1}^{K} \sum_{h_j \in \{0,1\}} exp(\sum_{i=1}^{D} v_i W_{ij} h_j)\\
&= \prod_{j=1}^{K} \left(1 + exp(\sum_{i=1}^{D} v_i W_{ij})\right).
\end{aligned}
$$

This version of the RBM is defined in terms of the so-called Free Energy $F(v,W)$:

$$p(v|W) = \frac{exp(-F(v,W))}{\sum_{v' \in V} exp(-F(v',W))}, \qquad (1.4)$$

$$F(v,W) = -\sum_{j=1}^{K} log\left(1 + exp\left(\sum_{i=1}^{D} W_{ij} v_i\right)\right). \qquad (1.5)$$

In this form, the partition function still has 2^D terms, and is thus still intractable for large values of D. One interesting feature of the RBM is that in the partition function, instead of summing over all visible states and marginalizing out the hidden states, we can instead do the opposite and sum over the hidden states while marginalizing out the visible states. That is, we can alternatively represent $Z(W)$ for a binary RBM as follows:

$$
\begin{aligned}
Z(W) &= \sum_{v \in V} \sum_{h \in H} exp\left(\sum_{i=1}^{D} \sum_{j=1}^{K} v_i W_{ij} h_j \right) \\
&= \sum_{h \in H} \sum_{v \in V} exp\left(\sum_{i=1}^{D} \sum_{j=1}^{K} v_i W_{ij} h_j \right) \\
&= \sum_{h \in H} \sum_{v \in V} \prod_{i=1}^{D} exp\left(\sum_{j=1}^{K} v_i W_{ij} h_j \right) \\
&= \sum_{h \in H} \prod_{i=1}^{D} \sum_{v_i \in \{0,1\}} exp\left(\sum_{j=1}^{K} v_i W_{ij} h_j \right) \\
&= \sum_{h \in H} \prod_{i=1}^{D} \left(1 + exp\left(\sum_{j=1}^{K} W_{ij} h_j \right) \right).
\end{aligned}
$$

In this form, $Z(W)$ has a computational complexity of $D \times 2^K$. If K is small enough then we can compute the partition function by directly summing over the 2^K hidden states. Keeping K small lessens the representational power of the RBM, but it will be useful for our simulation experiments later. For the sake of clarity, Equation 1.6 gives the RBM distribution in this alternative form.

$$
p(v|W) = \frac{exp\left(\sum_{i=1}^{D} \sum_{j=1}^{K} v_i W_{ij} h_j \right)}{\sum_{h' \in H} \prod_{i=1}^{D} \left(1 + exp\left(\sum_{j=1}^{K} W_{ij} h'_j \right) \right)}. \tag{1.6}
$$

It is useful at this time to show the gradient of the free energy for the binary RBM, which will be useful when learning the parameters W:

$$\begin{aligned}
\nabla_{W_{dk}} F(v,W) &= \nabla_{W_{dk}} - log\left(\sum_{h\in H} exp\left(\sum_{i=1}^{D}\sum_{j=1}^{K} v_i W_{ij} h_j\right)\right) \\
&= -\sum_{j=1}^{K} \nabla_{W_{dk}} log\left(1 + exp\left(\sum_{i=1}^{D} W_{ij} v_i\right)\right) \\
&= -\frac{v_d exp\left(\sum_{i=1}^{D} W_{ik} v_i\right)}{1 + exp\left(\sum_{i=1}^{D} W_{ik} v_i\right)} \\
&= -v_d E[h_k|v,W], \qquad (1.7)
\end{aligned}$$

where $E[h_k|v,W] \equiv p(h_k = 1|v,W)$ and $\nabla_{W_{dk}}$ represents the partial derivative with respect to W_{dk}.

1.3 Using RBMs for Classification

The free energy representation is useful when we wish to use an RBM to perform classification. In particular, if we let y represent a 1 of C vector where C is the number of classes, and let v represent the unsupervised component; we can append them to create a new vector $\begin{bmatrix} v \\ y \end{bmatrix}$. The new energy function is given by

$$E(v,y,h,W) = -\sum_{c=1}^{C}\sum_{j=1}^{K} y_c W_{cj} h_j - \sum_{i=1}^{D}\sum_{j=1}^{K} v_i W_{ij} h_j. \qquad (1.8)$$

This model is virtually identical to the fully unsupervised RBM with the exception that the class-units y are now sampled according to a softmax distribution:

$$p(y_c = 1, y_{-c} = 0|h,W) = \frac{exp(\sum_{j=1}^{K} W_{cj} h_j)}{\sum_{c'} exp(\sum_{j=1}^{K} W_{c'j} h_j)}, \qquad (1.9)$$

where y_{-c} denotes all other class units except c.

Using the Free Energy form, we can classify a test set point $\bar{v}$ according to the following rule:

$$class(\bar{v}|W) = \arg\max_c \frac{\prod_{j=1}^{K}\left(1+exp\left(W_{cj}+\sum_{i=1}^{D}\bar{v}_i W_{ij}\right)\right)}{\sum_{c'}\prod_{j=1}^{K}\left(1+exp\left((W_{c'j}+\sum_{i=1}^{D}\bar{v}_i W_{ij}\right)\right)}. \tag{1.10}$$

1.4 Gaussian Units

The RBM can be extended so that the visible units can take on a different exponential family distribution when conditioned on the hidden units and vice versa. A popular extension to the RBM for modelling natural images is to make the visible units conditionally Gaussian given the hidden units so that they can take on real values. We will call this model the Gaussian-Binary RBM to specify Gaussian visible units and binary hidden units. In order to define this new RBM, we add a new set of parameters $\sigma = \{\sigma_1, \sigma_2, ..., \sigma_D\}$. The energy for the Gaussian-binary RBM (GBRBM) is given by:

$$E(v,h,W,c,b,\sigma) = -\sum_{i=1}^{D}\sum_{j=1}^{K}\frac{v_i}{\sigma_i}W_{ij}h_j - \sum_{j=1}^{K}b_j h_j + \frac{1}{2}\sum_{i=1}^{D}\frac{(c_i-v_i)^2}{\sigma_i^2}. \tag{1.11}$$

From this we can also derive the corresponding free energy:

$$
\begin{aligned}
F(v,W,c,b,\sigma) &= -log\left(\sum_{h\in H} exp(-E(v,h,W,c,b,\sigma))\right) \\
&= -log\left(\sum_{h\in H} exp\left(\sum_{i=1}^{D}\sum_{j=1}^{K}\frac{v_i}{\sigma_i}W_{ij}h_j + \sum_{j=1}^{K} b_j h_j - \frac{1}{2}\sum_{i=1}^{D}\frac{(c_i-v_i)^2}{\sigma_i^2}\right)\right) \\
&= -log\left(exp\left(-\frac{1}{2}\sum_{i=1}^{D}\frac{(c_i-v_i)^2}{\sigma_i^2}\right)\sum_{h\in H} exp\left(\sum_{i=1}^{D}\sum_{j=1}^{K}\frac{v_i}{\sigma_i}W_{ij}h_j + \sum_{j=1}^{K} b_j h_j\right)\right) \\
&= \left(\frac{1}{2}\sum_{i=1}^{D}\frac{(c_i-v_i)^2}{\sigma_i^2}\right) - log\left(\sum_{h\in H}\prod_{j=1}^{K} exp\left(\sum_{i=1}^{D}\frac{v_i}{\sigma_i}W_{ij}h_j + b_j h_j\right)\right) \\
&= \left(\frac{1}{2}\sum_{i=1}^{D}\frac{(c_i-v_i)^2}{\sigma_i^2}\right) - log\left(\prod_{j=1}^{K}\left(1+exp\left(\sum_{i=1}^{D}\frac{v_i}{\sigma_i}W_{ij} + b_j\right)\right)\right) \\
&= \left(\frac{1}{2}\sum_{i=1}^{D}\frac{(c_i-v_i)^2}{\sigma_i^2}\right) - \sum_{j=1}^{K} log\left(1+exp\left(\sum_{i=1}^{D}\frac{v_i}{\sigma_i}W_{ij} + b_j\right)\right).
\end{aligned}
\tag{1.12}
$$

In this case it is more clear to describe the RBM model using explicit biases. Using this new energy function we can derive the conditional distributions:

$$
p(v_i|h,W,c,b,\sigma) \sim \mathcal{N}\left(c_i + \sigma_i\sum_j W_{ij}h_j, \sigma_i^2\right) \tag{1.13}
$$

$$
p(h_j = 1|v,W,c,b,\sigma) = sigm\left(b_j + \sum_{i=1}^{D} W_{ij}\frac{v_i}{\sigma_i}\right). \tag{1.14}
$$

Finally, it will be useful to also write the gradients of the free energy with respect to the different parameters.

$$\nabla_{W_{dk}} F(v,W,c,b,\sigma) = -\frac{v_d}{\sigma_d} sigm\left(b_k + \sum_{i=1}^{D} \frac{v_i}{\sigma_i} W_{ik}\right), \tag{1.15}$$

$$\nabla_{c_d} F(v,W,c,b,\sigma) = \frac{(c_d - v_d)}{\sigma_d^2}, \tag{1.16}$$

$$\nabla_{b_k} F(v,W,c,b,\sigma) = -sigm\left(b_k + \sum_{i=1}^{D} \frac{v_i}{\sigma_i} W_{ik}\right), \tag{1.17}$$

$$\nabla_{\sigma_d} F(v,W,c,b,\sigma) = -\frac{(c_d - v_d)}{\sigma_d^3} + \sum_{j=1}^{K} \frac{v_d W_{dj} sigm\left(b_j + \sum_{i=1}^{D} \frac{v_i}{\sigma_i} W_{ij}\right)}{\sigma_d^2}. \tag{1.18}$$

Similar to the binary case, we will define $E[h_j|v,W,c,b,\sigma] = p(h_j = 1|v,W,c,b,\sigma) = sigm\left(b_j + \sum_{i=1}^{D} \frac{v_i}{\sigma_i} W_{ij}\right)$.

We will make two simplifications to this model: we will not attempt to learn the visible variances, since we didn't notice any significant performance increase in doing this. We will also set all σ_i to the same value.

It is also possible to change the definition of the hidden units so that they are conditionally Gaussian. This approach is applied for use in collaborative filtering and dimensionality reduction [19, 66]. For simplicity when the hidden units are Gaussian, their σ parameters can be set to 1 so that their conditional distributions will correspond to Gaussian distributions with unit variance.

1.5 Thesis Contributions

This thesis makes several contributions toward improving our theoretical and practical knowledge of RBMs. First, we formulate several inference principles for RBMs in Chapter 2. Some of these principles, such as contrastive divergence, persistent contrastive divergence, and Markov chain Monte Carlo maximum likelihood estimation have been studied before. Others, such as ratio matching and score matching are novel in this context. Second, in Chapter 3 we connect the training of Gaussian-binary RBMs to the training of auto-encoders, showing that training a GBRBM can be seen as being equivalent to training an auto-encoder with a particular kind of regularization term. In Chapter 4 we explore in depth the

training of RBMs by stochastic gradient descent, and introduces iterate averaging [30, 52] as a way to accelerate convergence. In particular, it appears that averaging greatly improves the convergence of RBM training over current state of the art algorithms. In Chapter 5 we compare several inference principles for RBMs with binary data over several datasets and several tasks in order to determine the strengths and weaknesses of each algorithm in terms of learning unsupervised features from data. In Chapter 6 we extend this briefly to the case of continuous visible units, and show that score matching is competitive with currently used stochastic algorithms. Finally, in Chapter 7 we look at deep learning, and examine whether optimizing the likelihood of an RBM directly leads to the best performance in a deep belief network. We also examine the effects of fine-tuning a deep belief network, and suggest a way to train one in an entirely greedy manner in order to determine the correct number of layers.

Chapter 2

Inductive Principles for Parameter Estimation

In the previous chapter we discussed the Restricted Boltzmann Machine as a tool for probabilistic modelling and feature extraction. In this chapter we will discuss how we can fit the parameters of the RBM so that the features it infers will reflect the underlying patterns of a given dataset. We will find that the most popular method for doing so is also computationally infeasible, and discuss alternative strategies that can make learning feasible, while still producing good results.

The contents of this chapter are based on the results found in [42], which was work done with Benjamin Marlin, Bo Chen, and Nando de Freitas. In particular, we would like to acknowledge Benjamin Marlin for his concise description of why generalized score matching is impractical for real world situations, and Bo Chen for his work on developing the link function view of pseudo-likelihood and ratio matching.

2.1 Parameter Estimation

Consider a parametric model $p(\cdot|W) \equiv p_W(\cdot)$. $p_W(\cdot)$ can be any sort of probabilistic model, including an RBM. Parameter estimation is concerned with finding a setting of the parameters W such that the distribution described by $p_W(\cdot)$ matches the empirical distribution of $\mathbf{v}$ which we will call $p_e(\cdot)$. Another way of stating this

is that for some parameter setting W^*: $\mathbf{v} \sim p(\cdot|W^*)$. In order to do this we will look at several different estimation methods (also called estimators). Three properties that are desirable for any estimation method are asymptotic consistency, normality, and efficiency.

Suppose that for a dataset of size N, an estimation method were to recover parameters $\widehat{W}_N$. Asymptotic consistency means that as $N \to \infty$, $\widehat{W}_N \to W^*$. An estimator which is consistent is said to be unbiased, and this is generally an important property to consider when choosing between different methods.

Consider sampling datasets of size N and for each of these using an estimation method to find parameters $\widehat{W}_N$. If in the infinite limit these estimates follow the distribution $\sqrt{N}(W^* - \widehat{W}_N) \sim \mathcal{N}(0, \Sigma)$ then the estimator is considered asymptotically normal. That is, the estimates produced by this method will tend to cluster around the true parameter value according to a normal distribution with variance Σ.

For a consistent and asymptotically normal estimator, efficiency refers to how quickly the variance Σ diminishes as N increases. An efficient unbiased estimator will attain the minimum variance Σ possible for any unbiased estimator. An intuitive way of looking at this is that an estimator which is inefficient is effectively utilizing only a fraction of the data that an efficient estimator would use [74, p. 131].

In the next section we will consider the principle of maximum likelihood, which is an estimation method that posesses all three of the above properties. Unfortunately due to the partition function $Z(W)$, it is computationally infeasible to perform maximum likelihood for an RBM in general. We will therefore consider other estimation techniques that either seek to approximate maximum likelihood directly, or which sacrifices one or more of the above properties in order to become computationally feasible.

As an added caution, it is worthwhile to clarify that for many models (including an RBM), very few estimators admit closed-form solutions. In this case iterative approaches like gradient descent must be considered which start from some random parameter settings and converge to some local optimum. If the objective function resulting from applying an estimation method is non-convex, then there may exist many sub-optimal minima. These properties therefore only truly apply when we

have a perfect optimization routine which is able to reach the global minimum, but they are nevertheless useful in informing our decision of which estimators to use.

For the remainder of this chapter, we will use $\mathscr{L}$ to represent loss functions over general RBMs, and J to represent a loss function for an RBM with a specific distribution over its units.

2.2 Maximum Likelihood

The maximum likelihood principle states that we should assign the highest probability to the given data. In our case we assume the data is independent and identically distributed so that we can express the estimated parameters $\widehat{W}$ in terms of the following objective:

$$\widehat{W} = \arg\max_{W} \prod_{n=1}^{N} p(v_n|W), \tag{2.1}$$

where v_n refers to the n^{th} data point.

It is often more convenient to work with the logarithm of this objective when performing maximization. This does not change the solution as the logarithm is a monotonic transformation. To ensure the scaling doesn't change when we perform the maximization, we also divide the objective by N. As is commonly done in optimization literature, we can express this as a minimization problem by negating the objective. We can thus express the maximum likelihood problem as:

$$\widehat{W} = \arg\min_{W} -\frac{1}{N} \sum_{n=1}^{N} log\left(p(v_n|W)\right). \tag{2.2}$$

Maximum likelihood can be equivalently viewed as minimizing the Kullback-Leibler divergence $KL(p_e||p_W)$ between the empirical distribution p_e and the model distrubution p_W:

$$
\begin{aligned}
KL(p_e||p_W) &= \sum_{v\in V} p_e(v) log\left(\frac{p_e(v)}{p_W(v)}\right) \\
&= \sum_{v\in V} p_e(v) log\left(p_e(v)\right) - \sum_{v\in V} p_e(v) log\left(p_W(v)\right) \\
&= -\sum_{v\in V} p_e(v) log\left(p_W(v)\right) + const \\
&= -E_{p_e}\left[log\left(p_W(v)\right)\right] + const, \qquad (2.3)
\end{aligned}
$$

where $E_{p_e}\left[log\left(p_W(v)\right)\right]$ denotes the expected log-likelihood under the empirical distribution. Here we have set the terms that do not depend on p_W to be constant, since they do not involve the parameters W.

We can replace this expectation with it's Monte Carlo estimate by using a dataset which has been generated from the empirical distribution. In the limit of infinite data this will equal the expectation, and thus we get back the negative of Equation 2.2 (plus a constant term):

$$
KL(p_e||p_W) \approx const - \frac{1}{N}\sum_{n=1}^{N} log\left(p_W(v_n)\right). \qquad (2.4)
$$

We are now back to the original maximum likelihood objective. The important message here is that we are trying to minimize some form of difference between the empirical distribution and the model distribution. This is a recurring theme which will be present in most of the estimators introduced in this chapter.

Maximum likelihood is often the preferred method for learning model parameters since it is consistent, asymptotically normal, and efficient. However, it is known to suffer from several defects including unbounded likelihoods, a potentially large bias in the small-sample setting, and the possibility that the maximum of the likelihood is in a region with small probability mass [40]. Despite these defects, maximum likelihood still tends to perform quite well in practice, especially if we plan to use the likelihood to perform tasks like classification.

For a general RBM, the objective and its gradient would be expressed as:

$$\mathscr{L}_{ML}(W) = -\frac{1}{N}\sum_{n=1}^{N} -F(v_n,W) - log\,(Z(W)) \tag{2.5}$$

$$\begin{aligned}\nabla\mathscr{L}_{ML}(W) &= -\frac{1}{N}\sum_{n=1}^{N} -\nabla F(v_n,W) - \frac{\nabla Z(W)}{Z(W)} \\ &= -\frac{1}{N}\sum_{n=1}^{N} -\nabla F(v_n,W) + \sum_{v'\in v}\nabla F(v',W)\frac{exp\,(-F(v',W))}{Z(W)} \\ &= \frac{1}{N}\sum_{n=1}^{N}\nabla F(v_n,W) - \sum_{v'\in v}\nabla F(v',W)p(v'|W). \end{aligned} \tag{2.6}$$

This shows that computing the gradient also requires computing the partition function $Z(W)$. Thus, for large values of D it is also intractable.

Learning an RBM using Equation 2.6 is equivalent to matching the expected gradient of its free energy under the data distribution with the expected gradient under the model distribution. Intuitively, it is lowering the free energy (thus raising the probability) of the data while raising the free energy of everything else.

2.3 Stochastic Estimation Principles

While it may seem that finding the maximum likelihood solution is an infeasible problem, there are ways around the intractable partition function. In the following sections we will explore some methods that seek to approximate expectations over the model distribution by Markov chain Monte Carlo (MCMC) sampling. Using numerical sampling, we can generate reasonable estimates of the gradient, and using the guidelines provided by stochastic approximation [31], we can ensure that we reach a local minimum with a gradient descent procedure.

2.3.1 Markov Chain Monte Carlo Maximum Likelihood Estimation

An easy way to obtain gradient estimates is to approximate the model expectation by simulating M data points from $p(v,h|W)$ and averaging the resulting gradients of the free energy under these points. Since we cannot evaluate $p(v,h|W)$ directly, we can resort to using Markov Chain Monte Carlo (MCMC). In particular, we can use a block-Gibbs sampler [14] by repeatedly simulating v from $p(v|h,W)$ and using

1. Set $t = 1$.
2. Alternate sampling $\widetilde{h}$ from $p(h|\widetilde{v}, W^{(t)})$ and $\widetilde{v}$ from $p(v|\widetilde{h}, W^{(t)})$ until the equilibrium distribution is reached.
3. Collect M samples of $\widetilde{v}$.
4. set $W_{dk}^{(t+1)} = W_{dk}^{(t)} - \eta^{(t)} \left[-\frac{1}{N} \sum_{n=1}^{N} v_{dn} E[h_k|v_n, W^{(t)}] + \frac{1}{M} \sum_{m=1}^{M} \widetilde{v_{dm}} E[h_k|\widetilde{v_m}, W^{(t)}] \right]$.
5. Increase t to $t+1$ and iterate again.

Figure 2.1: The Markov-chain Monte Carlo Maximum Likelihood algorithm.

this to sample h from $p(h|v, W)$. We will refer to this algorithm as Markov chain Monte Carlo maximum likelihood estimation (MCMC-MLE) [15] and it is shown in Figure 2.1.

Note that in this algorithm, we can run several Markov chains in parallel. For simplicity, we run one chain per data instance. As long as $\eta^{(t)}$ is chosen carefully, such as a schedule following the Robbins Monro conditions [60], then this algorithm can be shown to converge to a local minimum of Equation 2.5. The issue with this approach, however, is running the Gibbs-sampler to equilibrium. The number of steps required to do this can be exponential in D, therefore this approach is still computationally demanding.

2.3.2 Contrastive Divergence

Contrastive divergence (CD) [17] is an alternative estimator based on the idea of starting the Markov chain at the data, and truncating it after only a few iterations, rather than letting it run to equilibrium. In fact, it has been observed to work quite well even when the chain is truncated after only a single iteration. Let $Q_W^t(v)$ be the distribution obtained by applying t steps of an MCMC kernel to the distribution p_e. Constrastive divergence is formally defined as minimizing the difference between $KL(p_e||p_W)$ and $KL(Q_W^t||p_W)$. For a general RBM we can express this as:

$$
\begin{aligned}
\mathscr{L}_{CD}(W) &= KL(p_e||p_W) - KL(Q_W^t||p_W) \\
&= \sum_{v\in V} p_e(v) log\left(\frac{p_e(v)}{p_W(v)}\right) - Q_W^t(v) log\left(\frac{Q_W^t(v)}{p_W(v)}\right) \\
&= const + \sum_{v\in V} -p_e(v) log\left(p_W(v)\right) - Q_W^t(v) log\left(Q_W^t(v)\right) + Q_W^t(v) log\left(p_W(v)\right) \\
&= const + \sum_{v\in V} \Bigg(p_e(v)F(v,W) + p_e(v)Z(W) - Q_W^t(v) log\left(Q_W^t(v)\right) \\
&\quad - Q_W^t(v)F(v,W) - Q_W^t(v)Z(W) \Bigg) \\
&= const + \sum_{v\in V} p_e(v)F(v,W) - Q_W^t(v) log\left(Q_W^t(v)\right) - Q_W^t(v)F(v,W).
\end{aligned}
\tag{2.7}
$$

Since for a general distribution $p(\cdot)$, $\sum_{v\in V} p(v)Z(W) = Z(W)\sum_{v\in V} p(v) = Z(W)$.

Notice that in this case the partition function is cancelled out, so we no longer have to compute it. We can now express the gradient of this objective as:

$$
\begin{aligned}
\nabla\mathscr{L}_{CD}(W) &= \sum_{v\in V} p_e(v)\nabla F(v,W) - \sum_{v'\in V} Q_W^t(v)\nabla F(v,W) \\
&\quad - \sum_{v\in V} \nabla Q_W^t(v)F(v,W) - \sum_{v'\in V} \nabla\left(Q_W^t(v) log\left(Q_W^t(v)\right)\right) \qquad (2.8) \\
&\approx \frac{1}{N}\sum_{n=1}^{N} \nabla F(v_n,W) - \frac{1}{N}\sum_{n=1}^{N} \nabla F(\widetilde{v_n},W), \qquad (2.9)
\end{aligned}
$$

where $\widetilde{v_n}$ is a sample obtained by starting from the data point v_n and applying t steps of Gibbs sampling.

Effectively, contrastive divergence replaces the intractable model expectation with an expectation that can be approximated by running t steps of a Gibbs sampler starting from the data. The last two terms of Equation 2.8, while generally intractable, have been determined empirically to be small compared to the first two terms and are therefore ignored [3, 17].

It has been shown in [10] that in general, the fixed points of CD will differ from

1. Set $t = 1$.
2. For each data point v_n: sample $\widetilde{h_n}^{(t)}$ from $p(h|v_n, W^{(t)})$, then sample $\widetilde{v_n}^{(t)}$ from $p(v|\widetilde{h_n}^{(t)}, W^{(t)})$.
3. Perform a CD-1 update:

$$W_{dk}^{(t+1)} = W_{dk}^{(t)} - \eta^{(t)} \left[-\frac{1}{N} \sum_{n=1}^{N} v_{dn} E[h_k|v_n, W^{(t)}] + \frac{1}{N} \sum_{n=1}^{N} \widetilde{v_{dn}}^{(t)} E[h_k|\widetilde{v_n}^{(t)}, W^{(t)}] \right]$$

4. Increase t to $t+1$ and iterate again.

Figure 2.2: The contrastive divergence algorithm with one-step sampling (CD-1).

those of maximum likelihood, but assuming the data is generated from an RBM it can be shown that asymptotically they both share the maximum likelihood solution as a fixed point [41]. In [77] conditions were given to guarantee convergence of CD, however they are difficult to satisfy in practice. Empirically, it appears that CD is a biased estimator, but that this bias is small [3, 10]. For a binary RBM the algorithm for taking one step, CD-1, is given in Figure 2.2.

CD has the effect of lowering the model free energy in regions around the training data. Although locally this is the correct thing to do, there is no guarantee that there won't be regions of higher probability mass far away from the data distribution that cannot be reached by the Markov chain.

One issue with CD is that since we cannot compute the objective function, it is difficult to measure convergence. For CD, reconstruction error $\sum_{n=1}^{N} ||v_n - \widetilde{v_n}||^2$ is generally used instead. Taking the difference between the free energies under the data and t-step distributions might also be an acceptable substitute, however to our knowledge this hasn't been used before.

2.3.3 Stochastic Maximum Likelihood

A beautiful alternative to MCMC-MLE given in [76], which is also computationally efficient, is the idea of stochastic maximum likelihood (SML); also known as

1. Set $t = 1$, and $\widetilde{h_n}^{(0)}$ to be a random K-dimensional binary vector.
2. Sample $\widetilde{v_n}^{(t)}$ from $p(v|\widetilde{h_n}^{(t-1)}, W^{(t)})$, and $\widetilde{h_n}^{(t)}$ from $p(h|\widetilde{v_n}^{(t)}, W^{(t)})$.
3. Update the parameters:

$$W_{dk}^{(t+1)} = W_{dk}^{(t)} - \eta^{(t)} \left[-\frac{1}{N} \sum_{n=1}^{N} v_{dn} E[h_k|v_n, W^{(t)}] + \frac{1}{N} \sum_{n=1}^{N} \widetilde{v_{dn}}^{(t)} E[h_k|\widetilde{v_n}^{(t)}, W^{(t)}] \right]$$

4. Increase t to $t+1$ and iterate again.

Figure 2.3: The stochastic maximum likelihood algorithm.

persistent contrastive divergence (PCD) [69]. This is quite similar to CD, except that instead of resetting the chain to the data after each parameter update, the previous state of the chain is kept and used for the next iteration of the algorithm. Unlike CD, this is a consistent estimator, and it can be shown that for an appropriate step size schedule $\eta^{(t)}$ this algorithm will almost surely converge to the maximum likelihood solution, even when only generating a single sample with each update. The SML algorithm is given in Figure 2.3.

While very similar to CD, this algorithm has the nice property that the Markov chain can explore regions away from the training data, since the chain isn't reset at each iteration. This means that if there are regions of higher probability mass than the training data, it is possible for the Markov chain to reach this area so that the distribution can be adjusted accordingly.

One small difference between SML and PCD is that in SML, we use the hidden state from the previous iteration as the starting point for the samples in the current iteration. In PCD, the state of the visible units from the previous iteration is used instead. We note that the same theory applies to both cases, and that this choice did not affect performance in our experiments.

Unfortunately, it is even more difficult to monitor the convergence of SML since it is based on directly minimizing the maximum likelihood objective. We have found that one-step reconstruction error also works reasonably well however.

2.4 Deterministic Estimation Principles

In the following sections, we will describe consistent estimation principles which do not need to rely on sampling. These work under the idea that matching some attribute of the model distribution to the empirical distribution, such as the conditionals $p(v_i|v_{-i})$, is equivalent to matching the distributions themselves (at least asymptotically). Unlike the approximate schemes above, these methods are all based on computable objective functions which means that it is relatively easy to assess their convergence. Another advantage is that we can apply powerful optimization routines such as L-BFGS [49, p. 224] to ensure that the objective is rapidly minmimized. In exchange, however, we sacrifice asymptotic efficiency. In many cases, the amount by which efficiency is actually reduced remains an open question.

2.4.1 Maximum Pseudo-Likelihood

The principle underlying maximum pseudo-likelihood (PL) [5] states that we should select parameters that maximize the product of all conditionals $p(v_i|v_{-i},W)$.

$$\begin{aligned}\widehat{W} &= \arg\max_W \prod_{n=1}^{N}\prod_{i=1}^{D} log(p(v_{in}|v_{-in},W)) \\ &= \arg\min_W -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} log(p(v_{in}|v_{-in},W)).\end{aligned} \tag{2.10}$$

This is equivalent to minimizing the pseudo-KL divergence between the data distribution and the model distribution:

$$\begin{aligned}PKL(p_e||p_W) &= \sum_{v\in V} p_e(v)\sum_{i=1}^{D}\left(log\left(p_e(v_i|v_{-i})\right) - log\left(p_W(v_i|v_{-i})\right)\right) \\ &= const - \sum_{v\in V} p_e(v)\sum_{i=1}^{D} log\left(p_W(v_i|v_{-i})\right) \\ &\approx const - \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} log\left(p_W(v_{in}|v_{-in})\right).\end{aligned} \tag{2.11}$$

For a general RBM, we can express the pseudo-likelihood objective as:

$$\mathscr{L}_{PL}(W) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} log\left(\frac{\frac{exp(-F(v_n,W))}{Z(W)}}{\sum_{v_i'\in V_i}\frac{exp(-F(v_{in}',v_{-in},W))}{Z(W)}}\right) \tag{2.12}$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} log\left(\frac{exp(-F(v_n,W))}{\sum_{v_i'\in V_i} exp(-F(v_{in}',v_{-in},W))}\right)$$

$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} F(v_n,W) + log\left(\sum_{v_i'\in V_i} exp(-F(v_{in}',v_{-in},W))\right), \tag{2.13}$$

where $v_n = (v_{in}, v_{-in})$, the set V_i denotes the domain of the i^{th} dimension of V, and $F(v_i', v_{-i}, W)$ denotes the free energy where the i^{th} dimension of v has been set to v_i'.

If the summation over a single dimension is tractable, then we can compute the exact PL objective. For example, in a binary RBM we can write the PL objective and its gradient as:

$$J_{PL}(W) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} log\left(\frac{p(v_n|W)}{p(v_n|W) + p(\neg^i v_n|W)}\right)$$

$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} F(v_n,W) + log\left(exp(-F(v_n,W)) + exp(-F(\neg^i v_n,W))\right) \tag{2.14}$$

$$\nabla J_{PL}(W) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} -\nabla F(v_n,W) - \nabla log\left(exp(-F(v_n,W)) + exp(-F(\neg^i v_n,W))\right)$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} -\nabla F(v_n,W) + \left(\frac{exp(-F(v_n,W))\nabla F(v_n,W) + exp(-F(\neg^i v_n,W))\nabla F(\neg^i v_n,W)}{exp(-F(v_n,W)) + exp(-F(\neg^i v_n,W))}\right)$$

$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} \nabla F(v_n,W) - \sum_{\xi_i\in\{0,1\}} \nabla F(\xi_i, v_{-in}, W)\, p(\xi_i|v_{-in},W), \tag{2.15}$$

where $p(\xi_i|v_{-i},W)$ is the conditional probability with dimension i equal to ξ_i, and $\neg^i v$ represents the variable v with the i^{th} bit "flipped":

$$\overset{\neg i}{v_i} = \begin{cases} 0, & v_i = 1 \\ 1, & v_i = 0 \end{cases}$$

Equation 2.15 shows that PL is quite similar to maximum likelihood, with the exception that instead of trying to match the expected free energies under the joint distribution $p(v|W)$, it tries to match them under the conditional distributions $p(\xi_i|v_{-i},W)$.

Another useful way of looking at PL for the binary RBM is through the ratio of probabilities $u_{in} \doteq \frac{p(v_n|W)}{p(\overset{\neg i}{v_n}|W)} = \frac{exp(-F(v_n,W))}{exp(-F(\overset{\neg i}{v_n},W))}$. Notice that since u_i is a ratio of probabilities, the normalization constants will cancel. We can represent the PL objective and its gradient in terms of this ratio as follows:

$$\begin{aligned} J_{PL}(W) &= -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} log\left(\frac{p(v_n|W)}{p(v_n|W)+p(\overset{\neg i}{v_n}|W)}\right) \\ &= -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} log\left(\frac{\frac{p(v_n|W)}{p(\overset{\neg i}{v_n}|W)}}{\frac{p(v_n|W)}{p(\overset{\neg i}{v_n}|W)}+1}\right) \\ &= -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D} log\left(\frac{u_{in}}{u_{in}+1}\right) \qquad (2.16) \\ \nabla J_{PL} &= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D}\frac{1}{1+u_{in}}\left(\nabla F(v_n,W)-\nabla F(\overset{\neg i}{v_n},W)\right). \qquad (2.17) \end{aligned}$$

2.4.2 Ratio Matching

Ratio matching (RM) [24] seeks the parameters that minimize the euclidean distance between the data conditional distributions and the model conditional distributions:

$$\widehat{W} = \arg\min_W \sum_{v\in V} p_e(v) \sum_{i=1}^{D} \sum_{\xi_i\in\{0,1\}} \left(p_e(\xi_i|v_{-i}) - p_W(\xi_i|v_{-i})\right)^2 \tag{2.18}$$

$$\begin{aligned} &= \arg\min_W \sum_{v\in V} p_e(v) \sum_{i=1}^{D} \sum_{\xi_i\in\{0,1\}} p_e(v)\left(p_e(\xi_i|v_{-i})^2 - 2p_e(\xi_i|v_{-i})p_W(\xi_i|v_{-i}) + p_W(\xi_i|v_{-i})^2\right) \\ &= \arg\min_W \sum_{v\in V} p_e(v) \sum_{i=1}^{D} \sum_{\xi_i\in\{0,1\}} p_e(v)p_e(\xi_i|v_{-i})^2 - 2p_e(v)p_e(\xi_i|v_{-i})p_W(\xi_i|v_{-i}) + p_e(v)p_W(\xi_i|v_{-i})^2. \end{aligned} \tag{2.19}$$

The first term in Equation 2.19 is constant with respect to W, and the third term can be approximated using samples from the empirical distribution. The second term, however, poses more of a challenge since it requires directly knowing the empirical distribution $p_e(v)$ in order to determine the conditionals $p_e(\xi_i|v_{-i})$. Luckily as shown in [24], this can be reduced to the following form:

$$\begin{aligned} \widehat{W} &= \arg\min_W \sum_{v\in V} \sum_{i=1}^{D} p_e(v) g^2(u_i) \\ &\approx \arg\min_W \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{D} g^2(u_{in}), \end{aligned} \tag{2.20}$$

where $u_i = \frac{p_W(v)}{p_W(\neg^i v)}$, and $g(u) = \frac{1}{1+u}$.

For a binary RBM, we can express the ratio matching gradient as:

$$\nabla J_{RM}(W) = \frac{2}{N} \sum_{n=1}^{N} \sum_{i=1}^{D} g^3(u_{in}) u_{in} \left(\nabla F(v_n, W) - \nabla F(\neg^i v_n, W)\right). \tag{2.21}$$

In this form, we can see the similarity between ratio matching and the form of pseudo-likelihood given in Equation 2.16. In particular, the gradients Equation 2.17 and Equation 2.21 are almost identical with the exception that they weight each term differently.

To get a better understanding of how these two methods differ, Figure 2.4 shows the link functions from pseudo-likelihood $\left(g(u) = -log\left(\frac{u}{1+u}\right)\right)$ and ratio

matching $\left(g(u) = \left(\frac{1}{1+u}\right)^2\right)$ as well as their derivatives, in terms of the ratio u.

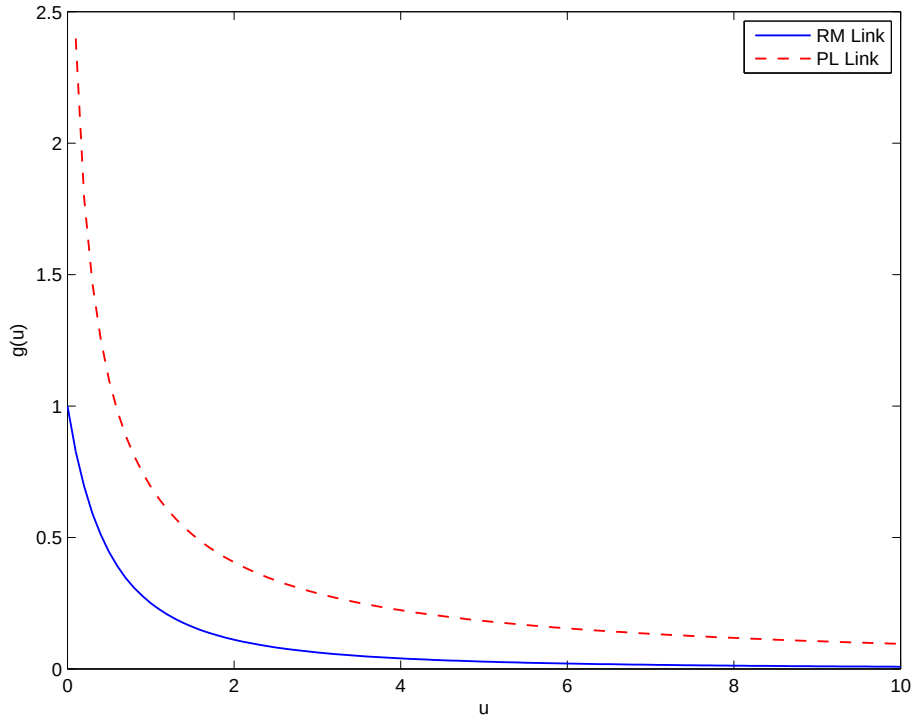
As $\frac{p_W(v)}{p_W(\neg^i v)}$ increases, the model does a better job of fitting the data and so the penalty decreases to 0. The major difference between the two methods occurs when the model does not fit the data well. As the ratio gets smaller, pseudo-likelihood will try exponentially harder to fit the model correctly, while ratio matching has a limit to how hard it will try.

Another way to see this is to express Equation 2.20 in the following form as shown in [39]:

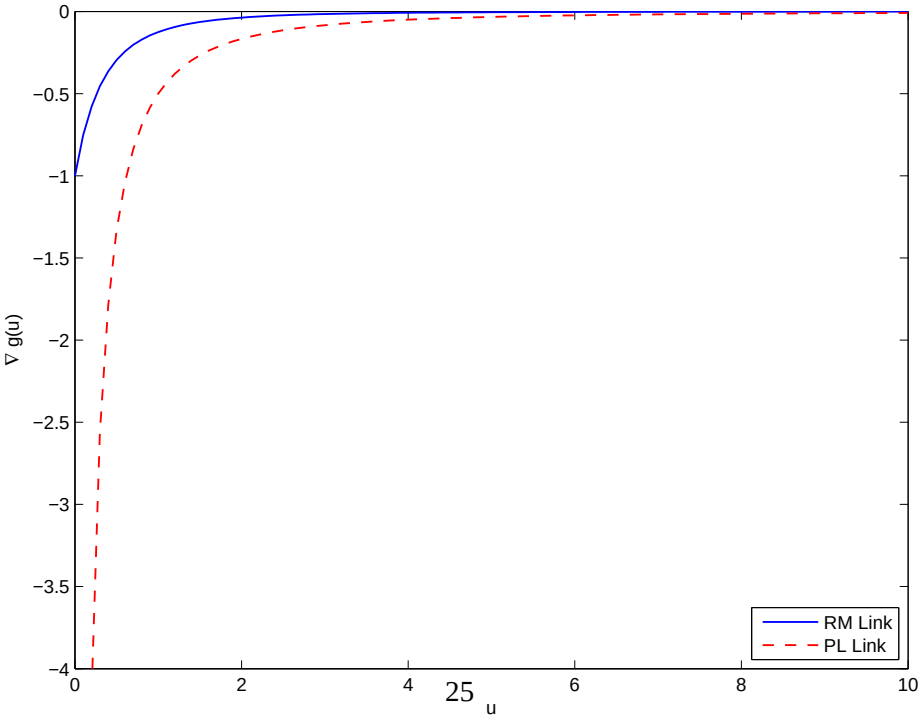
$$\begin{aligned}\widehat{W} &\approx \arg\min_W \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D}\left(\frac{p_W(\neg^i v_n)}{p_W(\neg^i v_n) + p_W(v_n)}\right)^2 \\ &= \arg\min_W \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D}(1 - p_W(v_{in}|v_{-in}))^2. \end{aligned} \tag{2.22}$$

Clearly both pseudo-likelihood and ratio matching try to maximize the probability of the conditional data distributions. Pseudo-likelihood does so by weighting each contribution in terms of the log-probability, while ratio matching weights each contribution by its euclidean distance to 1. From this it is easy to see that in pseudo-likelihood, if the conditional probability of a data point approaches 0 under the model distribution, then it will give an exponentially larger contribution to the objective, while in ratio matching the contribution of each term is bounded by a magnitude of 1.

There are some conjectures that can be made from this. First, it is clear that ratio matching may be more stable, at least in the initial phase of learning, since the magnitude of its gradient is bounded. Second, it may be that ratio matching is more robust to outliers since it will not try exceptionally hard to model every data point. This could be useful for modelling noisy or mislabelled data for instance. It is difficult to say for certain how the learned models will differ in practice, but this analysis does show that these two methods should produce different results with a finite amount of data.



(a) Link Functions



(b) Link Function Derivatives

Figure 2.4: Comparison of the link functions $g(u)$ between ratio matching and pseudo-likelihood.

2.4.3 Score Matching

So far we have restricted ourselves to estimation methods for binary RBMs. The algorithms CD and SML do not change very much when they are applied to the Gaussian-binary RBM. Ratio matching, on the other hand, strictly applies to the binary case and cannot be used when the visible units are continuous. We could use the pseudo-likelihood objective given in Equation 2.12, replacing the sum over the i^{th} variable with an integral. Unfortunately, this does not admit a closed form solution and so we would need to resort to numerical integration. Although this is certainly possible, it would require computing the free energy multiple times per dimension and could therefore become quite expensive in practice. An alternative estimator called score matching (SM) was proposed in [22] which overcomes these limitations by selecting parameters that minimize the difference between the so-called score functions $\psi_i(\cdot)$ of the data and model distributions:

$$\widehat{W}_{SM} = \arg\min_W \int_{v \in V} p_e(v) \sum_{i=1}^{D} \left(\psi_i(p_e(v)) - \psi_i(p_W(v))\right)^2 dv, \tag{2.23}$$

where $\psi_i(p(v)) \doteq \frac{\partial log(p(v))}{\partial v_i}$.

For an RBM, $\psi_i(p_W(v)) = \frac{\partial -F(v,W) - log(Z(W))}{\partial v_i} = \frac{\partial -F(v,W)}{\partial v_i}$ since $Z(W)$ does not depend on v. This means that computing the score matching objective does not depend on the partition function.

We can expand the score matching objective in a similar fashion to Equation 2.18:

$$\begin{aligned}
\widehat{W}_{SM} &= \arg\min_W \int_{v \in V} p_e(v) \sum_{i=1}^{D} \psi_i(p_e(v))^2 - 2\psi_i(p_e(v))\psi_i(p_W(v)) + \psi_i(p_W(v))^2 dv \\
&= \arg\min_W \int_{v \in V} p_e(v) \sum_{i=1}^{D} -2\psi_i(p_e(v))\psi_i(p_W(v)) + \psi_i(p_W(v))^2 dv.
\end{aligned} \tag{2.24}$$

Here we run into the same problem as before, where computing the objective would require knowledge of $p_e(v)$. In [22], however, it was shown that for distributions whose density decays to 0 quickly enough as any v_i approaches ∞, we can equivalently express the score matching objective as:

$$\begin{aligned}\widehat{W}_{SM} &= \arg\min_{W} \int_{v\in V} p_e(v) \sum_{i=1}^{D} \frac{1}{2}\psi_i(p_W(v))^2 + \frac{\partial \psi_i(p_W(v))}{\partial v_i} dv \\ &\approx \arg\min_{W} \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{D} \frac{1}{2}\psi_i(p_W(v_n))^2 + \frac{\partial \psi_i(p_W(v_n))}{\partial v_i}. \end{aligned} \tag{2.25}$$

Since Equation 2.25 does not depend on knowing $p_e(v)$, score matching provides a deterministic and consistent estimator for the GBRBM [22]. Indeed, we can write down the score matching objective and gradients for this model directly:

$$J_{SM}(W,c,b,\sigma) = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D}\left(\frac{1}{2}\left(-\frac{v_{in}}{\sigma_i^2}+\frac{c_i}{\sigma_i^2}+\sum_{j=1}^{K}\frac{W_{ij}}{\sigma_i}E[h_k|v_n,W,c,b,\sigma]\right)^2 - \frac{1}{\sigma_i^2}+\sum_{j=1}^{K}\frac{W_{ij}^2}{\sigma_i^2}E[h_j|v_n,W,c,b,\sigma]\,(1-E[h_j|v_n,W,c,b,\sigma])\right), \tag{2.26}$$

$$\begin{aligned}\nabla_{W_{dk}}J_{SM}(W,c,b,\sigma) = \frac{1}{N}\sum_{n=1}^{N}\Bigg[\Bigg[\sum_{i=1}^{D}\left(-\frac{v_{in}}{\sigma_i^2}+\frac{c_i}{\sigma_i^2}+\sum_{j=1}^{K}\frac{W_{ij}}{\sigma_i}E[h_j|v_n,W,c,b,\sigma]\right)\\ \times\left(\frac{v_{dn}W_{ik}}{\sigma_d\sigma_i}E[h_k|v_n,W,c,b,\sigma](1-E[h_k|v_n,W,c,b,\sigma])\right)\\ +\frac{v_{dn}W_{ik}^2}{\sigma_d\sigma_i^2}E[h_k|v_n,W,c,b,\sigma](1-E[h_k|v_n,W,c,b,\sigma])(1-2E[h_k|v_n,W,c,b,\sigma])\Bigg]\\ +\left(-\frac{v_{dn}}{\sigma_d^2}+\frac{c_d}{\sigma_d^2}+\sum_{j=1}^{K}\frac{W_{dj}}{\sigma_d}E[h_j|v_n,W,c,b,\sigma]\right)\frac{1}{\sigma_d}E[h_k|v_n,W,c,b,\sigma]\\ +\left(\frac{2W_{dk}}{\sigma_d^2}E[h_k|v_n,W,c,b,\sigma](1-E[h_k|v_n,W,c,b,\sigma])\right)\Bigg],\end{aligned} \tag{2.27}$$

$$\nabla_{c_d}J_{SM}(W,c,b,\sigma) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{\sigma_d^2}\left(-\frac{v_{dn}}{\sigma_d^2}+\frac{c_d}{\sigma_d^2}+\sum_{j=1}^{K}\frac{W_{dj}}{\sigma_d}E[h_j|v_n,W,c,b,\sigma]\right), \tag{2.28}$$

$$\begin{aligned}\nabla_{b_k}J_{SM}(W,c,b,\sigma) = \frac{1}{N}\sum_{n=1}^{N}\Bigg[\sum_{i=1}^{D}\left(-\frac{v_{in}}{\sigma_i^2}+\frac{c_i}{\sigma_i^2}+\sum_{j=1}^{K}\frac{W_{ij}}{\sigma_i}E[h_j|v_n,W,c,b,\sigma]\right)\\ \times\left(\frac{W_{ik}}{\sigma_i}E[h_k|v_n,W,c,b,\sigma](1-E[h_k|v_n,W,c,b,\sigma])\right)\\ +\frac{W_{ik}^2}{\sigma_i^2}E[h_k|v_n,W,c,b,\sigma](1-E[h_k|v_n,W,c,b,\sigma])(1-2E[h_k|v_n,W,c,b,\sigma])\Bigg],\end{aligned} \tag{2.29}$$

$$\nabla_{\sigma_d} J_{SM}(W,c,b,\sigma) = -\frac{1}{N}\sum_{n=1}^{N}\Bigg[\Bigg[\sum_{i=1}^{D}\left(-\frac{v_{in}}{\sigma_i^2}+\frac{c_i}{\sigma_i^2}+\sum_{j=1}^{K}\frac{W_{ij}}{\sigma_i}E[h_j|v_n,W,c,b,\sigma]\right)$$
$$\times\left(\sum_{j=1}^{K}\frac{v_{dn}W_{dj}W_{ij}}{\sigma_d^2\sigma_i}E[h_j|v_n,W,c,b,\sigma](1-E[h_j|v_n,W,c,b,\sigma])\right)$$
$$+\left(\sum_{j=1}^{K}\frac{v_{dn}W_{dj}W_{ij}^2}{\sigma_d^2\sigma_i^2}E[h_j|v_n,W,c,b,\sigma](1-E[h_j|v_n,W,c,b,\sigma])(1-2E[h_j|v_n,W,c,b,\sigma])\right)\Bigg]$$
$$+\left(-\frac{v_{dn}}{\sigma_d^2}+\frac{c_d}{\sigma_d^2}+\sum_{j=1}^{K}\frac{W_{dj}}{\sigma_d}E[h_j|v_n,W,c,b,\sigma]\right)$$
$$\times\left(-\frac{2v_{dn}}{\sigma_d^3}+\frac{2c_d}{\sigma_d^3}+\sum_{j=1}^{K}\frac{W_{dj}}{\sigma_d^2}E[h_j|v_n,W,c,b,\sigma]\right)$$
$$+\left(\sum_{j=1}^{K}\frac{2W_{dj}^2}{\sigma_d^3}E[h_j|v_n,W,c,b,\sigma](1-E[h_j|v_n,W,c,b,\sigma])\right)+\frac{2}{\sigma_d^3}\Bigg]. \tag{2.30}$$

In [23], contrastive divergence using Langevin Monte Carlo sampling [47] starting from the data can be viewed as optimizing the score matching objective with some added noise. Additionally, both are seen as approximations to pseudo-likelihood obtained by approximating the conditional expectation in Equation 2.15 by a single step of Langevin Monte Carlo. In the score matching case, this step would be infentesimally small.

Finally, it is worth noting that in [39] it was determined that unlike maximum likelihood which seeks a minimum of the KL divergence, score matching seeks parameters that lead to the least change in KL divergence when a small amount of noise is added to the training data. This would imply that score matching and by extension, contrastive divergence may be more robust in the case of noisy data.

2.4.4 Generalized Score Matching

Lyu [39] cast score matching as a specific application of the generalized Fisher divergence:

$$D_L(p_e||p_W) = \sum_{v\in V} p_e(v)\left\|\frac{L(p_e(v))}{p_e(v)}-\frac{L(p_W(v))}{p_W(v)}\right\|^2,$$

where L is a linear operator such that for all $v \in V$, $\frac{L(p_e(v))}{p_e(v)} = \frac{L(p_W(v))}{p_W(v)} \Rightarrow p_e(v) = p_W(v)$. In this case, L is said to be complete.

Using this interpretation, Lyu creates an estimator for discrete data by choosing the marginalization operation $\mathscr{M}_i(p(v)) = \sum_{\xi_i} p(\xi_i, v_{-i})$ so that the Fisher divergence becomes:

$$\begin{aligned} D_{\mathscr{M}}(p_e||p_W) &= \sum_{v\in V} p_e(v) \sum_{i=1}^{D} \left(\frac{\mathscr{M}_i(p_e(v))}{p_e(v)} - \frac{\mathscr{M}_i(p_W(v))}{p_W(v)} \right)^2 \\ &= \sum_{v\in V} p_e(v) \sum_{i=1}^{D} \left(\frac{1}{p_e(v_i|v_{-i})} - \frac{1}{p_W(v_i|v_{-i})} \right)^2, \end{aligned} \tag{2.31}$$

since $\frac{\mathscr{M}_i(p(v))}{p(v)} = \frac{\sum_{\xi_i} p(\xi_i, v_{-i})}{p(v)}$.

This estimator is called generalized score matching (GSM). We can proceed to write the GSM objective for the binary RBM as:

$$\begin{aligned} J_{GSM}(W) &= \sum_{v\in V} p_e(v) \sum_{i=1}^{D} \left(\frac{1}{p_e(v_i|v_{-i})} \right)^2 - 2\left(\frac{1}{p_e(v_i|v_{-i})} \frac{1}{p_W(v_i|v_{-i})} \right) + \left(\frac{1}{p_W(v_i|v_{-i})} \right)^2 \\ &= const + \sum_{v\in V} p_e(v) \sum_{i=1}^{D} \left(\frac{1}{p_W(v_i|v_{-i})} \right)^2 - 2\left(\frac{1}{p_e(v_i|v_{-i})} \frac{1}{p_W(v_i|v_{-i})} \right). \end{aligned} \tag{2.32}$$

We can further express the GSM objective in terms of a dataset sampled from the empirical distribution using the ratio of probabilities u:

$$J_{GSM}(W) = const + \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{D} g(u_{in}), \tag{2.33}$$

where $g(u) = u^{-2} - 2u$ and $u_i = \frac{p(v,W)}{p(\bar{v}^i,W)}$.

Note that in [39] there is a mistake in the derivation which incorrectly gives $g(u) = u^{-2} + u^2$. Equation 2.33 gives the corrected result.

Interestingly, Equation 2.33 is unbounded below as $u \to \infty$ since it is dominated by the linear term $-2u$. To see why this arises, note that Lyu begins by dropping

the constant term $\frac{p_e(v)}{p_e(v_i|v_{-i})^2} = p_e(v)\left(\frac{p_e(v)+p_e(\overset{\neg i}{v})}{p_e(v)}\right)^2$. Consider two visible configurations v and $\overset{\neg i}{v}$ which are equal everywhere except for dimension i. Now let $p_e(v) = \varepsilon$ and $p_e(\overset{\neg i}{v}) = \zeta$. In the limit where $\varepsilon \to 0$ and ζ is nonzero, the constant constribution $\frac{(\varepsilon+\zeta)^2}{\varepsilon}$ approaches infinity. Unfortunately, this makes it impractical to apply GSM to real datasets where many configurations v will have $p_e(v) = 0$. We cannot simply ignore these configurations either, since the GSM objective puts most of its emphasis on getting the corresponding model conditional distributions to equal exactly 0. We attempted an implementation anyway, and found that the objective value would effectively always diverge to $-\infty$ on real datasets.

2.4.5 Alternatives

In this chapter we detailed some of the straightforward and popular methods for learning Restricted Boltzmann Machines, and similar models. While we have mentioned several, this is by no means an exhaustive list of possible algorithms. In particular, variational methods [61], Bayesian inference [44], and maximum-margin learning [43] are other possible approaches one could take to solving the RBM learning problem. Most of these methods, however, come with their own set of challenges. The methods presented in this chapter have the advantage that their motivation, theory, and execution are all relatively simple and therefore it is worth understanding their behavior before considering more complicated schemes. We should give mention to a closely related method called minimum probability flow learning [68]. This can be seen as another generalization of score matching, and can be applied to both discrete and continuous data. While we do not consider it here, it would certainly be interesting to see how it compares to the methods we have already discussed.

Chapter 3

Connections Between Auto-Encoders and Gaussian Restricted Boltzmann Machines

In the previous chapters we focused on learning a probabilistic model whose corresponding distribution matches the distribution of a given dataset. While this is an effective approach, it is certainly not the only approach to modelling data. In particular, we may wish to learn models which are specifically designed for certain tasks such as classification, regression, or denoising. In these cases learning the full joint distribution over the data may not be the best way to achieve optimal results. For this reason, it is worth exploring alternatives to generative models.

In this chapter we focus on feed-forward neural networks, or multilayer perceptrons (MLPs) which are a general class of models designed to maximize the expected performance (or equivalently, minimize the espected error) on a particular task. The RBM can be considered an undirected model, in the sense that it attempts to learn correlations between its units. In contrast, MLPs are considered directed in that they attempt to predict some output given an input.

There are many links between MLPs and RBMs, for example training a specific type of MLP called an auto-encoder can be viewed as a biased approximation to contrastive divergence [3]. Another link of practical interest is that the weights of an RBM are often used as an initialization for MLP training [19]. In this sense the

MLP training is considered to be a fine-tuning stage for the RBM which can drastically improve its performance on a specific task. We will show that applying the score matching criterion to RBMs yields yet another close link with auto-encoders, and this will help to further elucidate the deep connections between these two models.

Finally, we will describe a recently proposed algorithm used to train MLPs involving training and stacking RBMs. The application of this algorithm is one of the main uses of the RBM in machine learning, and it has spawned the area of deep learning which involves learning heirarchies of features automatically from data.

3.1 Learning to Predict

Let us consider a scenario where we are given the pair (y,v) where $v \in V$ with dimension D represents an input, and $y \in \mathscr{Y}$ with dimension C represents a target. Our goal will be to predict y from v. Suppose we have a model that generates a response $\widehat{a_c},\ c = \{1,...,C\}$ from v. For example, given parameters $\Theta = \{W\}$ a simple response is the linear response:

$$\widehat{a_c} = \sum_{i=1}^{D} W_{ic} v_i. \tag{3.1}$$

Then we can use $\widehat{a}$ to form a prediction $\widehat{y}$. In general, suppose our model has a set of parameters Θ; then we can fit a distribution $p(y|v,\Theta)$ and choose as our prediction $\widehat{y} = E[y|v,\Theta]$. While there are certainly other choices, note that fitting this distribution effectively means performing maximum likelihood, and this approach will tie nicely with MLPs.

Our choice of model $p(y|v,\Theta)$ will depend on the kind of task we wish to perform. If our task is regression and $y \in \mathscr{R}$, then we can choose the distribution to be a Gaussian so that the negative log probability can be written as:

$$
\begin{aligned}
-log(p(y|v,\Theta)) &= const - log\left(exp\left(-\frac{1}{2}\left\|y-\widehat{a}\right\|^2\right)\right) \\
&= const + \frac{1}{2}\left\|y-\widehat{a}\right\|^2 \\
&= const + \frac{1}{2}\sum_{c=1}^{C}(y_c-\widehat{a_c})^2 \qquad (3.2) \\
&= const + \frac{1}{2}\sum_{c=1}^{C}(y_c-\widehat{y_c})^2. \qquad (3.3)
\end{aligned}
$$

Since $p(y|v,\Theta)$ is a Gaussian, we see that our prediction $\widehat{y}=\widehat{a}$ since $E[y|v,\Theta]$ under a Gaussian is simply the mean. When used for regression, this model is known as least squares. If we wish to perform classification, then we can choose instead:

$$
-log(p(y_c=1,y_{-c}=0|v,\Theta)) = -log\left(\widehat{a_c}\right). \qquad (3.4)
$$

To ensure that this forms a valid distribution, we need to add the constraint that $\sum_{c=1}^{C}\widehat{a_c}=1$. In this case, the prediction is again simply $\widehat{y}=\widehat{a}$. This is effectively a measure of confidence that the true class is c given that the input is v.

An alternative to constraining the model is to redefine it so that it automatically satisfies the constraint. One way to do this is to rewrite Equation 3.4 as:

$$
\begin{aligned}
-log(p(y_c=1,y_{-c}=0|v,\Theta)) &= -log\left(\frac{exp(\widehat{a_c})}{\sum_{c'=1}^{C}exp(\widehat{a_{c'}})}\right) \\
&= -log\left(\widehat{y_c}\right). \qquad (3.5)
\end{aligned}
$$

In this form, Equation 3.5 is known better as logistic regression, and the prediction now becomes $\widehat{y_c}=\frac{exp(\widehat{a_c})}{\sum_{c'=1}^{C}exp(\widehat{a_{c'}})}$.

Whatever distribution we choose, for a specific data distribution the goal of learning will then be to minimize the expected loss:

$$\begin{aligned}\widehat{\Theta} &= \arg\min_{\Theta} -E_{p_e(v,y)}[log\,(p(y|v,\Theta))] \\ &= \arg\min_{\Theta} -\sum_{v\in V, y\in\mathscr{Y}} p_e(v,y) log\,(p(y|v,\Theta)) \\ &\approx \arg\min_{\Theta} -\frac{1}{N}\sum_{n=1}^{N} log\,(p(y_n|v_n,\Theta))\,. \end{aligned} \tag{3.6}$$

If the predictions made by the model are deterministic and $log\,(p(y|v,\Theta))$ is smooth in Θ, then we can use gradient descent techniques to minimize Equation 3.6.

3.2 Multilayer Perceptrons

While the models given in Section 3.1 are widely used, they are limited in their ability to capture complicated distributions due to their linear nature. For example, one cannot perfectly classify data that is not linearly separable using ordinary logistic regression.

One solution that overcomes this is to make the function more flexible by introducing a nonlinear hidden layer h composed of hidden units $\widehat{h_j}$ where $j \in \{1,...,K\}$. These hidden units will then be the used to predict the output. Unlike the RBM in which the hidden units are random variables, in an MLP h_j is always computed deterministically according to the sigmoid function:

$$\widehat{h_j} = sigm\left(\sum_{i=1}^{D} W_{ij}^{(\ell_1)} v_i\right),$$

so that the final response becomes:

$$\widehat{a_c} = \sum_{j=1}^{K} W_{cj}^{(\ell_2)} \widehat{h_j}. \tag{3.7}$$

Here $W^{(\ell_1)}$ and $W^{(\ell_2)}$ refer to the parameters of each layer.

We did not necessarily have to use the sigmoid function, another popular vari-

ant is the tanh function. For our purposes, however, it will suffice to simply define MLPs in terms of the sigmoid nonlinearity.

Notice that if $W^{(\ell_1)}$ was learned according to an RBM, then this would be exactly equivalent to using a linear response on top of the RBM hidden features $E[h|v, W^{(\ell_1)}]$.

Of course, we don't have to stop at one hidden layer. We can use as many as we like, and each one will give the model more flexibility in terms of the kinds of predictions it can make. Let $\widehat{h}^{(\ell_m)}$, $m = \{1, ..., M\}$ denote the m^{th} hidden layer of size K_m, and $\Theta = \left\{W^{(\ell_1)}, ..., W^{(\ell_{M+1})}\right\}$ represent the corresponding parameters for each layer:

$$\begin{aligned}
\widehat{a_c} &= \sum_{j=1}^{K_M} W_{cj}^{(\ell_{M+1})} \widehat{h_j}^{(\ell_M)} \\
\widehat{h}^{(\ell_M)} &= E[h^{(\ell_M)} | \widehat{h}^{(\ell_{M-1})}, W^{(\ell_M)}] \\
\widehat{h}^{(\ell_{M-1})} &= E[h^{(\ell_{M-1})} | \widehat{h}^{(\ell_{M-2})}, W^{(\ell_{M-1})}] \\
&\vdots \\
\widehat{h}^{(\ell_1)} &= E[h^{(\ell_1)} | v, W^{(\ell_1)}].
\end{aligned} \tag{3.8}$$

Note that the expectations in the MLP equation refer to the application of the sigmoid nonlinearity. Using $\widehat{a}$, we can then compute $\widehat{y}$; a graphical representation of the MLP is shown in Figure 3.1

3.3 Auto-Encoders

One of the main uses of the MLP is that it can learn latent features which are useful for performing various tasks. Thus far we have described the MLP as accepting an input v and predicting a target $\widehat{y}$, which requires that we be given both v and the true target y at training time. Unfortunately, it is often difficult to obtain such pairs as this usually involves labelling examples by hand. Additionally, labellers may differ in their opinions, assigning different labels to the same data point, which injects unwanted noise into the learning process. Unlabelled data, on the other hand,

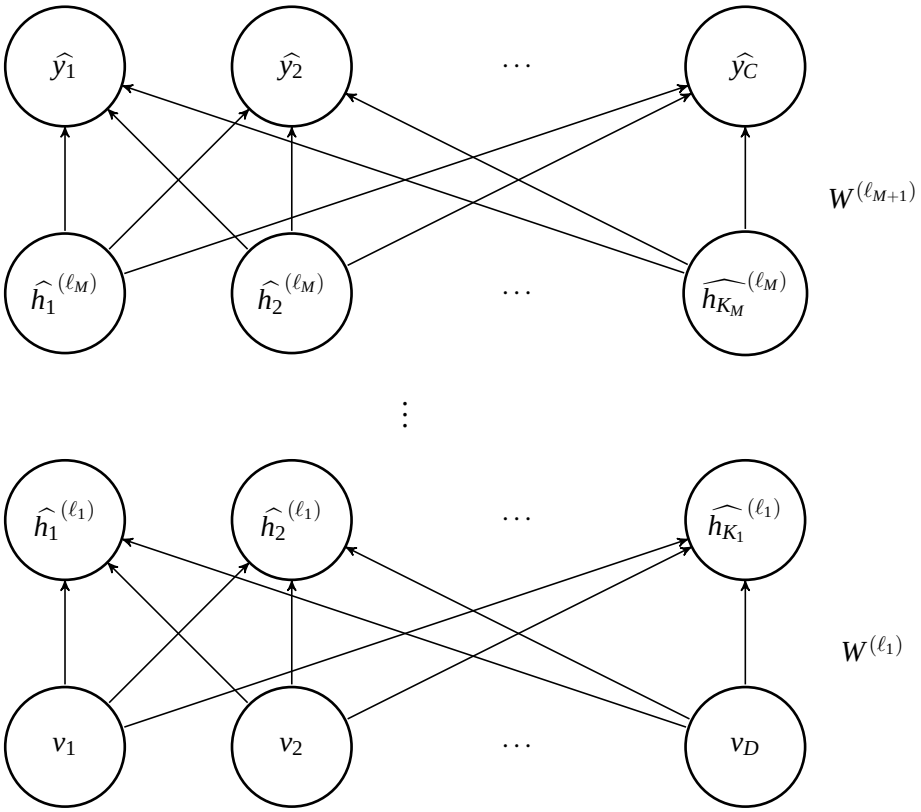


Figure 3.1: A multilayer perceptron with M hidden layers, inputs v, and outputs $\widehat{y}$.

is widely available and relatively easy to acquire. In addition, features learned from unlabelled data can often be applied to a wide variety of tasks, rather than being suited for something specific. We can easily adapt the MLP to learn from unlabelled data by setting the targets y to be the data points themselves. In this sense, we are taking data and trying to reconstruct v from its latent representation. An MLP which is trained for reconstruction is more commonly known as an auto-encoder.

Auto-encoders are most generally used for dimensionality reduction. If there is some form of information bottleneck, such as a hidden layer having fewer dimensions than the data ($K_m < D$ for some hidden layer m), then the autoencoder will be forced to learn common structure in order to minimize the amount of lost

information. We should note that if an auto-encoder has linear hidden units (i.e. it does not pass the weighted input through a sigmoid), then training it is equivalent to performing principal component analysis (PCA).

If the data is real-valued, then we can simply apply Equation 3.3 as our loss function for Equation 3.6. If the data is binary, then we can easily derive the associated distribution (note that we could have also derived Equation 3.5 in a similar manner):

$$
\begin{aligned}
-log(p(y|v,W)) &= -log\left(\frac{exp\left(\sum_{i=1}^{D} y_i \widehat{a_i}\right)}{\sum_{y' \in V} exp\left(\sum_{i'=1}^{D} y'_{i'} \widehat{a_{i'}}\right)}\right) \\
&= -log\left(\frac{exp\left(\sum_{i=1}^{D} y_i \widehat{a_i}\right)}{\sum_{y' \in V} \prod_{i'=1}^{D} exp\left(y'_{i'} \widehat{a_{i'}}\right)}\right) \\
&= -log\left(\frac{exp\left(\sum_{i=1}^{D} y_i \widehat{a_i}\right)}{\prod_{i'=1}^{D} 1 + exp\left(\widehat{a_{i'}}\right)}\right) \\
&= -log\left(\frac{\prod_{i=1}^{D} exp\left(y_i \widehat{a_i}\right)}{\prod_{i'=1}^{D} 1 + exp\left(\widehat{a_{i'}}\right)}\right) \\
&= -log\left(\prod_{i=1}^{D} \frac{exp\left(y_i \widehat{a_i}\right)}{1 + exp\left(\widehat{a_i}\right)}\right) \\
&= -log\left(\prod_{i=1}^{D} \left(\frac{exp\left(\widehat{a_i}\right)}{1 + exp\left(\widehat{a_i}\right)}\right)^{y_i} \left(\frac{1}{1 + exp\left(\widehat{a_i}\right)}\right)^{1-y_i}\right) \\
&= -\sum_{i=1}^{D} \left(y_i log\left(sigm(\widehat{a_i})\right) + (1 - y_i) log\left(1 - sigm(\widehat{a_i})\right)\right) \\
&= -\sum_{i=1}^{D} \left(v_i log\left(\widehat{v_i}\right) + (1 - v_i) log\left(1 - \widehat{v_i}\right)\right). \qquad (3.9)
\end{aligned}
$$

Equation 3.9 is commonly referred to as the cross-entropy loss.

The major reason for introducing nonlinearities into the MLP is to avoid linear solutions like PCA which are limited in their capacity to learn complex patterns. Interestingly, it has been shown that when we use the loss function Equation 3.3 with one hidden layer, then the optimal solution is still equivalent to the PCA solution despite the introduction of a nonlinear hidden layer [9]. In fact, one could add

as many hidden layers as they wished and if for all of the layers $K_m = K_p,\ m \neq p$, then the solution remains equivalent to PCA.

Luckily, this result doesn't apply if we use Equation 3.9, nor does it apply if we change the number of hidden units between layers. This means that as long as at least one hidden layer has a different number of units from the rest, the solution will be nonlinear. Additionally, it was shown in [25] that if the total input to a hidden unit falls outside of the linear range of the signoid function, then the PCA solution is also avoided. Another way which hasn't yet been explored in detail is to use the so-called stochastic reconstruction error (SRE) [3, 38], shown here with one hidden layer:

$$\begin{aligned} SRE(v, y = v) &= \sum_{h \in H} log\left(p(y|h, W^{(\ell_2)})\right) p(h|v, W^{(\ell_1)}) \\ &= \sum_{h \in H} log\left(p(v|h, W^{(\ell_2)})\right) p(h|v, W^{(\ell_1)}). \end{aligned} \tag{3.10}$$

Instead of using a deterministic reconstruction we define a distribution over possible reconstructions through all possible binary codes h and take the expected reconstruction error under the distribution $p(h|v, W^{(\ell_1)})$ to be our training criterion. We will show in Section 3.4 that there is a close relationship between the stochastic reconstruction error and the loss function given in Equation 3.3 when $p(v|h, W^{(\ell_2)})$ is Gaussian.

3.4 A Closer Look at Score Matching for the Gaussian-Binary RBM

Recall the objective that resulted from applying the score matching criterion to the GBRBM, given by Equation 2.26. For simplicity, we will set $\sigma_i = 1$ for every dimension, and absorb the bias terms into W:

$$J_{SM}(W) = const + \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D}\left(\frac{1}{2}\left(-v_{in} + \sum_{j=1}^{K} W_{ij}E[h_j|v_n,W]\right)^2 + \sum_{j=1}^{K} W_{ij}^2 E[h_j|v_n,W]\left(1-E[h_j|v_n,W]\right)\right)$$
$$= const + \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D}\left(\frac{1}{2}\left(\widehat{a_{in}} - v_{in}\right)^2 + \sum_{j=1}^{K} W_{ij}^2 E[h_j|v_n,W]\left(1-E[h_j|v_n,W]\right)\right). \tag{3.11}$$

We can immediately see that the first term $\frac{1}{2}\left(\widehat{v_{in}} - v_{in}\right)^2$ is the reconstruction term of an auto-encoder with predictions $\widehat{v_{in}} = \sum_{j=1}^{K} W_{ij}E[h_j|v_n,W]$ and quadratic loss. The second term, however, is a bit more curious. On close inspection, we can see that it is actually the variance of the reconstruction, represented by $var_{p(h|v,W)}\left(\sum_{j=1}^{K} W_{dj}h_j\right)$ since each h_j is a Bernoulli random variable. In fact, it is fairly straightforward to generalize the score matching objective to any distribution over the hidden units:

$$J_{SM}(W) = const + \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D}\frac{1}{2}(\psi_i(p(v_n|W))^2 + \frac{\partial \psi_d(p(v_n|W))}{\partial v_i}$$
$$= const + \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{D}\frac{1}{2}\left(-v_{in} + \sum_{h\in H}\left(\sum_{j=1}^{K} W_{ij}h_j\right)p(h|v_n,W)\right)^2 + var_{p(h|v_n,W)}\left(\sum_{j=1}^{K} W_{ij}h_j\right). \tag{3.12}$$

We see from Equation 3.11 and Equation 3.12 that training a GBRBM using score matching is equivalent to training a regularized auto-encoder network with a linear output and quadratic loss. In this case, the regularizer penalizes the variance of the reconstruction under the distribution induced by $p(h|v,W)$. On closer inspection, one can see that this penalty actually encourages binary codes, meaning that it will try to force the total input to the hidden units to fall outside of the linear range of the sigmoid nonlinearity. Based on the results in [25], this means that it encourages solutions to be different from those given by PCA. Amazingly, by the consistency property of score matching [22], training this regularized auto-encoder is an asymptotically consistent way to train a GBRBM. This analysis also reveals

another similarity to contrastive divergence, which itself tries to minimize a form of reconstruction error.

Now consider a single data instance v. We will take Equation 3.12 and drop the $\frac{1}{2}$ in the first term so that it becomes $\sum_{i=1}^{D}(\psi_i(p(v|W))^2$. This has the effect of paying more attention to the reconstruction, and less to the variance reduction. The coefficients on the reconstruction term and regularization term will now both be 1 and we will get:

$$
\begin{aligned}
&\sum_{i=1}^{D}\left(-v_i+\sum_{h\in H}\left(\sum_{j=1}^{K}W_{ij}h_j\right)p(h|v,W)\right)^2+var_{p(h|v,W)}\left(\sum_{j=1}^{K}W_{ij}h_j\right)\\
&=\sum_{i=1}^{D}v_i^2-2v_i\sum_{h\in H}\left(\sum_{j=1}^{K}W_{ij}h_j\right)p(h|v,W)+\left(\sum_{h\in H}\left(\sum_{j=1}^{K}W_{ij}h_j\right)p(h|v,W)\right)^2\\
&+\sum_{h\in H}\left(\sum_{j=1}^{K}W_{ij}h_j\right)^2p(h|v,W)-\left(\sum_{h\in H}\left(\sum_{j=1}^{K}W_{ij}h_j\right)p(h|v,W)\right)^2\\
&=\sum_{i=1}^{D}v_i^2-2v_i\sum_{h\in H}\left(\sum_{j=1}^{K}W_{ij}h_j\right)p(h|v,W)+\sum_{h\in H}\left(\sum_{j=1}^{K}W_{ij}h_j\right)^2p(h|v,W)\\
&=\sum_{i=1}^{D}\sum_{h\in H}v_i^2p(h|v,W)-\sum_{h\in H}2v_i\left(\sum_{j=1}^{K}W_{ij}h_j\right)p(h|v,W)+\sum_{h\in H}\left(\sum_{j=1}^{K}W_{ij}h_j\right)^2p(h|v,W)\\
&=\sum_{i=1}^{D}\sum_{h\in H}\left(v_i^2-2v_i\left(\sum_{j=1}^{K}W_{ij}h_j\right)+\left(\sum_{j=1}^{K}W_{ij}h_j\right)^2\right)p(h|v,W)\\
&=\sum_{h\in H}\sum_{i=1}^{D}\left(v_i-\sum_{j=1}^{K}W_{ij}h_j\right)^2p(h|v,W). \qquad (3.13)
\end{aligned}
$$

Notice that Equation 3.13 is exactly proportional to the stochastic reconstruction error in Equation 3.10 when the Gaussian distribution is used as $p(v|h,W)$. We can therefore see that the stochastic reconstruction error is very similar to Equation 3.12, except that it focuses more on reconstruction, or equivalently less on reducing the reconstruction variance. In this form, we also see that the SRE objective has an interesting interpretation as optimizing a bias/variance tradeoff over $p(h|v,W)$ with the reconstruction error representing the bias. We also see that it is

possible to marginalize out the hidden units for the SRE objective with Gaussian error. Unfortunately, for the cross entropy error in Equation 3.9 this is not possible. One could, however, sample h and use stochastic approximation to minimize the SRE in this case but we do not explore that here.

3.5 Deep Learning

We introduced the concept of the MLP in order to show how it is possible to go beyond linear models by adding hidden layers. These hidden layers act as features which are used as the basis for new features in subsequent hidden layers. Another way of interpreting this is that an MLP transforms the data to a new space where it is more easily represented by a linear model. The recipe then seems quite simple, just add as many hidden layers as you like until you can perfectly represent your target distribution. Unfortunately, there are a number of problems that make this more difficult than it seems.

As we mentioned in Section 3.3, it is much more difficult to come by labelled data than data without labels. If our goal is to measure the target distribution accurately, but we are not given enough data, then this problem becomes much more difficult. Consider that with a large enough MLP, we can model virtually any target distribution; see for instance [34]. This means that with very few labelled examples, we are likely to overfit the training set which would lead to poor generalization ability.

Even if we were given access to a large, labelled dataset, we may still run into a problem during optimization. As we add more hidden layers, optimization becomes much more difficult because this also introduces many suboptimal local minima. Most optimization schemes fare quite poorly in this domain, and finding a good minimum by directly minimizing the training error quickly becomes extremely difficult.

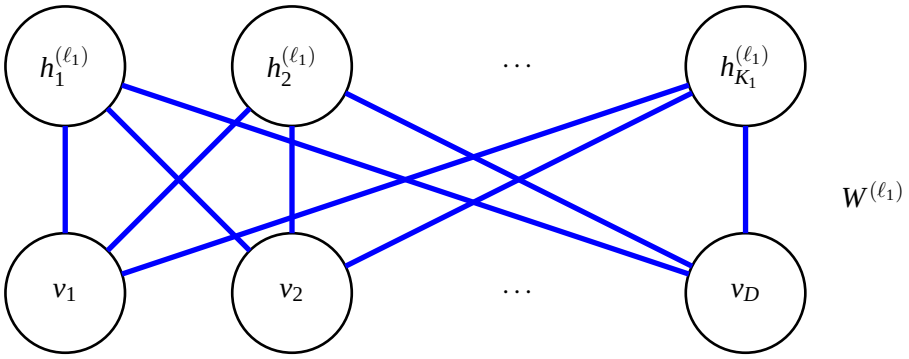
One way around this might be to perhaps use only one hidden layer, and to simply increase the number of hidden units within that layer. While this does add flexibility, there are several examples that show that deep architectures (with more than one hidden layer) are exponentially more efficient as representing a given distribution than a shallow architecture (one, or no hidden layers) [2]. That is,

a deep architecture would require exponentially fewer parameters than a shallow architecture. It is conjectured that this approach also applies to many other tasks, and deep architectures have been applied successfuly in many different domains.

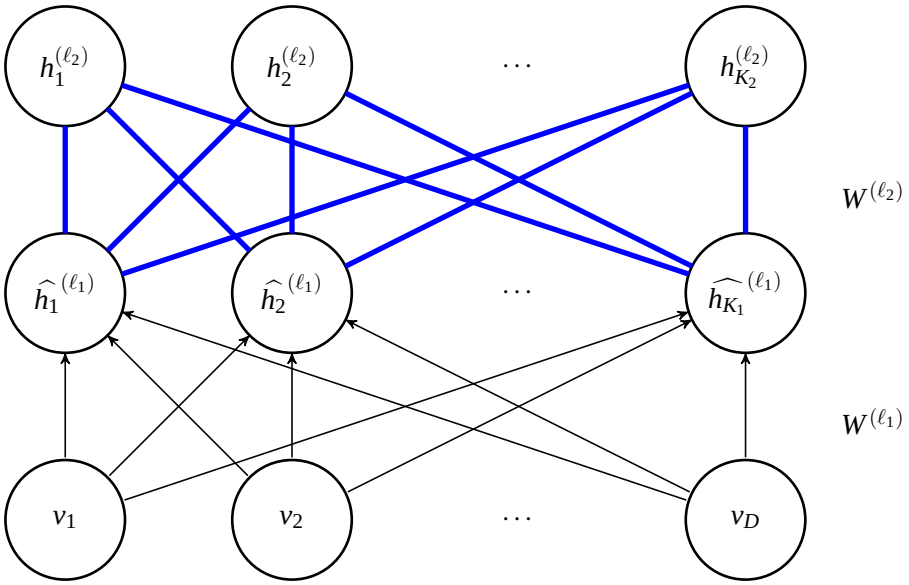
For a long time deep MLPs fell out of favour; the difficulty of training them, and their propensity to overfit made them less desireable than shallow methods like support vector machines [71] which were comparably easier to optimize, and yielded superior results. This changed in 2006 when Hinton et al. described a new way of training MLPs by training a series of RBMs. The idea is to first train an RBM with visible units v, hidden units $h^{(\ell_1)}$, and parameters $W^{(\ell_1)}$. We then take the hidden features $\widehat{h}^{(\ell_1)} = E[h^{(\ell_1)}|v, W^{(\ell_1)}]$ and use them as the visible units for another RBM with hidden units $h^{(\ell_2)}$ and parameters $W^{(\ell_2)}$. In this way, we are greedily training one RBM on top of another, and we can continue this procedure for as many layers as we like. Notice that after training an RBM, its parameters are used as if they were a layer in an MLP. A graphical illustration of this concept is given in Figure 3.2.

When RBMs are stacked in this way, the resulting architecture is called a deep belief network (DBN). Labelled data is often subsequently used to perform discriminative fine-tuning using backpropagation, and in this case a DBN and MLP have equivalent architectures, only differing in how they are trained. Using this strategy, DBNs seem to have overcome many of the problems that MLPs were associated with in the past, and they have achieved state of the art results on many different problems (an extensive, but by no means exhaustive list can be found in [2, p. 4]). The idea of greedily pre-training an MLP was further explored in [13], where surprisingly it was determined that this strategy doesn't necessarily help the optimization, but rather acts as a form of regularization. That is, we can achieve similar training error with and without pre-training, but pre-training significantly improves generalization performance.

Finally, the unsupervised nature of the greedy learning procedure means that we can benefit from learning from large amounts of unlabelled data. In particular, it has been shown that we can sometimes even utilize unlabelled data that isn't even relevant to the task at hand in order to improve generalization performance [53].



(a) Training the first layer



(b) Training the second layer

Figure 3.2: Greedy training of a deep belief network, where the weights being learned are shown in blue. Notice that in 3.2b, the weights in the first layer are frozen. The edges become directed and the first layer hidden units become deterministic, serving as the visible units for the next layer RBM. Also note that we can repeat this process to train as many layers as desired.

Chapter 4

Training Restricted Boltzmann Machines

In this chapter we will explore the many issues that arise when training Restricted Boltzmann Machines. These issues arise due to the stochastic and non-convex nature of the various training procedures outlined in Chapter 2. We will show that different choices made during training can have a large impact on how the final model will perform, and this will provide a guideline for training the models we will use in our experiments in Chapter 5 and Chapter 6.

Additionally, we will introduce a procedure from stochastic approximation called iterate averaging which can be used to accelerate the training of RBMs. We will show that this procedure outperforms the current state of the art techniques applied to the SML algorithm from Section 2.3.3. In addition, it carries the added benefit that the learning rate $\eta^{(t)}$ used at each iteration can be chosen as a constant, greatly reducing the amount of parameter tuning required to ensure good results.

4.1 Experiments Training RBMs on MNIST

In order to use RBMs effectively, it is essential that one chooses appropriate parameters. There are also many other factors involved in training RBMs that can have a significant impact on their performance. The following sections explore various strategies that can be applied during the training of RBMs which can be used to

improve classification performance.

There are many ways to assess the performance of an RBM, these include log-likelihood, train misclassification error, test misclassification error, reconstruction error, and samples generated from the model. While ideally one would choose log-likelihood as well as test error for assessing the performance of training an RBM for classification, calculating the log-likelihood is intractable for all but trivially small RBMs since the number of possible states in the model grows exponentially with the number of units. We choose test error as our performance measure since in addition to showing approximately how well the model is being trained, it also gives an indicator of how well the model is able to generalize to new data. Instructions on using RBMs for classification are given in Section 1.3.

It was shown in [66] that the time complexity to find the most likely class for a single data point is $O(D \times K \times C)$ where C is the number of possible classes. The error reported in our experiments corresponds to the proportion of properly classified points on the test set.

For each experiment, we use 500 hidden units. It is expected that more hidden units will improve classification performance, however 500 is generally used as a standard benchmark. Adding more hidden units gives the model more flexibility, however it also increases the computational cost and makes the model more difficult to optimize. Each training procedure uses mini-batches of 100 points. The idea of mini-batches is described in Section 4.1.3, and experiments justifying its usage are provided. The experiments in this section use the MNIST[1] training set and evaluate error on the provided test set since this allows us to directly ensure that our results are comparable to current published works. MNIST has the advantage of being a well studied dataset, and there are many papers providing benchmark results. This allows us to be reasonably confident that our results are indeed correct.

4.1.1 Iterate Averaging

One of the simplest ways to ensure convergence when using stochastic approximation is to follow a learning rate schedule η_t that satisfies the Robbins-Monro conditions [59], and a simple way to do this is to set $\eta_t \propto \frac{1}{t}$. Unfortunately, this can

[1] http://yann.lecun.com/exdb/mnist

often be slow in practice, and there are usually an additional one or two parameters that need to be tuned in order to govern how quickly η_t decays in a finite number of iterations. If η_t decays too quickly then the optimization will fall short of a local minimum, and if it decays too slowly then it won't converge within a reasonable amount of time. Luckily, there is a simple way to avoid this problem: averaging [51]. The idea of iterate averaging is to use a constant learning rate η, and to simply let the algorithm descend to a local minimum and then oscillate. If we take the average of these oscillations, then the resulting parameters will be closer to the true minimum than the parameters returned by the gradient descent routine. In order to do this, we will maintain a second set of parameters $\overline{W}$ representing the averaged parameters, in addition to the usual set of parameters. We can formally describe the stochastic gradient descent update using averaging as follows:

$$\begin{aligned} W_{dk}^{(t+1)} &= W_{dk}^{(t)} + \eta \nabla_{W_{dk}} log(p(v|W^{(t)})) \\ \overline{W_{dk}}^{(t+1)} &= \overline{W_{dk}}^{(t)} - \frac{1}{t}(\overline{W_{dk}}^{(t)} - W_{dk}^{(t+1)}). \end{aligned} \tag{4.1}$$

When the optimization terminates, we would return $\overline{W}$ as the final set of parameters. Amazingly, this update achieves the optimal asymptotically rate of convergence, even when the step size is chosen to be constant throughout [28, 29, 31, 52]. That is, it is equivalent to a Newton method, despite only using information from the gradient. We will use averaging frequently throughout this thesis, and we will show in Section 4.1.6 that it works quite well in training RBMs using SML.

4.1.2 Learning Rate, Momentum, and Weight Decay

To begin, we tested CD and SML over a range of parameter settings in order to determine reasonable values for our later experiments. We randomly selected 500 training instances, and 100 testing instances per class from the MNIST training set and ran each algorithm for 250 iterations using a grid search over the parameter settings. We ran experiments with several values for the learning rate parameter η: $\{0.1, 0.01, 0.001\}$. We also included momentum terms [50] to smooth the trajectory of the gradient descent rule as follows:

1. Set $t = 1$, $I_{dk}^{(1)} = 0$.

2. For each data point v_n: sample $\widetilde{h_n}^{(t)}$ from $p(h|v_n, W^{(t)})$, then sample $\widetilde{v_n}^{(t)}$ from $p(v|\widetilde{h_n}^{(t)}, W^{(t)})$.

3. Perform a CD-1 update with momentum:

$$I_{dk}^{(t+1)} = \beta I_{dk}^{(t)} - \eta^{(t)} \left[-\frac{1}{N} \sum_{n=1}^{N} v_{dn} E[h_k|v_n, W^{(t)}] + \frac{1}{N} \sum_{n=1}^{N} \widetilde{v_{dn}}^{(t)} E[h_k|\widetilde{v_n}^{(t)}, W^{(t)}] \right]$$
$$W_{dk}^{(t+1)} = W_{dk}^{(t)} + I_{dk}^{(t+1)}$$

4. Increase t to $t+1$ and iterate again.

Figure 4.1: The contrastive divergence algorithm with one-step sampling using momentum.

$$\begin{aligned} I_{dk}^{(1)} &= 0 \\ I_{dk}^{(t+1)} &= \beta I_{dk}^{(t)} + \eta^{(t)} \nabla_{W_{dk}} log(p(v|W^{(t)})) \\ W_{dk}^{(t+1)} &= W_{dk}^{(t)} + I_{dk}^{(t+1)}. \end{aligned}$$

Generally for an RBM, we found that momentum was most helpful in improving contrastive divergence, therefore we show the CD-1 algorithm in figure Figure 4.1

Momentum is a way to average the gradient of the current iteration with the gradients of previous iterations, thus smoothing the trajectory. We test β with the values $\{0, 0.3, 0.5, 0.8\}$. In addition to just using momentum, we also ran trials using iterate averaging without momentum, and iterate averaging in conjunction with momentum. This procedure is given by Equation 4.1 and we start it at iteration 150.

Finally, we experimented with weight decay, where an L2 penalty is added to the objective function to encourage smaller weights. This is mainly used as a tool

to improve test performance by preventing the model from overfitting the training set. The update with weight decay (not including momentum) is:

$$W_{dk}^{(t+1)} = W_{dk}^{(t)} + \eta^{(t)} \nabla_{W_{dk}} log(p(v|W^{(t)})) - \lambda W_{dk}^{(t)},$$

where λ regulates the strength of the weight decay term. We test with the following values for the weight decay: $\{10^{-3}, 10^{-4}, 10^{-5}\}$.

We found that CD works best with a learning rate of 0.1 and a weight decay of 10^{-3} and momentum of 0.8 while SML works best with a learning rate of 0.1, a weight decay of 10^{-4}, and no momentum. Both methods seemed to benefit from averaging.

4.1.3 Mini-Batches

Consider the scenario where instead of being given the entire training set at once, we only receive one or a handful of samples at a time, or "online". This may perhaps come from a sensor on a robot as it explores an environment. In this case one might not want to wait for all of the data to come in before starting the learning procedure. One way to facilitate learning here is to consider each example as a sample from the data distribution $p_e(v)$ and to employ stochastic approximation to ensure that the model is properly learned. One could in fact simulate this on a full training set by simply drawing points from it at random without replacement, and then repeating this once the set has been exhausted.

Now assume that we are blessed with the computational resources to compute the full maximum likelihood objective. In this case learning becomes deterministic, and we can employ the full set of tools from optimization literature to reach the optimal solution. This form of learning is also referred to as "batch optimization". From this perspective, it appears that there is no reason to consider the online case when the full objective is computable.

Surprisingly, there is considerable theoretical and empirical evidence that suggests many models can be trained substantially faster in the online scenario; see e.g. [8, 73]. The intuition behind this is that each iteration of stochastic approximation on a single point is N times faster than a gradient step on N points. While each iteration of stochastic approximation won't move exactly toward a local op-

timum, the overall trajectory will (on average), and the savings incurred in each iteration may greatly outweigh the cost of taking a few extra steps. Indeed there are some scenarios where learning can be done in just a few passes, or even a single pass through a dataset [21]. Another motivation is when there is redundancy in a dataset. Consider a taking a training set and doubling its size by replicating each instance. For a batch method, this will not provide any additional information, but it will have the effect of doubling the cost of each iteration. An online method, on the other hand, will still maintain the same per-iteration cost.

For some sets, the variance in the data might be high enough that there is too much noise for stochastic approximation to make sufficient progress. In these cases, it often helps to use more examples per iteration than just one. These collections of data points are called mini-batches, and they are created by randomly partitioning the data into non-overlapping groups. At each iteration, one group is randomly selected without replacement and when the groups have been exhausted, we repeat the cycle.

Although there is already an element of randomness when training an RBM using CD, it has been conjectured that online learning can still be quite beneficial. We test the effectiveness of this procedure using batch sizes $\{100, 1000, 10000\}$.

Figure 4.2 shows that using mini-batches provides an improvement in the convergence of the training procedure. What is more surprising is that smaller mini-batches seem to converge to a lower test error than the methods that use a higher batch size. Stochastic gradient descent is also able to escape local minima when there is enough noise and is therefore sometimes more suitable for global optimization than local, deterministic methods [7, p. 241]. For the remainder of this thesis we continue using the mini-batch procedure with batches of 100 points.

4.1.4 Binary v.s. Soft Data

While the binary RBM is designed for binary data, several tricks have been used that violate this principle in order to improve learning stability and overall results [63]. The first trick is to normalize the data so that each $v_i \in [0, 1]$, which we call "soft data". If we are dealing with images, this can be interpreted as the probability that a pixel in a given image will be turned on. For the second trick, instead of

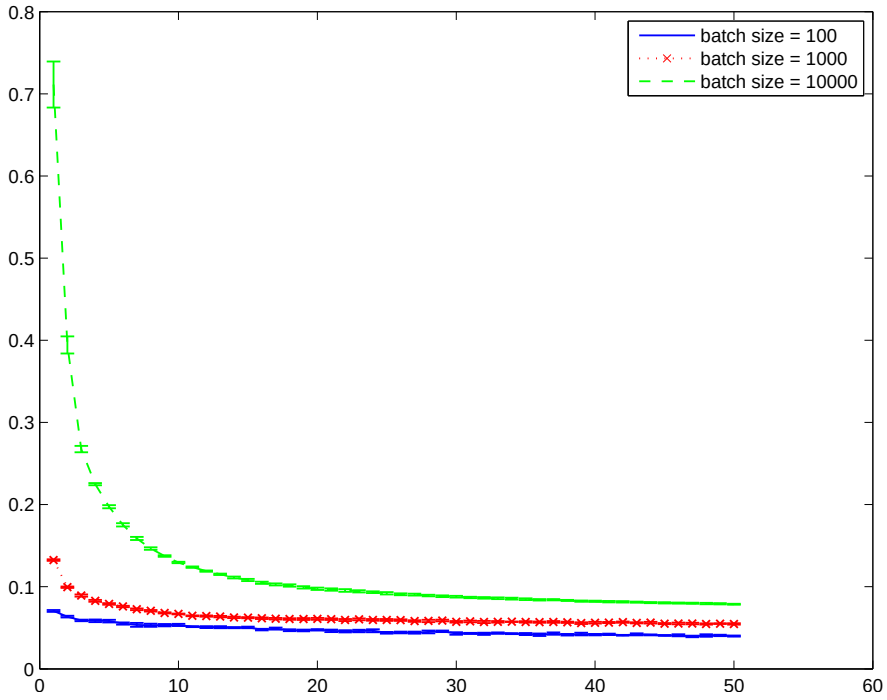


Figure 4.2: Test error vs iteration for training an RBM using stochastic approximation with different mini-batch sizes.

sampling $\widetilde{v} \sim p(v|\widetilde{h}, W)$, $E[v|\widetilde{h}, W]$ is used in the place of $\widetilde{v}$. We will call this using "soft samples". We experimented with all four combinations of using binary v.s. soft values with CD. In order to binarize the data, we thresholded the pixel values at 0.25.

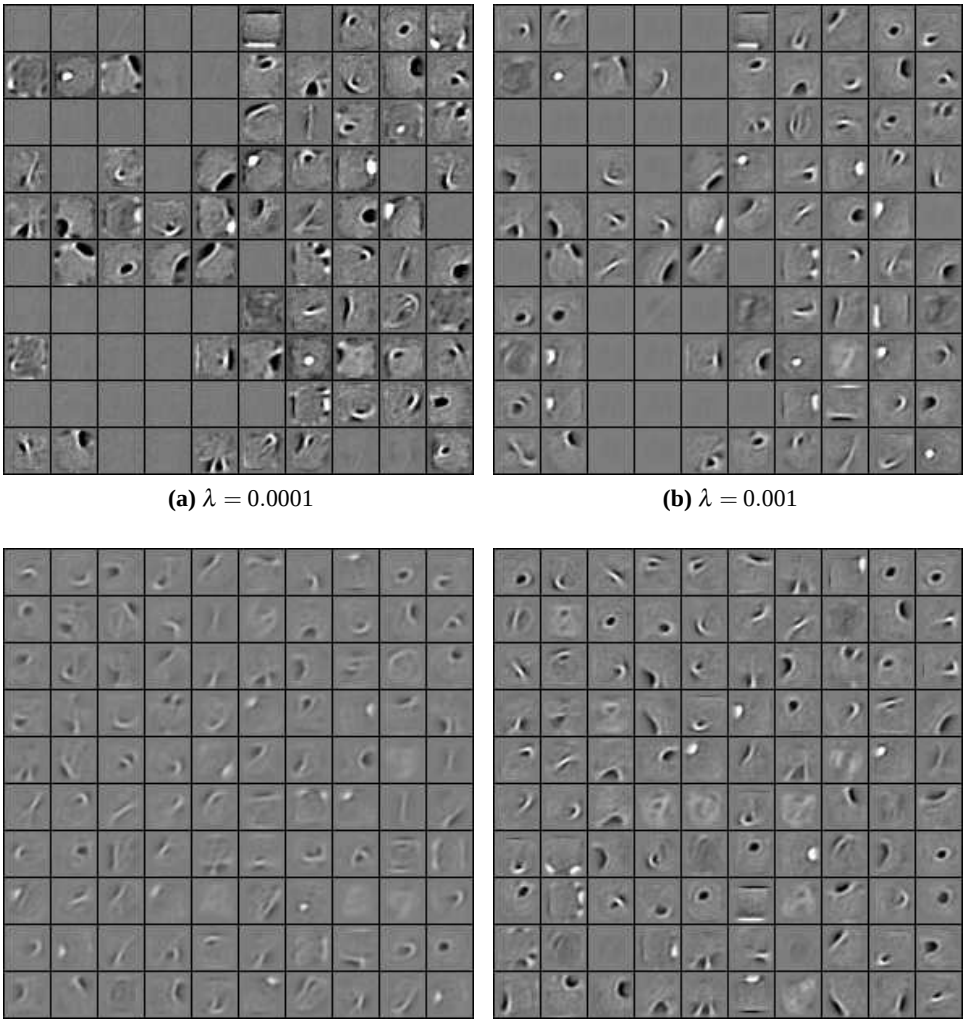
We found that sampling the second set of visible units properly is an important part of the algorithm, and that this trick should not be employed when seeking good classification performance. Normalizing the data appears to give better performance, perhaps because it retains more information about the original images which is lost in the binarization. As we showed in Section 1.4, the RBM can be modified to admit other distributions for the visible units which may be more suitable for this kind of data. An example of this is given in [33].

4.1.5 Continuation Methods

Figure 4.3 shows the receptive fields produced by CD. Also referred to as filters, these are plots of the weights coming into each hidden unit from each visible unit. We can display these as images, and we choose a randomly selected subset of hidden units to display. The pixels corresponding to weights larger than 1 are shown in white, and weights smaller than -1 are shown in black. All intermediate values are displayed in varying shades of grey. For a momentum of 0.8 and weight decay $\lambda = 0.0001$ some of the filters remain blank and uninteresting despite a good test error score. These blank filters correspond to dead units. They are never activated for any data points, and thus no learning occurs. It seems reasonable that each filter should contribute to the modelling of the data, otherwise they simply waste computational resources. A suprising result occurs when we change the value of the weight decay to 0.001: the previously blank filters become more interesting, though due to the higher penalty they are not quite as prominent. We see the same thing happen again when the weight decay is increased even more to 0.01. This suggests a new strategy where we anneal the weight decay from a high value to a low value over the course of training, in order to force the RBM to utilize as many hidden units as possible. We start with a weight decay of 0.1, and lower it to 0.01 after 5 iterations and finally 0.001 after 10 iterations.

This kind of strategy is referred to as a continuation method [1]. The idea behind this is that larger values of λ make the objective function more like a quadratic, essentially smoothing over shallow local minima and preserving deeper ones. This allows an optimization routine to move over suboptimal local minima that it might otherwise get stuck in. As training progresses, the value of λ is reduced so that a deep minimum of the original objective is eventually reached. Figure 4.4 shows that in addition to the qualitative difference in the receptive fields, test error results over the course of training also support this idea. Notice that as the weight decay is annealed, the test error rate quickly drops, eventually achieving less error than same model trained with a constant value for λ.

A similar strategy to this is to keep the weight decay constant, and to increase the momentum parameter β over the course of training instead [63]. As we mentioned in Section 4.1.3, the noise in stochastic gradient descent often allows it to



(a) $\lambda = 0.0001$ **(b)** $\lambda = 0.001$

(c) $\lambda = 0.01$ **(d)** Annealed λ

Figure 4.3: Receptive fields from RBMs trained on MNIST with various weight decay settings.

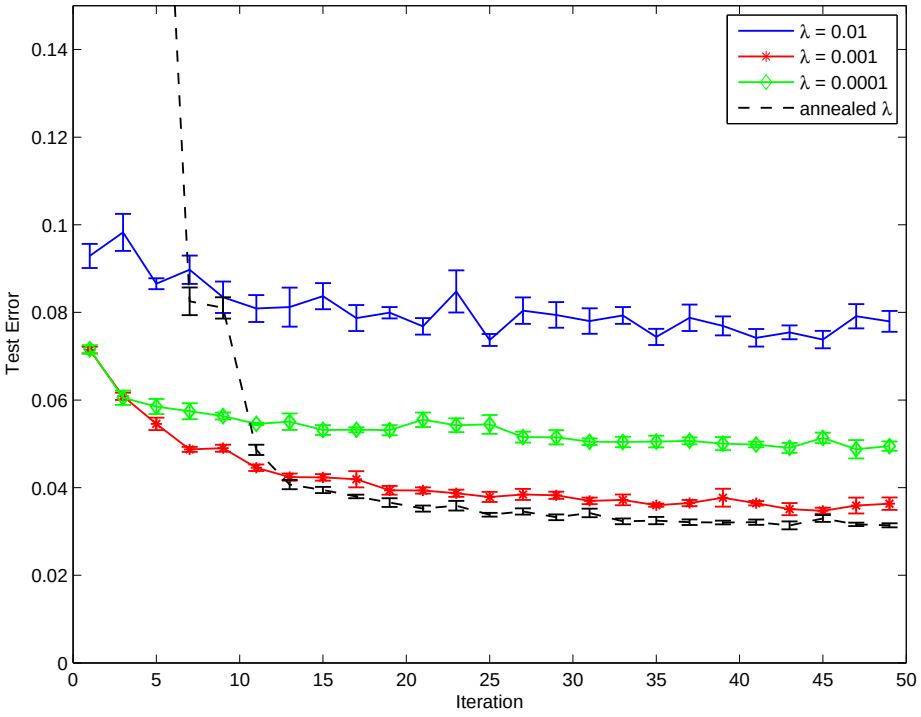


Figure 4.4: Test Error vs Iteration for various settings of the weight decay parameter λ, as well as an RBM trained by annealing λ.

skip over shallow local minima. A higher momentum will reduce the amount of noise, thereby reducing the chance that this will occur. It therefore makes sense to increase momentum slowly as training progresses so that a good region of the parameter space can be reached before the noise is reduced. Our experiments using this strategy gave simular results to those using annealed weight decay.

We should note that this technique isn't necessary for SML since it generally works best with a momentum of 0 throughout its entire training period.

4.1.6 Fast Weights and Averaging

While CD tends to be used more often for a wider variety of models, SML is considered the most effective stochastic algorithm for training RBMs when the goal is to directly maximize likelihood. This is supported by theory (since SML is

unbiased), and extensive experiments to support this are given in [69]. It was determined that although training an RBM with SML will generally provide a better solution, the process was quite noisy and required a relatively small learning rate to be successful whereas CD learning was rapid by comparison. Additionally, the learning rate would have to be decayed over the course of training as per standard stochastic approximation schemes to ensure that the magnitude of the sampling noise didn't overcome the magnitude of the gradient.

In [70], an extension to the standard learning algorithm for SML was proposed which learns a second set of weights with a much higher learning rate that are used in conjunction with the regular weights in the sampling phase of the algorithm. The idea is that as learning progresses, a high learning rate causes a rapid change in the energy landscape. Local modes that are concentrated away from the data are quickly removed, and their probability mass is transferred to vicinities around the data. In ordinary SML, as learning progresses and the learning rate is decayed, these changes become slower and the Markov chain will tend to spend more time in the same modes. By using an added set of weights with a fast learning rate, rapid exploration can continue, and since the regular weights have a small learning rate they will not be overwhelmed by the sampling noise. Additionally, the fast weights are decayed over the course of training using a decaying factor ρ so that toward the end of training their values will be reduced to 0, and only the regular weights will remain. We should note that fast weights SML is considered to be the state of the art algorithm for maximum likelihood training of RBMs. The formal algorithm for using SML with fast weights is given in Figure 4.5

Another alternative to training SML using a decaying learning rate is to use a constant rate and employ the averaging technique from Equation 4.1. This enables us to maintain a high learning rate throughout the course of training while still guaranteeing the optimal rate of convergence. Additionally, no schedule needs to be specified, thereby making parameter tuning significantly easier. We should also note that averaging and fast weights are not conflicting schemes, that is, averaging could certainly be used in conjunction with fast weights. The algorithm for training an RBM with SML using averaging is given in Figure 4.6.

It is clear that $\overline{W}$ acts as a completely parallel process and does not affect the actual optimization, this means that the samples are generated from the parameters

1. Set $t = 1$, and $\widetilde{h_n}^{(0)}$ to be a random K-dimensional binary vector. Set the fast weights $W_F^{(1)}$ to 0.

2. Sample $\widetilde{v_n}^{(t)}$ from $p(v|\widetilde{h_n}^{(t-1)}, W^{(t)} + W_F^{(t)})$, and $\widetilde{h_n}^{(t)}$ from $p(h|\widetilde{v_n}^{(t)}, W^{(t)} + W_F^{(t)})$.

3. Update the regular weights using the regular learning rate $\eta^{(t)}$:

$$W_{dk}^{(t+1)} = W_{dk}^{(t)} - \eta^{(t)} \left[-\frac{1}{N} \sum_{n=1}^{N} v_{dn} E[h_k|v_n, W^{(t)}] + \frac{1}{N} \sum_{n=1}^{N} \widetilde{v_{dn}}^{(t)} E[h_k|\widetilde{v_n}^{(t)}, W^{(t)} + W_F^{(t)}] \right]$$

4. Update the fast weights using the fast learning rate $\eta_F^{(t)}$:

$$W_{Fdk}^{(t+1)} = \rho W_{Fdk}^{(t)} - \eta_F^{(t)} \left[-\frac{1}{N} \sum_{n=1}^{N} v_{dn} E[h_k|v_n, W^{(t)}] + \frac{1}{N} \sum_{n=1}^{N} \widetilde{v_{dn}}^{(t)} E[h_k|\widetilde{v_n}^{(t)}, W^{(t)} + W_F^{(t)}] \right]$$

5. Increase t to $t+1$ and iterate again.

Figure 4.5: The stochastic maximum likelihood algorithm using fast weights.

W. This is similar to the fast weights algorithm only in the sense that we have one set of parameters that operate on a quickly decaying learning rate, and another that operates on a slowly decaying learning rate (in this case, with no decay at all).

One common problem with all of these methods is that the number of training epochs must be known in advance, especially in cases where a decreasing schedule is applied. Without averaging, the user needs to maintain a balance between choosing a large enough learning rate to make sufficient progress, and ensuring that it decays quickly enough that the training converges. With averaging, the user needs to consider when the averaging phase should begin. If averaging is employed too early then the averaged iterates may actually hurt convergence. In [30], a different form of averaging is suggested which is given by:

1. Set $t = 1$, and $\widetilde{h_n}^{(0)}$ to be a random K-dimensional binary vector.
2. Sample $\widetilde{v_n}^{(t)}$ from $p(v|\widetilde{h_n}^{(t-1)}, W^{(t)})$, and $\widetilde{h_n}^{(t)}$ from $p(h|\widetilde{v_n}^{(t)}, W^{(t)})$.
3. Update the parameters:

$$W_{dk}^{(t+1)} = W_{dk}^{(t)} - \eta^{(t)} \left[-\frac{1}{N} \sum_{n=1}^{N} v_{dn} E[h_k|v_n, W^{(t)}] + \frac{1}{N} \sum_{n=1}^{N} \widetilde{v_{dn}}^{(t)} E[h_k|\widetilde{v_n}^{(t)}, W^{(t)}] \right]$$

4. Update the averaged parameters:

$$\overline{W_{dk}}^{(t+1)} = \overline{W_{dk}}^{(t)} - \frac{1}{t}(\overline{W_{dk}}^{(t)} - W_{dk}^{(t+1)})$$

5. Increase t to $t+1$ and iterate again.
6. When convergence is reached, return $\overline{W}$ instead of W.

Figure 4.6: The stochastic maximum likelihood algorithm using iterate averaging.

$$\begin{aligned} W_{dk}^{(t+1)} &= W_{dk}^{(t)} - \eta^{(t)} \nabla_{W_{dk}} log(p(v|W^{(t)})) \\ \overline{W_{dk}}^{(t+1)} &= (1-\alpha)\overline{W_{dk}}^{(t)} + \alpha W_{dk}^{(t+1)}, \end{aligned} \tag{4.2}$$

for some small positive α.

This averaging can be run from the very beginning of the training process. Each

term that it adds is effectively decayed exponentially since:

$$\begin{aligned}
\overline{W}^{(t+1)} &= (1-\alpha)\overline{W}^{(t)} + \alpha W^{(t+1)} \\
&= (1-\alpha)((1-\alpha)\overline{W}^{(t-1)} + \alpha W^{(t)}) + \alpha W^{(t+1)} \\
&= (1-\alpha)^2\overline{W}^{(t-1)} + (1-\alpha)\alpha W^{(t)} + \alpha W^{(t+1)} \\
&\vdots \\
&= (1-\alpha)^{t+1}\overline{W}^{(0)} + (1-\alpha)^t \alpha W^{(1)} + \ldots + \alpha W^{(t+1)}.
\end{aligned}$$

That is, each successive iteration multiplies all previous iterates by $1-\alpha$. The lower the value of α, the longer it takes for an iterate's contribution to decay to 0, which means that the averaging encompasses a longer window into the past.

A major advantage of this form is that the user no longer has to specify the number of iterations in advance, or when to begin averaging. As long as the step size is chosen reasonably well, then it is simply a matter of waiting until the iterates converge. We tried this approach using $\alpha = 10^{-4}$ and found that it gives effectively the same results as using the traditional form of averaging. We also found that it is quite robust to the choice of α provided it is small enough. In order to see how averaging affects the results at each iteration, this section's experiments use the alternative form of averaging; though in general, it produces very similar results to the standard form.

Figure 4.7 shows the results of our comparison of SML with and without averaging, as well as fast weights with and without averaging. We trained each method 5 times using 300 iterations per trial. For the trials using fast weights without averaging, we linearly annealed the learning rate to 0 over the course of training as recommended in [70], and for the other variations we kept the learning rate constant. For each method, we tried initial learning rates of $\{0.01, 0.05, 0.1, 0.25, 0.5\}$ and weight decays $\left\{10^{-4}, 10^{-6}\right\}$. For the trials involving fast weights, we follow the recommendation given in [70] and set the fast learning rate $\eta_F^{(t)}$ to be e^{-1}. We report the results for the setting that gave the lowest final test error on each method separately.

Averaging appears to be extremely beneficial for training an RBM with SML. In particular, it is vastly more efficient than fast weights with a linearly decaying

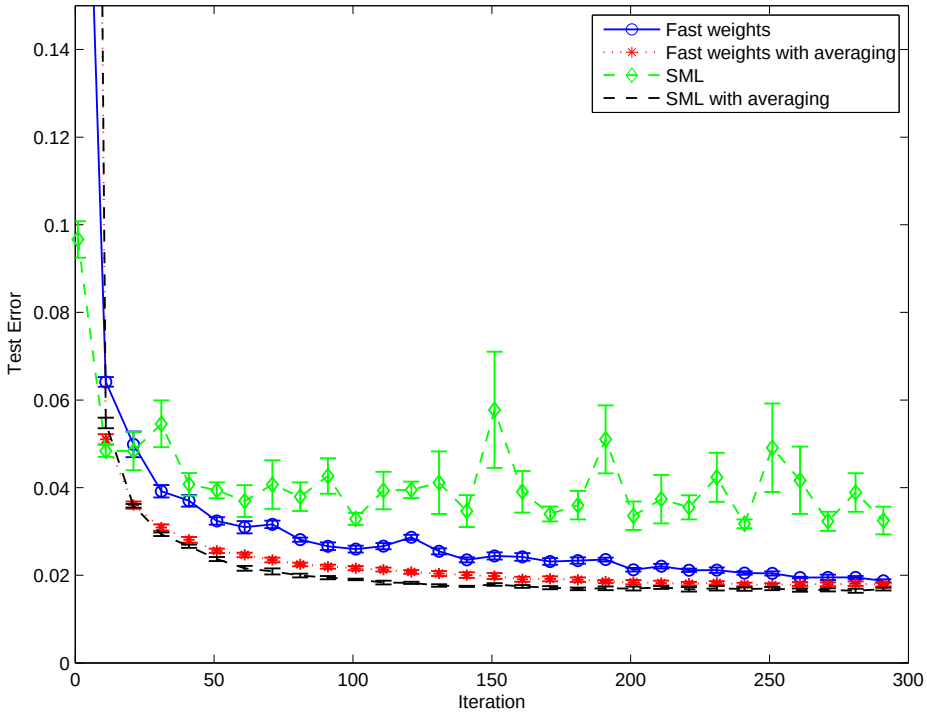


Figure 4.7: A comparison of fast weights and averaging for training an RBM using SML.

learning rate. While it also improves the convergence of fast weights, it seems that the use of fast weights isn't even necessary to begin with since averaging already allows the use of a more aggressive learning rate. The plot also shows the result of SML using a constant learning rate, which is what the test error would be if we simply used the parameter value at each iteration. In addition to giving much worse error, it is significantly more volatile.

It is worth noting that averaging can also be applied to CD learning, though we found that it doesn't necessarily improve test error results so much as reduces the amount of volatility associated with the final results.

4.2 Discussion

Learning the parameters of a Restricted Boltzmann Machine is a nontrivial problem. For stochastic training algorithms, noise and non-convexity can seriously hamper performance if care isn't taken to avoid suboptimal local minima. Although perfect optimization is far from guaranteed, the techniques introduced in this chapter can certainly help. Averaging in particular is a simple tool that simultaneously eases the burden on the user (in terms of parameter tuning), and greatly improves convergence.

There are many ways that the results from this chapter can be expanded and improved upon. For example, SML would benefit greatly from more efficient sampling schemes that can rapidly explore different states; examples of this are given in [11, 62]. Another approach is to adopt more efficient online learning algorithms utilizing second order information, such as in [67]. Again, it is likely that averaging can be used in conjunction with these methods, and could provide a very elegant way to ensure convergence.

Finally, although this chapter focused on the stochastic algorithms from Chapter 2, they can still apply to the deterministic algorithms if the mini-batch procedure is used. The reasons for considering this are: faster convergence compared to batch optimization, and the ability to handle massive datasets with millions or billions of examples which would not fit in main memory.

Chapter 5

Experiments using Inductive Principles for Binary Data

In the last chapter we saw further evidence that SML is superior to CD in terms of directly optimizing the likelihood of the data. Likelihood, however, is only one metric by which these methods can be compared. As we stated in Chapter 1, the main goal of this work, and the main utility of the Restricted Boltzmann Machine, is to derive features from data that can be used across many different tasks. In this chapter we will perform a number of experiments on several datasets in order to give a broader illustration of the strengths and weaknesses of each inductive principle from Chapter 2. The results in this chapter should help to inform future decisions about which training algorithm is suitable for a given task.

This chapter is based on content from [42], and was written in conjunction with Benjamin Marlin, Bo Chen, and Nando de Freitas.

5.1 Experiments with Simulated Data

Our first experiment attempts to assess the efficiency of each training method by learning a small RBM on a synthetic dataset. The data itself was generated from a small RBM with a known partition function, which means that the assumption that the data distribution is within the model class holds.

1. Sample a hidden configuration $\widetilde{h}$ from $p(h)$ using Equation 5.1.
2. Sample a visible configuration $\widetilde{v}$ from $p(v|\widetilde{h})$.

Figure 5.1: An algorithm to sample data from an RBM with a small number of hidden units.

5.1.1 Experiment Setup

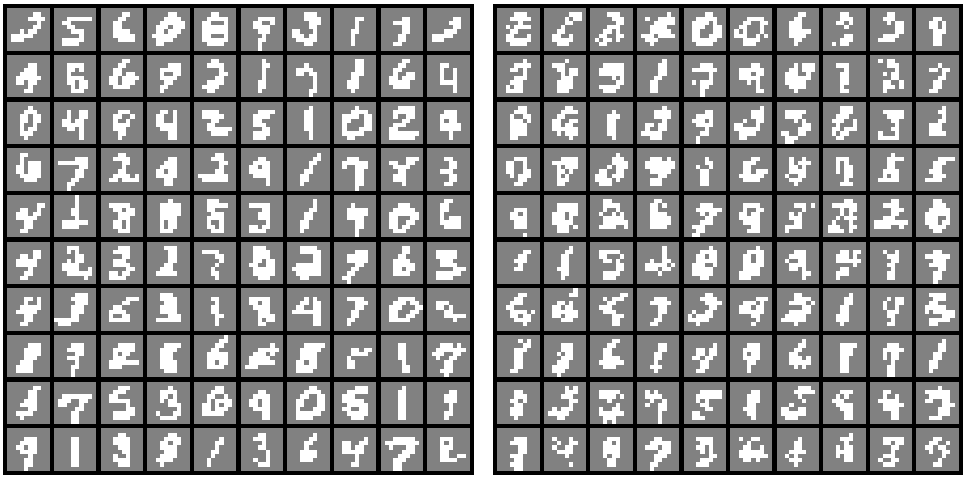
In order to ensure that the synthetic dataset represented a peaked distribution with interesting correlations between visible units, we downsampled $10,000$ MNIST digits from 28×28 images to 10×10 images. Using these, we learned an RBM with 100 visible units and 15 hidden units using exact maximum likelihood. This was possible, since the likelihood of the RBM given in Equation 1.6 is tractable for a model this small. In fact, in this model we can compute the probability of a hidden configuration $p(h)$ exactly since:

$$\begin{aligned} p(h) &= \frac{\sum_{v \in V} exp\left(\sum_{i=1}^{D}\sum_{j=1}^{K} v_i W_{ij} h_j\right)}{\sum_{h' \in H}\prod_{i=1}^{D}\left(1 + exp\left(\sum_{j=1}^{K} W_{ij} h'_j\right)\right)} \\ &= \frac{\prod_{i=1}^{D}\left(1 + exp\left(\sum_{j=1}^{K} W_{ij} h_j\right)\right)}{\sum_{h' \in H}\prod_{i=1}^{D}\left(1 + exp\left(\sum_{j=1}^{K} W_{ij} h'_j\right)\right)}. \end{aligned} \tag{5.1}$$

Using this, we are able to generate a new visible configuration by the procedure shown in Figure 5.1.

Some samples of downsized MNIST digits, as well as data points generated from this model are shown in Figure 5.2. Although they do not correspond to very pretty digits, they represent a sufficiently nontrivial, peaked distribution.

For each method SML, CD, PL, and RM, we tested their ability to minimize the KL divergence between an RBM and the true distribution. It should be noted that for this particular setup, even though we can evaluate $p_e(v)$ for a given v, it is infeasible to calculate the true KL since it would involve summing over 2^{100} states. We can't compare models using the hidden space H either, since RBMs



(a) Small MNIST digits **(b)** Synthetic digits

Figure 5.2: 10×10 digits from a downsized version of MNIST, and synthetic digits generated from an RBM with 100 visible units and 15 hidden units trained using maximum likelihood.

are unidentifiable. This means that after learning an RBM, one could switch the order of the hidden units and this would still define the same distribution over the visible space V. Effectively, this means that we cannot be sure of the relative order of the hidden units between two separately trained RBMs and so comparing the probability of a hidden configuration under two RBMs is meaningless. Instead of trying to evaluate the full KL, we approximate it empirically using a test set with one million examples:

$$KL(p_e(v)||p_W(v)) \approx \frac{1}{10^6} \sum_{m=1}^{10^6} log(p_e(v_m)) - log(p_W(v_m)).$$

We generated 4 datasets of sizes $\{100, 1000, 10000, 100000\}$, and for each of these we train each method using momentum values in the range $\{0, 0.3, 0.5, 0.8\}$, and learning rates in the range $\{0.01, 0.1\}$ for CD and SML, and $\{10, 100\}$ for RM and PL. For each parameter setting, we ran trials that used averaging, and trials that didn't, and all trials were repeated 5 times to get error measurements. At each

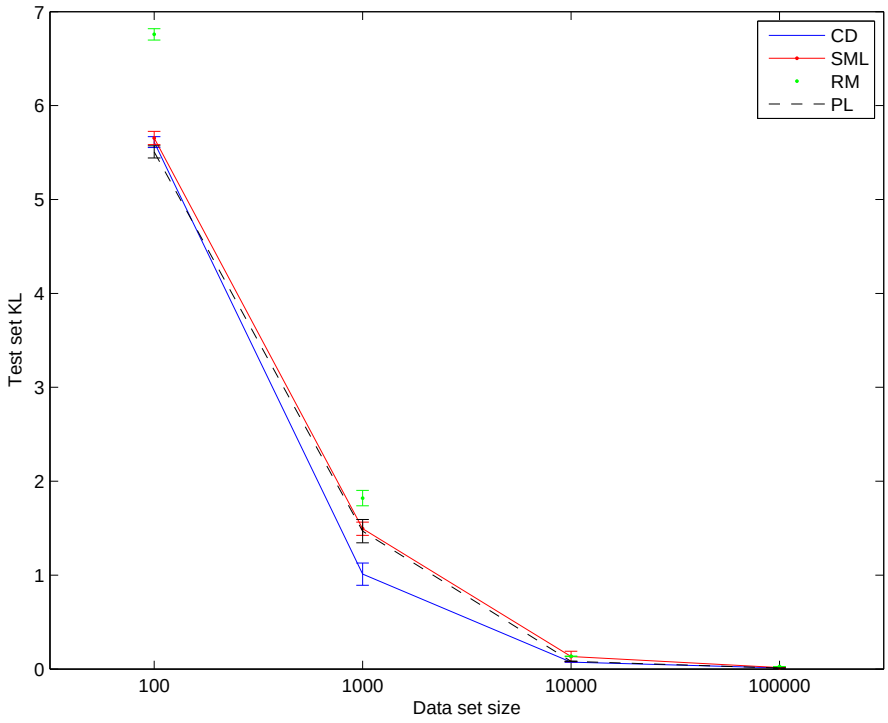


Figure 5.3: Test set KL vs dataset size for an RBM trained on synthetic data using various inductive principles.

dataset size, the best performing setting for each method individually is reported.

5.1.2 Results

Figure 5.3 shows that each of the tested methods appears to be consistent, since the *KL* divergence seems to decay toward 0 as the dataset size increases. Surprisingly, CD appears to be more efficient than SML when the dataset size is small, and ratio matching seems to be the least efficient of the different methods. It is worth mentioning that since the number of possible states is relatively small, this would make the local methods (everything except SML) much more efficient since there are fewer states far away from any data configuration that would need to be explored.

5.2 Experiments with Real Data

The synthetic experiment tested the ability of each method to match the data distribution to the model distribution in terms of minimizing KL divergence. While it is certainly important, it is not the only test of a model's ability. Indeed, there are many other ways by which we can evaluate different training measures depending on the task for which the model will eventually be used. For example, we have so far tested classification in an RBM by incorporating class bits directly into the likelihood and evaluating relative probabilities under the model. A different way is to consider the hidden units as features, and to apply a standard classifier to these features. In this section we will attempt to evaluate RBMs trained with different techniques on a range of tasks using several datasets.

5.2.1 Experiment Setup

We use three datasets in the experimental comparisons: MNIST, the small 20-Newsgroups[1] dataset, and a new binary image dataset derived from Caltech-101[2]. MNIST consists of 28×28-size images of hand-written digits from 0 through 9. We binarize the images and divide the database into a training set of $50,000$ examples and a validation set of $10,000$ examples, and use the standard $10,000$ example test set. The small 20-Newsgroups dataset contains newsgroup postings divided into four classes of groups. Each posting is represented by binary vectors over a vocabulary of 100 words. We split this dataset into 8500 training, 1245 validation, and 6497 test examples respectively. The final dataset we use is derived from the object outlines contained in the Caltech-101 annotations dataset. The object outlines were centered and scaled on a 28×28 image plane and rendered as filled black regions on a white background creating a silhouette of each object. We call this dataset Caltech-101 Silhouettes[3]. We show examples from several of the 101 classes in Figure 5.4. The training set contains 4100 examples with at least 20, and at most 100 examples from each class. The remaining images were split among a test set and validation set of size 2307 and 2264 respectively.

[1] http://www.cs.toronto.edu/~roweis/data.html
[2] http://www.vision.caltech.edu/Image_Datasets
[3] http://www.cs.ubc.ca/~bmarlin/data/

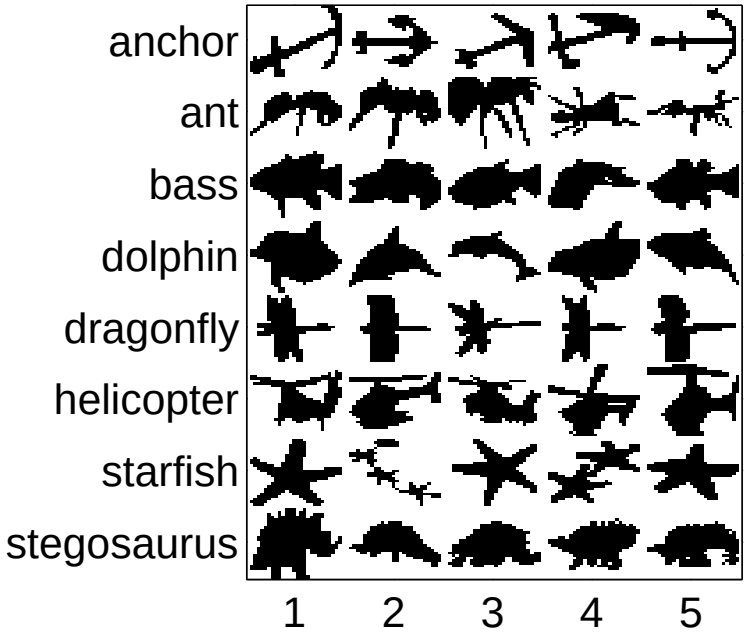


Figure 5.4: Examples from the Caltech-101 Silhouettes dataset

Our training protocol begins with 100 iterations of stochastic gradient descent (SGD) on mini-batches of 100 data cases using momentum and iterate averaging as acceleration techniques. To reduce computation time, we select the SGD learning rate, momentum parameter and whether or not to use iterate averaging separately for each method by minimizing a performance measure on a smaller subset of each dataset. For CD and SML, we chose one-step reconstruction error as the performance measure, while for RM and PL we used their objective function values.

We fine-tune methods with computable objective functions, using an L-BFGS optimizer from Mark Schmidt's minFunc package[4]. We select a weight decay setting specific to each dataset, experiment, and method by learning using the full training set with a validation set held-out. Lastly, we do five full training runs for each dataset, experiment, and method using the selected learning rate, momentum, iterate averaging and weight decay settings and average the results. The

[4]http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html

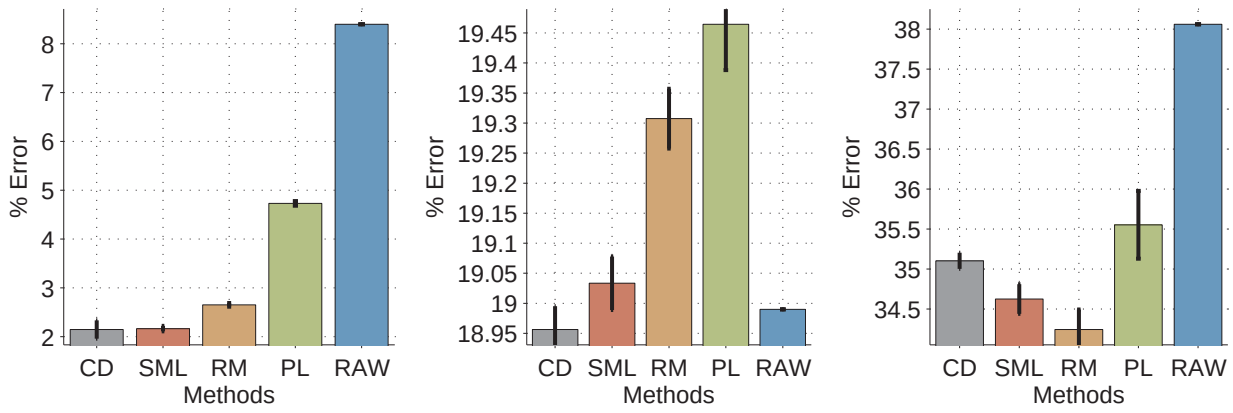
learning rate range was $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$. The momentum range was $\{0, 0.3, 0.5, 0.8\}$. The weight decay range was $\{10^{-5}, 10^{-4}, 10^{-3}\}$. We use 500 hidden units for all experiments.

5.2.2 Classification

We tested the ability of each method to produce useful features for classification by training a support vector machine (SVM) classifier [71] on the class labels y_c and expected hidden activations $E[h|v,W]$. We use a smooth, multi-class, linear SVM with an L2-penalty selected to minimize validation set error [37]. We report the corresponding test set error for each dataset in Figure 5.5. In addition, we include the results of applying the SVM to the raw data to see whether the RBM features actually improve results. From this we see that SML does consistently well across all datasets, and that pseudo-likelihood is consistently the worst performer. Another interesting result is that CD tends to do quite well in comparison to SML on MNIST, in particular we see that CD can actually produce features that deliver good classification performance compared to SML when the likelihood is substituted for a separate classifier. The Caltech results are a bit less obvious, but as shown in Figure 5.4, there is considerable variation between examples, both within class and between classes. One possible explanation for the success of ratio matching on this dataset is that it might spend less time trying to model difficult outliers, focusing instead on modelling more common examples. Interestingly, the RBM features actually tend to either not affect, or even hurt performance on the 20-Newsgroups dataset. One possible explanation is that since the dataset has been filtered down to the 100 most commonly occuring words, there is very little that additional feature extraction would do. Another is that 500 hidden units are too many for this dataset, and that the optimization routine consistently gets stuck in a poor minimum.

5.2.3 Likelihood

There have been significant recent advances in Monte Carlo methods for estimating the log partition function of an RBM that are computationally feasible for post-training analysis. We compute test set log-likelihoods by estimating the log parti-



(a) MNIST **(b)** 20-Newsgroups **(c)** Caltech

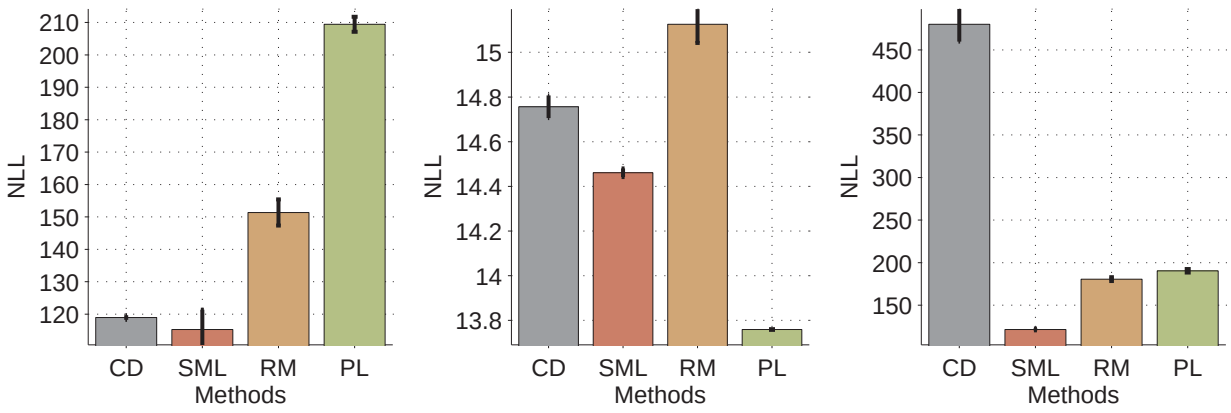
Figure 5.5: Test set classification error from various RBM training methods using an SVM classifier on hidden activations.

tion function for each trained model using the annealed importance sampling (AIS) method proposed in [45] using the provided code[5]. We show the results of this evaluation in Figure 5.6. We see a win for the stochastic approximate maximum likelihood (SML) method, which achieves the best average test set log likelihood on MNIST and Caltech and the second best on 20-Newsgroups. This is not surprising since SML is the only method specifically trained to optimize the likelihood. Interestingly, PL gives the best test set likelihood on the 20-Newsgroups dataset. While it is difficult to say for certain why this is (especially in light of the poor classification performance), it could either be an issue with the parameters of the AIS sampler, or a product of this particular dataset. One way to perhaps find out would be to repeat this experiment on the full 20-Newsgroups dataset[6].

5.2.4 Denoising

Reconstruction and de-noising are commonly used to asses the performance of RBM models and auto-encoders. We consider a de-noising task where we select a certain fraction of bits in each test data case v_n and set them to 0 or 1 with even

[5] http://www.cs.toronto.edu/~rsalakhu/rbm_ais.html
[6] http://people.csail.mit.edu/jrennie/20Newsgroups/



(a) MNIST **(b)** 20-Newsgroups **(c)** Caltech

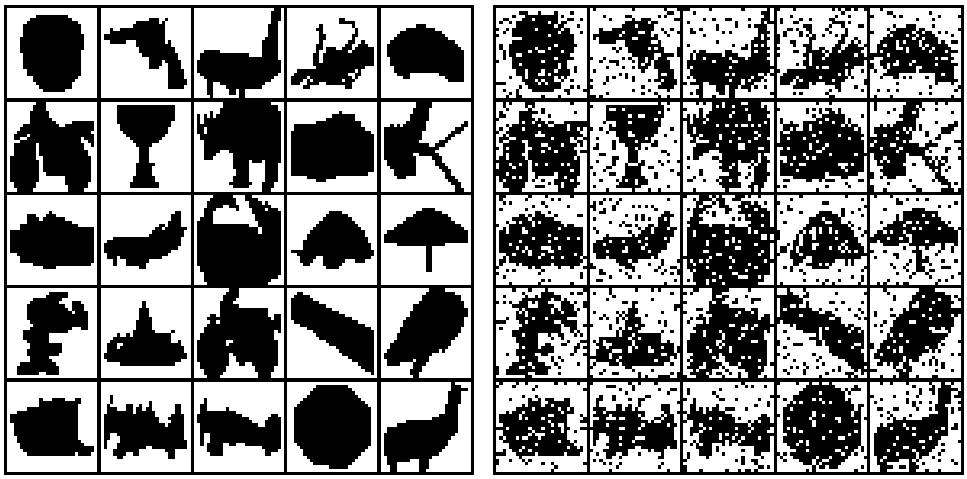
Figure 5.6: Test set negative log-likelihood from various RBM training methods estimated with AIS.

probability, creating a noisy version of the data case $\widetilde{v_n}$. We then compute the 1-step mean field reconstruction of v_n using $\widehat{h_n} = E[h|\widetilde{v_n}]$ and $\widehat{v_n} = E[v_n|\widehat{h_n}]$. We measure the average per-dimension reconstruction error using mean squared error between v_n and $\widehat{v_n}$: $\frac{1}{ND}\sum_{n=1}^{N}\sum_{i=1}^{D}(v_{in} - \widehat{v_{in}})^2$. Figure 5.7 shows several examples from the Caltech Silhouettes dataset in their original, corrupted, and repaired states. In this case, the denoising was performed by an RBM trained with CD, and the noise level was set to 25%.

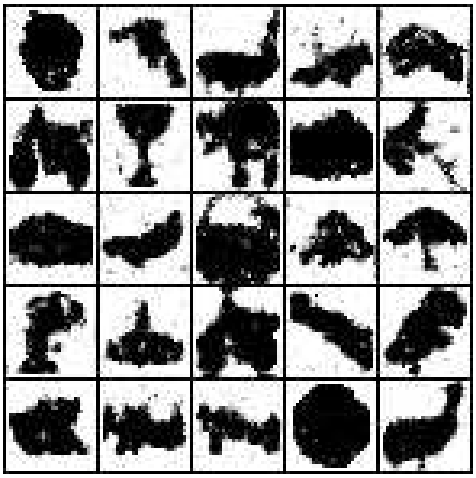
The results of this analysis are presented in Figure 5.8 (note that in the error bars are inside the line markers and were not plotted). These results show a consistent advantage for ratio matching across the three datasets.

5.2.5 Novelty Detection

We consider a novelty detection task that looks at how the free energy of test cases varies as we add random noise. We select bits at random and set them to 0 or 1 with even probability. We report the relative free energy defined as the difference between the free energy at a given noise level and the free energy at the zero-noise baseline. The results for this task are presented in Figure 5.9 (note that in the error bars are inside the line markers and were not plotted). Pseudo-likelihood

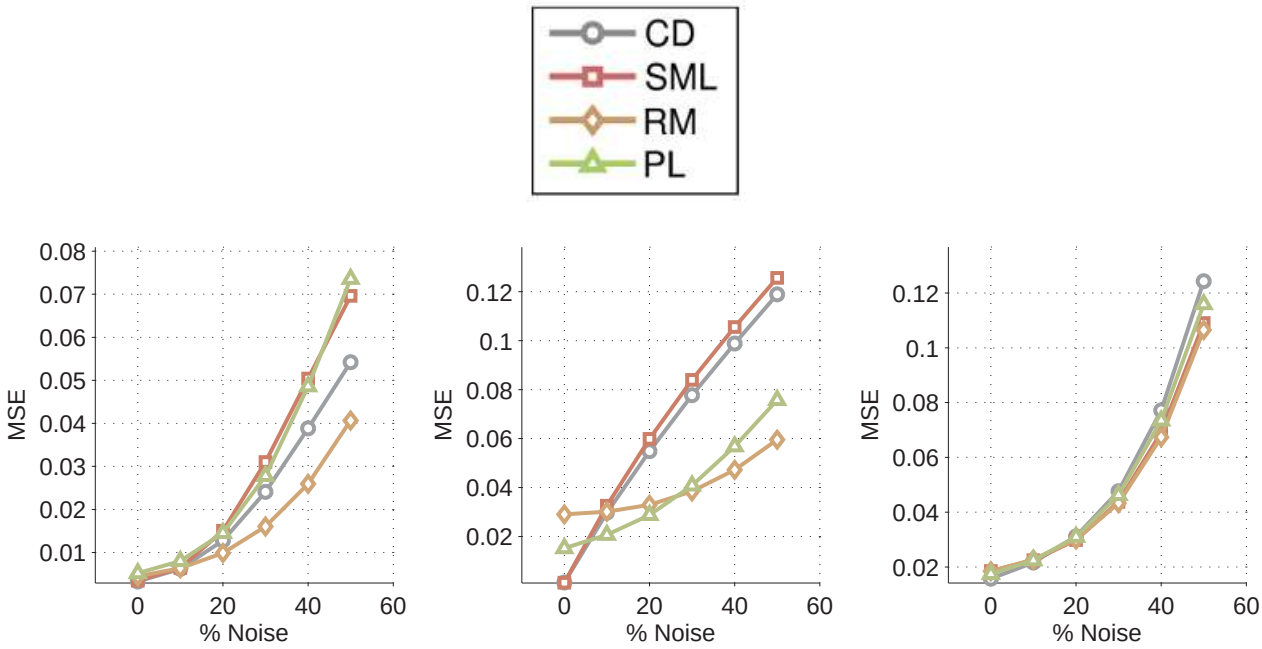


(a) Original data **(b)** Corrupted data



(c) Repaired data

Figure 5.7: Examples from Caltech Silhouettes after being corrupted with noise, then repaired by an RBM trained with CD.



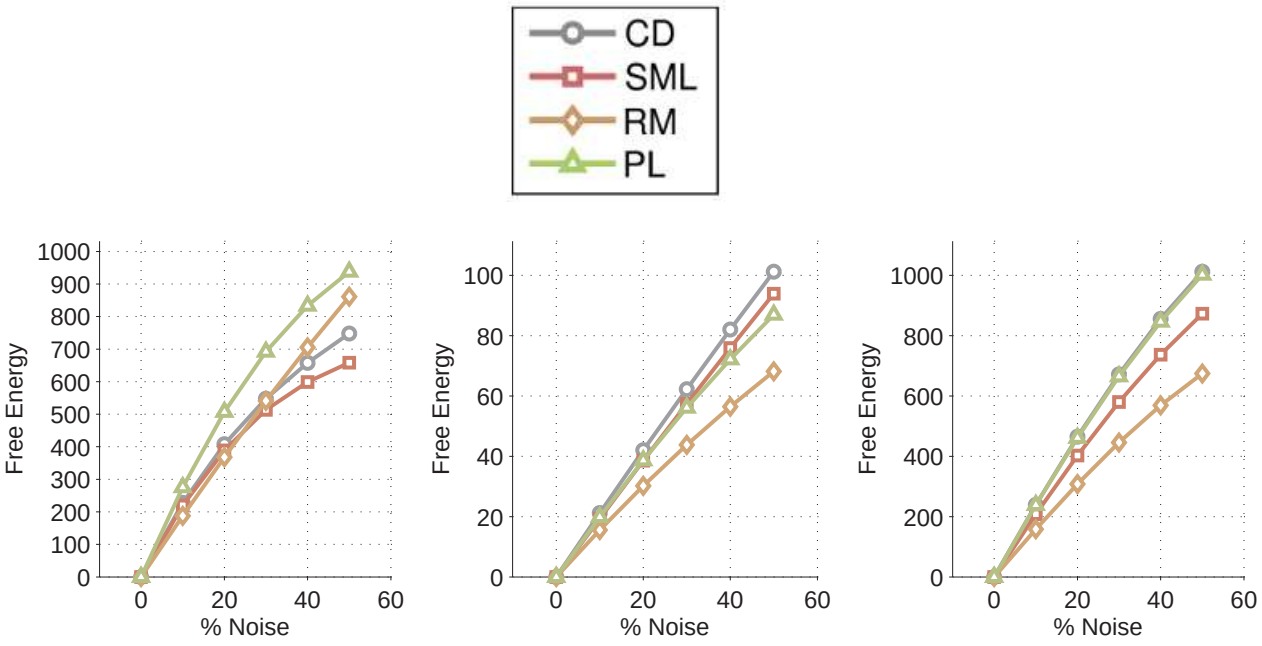
(a) MNIST **(b)** 20-Newsgroups **(c)** Caltech

Figure 5.8: Test set denoising mean squared error as a function of the percent of bits corrupted by noise.

appears to be the most sensitive to noise, with a rate of free-energy increase that is either similar to or above the other methods. Conversely, ratio matching appears to generally be the least sensitive. This provides evidence that pseudo-likelihood will tend to create a much sharper distribution than ratio matching.

5.2.6 Visualization of Learned Features

As a final qualitative assessment of the trained RBM models, we display the learned weights W and visible biases c for each training method on the MNIST dataset. We select the regularization setting that results in the lowest classification error. A random subset of the 500 learned weight vectors are shown in Figure 5.10 where the top left cell in each figure is the visible bias vector. The weight matrix for a hidden unit h_j can be thought of as a linear filter or feature detector where a larger filter response will result a larger value of $E[h_j|v]$.



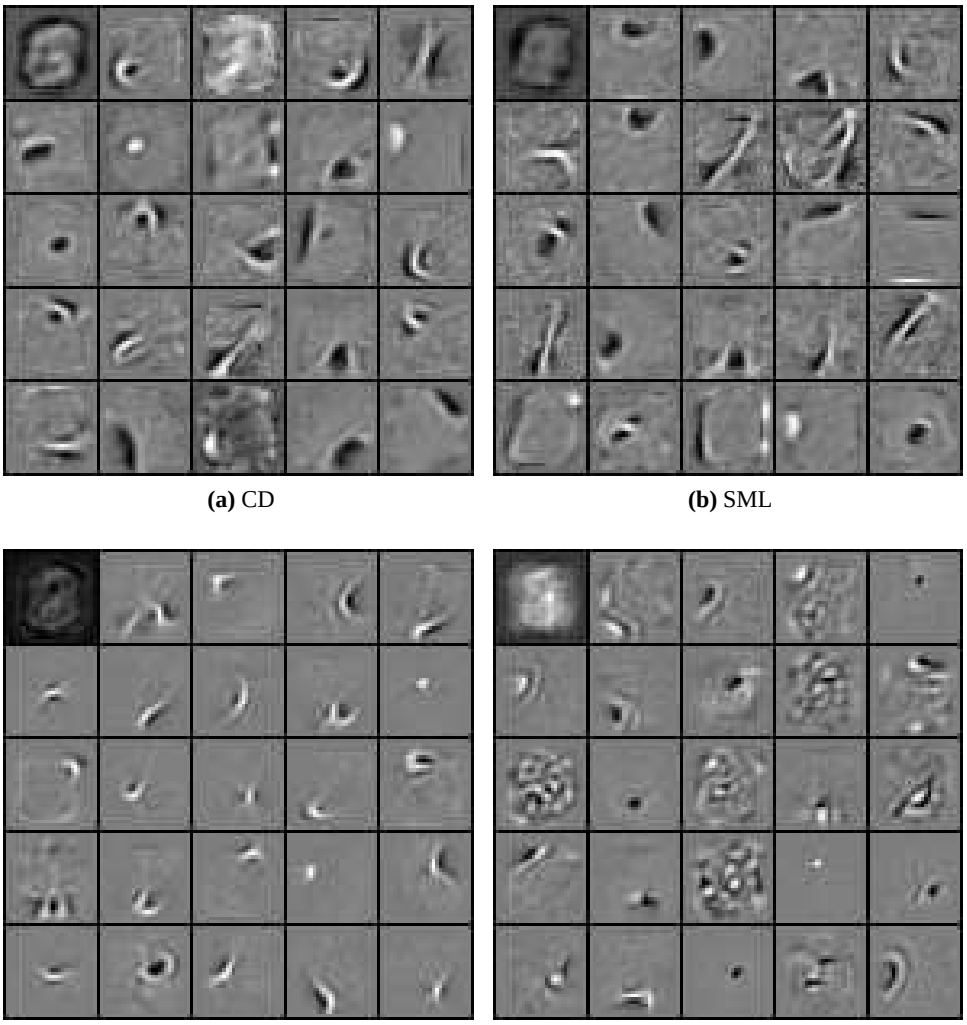
(a) MNIST **(b)** 20-Newsgroups **(c)** Caltech

Figure 5.9: Relative test set free energy as a function of the percent of bits corrupted by noise.

CD and SML appear to learn a combination of localized spot filters, stroke filters, and assorted non-localized filters. These results are in excellent agreement with previous results in both [18, Figure 3] and [57, Figure 1(f)], which show an almost identical mixture of spot filters and non-localized filters. Pseudo-likelihood seems to learn a combination of extremely localized spot filters, and non-localized noisy filters. Interestingly, the ratio matching filters are very similar to previous results on MNIST obtained using sparse RBMs [35, Figure 2] and other sparse coding models [57, Figure 2(d)], even though the ratio matching objective does not include an explicit sparsity term.

5.3 Computation Time

Basic implementations of pseudo-likelihood and ratio matching have a computational complexity that is approximately D times higher than stochastic maximum



(a) CD **(b)** SML

(c) RM **(d)** PL

Figure 5.10: Randomly selected receptive fields and visible biases learned using various training methods on the MNIST dataset. The top left cell in each figure corresponds to the visible bias vector.

likelihood SML and contrastive divergence due to the fact that CD and SML consider one fantasy data case per training case, while PL and RM consider D fantasy cases corresponding to the D possible single bit flips. A more careful implementation of PL and RM can reduce this gap by re-using intermediate computations. On the 100 dimensional 20 Newsgroups data we observe that PL and RM are approximately 10 times slower than SML and CD. On the 784 dimensional MNIST and CalTech101 datasets, they are approximately 16 times slower. One way these could be sped up is by sampling a subset of dimensions randomly at each iteration instead of iterating over every dimension.

5.4 Discussion

In this section we did an extensive empirical comparison of the different training methods from Chapter 2. We found that SML is the best algorithm in terms of directly optimizing the likelihood of an RBM, and that it is generally the best performing algorithm for classification. Ratio matching appears to be the best algorithm for denoising, and this corroborates well with the observation that it produces localized stroke filters instead of spot filters on MNIST. On the novelty detection task, pseudo-likelihood exhibits the most sensitivity. While CD and SML tended to dominate on MNIST, they did not fare nearly as well on Caltech which is a more difficult dataset. Caltech also happens to be quite small, and it is possible that given more examples their relative performance might change. It would be interesting to repeat these experiments on another binary dataset the size of MNIST or larger. An interesting direction for future work would be to extend these algorithms to the deep learning case, to see how they fare in pre-training a DBN.

Chapter 6

Experiments using Inductive Principles for Continuous Data

In this chapter, we test whether score matching can produce good features on an RBM with Gaussian visible units, and binary hidden units. This RBM is much slower to train than the standard binary RBM, as it is less stable with high learning rates. The experiments provided here serve as a proof of concept that the score matching criterion is suitable for the GBRBM.

6.1 Experiment Setup

For our experiments, we used the 32×32 images from CIFAR-10[1] dataset. This is a dataset of small photographic images with each image belonging to one of 10 possible classes. In order to keep the size of the experiments small, we converted the images from colour into greyscale. We further split the dataset into $40,000$ training examples, $10,000$ validation examples, and used the default $10,000$ example test set.

We trained several GBRBMs using contrastive divergence, stochastic maximum likelihood, and score matching, over a wide range of parameter settings. We used learning rates in the range $\left\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\right\}$, weight decay in the range $\left\{10^{-5}, 10^{-4}, 10^{-3}\right\}$, and the variance parameter σ in the range

[1] http://www.cs.utoronto.ca/~kriz/cifar.html

$\{1, 10, 25\}$. Additionally, we trained each method using stochastic gradient descent on mini-batches of 100 points, using momentum in the range $\{0, 0.5, 0.8\}$, both with and without averaging. For each GBRBM, we used 1000 hidden units. In order to obtain error bars, we took the best performing settings for each method, and repeated the experiments for 5 trials.

Examples of the filters learned by each method are shown in Figure 6.1 which are in excellent agreement with filters learned on natural image patches using CD in [12, Figure 2]. In particular, there are a mixture of high frequency, and low frequency patterns, but very few appear to be localized.

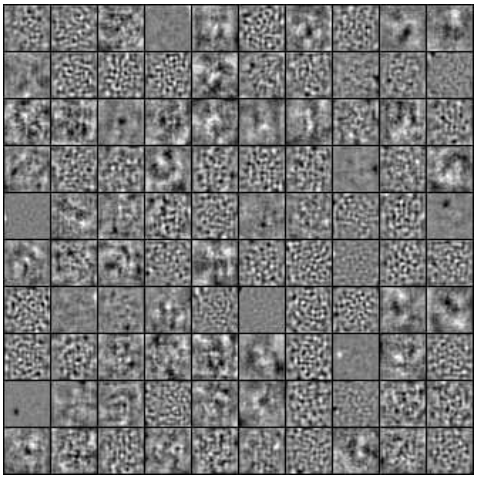
6.2 Classification

Our first experiment involved using the hidden units of the GBRM for classification. As in Section 5.2.2, we use a linear multi-class support vector machine classifier with an L2-penalty selected to minimize error on the validation set. We also tested the classifier on the raw image pixels since there were no benchmarks for CIFAR-10 with colour removed.

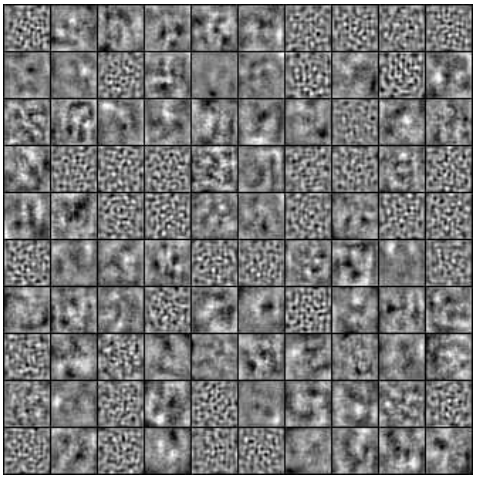
The results shown in Figure 6.2 indicate that the features learned by the GBRBMs significantly improve the classification error regardless of the training algorithm used. We can also see that features learned with SML performed the best, followed by score matching, and CD.

6.3 Denoising

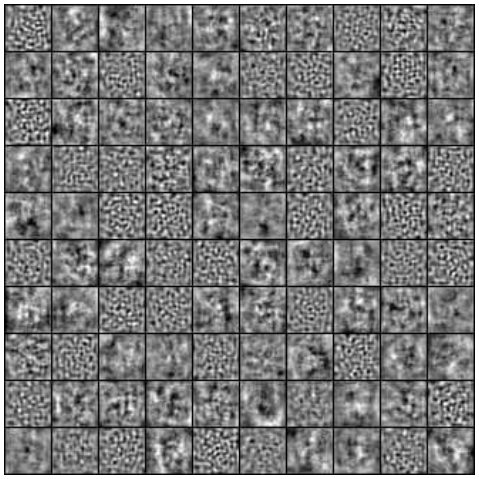
Our next experiment involved repairing images which had been corrupted by noise. Since the pixel values are continuous, we corrupted each image using zero-mean additive Gaussian noise. For all methods, the reconstructions were made by first computing the expected hidden activations $E[h|v, W, c, b, \sigma]$, then computing the reconstructed image $E[v|E[h|v, W, c, b, \sigma], W, c, b, \sigma]$. Figure 6.3 shows that, surprisingly, score matching doesn't perform as well as CD or SML in the zero noise case, but as the noise increases it performs very similarly to CD.



(a) CD



(b) SML



(c) score matching

Figure 6.1: Receptive fields from Gaussian-binary RBMs trained on greyscale CIFAR-10 using various training methods.

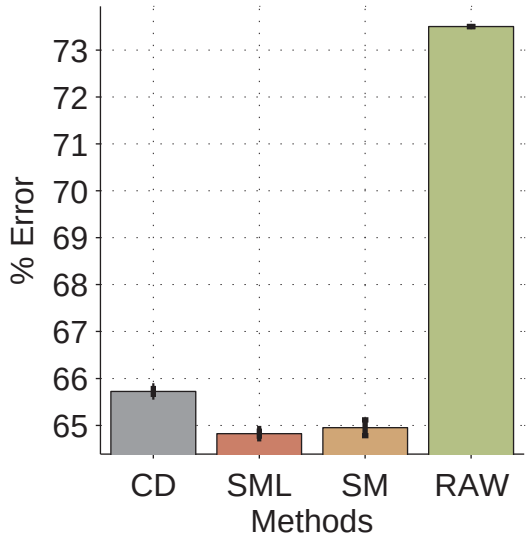


Figure 6.2: Test set classification error on greyscale CIFAR-10 from various Gaussian-binary RBM training methods using an SVM classifier on hidden activations. The method labelled "RAW" represents the application of the SVM classifier to the raw image pixels.

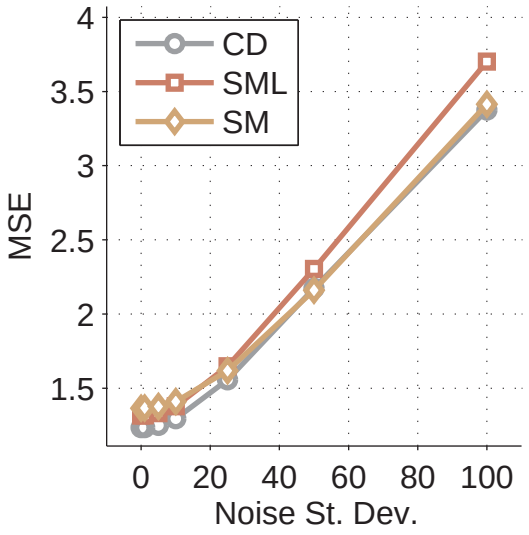


Figure 6.3: Test set denoising mean squared error as a function of the variance of zero-mean corrupting additive Gaussian noise.

6.4 Discussion

In this section, we performed a limited set of experiments designed to illustrate that score matching can be a viable option in for training Gaussian-binary RBMs. Score matching is nearly as computationally efficient as an auto-encoder, and is quite similar to CD and SML. More experiments should certainly be done to verify these results on a variety of datasets, and extensions to convolutional architectures as in [36], as well as parallel training on a graphics processing unit [54] should be considered.

Chapter 7

Deep Learning

We have spent a good deal of effort investigating the performance of different RBM training algorithms. One the most popular aspect of the RBM is that it can easily be used to provide a good starting point for training a deep belief network. In this chapter we will explore whether the performance of a training algorithm in the shallow case can affect its performance in the deep case. We will also briefly look at the effects of supervised fine-tuning in order to gain a better understanding of how it changes the model after unsupervised pre-training.

7.1 Comparing Algorithms for Training Deep Belief Networks

We trained a DBN on MNIST to compare how well the features learned by CD and SML would perform. The main question is: does optimizing the likelihood of an RBM lead to better classification performance? We used the best parameter settings based on the results in Chapter 4, and we trained a DBN consisting of 3 hidden layers with 500 hidden units in the first two layers, and 2000 units in the third (we can refer to this as a 784-500-500-2000 DBN), which is the architecture given in [19]. Each layer was trained for 50 iterations with Polyak averaging beginning at the 40^{th} iteration. Fine-tuning with backpropagation was applied for 30 iterations using Conjugate Gradient with 3 line searches on mini-batches of 1000 examples. The results are given in Table 7.1

Algorithm	Test Error (%)
CD	1.2
SML	1.13
CD weight decay 0.0001	1.12

Table 7.1: Test Error Rates based on greedily training a DBN with CD and SML, followed by supervised fine-tuning.

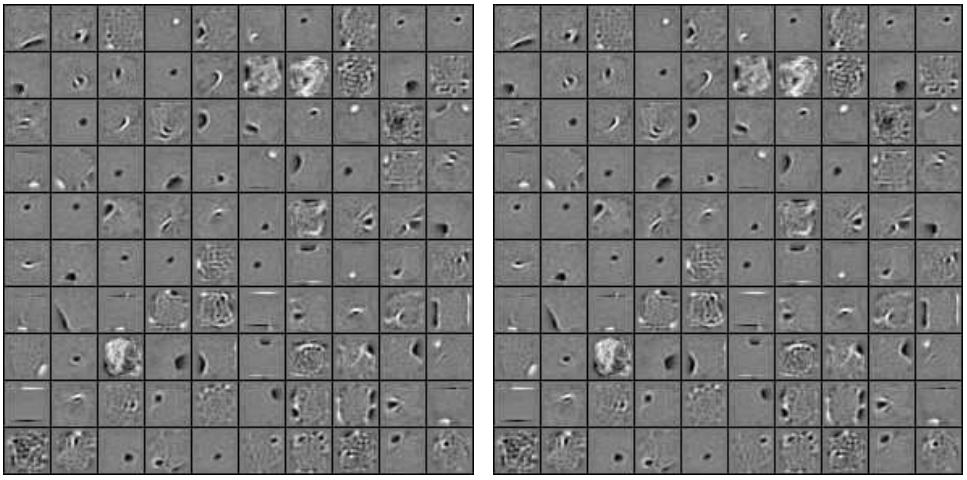
Algorithm	Test Error (%)
MLR original data	7.78
MLR augmented data	1.59
MLR top hidden layer only	1.67
DBN fine tuned	1.12

Table 7.2: The effect of features learned by a stack of RBMs, and by fine-tuning a DBN

Surprisingly, the parameters that worked best for the RBM don't necessarily work best when pre-training a DBN. In particular, lowering the weight decay for CD slightly lowers the test set error in the deep case, even though it raised test set error for the RBM. This indicates that perhaps test error is not the ideal metric to use when choosing the parameters of an RBM to initialize a DBN. Consistent with our results on MNIST from Figure 5.5 and Figure 5.6, it appears that optimizing the likelihood is not necessarily equivalent to learning good features, as CD and SML perform about the same in the DBN.

7.2 The Effects of Fine-Tuning

We now investigate the effect of fine-tuning on a DBN by greedily training a stack of RBMs with CD, and then comparing the difference in error before and after performing backpropagation. In order to assess the degree to which unsupervised learning of higher level features improves classification, and how much backpropagation helps, we augmented the original data with the higher level features obtained before backpropagation and trained a multinomial logistic regression (MLR) classifier. We also trained MLR classifiers on the original data, and on the features from the top hidden layer only. The results are shown in Table 7.2



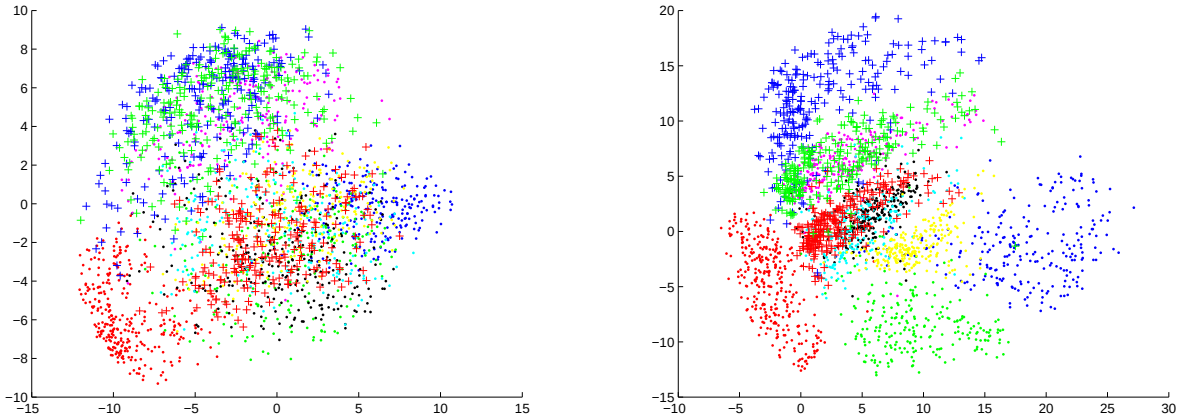
(a) Before fine-tuning **(b)** After fine-tuning

Figure 7.1: Comparison of the first layer weights of a DBN before and after performing fine-tuning with backpropagation.

Clearly the additional features improve classification significantly, and using more features including the original data seems to help. Supervised backpropagation changes the intermediate weights (and thus, features) to allow for greater linear separability in the top level of features, and gives the best results. Since most of the gain seems to come from the initial greedy learning, it may be possible to use these features directly with a nonlinear classifier such as support vector machines with a nonlinear kernel, or random forests, instead of using backpropagation. Such procedures may yield improved results, or at least they might be faster to train.

We can visually inspect the effects of fine-tuning to determine whether it drastically changes the features learned by the RBM. In particular, Figure 7.1 shows that the the changes in the first layer weights before and after backpropagation are hardly noticeable. This suggests that the weights learned by the RBM were already close to a local optimum.

While it appears that the weights have been perturbed only slightly, there is certainly some degree of change which is evident in the change in classification performance. A different way of visualizing this effect is to see what happens to the actual data in the last hidden layer before and after backpropagation. To do



(a) Before fine-tuning

(b) After fine-tuning

Figure 7.2: Comparison of 2500 Data points from MNIST in the 2-dimensional hidden layer of a deep auto-encoder before and after fine-tuning with backpropagation. Each point represents a different example, and each class is represented with a different colour/marker combination.

this, we appended class labels to the data and trained a 794-1000-500-250-2 DBN with Gaussian activations in the last hidden layer [19]. We turn this into an auto-encoder by "unrolling" the network after greedy training so that we end up with a 794-1000-500-250-2-250-500-1000-794 deep auto-encoder. The unrolled auto-encoder initially has symmetric weights (so $W^{(\ell_1)} = W^{(\ell_8)}$, etc.), but this constraint is not maintained during backpropagation. A data point is reconstructed by sending it through the auto-encoder to the 794-unit output layer. After initial greedy training, we applied 30 epochs of backpropagation to minimize the cross-entropy error between the reconstructed data, and the actual data. Note that we kept the class labels appended to the data to encourage the classes to further separate. In figure 7.2 we plot the 2-dimensional projection of each test example to the middle layer, where each class is given a separate colour/marker combination.

Amazingly, it appears that fine-tuning greatly contributes to enhancing class separation. We should note that there is one additional variable, however: the vi-

sualization was obtained by training an RBM with Gaussian hidden units, which is generally more difficult to train than an RBM with binary hidden units. Regardless, it is clear that fine-tuning can have a significant impact on the performance of a DBN, even though it does not significantly change the parameters.

7.3 Greedy Fine-Tuning

One issue with building a DBN is that we can never really be sure how many hidden layers to use. Adding new hidden layers is a slow, and costly process, and it is for this reason that it is worthwhile to explore ways to infer the number of layers before doing any sort of global fine-tuning.

In the case where we have labelled data and would like to perform classification, we can fine-tune each layer greedily after training its associated RBM. A completely greedy approach was used to train a document classifier in [56] with good results. If used in conjunction with a validation set, this would give a natural stopping point, as we could cease training when the error stops decreasing.

We experimented by training a DBN greedily, but also fine-tuning it immediately after, only focusing on the weights in that particular layer. We then continue training the rest of the DBN in this manner. Because we use unsupervised training first, we avoid being "too greedy", as in [4] where they train each layer as a neural network classifier without the unsupervised step. This sort of greedy tuning is another way of incorporating class information at each level during training, which might make classification easier at the higher levels, even without a global procedure.

We trained a 784-1000-1000-1000 network with this procedure since it seemed to require a larger number of units in the earlier layers. Our results indicated that the model is unable to improve beyond the first layer, and that the network is possibly overfitting very early. To test this, we first tried tuning the strength of the L2 penalty used in the training, but this did not improve results. Next, we tried injecting noise in the form of randomly zeroing out 10% of the features of each data point, as in [72]. Adding noise is a well known form of regularization that has been used in neural network training before [6]. Note that in all cases, we applied averaging during the fine-tuning, since even without added noise the training was

	Test Error (%)	
Layer	No Noise	Noise
1	1.23	1.20
2	1.22	1.14
3	1.25	1.11

Table 7.3: Error rates for a DBN of various sizes trained greedily with an RBM followed immediately by greedy fine-tuning.

performed on mini-batches.

From the results in Table 7.3, it appears that the overfitting hypothesis may be correct. After applying this greedy procedure, we checked to see if further improvements could be made by a final stage of global backpropagation, but this did not improve the results. This seems to indicate that it is possible to train a deep network in an entirely greedy fashion, and that each layer is able to increase the linear separability of the data. It would be interesting to see whether this approach can be used with respect to reconstruction error in a deep autoencoder.

Chapter 8

Conclusion and Future Work

In this thesis we explored training methods for the Restricted Boltzmann Machine. For stochastic training methods, we carefully investigated a number of factors involved in their learning procedures in order to improve the performance of the RBM. Key among these is the usage of iterate averaging, which showed significant improvement over the state of the art procedures in terms of speeding up the convergence of stochastic maximum likelihood. We derived a number of estimators for the RBM and tested their effectiveness over a range of tasks. We found that in the case of binary data, the stochastic methods are faster, but that there are cases such as denoising where ratio matching could be useful. We should note that in the case of Gaussian hidden units, the computational complexity of these methods all become similar, and it would be interesting to see a comparison of the different algorithms in this scenario. It would also be interesting to see the effects of sampling a subset of data dimensions in each iteration, rather than considering them all. This might be a useful way to greatly speed up the RBM learning methods, and if the performance isn't terribly affected, then it may be an interesting approach when the number of data dimensions is huge.

Although ratio matching appeared to be less efficient than pseudo-likelihood in our simulations, it generally performed quite a bit better than PL in several performance categories across several datasets. It would be interesting to see this estimator applied in other models where pseudo-likelihood is commonly used such as conditional random fields [27].

For modelling continuous data, we derived the score matching estimator for an RBM with Gaussian visible units, and binary hidden units. We showed that score matching is comparable to CD and SML in terms of task-related performance, and that its computational complexity is also comparable. This principle can also be applied to more complicated models, such as the higher order RBM found in [55]. In this case, direct sampling of the visible units given the hidden units is no longer computationally feasible, and so hybrid Monte-Carlo [48] is used. Score matching might be an effective way to bypass the need for sampling altogether, potentially making it much more computationally efficient.

We discussed the multilayer perceptron and the auto-encoder, and showed how applying score matching to a Gaussian-binary RBM directly yielded an auto-encoder with a particular kind of regularization function which penalizes the variance of the reconstructions. We further showed that this is quite close to a form of stochastic reconstruction error, which has previously been shown to be closely related to CD. This analysis further shows that optimizing for one-step reconstruction error can actually lead to a consistent estimator for the RBM, and provides a closer link between score matching and contrastive divergence. It also potentially provides a way to incorporate denoising directly into the training objective, as in [72]. This method has been shown to greatly improve performance in an auto-encoder, and it might do the same for an RBM or a deep belief network (DBN) trained with RBMs.

Finally, we looked at the issue of deep learning. We found that optimizing the likelihood of an RBM does not necessarily lead to the best performing features for classification in a DBN. We also found that fine-tuning can have a significant impact on the performance of a DBN, despite the fact that it appears to very minimally change the parameters learned by stacking RBMs. We also briefly investigated ways to learn the number of layers of a deep belief network. We suggested a procedure which involves injecting noise into the data in a similar manner to [72], and fine tuning each layer individually after training with an RBM. In this way, one can hopefully build the DBN in a completely greedy fashion.

One major issue we did not discuss was determining the number of hidden units in an RBM. At this point in time, the most effective method is to simply test a few different values, but this can be labour, as well as computationally intensive.

Adding more hidden units can help, but it also makes learning more difficult, as well as slower. Another issue is the sensitivity of the RBM to weight decay, as performance can vary greatly depending on how this is set, and what task is being considered. Bayesian methods could certainly be useful in this instance, but these bring a whole new set of challenges.

Finally, another area that hasn't been explored here are sparsity inducing priors. It has often been found that encouraging sparse activations in the hidden units improves performance, especially when the model is overcomplete (more hidden units than visible units). Adding a prior over the hidden unit activations can help to encourage sparsity, however there are many ways in which this can be done; some examples are given in [26, 32, 35, 46]. It would be useful to have a comparison of these methods to determine their effects, and to quantify the benefits that sparsity might bring. An example of work along this line is given in [16].

While the RBM is actually a fairly simple model, it is obvious that there are many challenges inherent in learning it, and understanding its behaviour. This thesis should serve as a small step toward overcoming those challenges.

Bibliography

[1] E. Allgower and K. Georg. *Numerical continuation methods*. Springer, 1990. → pages 52

[2] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. → pages 42, 43

[3] Y. Bengio and O. Delalleau. Justifying and generalizing contrastive divergence. *Neural Computation*, 21(6):1601–1621, 2009. → pages 17, 18, 32, 39

[4] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, and U. Montreal. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*. MIT Press, 2007. → pages 84

[5] J. Besag. Statistical analysis of non-lattice data. *The Statistician*, 24(3): 179–195, 1975. → pages 20

[6] C. M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural computation*, 7(1):108–116, 1995. → pages 84

[7] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006. ISBN 0387310738. → pages 50

[8] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20, 2007. → pages 49

[9] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4):291–294, 1988. → pages 38

[10] M. Carreira-Perpinan and G. Hinton. On contrastive divergence learning. In *Artificial Intelligence and Statistics*, volume 2005, 2005. → pages 17, 18

[11] G. Desjardins, A. Courville, Y. Bengio, P. Vincent, and O. Delalleau. Tempered markov chain monte carlo for training of restricted boltzmann machines. Technical Report 1345, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, 2009. → pages 60

[12] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. Technical Report 1341, University of Montreal, 2009. → pages 76

[13] D. Erhan, P. Manzagol, Y. Bengio, S. Bengio, and P. Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. *AISTATS*, 2009. → pages 43

[14] S. Geman, D. Geman, K. Abend, T. Harley, and L. Kanal. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images*. *Journal of Applied Statistics*, 20(5):25–62, 1993. → pages 15

[15] C. Geyer. Markov chain Monte Carlo maximum likelihood, 1992. → pages 16

[16] I. Goodfellow, Q. Le, A. Saxe, and A. Ng. Measuring invariances in deep networks. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 646–654. 2009. → pages 88

[17] G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14, 2000. → pages 16, 17

[18] G. Hinton. To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547, 2007. → pages 72

[19] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. → pages 9, 32, 80, 83

[20] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. → pages 43

[21] C.-N. Hsu, Y.-M. Chang, H. Huang, and Y.-J. Lee. Periodic step size adaptation for single pass on-line learning. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 763–771. 2009. → pages 50

[22] A. Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709, 2005. → pages 26, 27, 40

[23] A. Hyvärinen. Connections between score matching, contrastive divergence, and pseudolikelihood for continuous-valued variables. *IEEE Transactions on Neural Networks*, 18(5):1529–1531, 2007. → pages 29

[24] A. Hyvärinen. Some extensions of score matching. *Computational Statistics & Data Analysis*, 51(5):2499–2512, 2007. → pages 22, 23

[25] N. Japkowicz, S. Hanson, and M. Gluck. Nonlinear autoassociation is not equivalent to PCA. *Neural Computation*, 12(3):531–545, 2000. → pages 39, 40

[26] H. J. Kappen. Deterministic learning rules for Boltzmann machines. *Neural Networks*, 8(4):537–548, 1995. → pages 88

[27] S. Kumar and M. Hebert. Discriminative random fields: A discriminative framework for contextual interaction in classification. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, pages 1150–1157. IEEE Computer Society, 2003. → pages 86

[28] H. J. Kushner and H. Huang. Averaging methods for the asymptotic analysis of learning and adaptive systems, with small adjustment rate. *SIAM J. Control Optim.*, 19(5):635–650, 1981. → pages 47

[29] H. J. Kushner and H. Huang. Asymptotic properties of stochastic approximations with constant coefficients. *SIAM J. Control Optim.*, 19(1): 87–105, 1981. → pages 47

[30] H. J. Kushner and J. Yang. Stochastic approximation with averaging and feedback: rapidly convergent "on-line" algorithms. *Institute for Mathematics and Its Applications*, 40(1):24–34, 1995. → pages 10, 56

[31] H. J. Kushner and G. G. Yin. *Stochastic Approximation Algorithms and Applications*. Springer-Verlag, 1997. → pages 15, 47

[32] H. Larochelle and Y. Bengio. Classification using discriminative restricted Boltzmann machines. In *International Conference on Machine learning*, pages 536–543, 2008. → pages 88

[33] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring Strategies for Training Deep Neural Networks. *Journal of Machine Learning Research*, 1:1–40, 2009. → pages 51

[34] N. Le Roux and Y. Bengio. Deep Belief Networks are Compact Universal Approximators. *To appear in Neural Computation*, 2010. → pages 42

[35] H. Lee, C. Ekanadham, and A. Ng. Sparse deep belief net model for visual area V2. *Advances in neural information processing systems*, 20, 2008. → pages 72, 88

[36] H. Lee, R. Grosse, R. Ranganath, and A. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *International Conference on Machine Learning*, 2009. → pages 79

[37] Y. Lee and O. Mangasarian. SSVM: A smooth support vector machine for classification. *Computational optimization and Applications*, 20(1):5–22, 2001. → pages 67

[38] S. P. Luttrell. Using stochastic encoders to discover structure in data. *CoRR*, cs.NE/0408049, 2004. → pages 39

[39] S. Lyu. Interpretation and generalization of score matching. In *Uncertainty in Artificial Intelligence 25*, 2009. → pages 24, 29, 30

[40] D. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. → pages 14

[41] B. M. Marlin. A direct proof that the true RBM parameters are a fixed point of both ML and CD1 in the asymptotic setting. 2008. → pages 18

[42] B. M. Marlin., K. Swersky, B. Chen., and N. de Freitas. Inductive principles for Restricted Boltzmann Machine learning. *AISTATS*, 2009. → pages 11, 61

[43] X. Miao and R. Rao. Large margin Boltzmann machines. In *Proceedings of the 21st international joint conference on Artifical intelligence*, pages 1156–1162, 2009. → pages 31

[44] I. Murray and Z. Ghahramani. Bayesian learning in undirected graphical models: approximate MCMC algorithms. In *Uncertainty in Artificial Intelligence 20*, pages 392–399, 2004. → pages 31

[45] I. Murray and R. Salakhutdinov. Evaluating probabilities under high-dimensional latent variable models. In *Advances in Neural Information Processing Systems 21*, pages 1137–1144, 2009. → pages 68

[46] V. Nair and G. Hinton. 3d object recognition with deep belief nets. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1339–1347. 2009. → pages 88

[47] R. Neal. *Probabilistic inference using Markov chain Monte Carlo methods*. Citeseer, 1993. → pages 29

[48] R. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, 1996. ISBN 0387947248. → pages 87

[49] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, Second edition, 2000. → pages 20

[50] B. T. Polyak. Some methods of speeding up the convergence of iterative methods. *USSR Computational Mathematics and Mathematical Physics.*, 4: 1–17, 1964. → pages 47

[51] B. T. Polyak. A new method of stochastic approximation type. *Avtomat. i Telemekh.*, (7):98–107, 1990. → pages 47

[52] B. T. Polyak and A. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30:838, 1992. → pages 10, 47

[53] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng. Self-taught learning: transfer learning from unlabeled data. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 759–766. ACM, 2007. ISBN 9781595937933. → pages 43

[54] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 873–880. ACM, 2009. ISBN 978-1-60558-516-1. → pages 79

[55] M. Ranzato and G. Hinton. Factored 3-Way Restricted Boltzmann Machines for modeling natural images. *AISTATS*, 2009. → pages 87

[56] M. Ranzato and M. Szummer. Semi-supervised learning of compact document representations with deep networks. *ICML*, 2008. → pages 84

[57] M. Ranzato, Y.-L. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems 19*. MIT Press, 2007. → pages 72

[58] M. A. Ranzato and M. Szummer. Semi-supervised learning of compact document representations with deep networks. In *ICML*, pages 792–799. ACM, 2008. → pages 3

[59] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22:400–407, 1951. → pages 46

[60] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951. → pages 16

[61] R. Salakhutdinov. Learning and evaluating Boltzmann machines. Technical Report UTML TR 2008-002, Department of Computer Science, University of Toronto, June 2008. → pages 31

[62] R. Salakhutdinov. Learning in markov random fields using tempered transitions. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1598–1606. 2009. → pages 60

[63] R. Salakhutdinov and G. Hinton. Training a deep autoencoder or a classifier on MNIST digits — source code, 2006. → pages 50, 52

[64] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 2008. → pages 3

[65] R. Salakhutdinov and G. Hinton. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455, 2009. → pages 3

[66] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann machines for collaborative filtering. In *International Conference on Machine learning*, pages 791–798, 2007. → pages 9, 46

[67] N. Schraudolph, J. Yu, and S. Gunter. A Stochastic Quasi-Newton Method for Online Convex Optimization. *Proc. 11th Intl. Conf. Artificial Intelligence and Statistics (AIstats), San Juan, Puerto Rico, March*, 2007. → pages 60

[68] J. Sohl-Dickstein, P. Battaglino, and M. R. DeWeese. Minimum probability flow learning. *CoRR*, abs/0906.4779, 2009. → pages 31

[69] T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *International conference on Machine Learning*, pages 1064–1071, 2008. → pages 19, 55

[70] T. Tieleman and G. Hinton. Using Fast Weights to Improve Persistent Contrastive Divergence. In *Proceedings of the 26th international conference on Machine learning*, pages 1033–1040. ACM, 2009. → pages 55, 58

[71] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995. ISBN 0387945598. → pages 43, 67

[72] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine learning*, pages 1096–1103, 2008. → pages 84, 87

[73] S. V. N. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 969–976. ACM, 2006. → pages 49

[74] L. Wasserman. *All of Statistics: a Concise Course in Statistical Inference*. Springer, 2004. → pages 12

[75] M. Welling, M. Rosen-Zvi, and G. Hinton. Exponential family harmoniums with an application to information retrieval. *Advances in neural information processing systems*, 17:1481–1488, 2005. → pages 3

[76] L. Younes. Parametric inference for imperfectly observed Gibbsian fields. *Probability Theory and Related Fields*, 82(4):625–645, 1989. → pages 18

[77] A. Yuille. The convergence of contrastive divergences. In *Advances in Neural Information Processing Systems*, 2004. → pages 18