(DRAFT)

Advances in Deep Learning

Kyunghyun Cho

July 17, 2013

Abstract

Deep neural networks have become increasingly more popular under the name of deep learning recently due to their success in challenging machine learning tasks. Although the popularity is mainly due to the recent successes, the history of neural networks goes as far back as 1958 when Rosenblatt presented a perceptron learning algorithm. Since then, various kinds of artificial neural networks have been proposed. They include Hopfield network, self-organizing maps, neural principal component analysis, Boltzmann machines, multi-layer perceptrons, radial-basis function networks, autoencoders, sigmoid belief network, support vector machines and deep belief networks.

In the first part of this thesis, the author aims at investigating these models and finding a common set of basic principles for deep neural networks. The thesis starts from some of the earlier ideas and models in the field of artificial neural networks and arrive at autoencoders and Boltzmann machines which are two most widely studied neural networks these days. The author thoroughly discusses how those various neural networks are related to each other and how the principles behind those networks form foundation for autoencoders and Boltzmann machines.

The second part is the collection of the ten recent publications by the author. These publications mainly focus on learning and inference algorithms of Boltzmann machines and autoencoders. Especially, Boltzmann machines which are known to be difficult to train have been in the main focus. Throughout several publications the author and the co-authors have devised and proposed a new set of learning algorithms which includes the enhanced gradient, adaptive learning rate and parallel tempering. These algorithms are further applied to a restricted Boltzmann machine with Gaussian visible units.

In addition to these algorithms for restricted Boltzmann machines the author proposed a two-stage pretraining algorithm that initializes the parameters of a deep Boltzmann machine to match the variational posterior distribution of a similarly structured deep autoencoder. Finally, deep neural networks are applied to image denoising and speech recognition.

Preface

This work has been carried out in the Department of Information and Computer Science at Aalto University School of Science under the supervision of Prof. Juha Karhunen, Dr. Tapani Raiko and Dr. Alexander Ilin while being fully funded by the Finnish Doctoral Programme in Computational Sciences (FICS).

Espoo, July 17, 2013,

Kyunghyun Cho

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Kyunghyun Cho, Tapani Raiko and Alexander Ilin. Enhanced Gradient for Training Restricted Boltzmann Machines. *Neural Computation*, Volume 25 Issue 3 Pages 805–831, March 2013.

II Kyunghyun Cho, Tapani Raiko and Alexander Ilin. Enhanced Gradient and Adaptive Learning Rate for Training Restricted Boltzmann Machines. In *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 105–112, June 2011.

III Kyunghyun Cho, Tapani Raiko and Alexander Ilin. Parallel Tempering is Efficient for Learning Restricted Boltzmann Machines. In *Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN 2010)*, pages 1–8, July 2010.

IV Kyunghyun Cho, Alexander Ilin and Tapani Raiko. Tikhonov-Type Regularization for Restricted Boltzmann Machines. In *Proceedings of the 22nd International Conference on Artificial Neural Networks (ICANN 2012)*, pages 81–88, September 2012.

V Kyunghyun Cho, Alexander Ilin and Tapani Raiko. Improved Learning of Gaussian-Bernoulli Restricted Boltzmann Machines. In *Proceedings of the 21st International Conference on Artificial Neural Networks (ICANN 2011)*, pages 10–17, June 2011.

VI Kyunghyun Cho, Tapani Raiko and Alexander Ilin. Gaussian-Bernoulli Deep Boltzmann Machines. In *Proceedings of the 2013 International Joint Conference on Neural Networks (IJCNN 2013)*, August 2013.

VII Kyunghyun Cho, Tapani Raiko, Alexander Ilin and Juha Karhunen. A Two-Stage Pretraining Algorithm for Deep Boltzmann Machines. In *Proceedings of the 23rd International Conference on Artificial Neural Networks (ICANN 2013)*, September 2013.

VIII Kyunghyun Cho. Simple Sparsification Improves Sparse Denoising Autoencoders in Denoising Highly Corrupted Images. In *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, 432–440, June 2013.

IX Kyunghyun Cho. Boltzmann Machines for Image Denoising. In *Proceedings of the 23rd International Conference on Artificial Neural Networks (ICANN 2013)*, September 2013.

X Sami Keronen, Kyunghyun Cho, Tapani Raiko, Alexander Ilin and Kalle Palomäki. Gaussian-Bernoulli Restricted Boltzmann Machines and Automatic Feature Extraction for Noise Robust Missing Data Mask Estimation. In *Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2013)*, May 2013.

List of Abbreviations

BM	Boltzmann machine
CD	Contrastive divergence
DBM	Deep Boltzmann machine
DBN	Deep belief network
DEM	Deep energy model
ELM	Extreme learning machine
EM	Expectation-Maximization
GDBM	Gaussian-Bernoulli deep Boltzmann machine
GP	Gaussian Process
GP-LVM	Gaussian process latent variable model
GRBM	Gaussian-Bernoulli restricted Boltzmann machine
ICA	Independent component analysis
KL	Kullback-Leibler divergence
lasso	Least absolute shrinkage and selection operator
MAP	Maximum-a-posteriori estimation
MCMC	Markov Chain Monte Carlo
MLP	Multi-layer perceptron
MoG	Mixture of Gaussians
MRF	Markov random field
OMP	Orthogonal matching pursuit
PCA	Principal component analysis
PoE	Product of Experts
PSD	Predictive sparse decomposition
RBM	Restricted Boltzmann machine
SESM	Sparse encoding symmetric machine
SVM	Support vector machine
XOR	Exclusive-OR

Mathematical Notation

As the author has tried to make mathematical notations consistent throughout this thesis, in some parts they may look different from how they are used commonly in the original research literature. Before entering the main text of the thesis, the author would like to declare and clarify the mathematical notations which will be used repeatedly.

Variables and Parameters

A vector, which is always assumed to be a column vector, is mostly denoted by a bold, lower-case Roman letter such as $\mathbf{x}$, and a matrix by a bold, upper-case Roman letter such as $\mathbf{W}$. Two important exceptions are $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$ which denote a vector of parameters and a vector of variational parameters, respectively.

A component of a vector is denoted by a (non-bold) lower-case Roman letter with the index of the component as a subscript. Similarly, an element of a matrix is denoted by a (non-bold) lower-case Roman letter with a pair of the indices of the component as a subscript. For instance, x_i and w_{ij} indicate the i-th component of $\mathbf{x}$ and the element of $\mathbf{W}$ on its i-th row and j-th column, respectively.

Lower-case Greek letters are used, in most cases, to denote scalar variables and parameters. For instance, η, λ and σ mean learning rate, regularization constant and standard deviation, respectively.

Functions

Regardless of the type of its output, all functions are denoted by non-bold letters. In the case of vector functions, the dimensions of the input and output will be explicitly explained in the text, unless they are obvious from the context. Similarly to a vector notation, a subscript may be used to denote a component of a vector function such that $f_i(\mathbf{x})$ is the i-th component of a vector function f.

Some commonly used functions include a component-wise nonlinear activation function ϕ, a stochastic noise operator κ, an encoder function f, and a decoder function g.

A component-wise nonlinear activation function ϕ is used for different types of activation functions depending on the context. For instance, ϕ is a Heaviside function (see Eq. (2.5)) when used in a Hopfield network, but is a logistic sigmoid function (see Eq. (2.7)) in the case of Boltzmann machines. However, there should be no confusion, as its definition will always be explicitly given at each usage.

Probability and Distribution

A probability density/mass function is often denoted by p or P and the corresponding unnormalized probability by p^* or P^*. By dividing p^* by the normalization constant Z, one recovers p. Additionally, q or Q are often used to denote a (approximate) posterior distribution over hidden or latent variables.

An expectation of a function $f(\mathbf{x})$ over a distribution p is denoted either by $\mathbb{E}_p\left[f(\mathbf{x})\right]$ or by $\left\langle f(\mathbf{x})\right\rangle_p$. A cross-covariance of two random vectors $\mathbf{x}$ and $\mathbf{y}$ over probability density p is often denoted by $\mathrm{Cov}_p(\mathbf{x}, \mathbf{y})$. $\mathrm{KL}\left(Q\|P\right)$ means a Kullback-Leibler divergence (see Eq. (2.26)) between distributions Q and P.

Two important types of distributions that will be used throughout this thesis are data distribution and model distribution. The data distribution is the distribution from which training samples are sampled, and the model distribution is the one that is represented by a machine learning model. For instance, a Boltzmann machine defines a distribution over all possible states of visible units, and that distribution is referred to as the model distribution.

The data distribution is denoted by either d, p_D or P_0, and the model distribution by either m, p or P_∞. Reasons for using different notations for the same distribution will be made clear throughout the text.

Superscripts and Subscripts

In machine learning, it is usually either explicitly or implicitly assumed that a set of training samples are given. N is often used to denote the size of the training set, and each sample is denoted by its index in the super- or subscript such that $\mathbf{x}^{(n)}$ is the n-th training sample. However, as it is a *set*, it should be understood that the order of the elements is arbitrary.

In a neural network, units or parameters are often divided into multiple layers.

Then we use either a superscript or subscript to indicate the layer to which each unit or a vector of units belongs. For instance, $\mathbf{h}^{[l]}$ and $\mathbf{W}^{[l]}$ are respectively a vector of (hidden) units and a matrix of weight parameters in the l-th layer. Whenever it is necessary to make an equation less cluttered, $\mathbf{h}^{[l]}$ (superscript) and $\mathbf{h}_{[l]}$ (subscript) may be used interchangeably.

Occasionally, there appears an ordered sequence of variables or parameters. In that case, a super- or subscript $\langle t \rangle$ is used to denote the temporal index of a variable. For example, both $\mathbf{x}^{\langle t \rangle}$ and $\mathbf{x}_{\langle t \rangle}$ mean the t-th vector $\mathbf{x}$ or the value of a vector $\mathbf{x}$ at time t.

The latter two notations $[l]$ and $\langle t \rangle$ apply also to functions as well as probability density/mass functions. For instance, $f^{[l]}$ is an encoder function that projects units in the l-th layer to the $(l+1)$-th layer. In the context of Markov Chain Monte Carlo sampling, $p^{\langle t \rangle}$ denotes a probability distribution over the states of a Markov chain after t steps of simulation.

In many cases, $\boldsymbol{\theta}^*$ and $\hat{\boldsymbol{\theta}}$ denote an unknown optimal value and a value estimated by, say, an optimization algorithm, respectively. However, one should be aware that these notations are not strictly followed in some parts of the text. For example, $\mathbf{x}^*$ may be used to denote a novel, unseen sample other than training samples.

1. Introduction

1.1 Aim of this Thesis

A research field, called *deep learning*, has gained its popularity recently as a way of learning deep, hierarchical artificial neural networks (see, for example, Bengio, 2009). Especially, deep neural networks such as a deep belief network (Hinton et al., 2006), deep Boltzmann machine (Salakhutdinov and Hinton, 2009a), stacked denoising autoencoders (Vincent et al., 2010) and many other variants have been applied to various machine learning tasks with impressive improvements over conventional approaches. For instance, Krizhevsky et al. (2012) significantly outperformed all other conventional methods in classifying a huge set of large images. Speech recognition also benefited significantly by using a deep neural network recently (Hinton et al., 2012). Also, many other tasks such as traffic sign classification (Ciresan et al., 2012c) have been shown to benefit from using a large, deep neural network.

Although the recent surge of popularity stems from the breakthrough, involving a layer-wise pretraining, in 2006 (Hinton and Salakhutdinov, 2006; Bengio et al., 2007; Ranzato et al., 2007b), research on artificial neural networks in general has begun as early as 1958 when Rosenblatt (1958) proposed a perceptron learning algorithm. Since then, various kinds of artificial neural networks have been proposed. They include, but are not limited to Hopfield networks (Hopfield, 1982), self-organizing maps (Kohonen, 1982), neural networks for principal component analysis (Oja, 1982), Boltzmann machines (Ackley et al., 1985), multi-layer perceptrons (Rumelhart et al., 1986), radial-basis function networks (Broomhead and Lowe, 1988), autoencoders (Baldi and Hornik, 1989), sigmoid belief networks (Neal, 1992) and support vector machines (Cortes and Vapnik, 1995).

These types of artificial neural networks are interesting not only on their own, but by connections among themselves and with other machine learning approaches. For instance, principal component analysis (PCA) which may be considered a linear alge-

braic method, arises also from an unsupervised neural network with Oja's rule (Oja, 1982), and at the same time, can be recovered from a latent variable model (Tipping and Bishop, 1999; Roweis, 1998). Also, the solution of a linear autoencoder with a single hidden layer corresponds exactly to the solution of PCA. PCA can be further generalized to nonlinear PCA through, for instance, an autoencoder with multiple nonlinear hidden layers (Kramer, 1991).

Due to the recent popularity of deep learning, two of the most widely studied artificial neural networks these days are autoencoders and Boltzmann machines. An autoencoder with a single hidden layer as well as a structurally restricted version of the Boltzmann machine, called a restricted Boltzmann machine, have become popular due to their application in layer-wise pretraining of deep multi-layer perceptrons.

Thus, this thesis starts from some of the earlier ideas in the artificial neural networks and arrive at those two currently popular models. In due course, the author will try to explain how various types of artificial neural networks are related to each other, ultimately leading to autoencoders and Boltzmann machines. Furthermore, this thesis will include underlying methods and concepts that have led to those two models' popularity, which include, for instance, layer-wise pretraining and manifold learning. Whenever it is possible, informal mathematical justification for each model or method is provided alongside.

Since the main focus of this thesis is on *general* principles of deep neural networks, the thesis tries to avoid describing any method that is specific to a certain task. In other words, the explanations as well as the models in this thesis assume no prior knowledge about data, except that each sample is independent and identically distributed and that its length is fixed.

Ultimately, the author hopes that the reader, even without much background in deep learning, will understand the basic principles and concepts of deep neural networks.

1.2 Outline

This dissertation aims to provide an introduction to deep neural networks throughout which the author's contributions are placed. Starting from simple neural networks that were introduced as early as 1958, we gradually move toward the recent advances in deep neural networks.

For clarity, contributions that have been proposed and presented by the author are emphasized with bold-face. A separate list of author's contributions is given in Section 1.3.

1.2.1 Shallow Neural Networks

In Chapter 2, the author tries to give a background on neural networks that are considered shallow. By shallow neural networks we refer, in the case of supervised models, to those neural networks that have only input and output units. No intermediate hidden units are considered. A *linear regression* network and *perceptron* are described as representative examples of supervised, shallow neural networks in Section 2.1.

Unsupervised neural networks are considered shallow when either there are no hidden units or there are only linear hidden units. A *Hopfield network* is one example of having no hidden units, and a *linear autoencoder*, or equivalently principal component analysis, is an example of having linear hidden units only. Both of them are briefly described in Section 2.2.

All these shallow neural networks are then in Section 2.3 further described in relation with probabilistic models. From this probabilistic perspective, forward computations in neural networks are interpreted as computing the conditional probability of other units given an input sample. In supervised neural networks, these forward computations correspond to computing the conditional probability of output variables, while in unsupervised neural networks, they are shown to be equivalent to inferring the posterior distribution of hidden units under certain assumptions.

Based on this preliminary knowledge on shallow neural networks, the author tries to loosely define *two conditions* which must be satisfied by a neural network to be considered *deep* in Section 2.4. These conditions will be evaluated for each deep neural network introduced in later chapters. The chapter ends by briefly describing how the parameters of a neural network can be efficiently estimated by the stochastic gradient method.

1.2.2 Deep Feedforward Neural Networks

The first family of deep neural networks is introduced and discussed in detail in Chapter 3. This family consists of feedforward neural networks that have multiple layers of nonlinear hidden units. A *multi-layer perceptron* is introduced, and two related, but not-so-deep feedforward neural networks, a kernel *support vector machine* and an *extreme learning machine* are briefly discussed in Section 3.1.

The remaining part of the chapter begins by describing *deep autoencoders*. With its basic description, a probabilistic interpretation of the encoder and decoder of a deep autoencoder is provided in connection with a *sigmoid belief network* and its learning algorithm called wake-sleep algorithm in Section 3.2.1. This allows one to view the encoder and decoder as inferring an approximate posterior distribution and computing a conditional distribution. Under this view, a related approach called

sparse coding is discussed, and an ***explicit sparsification***, proposed by the author in Publication VIII, for a sparse deep autoencoder is introduced in Section 3.2.5.

Another view of an autoencoder is provided afterward based on the manifold assumption in Section 3.3. In this view, it is explained how some variants of autoencoders such as a *denoising autoencoder* and a *contractive autoencoder* are able to capture the manifold on which data lies.

A universal learning algorithm, called *backpropagation*, for estimating the parameters of a feedforward neural network is presented in Section 3.4. After a brief description of the algorithm, the section further discusses the difficulty of training deep feedforward neural networks with the backpropagation by introducing some of the hypotheses proposed recently. Furthermore, for each hypothesis, a potential remedy is described.

1.2.3 Boltzmann Machines with Hidden Units

The second family of deep neural networks considered in this dissertation consists of a Boltzmann machine and its structurally restricted variants. The author classifies the Boltzmann machines as deep neural networks based on the observation that any *recurrent neural network* with nonlinear hidden units is deep, satisfying the conditions defined earlier.

The chapter proceeds by describing a *general Boltzmann machine* of which all units, regardless of their types, are fully connected by undirected edges in Section 4.1. One important consequence of formulating the probability distribution of a Boltzmann machine with a Boltzmann distribution (see Section 2.3.2) is that an equivalent Boltzmann machine can always be constructed when the variables or units are transformed with, for instance, a bit-flipping transformation. Based on this, in Section 4.1.1 the ***enhanced gradient*** which was proposed by the author in Publication I is introduced.

In Section 4.3, three basic estimation principles needed to train a Boltzmann machine are introduced. They are *Markov Chain Monte Carlo sampling*, *variational approximation*, and *stochastic approximation procedure*. An advanced sampling method, called ***parallel tempering***, whose use for training variants of Boltzmann machines was proposed in Publication III, Publication V and Publication VI for training variants of Boltzmann machines, is described further in Section 4.3.1.

The remaining part of this chapter concentrates on more widely used variants of Boltzmann machines. In Section 4.4.1, an underlying mechanism based on the conditional independence property of a Markov random field is explained that justifies restricting the structure of a Boltzmann machine. Based on this mechanism, a *restricted Boltzmann machine* and *deep Boltzmann machine* are explained in Section 4.4.2–

4.4.3.

After describing the restricted Boltzmann machine in Section 4.4.2, the author discusses the connection between a product of experts and the restricted Boltzmann machine. This connection further leads to the learning principle of minimizing *contrastive divergence* which is based on constructing a sequence of distributions using Gibbs sampling.

At the end of this chapter, in Section 4.5, the author discusses the connections between the autoencoder and the Boltzmann machine found earlier by other researchers. The close equivalence between the restricted Boltzmann machine and the autoencoder with a single hidden layer is described in Section 4.5.1. In due course, a ***Gaussian-Bernoulli restricted Boltzmann machine*** is discussed with its **modified energy function** proposed in Publication V. A *deep belief network* is subsequently discussed as a composite model of a restricted Boltzmann machine and a stochastic deep autoencoder in Section 4.5.2.

1.2.4 Unsupervised Neural Networks as the First Step

The last chapter before the conclusion deals with an important concept of pretraining, or initializing another potentially more complex neural network with unsupervised neural networks. This is first motivated by the difficulty of training a deep multi-layer perceptron using the plain backpropagation discussed in Section 3.4.1.

The first section (Section 5.1) describes stacking multiple layers of unsupervised neural networks with a single hidden layer to initialize a multi-layer perceptron, called *layer-wise pretraining*. This method is motivated in the framework of incrementally, or recursively, transforming the coordinates of input samples to obtain better representations. In this framework, several alternative building blocks are introduced in Sections 5.1.1–6.3.5.

In Section 5.2, we describe how the unsupervised neural networks such as Boltzmann machines and deep belief networks can be used for discriminative tasks. A direct method of learning a joint distribution between an input and output is introduced in Section 5.2.1. A *discriminative restricted Boltzmann machine* and a deep belief network with the top pair of layers augmented with labels are described. A non-trivial method of initializing a multi-layer perceptron with a deep Boltzmann machine is further explained in Section 5.2.2.

The author wraps up the chapter by describing in detail how more complex generative models, such as deep belief networks and deep Boltzmann machines, can be initialized with simpler models such as restricted Boltzmann machines in Section 5.3. Another perspective based on maximizing variational lower bound is introduced to motivate pretraining a deep belief network by stacking multiple layers of restricted

Boltzmann machines in Section 5.3.1–5.3.2. Section 5.3.3 explains two pretraining algorithms for deep Boltzmann machines. The second algorithm, called the ***two-stage pretraining algorithm***, was proposed by the author in Publication VII.

1.2.5 Discussion

The author finishes the thesis by summarizing the current status of academic research and commercial applications of deep neural networks. Also, the overall content of this thesis is summarized. This is immediately followed by five subsections that discuss some topics that have *not* been discussed in, but are relevant to this thesis.

The field of deep neural networks, or deep learning, is expanding rapidly, and it is impossible to discuss everything in this thesis. Multi-layer perceptrons, autoencoders and Boltzmann machines, which are main topics of this thesis, are certainly not the only neural networks in the field of deep neural networks. However, as the aim of this thesis is to provide a brief overview of and introduction to deep neural networks, the author intentionally omitted some models, even though they are highly related to the neural networks discussed in this thesis. One of those models is independent component analysis (ICA), and the author provides a list of references that present the relationship between the ICA and the deep neural networks in Section 6.3.1.

Although deep neural networks have shown extremely competitive performance in various machine learning tasks, the theoretical motivation for using them is still debated. At least, one well-founded theoretical property of most of deep neural networks discussed in this thesis is the universal approximator property, stating that a model with this property can approximate the target function, or distribution, with arbitrarily small error. In Section 6.3.2, the author provides the references to some earlier works that proved or described this property of various deep neural networks.

Compared to the feedforward neural networks such as autoencoders and multi-layer perceptrons, it is difficult to evaluate Boltzmann machines. Even when the structure of the network is highly restricted, the existence of the intractable normalization constant requires using a sophisticated sampling-based estimation method to evaluate Boltzmann machines. In Section 6.3.3, the author tries to point out some of the recent advances in evaluating Boltzmann machines.

The chapter ends by presenting recently proposed solutions to two practical matters concerning training and building deep neural networks. First, a recently proposed method of hyper-parameter optimization is briefly described, which relies on Bayesian optimization. Second, a standard approach to building a deep neural network that explicitly exploits spatial structure of data is presented.

1.3 Author's Contributions

This thesis contains ten publications that are closely related to and based on the basic principles of deep neural networks. This section lists for each publication the author's contribution.

In **Publication I**, **Publication II**, **Publication III** and **Publication IV**, the author extensively studied learning algorithms for restricted Boltzmann machines (RBM) with binary units. By investigating potential difficulties of training RBMs, the author together with the co-authors of Publication I and Publication II designed a novel update direction called enhanced gradient, that utilizes the transformation invariance of Boltzmann machines (see Section 4.1.1). Furthermore, to alleviate selecting the right learning rate scheduling, the author proposed an adaptive learning rate algorithm based on maximizing the locally estimated likelihood that can adapt the learning rate on-the-fly (see Section 6.3.3), in Publication II. In Publication III, parallel tempering which is an advanced Markov Chain Monte Carlo sampling algorithm, was applied to estimating the statistics of the model distribution of an RBM (see Section 4.3.1). Additionally, the author proposed and tested empirically novel regularization terms for RBMs that were motivated by the contractive regularization term recently proposed for autoencoders (see Section 3.3).

The author further applied these novel algorithms and approaches, including the enhanced gradient, the adaptive learning and the parallel tempering to Gaussian-Bernoulli RBMs (GRBM) which employ Gaussian visible units in place of binary visible units in **Publication V**. In this work, those approaches as well as a modified form of the energy function (see Section 4.5.1) were empirically found to facilitate estimating the parameters of a GRBM. These novel approaches were further applied to a more complex model, called a Gaussian-Bernoulli deep Boltzmann machine (GDBM), in **Publication VI**.

In **Publication VII**, the author proposed a novel two-stage pretraining algorithm for deep Boltzmann machines (DBM) based on the fact that the encoder of a deep autoencoder performs approximate inference of hidden units (see Section 5.3.3). A deep autoencoder trained during the first stage is used as an approximate posterior distribution during the second stage to initialize the parameters of a DBM to maximize the variational lower bound of a marginal log-likelihood.

Unlike the previous work, the author moved his focus to a denoising autoencoder (see Section 3.3) trained with a sparsity regularization, in **Publication VIII**. In this work, mathematical motivation is given for sparsifying the states of hidden units when the autoencoder was trained with a sparsity regularization (see Section 3.2.5. The author proposes a simple sparsification based on a shrinkage operator that was

empirically shown to be effective when an autoencoder is used to denoise a corrupted image patch with high noise.

In **Publication X** and **Publication IX**, two potential applications of deep neural networks were investigated. An RBM with Gaussian visible units was used to extract features from speech signal for speech recognition in highly noisy environment, in Publication X. This work showed that an existing system can easily benefit from simply adopting a deep neural network as an additional feature extractor. In Publication IX, the author applied a denoising autoencoder, a GRBM and a GDBM to a blind image denoising task.

2. Preliminary: Simple, Shallow Neural Networks

In this chapter, we review several types of simple artificial neural networks that form the basis of deep neural networks[1]. By the term *simple* neural network, we refer to the neural networks that do not have any hidden units, in the case of supervised models, or have zero or one single layer of hidden units, in the case of unsupervised models.

Firstly, we look at a supervised model that consists of several visible, or input, units and a single output unit. There is a feedforward connection from each input unit to the output unit. Depending on the type of the output unit, this model can perform linear regression as well as a (binary) classification.

Secondly, unsupervised models are described. We begin with a linear autoencoder that consists of several visible units and a single layer of hidden units, and show the connection with principal component analysis (PCA). Then, we move on to Hopfield networks.

These models will be further discussed in a probabilistic framework. Each model will be re-formulated as a probabilistic model, and the correspondence between the parameter estimation from the perspectives of neural networks and probabilistic models will be found. This probabilistic perspective will be useful later in interpreting a deep neural networks as a machine performing probabilistic inference and generation.

At the end of this section, we introduce conditions that distinguish deep neural networks from the simple neural networks introduced in the earlier part of this chapter. Those conditions will be used as a basis for constructing deep neural networks from the simpler, shallower neural networks.

[1] Note that we use the term *neural network* instead of *artificial* neural network. There should not be any confusion, as this thesis specifically focuses only on artificial neural networks.

2.1 Supervised Model

Let us consider a case where a set D of N input/output pairs is given:

$$D = \left\{ \left(\mathbf{x}^{(n)}, y^{(n)} \right) \right\}_{n=1}^{N}, \tag{2.1}$$

where $\mathbf{x}^{(n)} \in \mathbb{R}^p$ and $y^{(n)} \in \mathbb{R}$ for all $n = 1, \ldots, N$.

It is assumed that each y is a noisy observation of a value generated by an unknown function f with $\mathbf{x}$:

$$y = f(\mathbf{x}) + \epsilon, \tag{2.2}$$

where ϵ indicates noise. Furthermore, it may be assumed that $\mathbf{x}^{(n)}$ is a noisy sample of an underlying distribution. Under this setting, a supervised model aims to estimate f using the given training set D.

Often when y has many, potentially infinite possible outcomes, the task is called a *regression* problem. On the other hand, when y is a discrete variable corresponding to a class of y with only a small number of possible outcomes, it is called a *classification* task.

Now, we look at how simple neural networks can be used to solve these two tasks.

2.1.1 Linear Regression

A directed edge between two units or neurons indicates that the output of one unit flows into the other one via the edge[2]. It is possible to have multiple edges going out from a single unit and to have multiple edges coming in. Each edge has a weight value that amplifies the signal carried by the edge.

A linear unit u gathers all p incoming values amplified by the associated weights and outputs their sum:

$$u(\mathbf{x}) = \sum_{i=1}^{p} x_i w_i + b, \tag{2.3}$$

where w_i is a weight of the i-th incoming edge, and b is a bias of the unit. With this linear unit as an output unit, we can construct a simple neural network that can simulate the unknown function f, given a training set D.

We can arrange the input and output units with the described linear units, as shown in Figure 2.1 (a). With a proper set of weights, this network then simulates the unknown function f given an input x.

[2]Although it is common to use the terms *neuron*, *node* and *unit* to indicate each variable in a neural network, from here on, we use the term *unit*, only. An edge in a neural network is also commonly referred to as a synapse, synaptic connection or edge, but we use the term *edge* only in this thesis.

(a) Linear Regression (b) Perceptron

Figure 2.1. Illustrations of linear regression and perceptron networks. Note that the outputs of these two networks use different activation functions.

The aim now becomes to find a vector of weights $\mathbf{w} = [w_1, \ldots, w_p]^\top$ such that the output u of this neural network estimates the unknown function f as closely as possible. If we assume Gaussian noise ϵ, this can be done by minimizing the squared error between the desired outputs $\left\{y^{(n)}\right\}$ and the simulated output $\left\{u(\mathbf{x}^{(n)})\right\}$ with respect to $\mathbf{w}$:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{n=1}^{N} \left(y^{(n)} - u\left(\mathbf{x}^{(n)}\right)\right)^2 + \lambda\Omega\left(\mathbf{w}, D\right), \tag{2.4}$$

where Ω and λ are the regularization term and its strength. Regularization is an oft-used method for preventing an estimated model from overfitting to training samples.

If we assume the case of no regularization ($\lambda = 0$), we can find the analytical solution of $\hat{\mathbf{w}}$ by a simple linear least-squares method (see, e.g., Golub and van Van Loan, 1996; Haykin, 2009). For instance, $\hat{\mathbf{w}}$ is obtained by multiplying $\mathbf{y} = \left[y^{(1)}, y^{(2)}, \ldots, y^{(N)}\right]^\top$ from left a pseudo-inverse of $\mathbf{X}^\top = \left[\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\right]^\top$.

However, when there exists a regularization term, the problem in Eq. (2.4) may not have an analytical solution depending on the type of the regularization term. In this case, one must resort to using an iterative optimization algorithm (see, e.g., Fletcher, 1987). We iteratively compute updating directions to update $\mathbf{w}$ such that eventually $\mathbf{w}$ will converge to a solution $\hat{\mathbf{w}}$ that (locally) minimizes the cost function.

One exception is the ridge regression which regularizes the growth of the L_2-norm of the weight vector $\mathbf{w}$ such that

$$\Omega(\mathbf{w}, D) = \sum_{i=1}^{p} w_i^2.$$

In this case, we still have an analytical solution

$$\hat{\mathbf{w}} = \left(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\right)^{-1}\mathbf{X}\mathbf{y}.$$

With other regularization terms, however, it is usual that there is no analytical solution. For instance, the *least absolute shrinkage and selection operator* (lasso) (Tibshirani, 1994) regularizes the L_1-norm of the weights, and the regularization term

$$\Omega(\mathbf{w}, D) = \sum_{i=1}^{p} |w_i|.$$

does not have an exact analytical solution.

Although we have considered the case of a one-dimensional output y, it is easy to extend this network to predict a multi-dimensional output. Simply, the network will require as many output units as is the dimensionality of the output $\mathbf{y}$. The solution for the weights $\mathbf{w}$ can be found in exactly the same way as before by solving weights corresponding to each output simultaneously.

This simple linear neural network is highly restrictive in a sense that it can only approximate, or simulate, a *linear* function arbitrary well. When the unknown function f is not linear, this network most likely fails to simulate it. This is one of the motivations for considering a deep neural network instead.

2.1.2 Perceptron

The basic idea of the perceptron introduced by Rosenblatt (1958) is to insert a Heaviside step function ϕ after the summation in a linear unit, where

$$\phi(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{otherwise} \end{cases} . \tag{2.5}$$

The unit u then becomes nonlinear:

$$u(\mathbf{x}) = \phi\left(\sum_{i=1}^{p} x_i w_i + b\right) . \tag{2.6}$$

This formula allows us to perform a binary classification, where each sample is either classified as *negative* (0) or *positive* (1)[3].

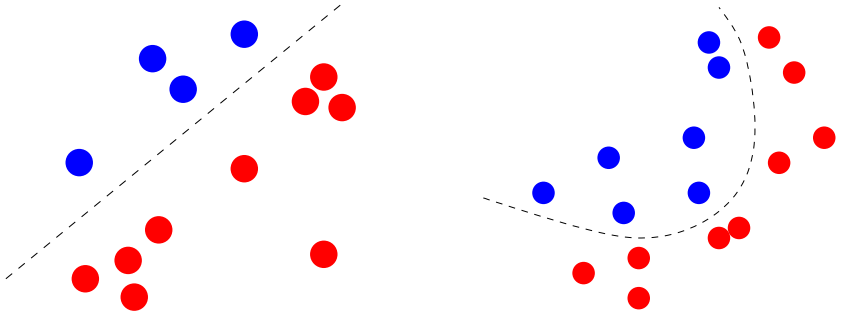
It is clear from the illustration of a perceptron in Fig. 2.1 (b) that the perceptron is identical to the linear regression network except that the activation function of the output is a nonlinear step function (see, e.g., Haykin, 2009).

Consider a case where we have again a training set D of input/output pairs. However, now each output $y^{(n)}$ is either 0 or 1. Furthermore, each $y^{(n)}$ was generated from $\mathbf{x}^{(n)}$ by an unknown function f, as in Eq. (2.2). As before, we want to find a set of weights $\mathbf{w}$ such that the perceptron can approximate the unknown function f as closely as possible.

In this case, this is considered a *classification* task rather than a *regression* as there is a finite number of possible values for y. The task of the perceptron is to figure out to which class each sample $\mathbf{x}$ belongs.

A perceptron can perfectly simulate the unknown function f, when the training samples are *linearly* separable (see, e.g., Haykin, 2009). It means that there exists a linear hyperplane that separates $\mathbf{x}^{(n)}$ that belongs to the positive class from those

[3]Originally, instead of the Heaviside step function, $\phi(x)$ was defined to give either -1 or 1.



(a) Linearly separable (b) Nonlinearly separable

Figure 2.2. (a) Samples are linearly separable. (b) They are nonlinearly separable.

that belong to the negative class (see Fig. 2.2). With a correct set of weights $\mathbf{w}^*$, the linear *separating hyperplane* can be characterized by

$$\sum_{i=1}^{p} x_i w_i^* + b^* = 0$$

The perceptron learning algorithm can be used to estimate the set of weights, assuming -1 and 1 outputs in the Heaviside function. The algorithm iteratively updates the weights $\mathbf{w}$ over N training samples by the following rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \left[y^{(n)} - u\left(\mathbf{x}^{(n)}\right) \right] \mathbf{x}^{(n)}.$$

This will converge to the correct solution as long as the given training set is linearly separable.

Note that it is not necessary to use the Heaviside step function. It is possible to use any other nonlinear saturating function whose range is limited from up and down so that it can approximate the Heaviside function. One such example is a sigmoid function whose range is $[0, 1]$:

$$\phi(x) = \frac{1}{1 + \exp(-x)}. \tag{2.7}$$

In this case, a given sample $\mathbf{x}$ is classified positive if the output is greater than, or equal to, 0.5, and otherwise as negative. Another possible choice is a hyperbolic tangent function whose range is $[-1, 1]$:

$$\phi(x) = \tanh(x). \tag{2.8}$$

The set of weights can be estimated in another way by minimizing the difference between the desired output and the output of the network, just like in the simple linear neural network. However, in this case the cross-entropy cost function (see, e.g. Bishop, 2006) can be used instead of the mean squared error:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{n=1}^{N} \Big(-y^{(n)} \log u\left(\mathbf{x}^{(n)}\right) - \left(1 - y^{(n)}\right) \log\left(1 - u\left(\mathbf{x}^{(n)}\right)\right) \Big) + \lambda \Omega\left(\mathbf{w}, D\right). \tag{2.9}$$

Unlike the simple linear neural network, this does not have an analytical solution, and one needs to use an iterative optimization algorithm.

As was the case with the simple linear neural network, the capability of the perceptron is limited. It only works well when the classes are *linearly* separable (see, e.g., Minsky and Papert, 1969). For instance, a perceptron cannot learn to compute an exclusive-or (XOR) function. In this case, any non-noisy samples from the XOR function are clearly separable, however, with a nonlinear boundary.

It has been known that a network of perceptrons, having between the input units and the output unit one or more layers of nonlinear hidden units that do not correspond to either inputs or outputs, can solve the XOR function (see, e.g., Touretzky and Pomerleau, 1989). This makes us consider a *deep* neural network also in the context of classification.

2.2 Unsupervised Model

Unlike in supervised learning, we now consider a case where there is no target value. Hence, the training set D consists of only input vectors:

$$D = \left\{ \mathbf{x}^{(n)} \right\}_{n=1}^{N}. \tag{2.10}$$

Similarly to the supervised case, we may assume that each $\mathbf{x}$ in D is a noisy observation of an unknown hidden variable such that

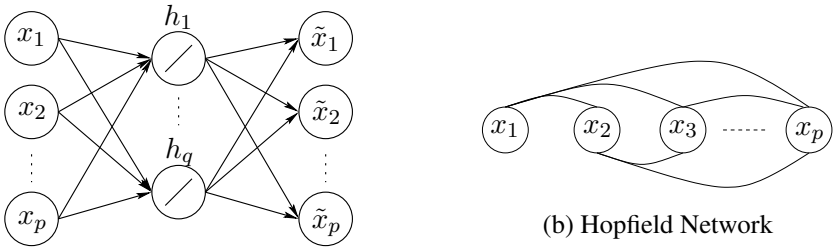
$$\mathbf{x} = f(\mathbf{h}) + \epsilon, \tag{2.11}$$

where ϵ indicates noise. Whereas we aimed to find the function or mapping f given both input and output previously in supervised models, our aim here is to find both the unknown function f and the hidden variables $\mathbf{h} \in \mathbb{R}^q$. This leads to *latent variable models* in statistics (see, e.g., Murphy, 2012).

However, this is not the only way to formulate an unsupervised model. Another way is to build a model that learns direct relationships among the input components $x_1, \ldots, x_p$. This does not require any hidden variable, but still learns an (unknown) structure of the model.

2.2.1 Linear Autoencoder and Principal Component Analysis

Firstly, we look at the case where hidden variables are assumed to have generated training samples. In this case, it is desirable to learn not only an unknown function f, but also another function g which is an inverse function of f. Opposite to f, the inverse function g recognizes a given sample by finding a corresponding state of the



(a) Linear Autoencoder

(b) Hopfield Network

Figure 2.3. Illustrations of a linear autoencoder and Hopfield network. An undirected edge in the Hopfield network indicates that signal flows in both ways.

hidden variables[4].

Let us start by constructing a neural network with linear units. There are as many input units as p corresponding to components of an input vector, denoted by $\mathbf{x}$, and q linear units that correspond to the hidden variables, denoted by $\mathbf{h}$. Additionally, we add another set of p linear units, denoted by $\tilde{\mathbf{x}}$. We connect directed edges from $\mathbf{x}$ to $\mathbf{h}$ and from $\mathbf{h}$ to $\tilde{\mathbf{x}}$. Each edge e_{ij} which connects the i-th input unit to the j-th hidden unit has a corresponding weight w_{ij}. Also, edge e_{jk} which connects the j-th hidden unit to the k-th output unit has its weight u_{jk}. See Fig. 2.3 (a) for the illustration.

This model is called a linear autoencoder[5]. The encoder of the autoencoder is

$$\mathbf{h} = f(\mathbf{x}) = \mathbf{W}^\top \mathbf{x} + \mathbf{b}, \tag{2.12}$$

and the decoder is

$$\tilde{\mathbf{x}} = g(\mathbf{h}) = \mathbf{U}^\top \mathbf{h}(\mathbf{x}) + \mathbf{c}, \tag{2.13}$$

where we uses the matrix-vector notation for simplicity. $\mathbf{W} = [w_{ij}]_{p\times q}$ are the encoder weights, $\mathbf{U} = [u_{jk}]_{q\times p}$ the decoder weights, and $\mathbf{b}$ and $\mathbf{c}$ are the hidden biases and the visible biases, respectively. It is usual to call the layer of the hidden units a *bottleneck*[6]. Note that without loss of generality we will omit biases whenever it is necessary to make equations uncluttered.

In this linear autoencoder, the encoder in Eq. (2.12) acts as an inverse function g that recognizes a given sample, whereas the decoder in Eq. (2.13) simulates the unknown function f in Eq. (2.11).

If we tied the weights of the encoder and decoder so that $\mathbf{W} = \mathbf{U}^\top$, we can see the connection between the linear autoencoder and the principal component analysis

[4]Note that it is not necessary for g to be an explicit function. In some models such as sparse coding in Section 3.2.5, g may be defined implicitly.

[5]The same type of neural networks is also called *autoassociative* neural networks. In this thesis, however, we use the term *autoencoder* which has become more widely used recently.

[6]Although the term *bottleneck* implicitly implies that the size of the layer is smaller than that of either the input or output layers, it is not necessarily so.

(PCA). Although there are many ways to formulate PCA (see, e.g. Bishop, 2006), one way is to use a minimum-error formulation[7] that minimizes

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{n=1}^{N}\left\|\mathbf{x}^{(n)} - \tilde{\mathbf{x}}^{(n)}\right\|_2^2. \tag{2.14}$$

The minimum of Eq. (2.14) is in fact exactly the solution the linear autoencoder aims to find.

This connection and equivalence between the linear autoencoder and the PCA have been noticed and shown by previous research (see, for instance, Oja, 1982; Baldi and Hornik, 1989). However, it should be reminded that minimizing the cost function in Eq. (2.14) by an optimization algorithm is unlikely to recover the principal components, but an arbitrary basis of the subspace spanned by the principal components, unless explicitly constraining the weight matrix to be orthogonal.

This linear autoencoder has several restrictions. The most obvious one is that it is only able to learn a correct model when the unknown function f is linear. Secondly, due to its linear nature, it is not possible to model any hierarchical generative process. Adding more hidden layers is equivalent to simply multiplying the weight matrices of additional layers, and this does not help in any way.

Another restriction is that the number of hidden units q is upper-bounded by the input dimensionality p. Although it is possible to use $q > p$, it will not make any difference, as it does not make any sense to use more than p principal components in PCA. This could be worked around by using regularization as in, for instance, sparse coding (Olshausen and Field, 1996) or independent component analysis (ICA) with reconstruction cost (Le et al., 2011b).

As was the case with the supervised models, this encourages us to investigate more complex, overcomplete models that have multiple layers of nonlinear hidden units.

2.2.2 Hopfield Networks

Now let us consider a neural network consisting of visible units only, and each visible unit is a *nonlinear*, *deterministic* unit, following Eq. (2.6), that corresponds to each component of an input vector $\mathbf{x}$. We connect each pair of the units x_i and x_j with an undirected edge e_{ij} that has the weight w_{ij}, as in Fig. 2.3 (b). We add to each unit x_i a bias term b_i. Furthermore, let us define an *energy* of the constructed neural network as

$$-E(\mathbf{x} \mid \boldsymbol{\theta}) = \frac{1}{2}\sum_{i\neq j} w_{ij}x_i x_j + \sum_i x_i b_i, \tag{2.15}$$

[7] Actually the minimum-error formulation minimizes the mean-squared error $\mathbb{E}\left[\|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2\right]$ which is in most cases not available for evaluation. The cost function in Eq. (2.14) is an approximation to the mean-squared error using a finite number of training samples.

where $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$. We call this neural network a Hopfield network (Hopfield, 1982).

The Hopfield network aims to finding a set of weights that makes the energy of the presented patterns low via the training set $D = \left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\right\}$ (see, e.g., Mackay, 2002). Given a fixed set of weights and an unseen, possibly corrupted input, the Hopfield network can be used to find a clean pattern by finding the nearest *mode* in the energy landscape.

In other words, the weights of the Hopfield network can be obtained by minimizing the following cost function given a set D of training samples:

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} E\left(\mathbf{x}^{(n)} \middle| \boldsymbol{\theta}\right). \tag{2.16}$$

The learning rule for each weight w_{ij} can be derived by taking a partial derivative of the cost function J with respect to it. The learning rule is

$$\begin{aligned} w_{ij} &\leftarrow w_{ij} - \frac{\eta}{N} \sum_{n=1}^{N} \frac{\partial E\left(\mathbf{x}^{(n)} \mid \boldsymbol{\theta}\right)}{\partial w_{ij}} \\ &= w_{ij} + \frac{\eta}{N} \sum_{n=1}^{N} x_i^{(n)} x_j^{(n)} = w_{ij} + \eta \left\langle x_i x_j \right\rangle_{\mathrm{d}}, \end{aligned} \tag{2.17}$$

where η is a learning rate and $\langle x \rangle_P$ refers to the expectation of x over the distribution P. We denote by d the data distribution from which samples in the training set D come. Similarly, a bias b_i can be updated by

$$b_i = b_i + \eta \left\langle x_i \right\rangle_{\mathrm{d}}. \tag{2.18}$$

This learning rule is known as the Hebbian learning rule (Hebb, 1949). This rule states that the weight between two units, or neurons, increases if they are active together. After learning, the weight will be strongly positive, if the activities of the two connected units are highly correlated.

With the learned set of weights, we can simulate the network by updating each unit x_i with

$$x_i = \phi\left(\sum_{j \neq i} w_{ij} x_j + b_i\right), \tag{2.19}$$

where ϕ is a Heaviside function as in Eq. (2.6).

It should be noticed that because the energy function in Eq. (2.15) is not lower bounded and the gradient in Eq. (2.17) does not depend on the parameters, we may simply set each weight by

$$w_{ij} = c \left\langle x_i x_j \right\rangle_{\mathrm{d}},$$

where c is an arbitrary , positive constant, given a fixed set of training samples. An arbitrary c is possible, since the output of (2.19) is invariant to the scaling of the parameters. Hence, the parameters of the Hopfield network do not require any iterative estimation, but a single computation.

In summary, the Hopfield network memorizes the training samples and is able to retrieve them starting from either a corrupted input or a random sample. This is one way of learning an internal structure of a given training set in an unsupervised manner.

The Hopfield network learns the unknown structure of training samples. However, it is limited in a sense that only direct correlations among visible units are modeled. In other words, the network can only learn a second-order statistics. Furthermore, the use of the Hopfield network is highly limited by a few fundamental deficiencies including the emergence of spurious states (for more details, see Haykin, 2009). These encourage us to extend the models by introducing multiple hidden units as well as making them stochastic.

2.3 Probabilistic Perspectives

All neural network models we have described in this chapter can be re-interpreted from a probabilistic perspective. This interpretation helps understanding how neural networks perform *generative* modeling and *recognize* patterns in a novel sample. In this section, we briefly explain the basic ideas involving probabilistic approaches to machine learning problems and their relations to neural networks.

For more details in probabilistic approaches, we refer the readers to, for instance, (Murphy, 2012; Barber, 2012; Bishop, 2006).

2.3.1 Supervised Model

Let us start by looking at the discriminative modeling from the probabilistic perspective. Again, we assume that a set D of N input/output pairs, as in Eq. (2.1), is given. The same model in Eq. (2.2) is used to describe how the set D was generated. In this case, we can directly plug in a probabilistic interpretation.

Let each component x_i of $\mathbf{x}$ be a random variable, but for now fixed to a given value. Also, we assume that ϵ is another random variable. Then, the aim of discriminative modeling in a probabilistic approach is to estimate or approximate the conditional distribution of yet another random variable y given the input $\mathbf{x}$ and the noise ϵ parameterized[8] by $\boldsymbol{\theta}$, that is, $p(y \mid \mathbf{x}, \boldsymbol{\theta})$.

With the estimated parameters $\tilde{\boldsymbol{\theta}}$, the prediction of the output $\hat{y}$ given a new sample $\mathbf{x}$ can be computed from the conditional distribution $p(y \mid \mathbf{x}, \tilde{\boldsymbol{\theta}})$. It is typical to use

[8]It is possible to use non-parametric approaches, such as Gaussian Process (GP) (see, e.g., Rasmussen and Williams, 2006), which do not have in principle any explicit parameter. However, we may safely use the parameters $\boldsymbol{\theta}$ by including the hyper-parameters of, for instance, kernel functions and potentially even (some of) the training samples.

the mean of the distribution as a prediction and its variance as a confidence.

Linear Regression

A probabilistic model equivalent to the previously described linear regression network can be easily built by assuming that noise ϵ follows a Gaussian distribution with zero mean and its variance fixed to s^2. Then, the conditional distribution of y given a fixed input $\mathbf{x}$ becomes

$$p(y \mid \mathbf{x}, s^2) = \mathcal{N}\left(y \middle| \sum_{i=1}^{p} x_i w_i + b, s^2\right),$$

where $\mathcal{N}\left(y \middle| m, s^2\right)$ is a probability density of the scalar variable y following a Gaussian distribution with the mean m and variance s^2. A linear relationship between the input and output variables has been assumed in computing the mean of the distribution.

The parameters w_i and b can be found by maximizing the log-likelihood function

$$\mathcal{L}(\mathbf{w}, b) = -\sum_{n=1}^{N} \frac{\left(y^{(n)} - \sum_{i=1}^{p} x_i^{(n)} w_i - b\right)^2}{2s^2} + \mathrm{C}, \tag{2.20}$$

where the constant C does not depend on any parameter. This way of estimating $\hat{w}_i$ and $\hat{b}$ to maximize $\mathcal{L}$ is called maximum-likelihood estimation (MLE).

If we assume a fixed constant s^2, maximizing $\mathcal{L}$ is equivalent to minimizing

$$\sum_{n=1}^{N} \left(y^{(n)} - u(\mathbf{x}^{(n)})\right)^2$$

using the definition of the output of a linear unit $u(\mathbf{x})$ from Eq. (2.3). This is identical to the cost function of the linear regression network given in Eq. (2.4) without a regularization term.

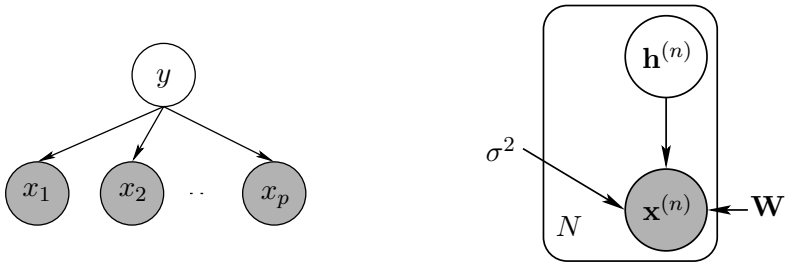
A regularization term can be inserted by considering the parameters as random variables. When each weight parameter w_i is given a prior distribution, the log-posterior distribution $\log p(\mathbf{w} \mid \mathbf{x}, y)$ of the weights can be written, using the Bayes' rule[9], as

$$\log p(\mathbf{w} \mid \mathbf{x}, y) = \log p(y \mid \mathbf{x}, \mathbf{w}) + \log p(\mathbf{w}) + \text{const.},$$

[9]The Bayes' rule states that

$$p(X \mid Y) = \frac{p(Y \mid X)p(X)}{p(Y)}, \tag{2.21}$$

where both X and Y are random variables. One interpretation of this rule is that the posterior probability of X given Y is proportional to the product of the likelihood (or conditional probability) of Y given X and the prior probability of X. Hence, if both the conditional and prior distributions are specified, the posterior probability can be evaluated as their product up to the normalization constant, or evidence, $p(Y)$.



(a) Naive Bayes (b) Probabilistic PCA

Figure 2.4. Illustrations of the naive Bayes classifier and probabilistic principal component analysis. The naive Bayes classifier in (a) describes the conditional independence of each component of input given its label. In both figures, the random variables are denoted with circles (a gray circle indicates an *observed* variables), and other parameters are without surrounding circles. The plate indicates that there are N copies of a pair of $\mathbf{x}^{(n)}$ and $\mathbf{h}^{(n)}$. For details on probabilistic graphical models, see, for instance, (Bishop, 2006).

where the constant term does not depend on the weights. If, for instance, the prior distribution of each weight w_i is a zero-mean Gaussian distribution with its variance fixed to $\frac{1}{2\lambda}$, the log-posterior distribution given a training set D becomes

$$\log p\left(\mathbf{w} \,\middle|\, \left\{(\mathbf{x}^{(n)}, y^{(n)})\right\}_{n=1}^{N}\right) = \mathcal{L}(\mathbf{w}, b) - \lambda \sum_{i=1}^{p} w_i^2,$$

which is equivalent to ridge regression (Hoerl and Kennard, 1970). When the log-posterior is maximized instead of the log-likelihood, we call it a maximum-a-posteriori estimation (MAP), or in some cases, penalized maximum-likelihood estimation (PMLE).

Logistic Regression: Perceptron

As was the case in our discussion on perceptrons in Section 2.1.2, we consider a binary classification task.

Instead of Eq. (2.2) where it was assumed that the output y was generated from an input $\mathbf{x}$ through an unknown function f, we can think of a probabilistic model where the sample $\mathbf{x}$ was generated according to the conditional distribution given its label[10] y, where y was chosen according to the prior distribution. In this case, we assume that we know the forms of the conditional and prior distributions *a priori*. See Fig. 2.4 (a) for the illustration of this model which is often referred to as the naive Bayes model (see, e.g., Bishop, 2006).

Based on this model, the aim is to find a class, or a label, that has the highest posterior probability given a sample. In other words, a given sample belongs to a class 1, if

$$p(y = 1 \mid \mathbf{x}) \geq \frac{1}{2},$$

[10] A *label* of a sample tells which *class* the sample belong to. It is often that these two terms are interchangeable.

since

$$p(y = 1 \mid \mathbf{x}) + p(y = 0 \mid \mathbf{x}) = 1$$

in the case of a binary classification.

Using the Bayes' rule in Eq. (2.21), we may write the posterior probability as

$$\begin{aligned} p(y = 1 \mid \mathbf{x}) &= \frac{p(\mathbf{x} \mid y = 1)p(y = 1)}{p(\mathbf{x} \mid y = 1)p(y = 1) + p(\mathbf{x} \mid y = 0)p(y = 0)} \\ &= \frac{1}{1 + \frac{p(\mathbf{x}|y=0)p(y=0)}{p(\mathbf{x}|y=1)p(y=1)}} \\ &= \frac{1}{1 + \exp\left(-\log \frac{p(\mathbf{x}|y=1)p(y=1)}{p(\mathbf{x}|y=0)p(y=0)}\right)}. \end{aligned}$$

It is easy to see that the posterior probability is in a form of a sigmoid function with an input

$$a = \log \frac{p(\mathbf{x} \mid y = 1)p(y = 1)}{p(\mathbf{x} \mid y = 0)p(y = 0)}.$$

A logistic regression approximates this input a with a weighted linear sum of the components of an input. The posterior probability of $y = 1$ is then

$$u(\mathbf{x}) = p(y = 1 \mid \mathbf{x}, \boldsymbol{\theta}) = \phi\left(\mathbf{W}^\top \mathbf{x} + b\right),$$

where $\boldsymbol{\theta} = (\mathbf{W}, b)$ is a set of parameters. We put $u(\mathbf{x})$ to show the equivalence of the posterior probability to the nonlinear output unit used in the perceptron (see Eq. (2.6)). Since the posterior distribution of y is simply a Bernoulli random variable, we may write the log-likelihood of the parameters $\boldsymbol{\theta}$ as

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} y^{(n)} \log u(\mathbf{x}^{(n)}) + (1 - y^{(n)}) \log(1 - u(\mathbf{x}^{(n)})). \tag{2.22}$$

This is identical to the cross-entropy cost function in Eq. (2.9) that was used to train a perceptron, except for the regularization term. The regularization term can be, again, incorporated by introducing a prior distribution to the parameters, as was done with the probabilistic linear regression in the previous section.

2.3.2 Unsupervised Model

The aim of unsupervised learning in a probabilistic framework is to let the model approximate a distribution $p(\mathbf{x} \mid \boldsymbol{\theta})$ of a given set of training samples, parameterized by $\boldsymbol{\theta}$. As was the case without any probabilistic interpretation, two approaches are often used. The first approach utilizes a set of hidden variables to describe the relationships among visible variables. On the other hand, the other approach does not require introducing hidden variables.

Principal Component Analysis and Expectation-Maximization

A PCA can be viewed in a probabilistic perspective (see, e.g., Tipping and Bishop, 1999; Roweis, 1998) by considering both $\mathbf{x}$ and $\mathbf{h}$ in Eq. (2.11) as random variables and assuming that f is a linear function parameterized by the projection matrix $\mathbf{W}$. The noise term ϵ follows a zero-mean Gaussian distribution with its variance σ^2. Furthermore, we may assume that the training samples $\mathbf{x}^{(n)}$ are centered so that their mean is zero[11].

The hidden variables $\mathbf{h}$ follow a zero-mean unit-covariance multivariate Gaussian distribution such that

$$\mathbf{h} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

and the conditional distribution over $\mathbf{x}$ given $\mathbf{h}$ is also a multivariate Gaussian with a diagonal covariance:

$$\mathbf{x} \mid \mathbf{h} \sim \mathcal{N}(\mathbf{W}\mathbf{h}, \sigma^2\mathbf{I}).$$

This is illustrated in Fig. 2.4 (b).

The aim is to estimate $\mathbf{W}$ and σ^2 that maximize the marginal log-likelihood given by

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \int_{\mathbf{h}} p(\mathbf{x} \mid \mathbf{h}) p(\mathbf{h}) \mathrm{d}\mathbf{h}, \tag{2.23}$$

where $\boldsymbol{\theta} = \left(\mathbf{W}, \sigma^2\right)$. Because both the prior distribution of $\mathbf{h}$ and the conditional distribution of $\mathbf{x} \mid \mathbf{h}$ are Gaussian distributions, the marginal distribution of $\mathbf{x}$ is also a Gaussian distribution with the following sufficient statistics:

$$\begin{aligned} \mathbb{E}\left[\mathbf{x}\right] &= 0, \\ \mathrm{Cov}\left[\mathbf{x}\right] &= \mathbf{C} = \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I}. \end{aligned}$$

In this case, Tipping and Bishop (1999) showed that all the stationary points of the marginal log-likelihood function $\mathcal{L}$ are given by

$$\hat{\mathbf{W}} = \mathbf{U}_q \left(\mathbf{L}_q - \sigma^2\mathbf{I}\right)^{1/2} \mathbf{R}$$

and

$$\hat{\sigma}^2 = \frac{1}{p-q} \sum_{i=q+1}^{p} \lambda_i,$$

where $\mathbf{U}_q$, $\mathbf{L}_q$ and λ_i are the any subset of q eigenvectors of the data covariance, the diagonal matrix of corresponding eigenvalues, and the i-th eigenvalue, respectively. $\mathbf{R}$ is an arbitrary orthogonal matrix[12]. Furthermore, in (Tipping and Bishop, 1999),

[11] This is not necessary, but for simplicity, it is assumed here.

[12] Any orthogonal matrix can be used as $\mathbf{R}$, since it does not change the sufficient statistics, specifically the covariance of the observation, as $\mathbf{W}\mathbf{R}\left(\mathbf{W}\mathbf{R}\right)^\top = \mathbf{W}\mathbf{R}\mathbf{R}^\top\mathbf{W}^\top = \mathbf{W}\mathbf{W}^\top$.

it was shown that $\mathcal{L}$ is maximal when the q *largest* eigenvectors and eigenvalues of the data covariance were used.

However, we are more interested in solving PCA using expectation-maximization (EM) algorithm (Dempster et al., 1977), which was proposed by Roweis (1998). It will make the connection to the linear autoencoder introduced in Section 2.2.1 more clear.

The EM algorithm is an iterative algorithm used to find a maximum-likelihood estimation when the probabilistic model has a set of unobserved, hidden variables. Let us state the algorithm based on (Neal and Hinton, 1999; Bishop, 2006).

We first note that the marginal log-likelihood in Eq. (2.23) can be decomposed by

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_Q(\boldsymbol{\theta}) + \mathrm{KL}(Q\|P), \tag{2.24}$$

where

$$\mathcal{L}_Q(\boldsymbol{\theta}) = \mathbb{E}_Q\left[\log p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})\right] + \mathcal{H}(Q) \tag{2.25}$$

and

$$\mathrm{KL}(Q\|P) = -\mathbb{E}_Q\left[\log \frac{p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta})}{Q(\mathbf{h})}\right], \tag{2.26}$$

and P and Q denote the true posterior distribution $p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta})$ and any arbitrary distribution, respectively. $\mathrm{KL}(Q\|P)$ is a Kullback-Leibler (KL) divergence that measures the difference between two distributions Q and P (Kullback and Leibler, 1951).

The first term is a complete-data log-likelihood over the posterior distribution of $\mathbf{h}$ given $\mathbf{x}$, and the second term is a KL divergence between a distribution Q and the posterior distribution over $\mathbf{h}$. $\mathcal{H}(Q)$ is an entropy functional of the distribution Q.

It is easy to see that $\mathcal{L}_Q(\boldsymbol{\theta})$ is a lower bound of $\mathcal{L}(\boldsymbol{\theta})$ in Eq. (2.24), since the KL-divergence $\mathrm{KL}(Q\|P)$ is always non-negative. The lower bound $\mathcal{L}_Q(\boldsymbol{\theta})$ is equal to the marginal log-likelihood when an arbitrary distribution Q is equivalent to the posterior distribution ($\mathrm{KL}(Q\|P) = 0$).

From this decomposition, we now state the EM algorithm consisting of two steps; expectation (E) and maximization (M) steps:

(E) Maximize $\mathcal{L}_Q(\boldsymbol{\theta})$ with respect to $Q(\mathbf{h})$ while $\boldsymbol{\theta}$ is fixed.
(M) Maximize $\mathcal{L}_Q(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ while $Q(\mathbf{h})$ is fixed.

In other words, we repeatedly update the approximate posterior distribution Q and the set of parameters $\boldsymbol{\theta}$ sequentially.

Since the original $\mathcal{L}(\boldsymbol{\theta})$ is not dependent on the choice of Q, the E-step effectively minimizes the KL-divergence between Q and the true posterior distribution. This is

equivalent to saying that $Q(\mathbf{h})$ becomes a better approximate of the true posterior distribution $p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta})$.

In probabilistic PCA, the E-step simply estimates the sufficient statistics of the posterior distribution $p(\mathbf{h} \mid \mathbf{x}, \tilde{\boldsymbol{\theta}})$ with the fixed set of parameters $\tilde{\boldsymbol{\theta}}$. The posterior distribution follows in this case again a Gaussian distribution, and the sufficient statistics are

$$\begin{aligned}\mathbb{E}\left[\mathbf{h}^{(n)}\right] &= (\tilde{\mathbf{W}}^\top \tilde{\mathbf{W}} + \tilde{\sigma}^2 \mathbf{I})^{-1} \tilde{\mathbf{W}}^\top \mathbf{x}^{(n)} \\ \mathbb{E}\left[\mathbf{h}^{(n)} \mathbf{h}^{(n)\top}\right] &= \tilde{\sigma}^2 (\tilde{\mathbf{W}}^\top \tilde{\mathbf{W}} + \tilde{\sigma}^2 \mathbf{I})^{-1} + \mathbb{E}\left[\mathbf{h}^{(n)}\right] \mathbb{E}\left[\mathbf{h}^{(n)}\right]^\top .\end{aligned}$$

These are simplified in the limiting case of $\sigma^2 \to 0$ as

$$\mathbb{E}\left[\mathbf{h}^{(n)}\right] = \mathbf{W}_*^\top \mathbf{x}^{(n)} \tag{2.27}$$

$$\mathbb{E}\left[\mathbf{h}^{(n)} \mathbf{h}^{(n)\top}\right] = \mathbb{E}\left[\mathbf{h}^{(n)}\right] \mathbb{E}\left[\mathbf{h}^{(n)}\right]^\top . \tag{2.28}$$

The mean of each hidden unit in Eq. (2.27) clearly corresponds to the encoder of a linear autoencoder in Eq. (2.12).

Again, in the limit of $\sigma^2 \to 0$, at the M-step, the weights $\mathbf{W}$ that maximize $\mathcal{L}_{\tilde{Q}}(\boldsymbol{\theta})$ with the fixed $\tilde{Q}$ from the E-step, are obtained by

$$\mathbf{W} = \left[\sum_{n=1}^{N} \mathbf{x}^{(n)} \mathbb{E}\left[\mathbf{h}^{(n)}\right]^\top\right] \left[\sum_{n=1}^{N} \mathbb{E}\left[\mathbf{h}^{(n)}\right] \mathbb{E}\left[\mathbf{h}^{(n)}\right]^\top\right]^{-1} . \tag{2.29}$$

We ignore σ^2 as we consider the case of $\sigma^2 \to 0$. This update corresponds to minimizing the squared reconstruction error between $\mathbf{x}^{(n)}$ and $\mathbf{W}\mathbb{E}\left[\mathbf{h}^{(n)}\right]$.

This EM iteration is equivalent to the fixed-point iteration of the weights matrix $\mathbf{W}$ that minimizes the following cost function

$$\sum_{n=1}^{N} \left(\mathbf{x}^{(n)} - \mathbf{W}\left(\mathbf{W}^\top \mathbf{x}^{(n)}\right)\right)^2 ,$$

which is identical to the cost function of the linear autoencoder in Eq. (2.14). We omitted biases without loss of generality.

This gives us another possible interpretation of the components of an autoencoder, in general. The encoder *infers* the state of the hidden variables given an input, and the decoder *generates* a visible sample given a state of the hidden variables.

Fully-Visible Boltzmann Machines

The Hopfield network described in Section 2.2.2 becomes *stochastic* if each unit is stochastic. By a *stochastic* unit, we mean that the activity of the unit is not deterministically decided based on the output of each unit but is randomly sampled from its distribution. A resulting stochastic version of the Hopfield network is called a

Boltzmann machine[13], proposed by Ackley et al. (1985).

Instead of the energy of the Hopfield network, the Boltzmann machine is defined by a probability distribution. The probability of a state $\mathbf{x}$ follows a *Boltzmann distribution*[14] (with the temperature T fixed to 1) which is defined by

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left\{-E\left(\mathbf{x} \mid \boldsymbol{\theta}\right)\right\}, \tag{2.30}$$

where $Z(\boldsymbol{\theta}) = \sum_{\mathbf{x}} \exp\left\{-E(\mathbf{x} \mid \boldsymbol{\theta})\right\}$ is a normalization constant that ensures the probability sums up to one. Although it is possible to have hidden units that do not correspond to any component of an input in Boltzmann machines, we do not consider them in this section.

The conditional probability of a unit x_i given the states of all other units $\mathbf{x}_{-i}$ is

$$p(x_i = 1 \mid x_1, \cdots, x_{i-1}, x_{i+1}, \ldots, x_N, \boldsymbol{\theta}) = \phi\left(\sum_{j \neq i} w_{ij} x_j + b_i\right), \tag{2.31}$$

where ϕ is a sigmoid function. It is easy to see the equivalence of this equation and Eq. (2.19) after substituting the Heaviside function with the sigmoid function.

Since it is possible to define a proper distribution, learning the weights is done by maximizing the log-likelihood. The log-likelihood function is defined to be

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)} \mid \boldsymbol{\theta}) \\ &= \sum_{n=1}^{N} -E\left(\mathbf{x}^{(n)} \mid \boldsymbol{\theta}\right) - \log Z(\boldsymbol{\theta}). \end{aligned} \tag{2.32}$$

The learning rule of the Boltzmann machine is quite similar to that of the Hopfield network. However, due to the existence of the normalization constant $Z(\boldsymbol{\theta})$, an additional term appears. The learning rule for the weight w_{ij} of the Boltzmann machine is

$$w_{ij} \leftarrow w_{ij} + \eta\left(\langle x_i x_j \rangle_{\mathrm{d}} - \langle x_i x_j \rangle_{\mathrm{m}}\right), \tag{2.33}$$

[13] An equivalent model of the fully-visible Boltzmann machine, called an Ising model, is used in physics to investigate a phase transition of a large system consisting of small, locally interacting particles. We refer any interested reader to (Cipra, 1987, and references therein).

[14] The Boltzmann distribution is often referred to as a *Gibbs distribution* as well. This distribution was discovered for describing the statistical distribution of any macroscopic (small) part of a large closed system in statistical physics (see, e.g., Landau and Lifshitz, 1980). Under this distribution, the probability of a state $\mathbf{x}$ is

$$p(\mathbf{x}) = \frac{1}{Z} \exp\left\{-\frac{E(\mathbf{x})}{T}\right\},$$

where T is the temperature of the system and Z is a normalization constant which does not depend on $E(\mathbf{x})$.

where d and m refer to the data distribution defined by the training set D and the model distribution defined by the Boltzmann machine (see Eq. (2.30)), respectively[15].

Although computing the statistics of the distribution learned by the Boltzmann machine, called the model distribution, is computationally intractable, one can approximate it with Markov Chain Monte Carlo (MCMC) sampling (see, e.g., Neal, 1993). As the conditional distribution of each unit can be exactly and efficiently computed by Eq. (2.31), it is possible to gather unbiased samples from the model distribution by Gibbs sampling introduced by Geman and Geman (1984). More on how to compute the statistics of the model distribution will be discussed later in Section 4.3.1.

Even though each unit is now stochastic and a valid probability distribution of input samples can be learned, the same limitation of the Hopfield network persists in the fully-visible Boltzmann machine. It also can only model the second-order statistics of the training samples. Hence, this again encourages us to consider introducing hidden units.

2.4 What Makes Neural Networks Deep?

The neural networks discussed in this chapter are *shallow* in a sense that the number of layers of units, regardless of their types, is usually at most two. Logistic regression, for instance, consists of two layers corresponding to input and output layers. A linear autoencoder or PCA consists of an input layer and a single hidden layer, considering that the input and the output layers of the linear autoencoder represent the same variable. Then we must ask ourselves: *what must a neural network satisfy in order to be called a* ***deep neural network****?*

A straightforward requirement of a deep neural network follows from its name. A deep neural network is *deep*. That is, it has multiple, usually more than three, layers of units. However, this does not fully characterize the deep neural networks we are interested in.

In essence, we say that a neural network is deep when the following two conditions are met (see, e.g., Bengio and LeCun, 2007):

1. The network can be extended by adding layers consisting of multiple units.
2. The parameters of each and every layer are trainable.

From these conditions, it should be understood that there is no absolute number of layers that distinguishes deep neural networks from shallow ones. Rather, the depth

[15]For the derivations of the conditional distribution in Eq. (2.31) and the gradient in Eq. (2.33), see, for instance, (Cho, 2011).

of a deep neural network grows by a generic procedure of adding and training one or more layers, until it can properly perform a target task with a given dataset. In other words, *data decides how many layers a deep neural network needs*[16].

One important consequence of the first condition is that the units in an added layer *share* the computation results of the units in the existing layers. In other words, there exist more than one computational path from the input layer to a unit in the added, upper layers.

In the following chapters of this thesis, we will look at deep neural networks that have become widely used recently, and discuss how they satisfy these conditions given above. For each of those networks, the structure as well as learning algorithms to estimate its parameters will be presented and explained.

2.5 Learning Parameters: Stochastic Gradient Method

Before ending this chapter and moving on to discuss deep neural networks, we briefly look at one of the most widely used techniques for learning parameters of a neural network, or any parameterized machine learning model. This can be applied not only to the simple models introduced earlier in this chapter, but also to deep neural networks that will be discussed later.

The cost functions, or *negative* log-likelihood functions, we discussed in this chapter so far (See Eqs. (2.4), (2.9), (2.14), (2.16), (2.20), (2.22), (2.23) and (2.32)), can be generalized as a sum of losses over training samples such that

$$R_e(\boldsymbol{\theta}) = \sum_{n=1}^{N} L(\mathbf{x}^{(n)}, \boldsymbol{\theta}), \tag{2.34}$$

where $\mathbf{x}^{(n)}$ can be either a sample-label pair (supervised models) or just a sample (unsupervised models). $L(\mathbf{x}, \boldsymbol{\theta})$ is a non-negative loss function.

If we follow the approach of the statistical learning theory (see, e.g. Vapnik, 1995), the empirical cost function R_e approximates the expected cost function

$$R(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}}\left[L(\mathbf{x}, \boldsymbol{\theta})\right] = \int p(\mathbf{x}) L(\mathbf{x}, \boldsymbol{\theta}) d\mathbf{x}, \tag{2.35}$$

where $p(\mathbf{x})$ is the probability of $\mathbf{x}$ or the data distribution. Clearly, $R(\boldsymbol{\theta})$ cannot be evaluated directly as the distribution from which the training samples $\left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\right\}$ were samples is *unknown*, while the empirical cost $R_e(\boldsymbol{\theta})$ can be in many cases exactly computed.

However, there is another way to look at the empirical cost function. Instead of considering a set of training samples as an initially *given* fixed set, we may consider

[16] Yoshua Bengio, personal communication

them as a sequence of sampled points from an unknown data distribution $D(\mathbf{x})$. For instance, at time t we approximate the expected cost function $R(\boldsymbol{\theta})$ by

$$R(\boldsymbol{\theta}) \approx R_e^{\langle t \rangle}(\boldsymbol{\theta}) = \frac{1}{t} \sum_{i=1}^{t} L(\mathbf{x}^{\langle i \rangle}, \boldsymbol{\theta}),$$

where $\mathbf{x}^{\langle i \rangle}$ is the sample collected from D at time i, and we denote the empirical cost at time t by $R_e^{\langle t \rangle}$. As $t \to \infty$, the empirical cost will converge to $R(\boldsymbol{\theta})$, assuming that the samples are unbiased.

Under this perspective, we may justify using the stochastic approximation method initially proposed by Robbins and Monro (1951). According to this method, the set of parameters $\boldsymbol{\theta}$ that minimizes the expected loss can be found by updating them iteratively by

$$\boldsymbol{\theta}^{\langle t+1 \rangle} = \boldsymbol{\theta}^{\langle t \rangle} - \eta_{\langle t \rangle} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^{\langle t \rangle}, \boldsymbol{\theta}^{\langle t \rangle}), \tag{2.36}$$

where the superscript $\langle t \rangle$ indicates the value of the variable at time t. Bottou (1998) provides a proof of almost sure convergence of this iteration to the true solution with a *convex* $R(\boldsymbol{\theta})$ with the following constraints on the learning rate

$$\sum_{t=1}^{\infty} \eta_{\langle t \rangle}^2 < \infty, \tag{2.37}$$

$$\sum_{t=1}^{\infty} \eta_{\langle t \rangle} = \infty, \tag{2.38}$$

assuming that $\mathbb{E}_{\mathbf{x}} \left[\nabla_{\boldsymbol{\theta}} L(\mathbf{x}, \boldsymbol{\theta}) \right] = \nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta})$. We call $\nabla_{\boldsymbol{\theta}} L(\mathbf{x}^{\langle t \rangle}, \boldsymbol{\theta}^{\langle t \rangle})$ a *sample gradient* at time t.

Furthermore, Bottou (1998) also proves a more general case where the convexity of $R(\boldsymbol{\theta})$ is *not* assumed. With some more assumptions, he was able to show that the iteration in Eq. (2.36) converges to one of the extremal points of $R(\boldsymbol{\theta})$ that include global/local minima and saddle points.

All these proofs, however, essentially require that the number of training samples approaches infinity, assuming that $\mathbf{x}$ is continuous. In most of machine learning tasks, this assumption does not hold. However, it is possible to use this approach by choosing a single sample, or a (mini-)batch of a fixed-number of samples, at each iteration from the whole training set uniform-randomly to compute the sample gradient. It is also a usual practice to simply cycle through all (randomly shuffled) training samples several times.

This method of iteratively updating parameters using the stochastic iterate in Eq. (2.36) is often referred to as a *stochastic gradient method*. As one may easily guess from its name, its deterministic counterpart is a simple, *batch* gradient method.

The most important advantage of using this method is that the computational resource required at each iteration can be bounded by constant with respect to the

number of training samples. A batch steepest gradient method requires time linearly proportional to the number of all training samples. On the other hand, it takes constant time to compute a stochastic gradient with respect to the number of training samples. Furthermore, since only few samples are used for computing a stochastic gradient, the memory requirement is also much smaller than in the batch method.

Many studies (see, e.g., Bottou and Bousquet, 2008; Bottou and LeCun, 2004, and references therein) have shown that the stochastic gradient method converges to at least a general area in the parameter space with a relatively low cost function very rapidly. Furthermore, some even argued that it is easier to achieve a better solution in terms of the expected cost rather than the empirical cost with the stochastic gradient method compared to the batch gradient method in the case of neural networks (LeCun et al., 1998b).

Hence, when we talk about estimating parameters in this thesis, it is implicitly assumed that the stochastic gradient method is used together with a cost, or objective function of each model. Most arguments on the advantages and disadvantages of learning algorithms specific to different neural networks presented later will also assume that the stochastic gradient method is used.

Recently there have been many approaches that try to overcome the weaknesses of the naive stochastic gradient method, such as the slow convergence rate and inability to easily escape from a plateau or a saddle point. Tonga proposed by Le Roux et al. (2008) speeds up the convergence of the stochastic gradient method by utilizing the online approximation to the natural gradient. Schraudolph et al. (2007) proposed an online variant of a popular Quasi-Newton method. Also, Le et al. (2011a) compared the performances of various second-order optimization method in training deep neural networks against the stochastic gradient descent.

However, we do not further discuss on the stochastic gradient method and its extensions, as they are out of the scope of this thesis.

3. Feedforward Neural Networks:

Multi-layer Perceptron and Deep Autoencoder

In this chapter, we describe deep feedforward neural networks. A feedforward neural network consists of units that are connected to each other with directed edges where each edge propagates the activation of one unit to another. The network is *feedforward* in a sense that there are no feedback connections in the network, which makes it efficient to evaluate the activations of all units in the network with a single sweep from the input units to the output units.

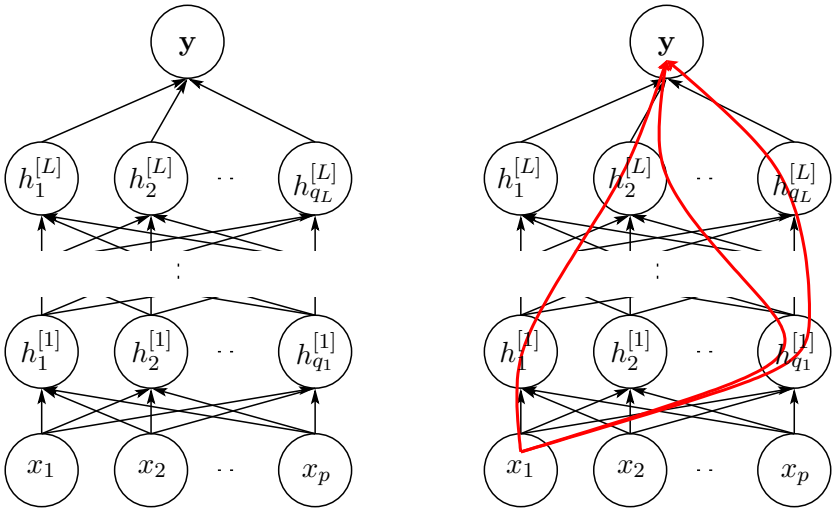
The linear regression network, perceptron, and linear autoencoder, introduced in the previous chapter, are shallow realizations of feedforward neural networks. If we group, starting from the input units, each set of the disjoint units as a layer, then one can evaluate the activations, or states, of the output units by letting the input vector propagate through the network layer by layer to the output layer.

Here we discuss in more detail two types of deep feedforward neural networks which are a multi-layer perceptron and autoencoder. Unlike the ones from the previous chapter, we consider more generalized versions that have *multiple layers* of *nonlinear* units. In due course, we introduce other machine learning models that are closely related to these neural networks and provide different aspects from which these networks can be viewed.

3.1 Multi-layer Perceptron

Let us start by extending a linear regression network and a perceptron introduced in Section 2.1. Based on the structures shown in Fig. 2.1, we may add intermediate layers of nonlinear hidden units between the input and output units. A shallow, simple perceptron, for instance, can be extended to a deep neural network by adding multiple intermediate layers, and this deep neural network is often called a *multi-layer perceptron* (MLP) (see, e.g., Haykin, 2009, and references therein). See Fig. 3.1(a) for an example of the neural network.

An MLP can approximate a nonlinear function f in Eq. (2.2) with a set D of train-



(a) Multi-layer Perceptron (b) Multiple Computational Paths

Figure 3.1. Illustrations of (a) a multi-layer perceptron and (b) an example of multiple computational paths starting from x_1 to $\mathbf{y}$. In (b), the red arrows illustrate three different paths sharing hidden units.

ing samples. Assuming linear output units (see Eq. (2.3)), the output of an MLP with L intermediate hidden layers is

$$u(\mathbf{x}) = \mathbf{U}^\top \phi\left(\mathbf{W}_{[L]}^\top \phi\left(\mathbf{W}_{[L-1]}^\top \cdots \phi\left(\mathbf{W}_{[1]}^\top \mathbf{x}\right)\cdots\right)\right),$$

where $\mathbf{U}$ and $\mathbf{W}_{[l]}$ are the weights of the edges between the L-th intermediate layer and the output layer and between $(l-1)$-th and l-th layers, respectively. ϕ is a component-wise nonlinear function such as the sigmoid function in Eq. (2.7). We have omitted biases for notational simplicity.

This can be re-written by considering each intermediate hidden layer l as a nonlinear module $g_{[l]}$ such that

$$g_{[l]}(\mathbf{x}) = \phi(W_{[l]}^\top \mathbf{x}),$$

and

$$u(\mathbf{x}) = \mathbf{U}^\top \left(g_{[L]} \circ g_{[L-1]} \circ \cdots \circ g_{[1]}(\mathbf{x})\right).$$

In this way, we can say that the MLP is a composition of multiple layers of nonlinear modules (Bengio and LeCun, 2007).

Each intermediate nonlinear layer transforms the coordinate of the input from the layer below nonlinearly. For instance, the vector of a hidden activations one layer below $\mathbf{h}_{[l-1]} \in [0,1]^p$ is transformed into $\mathbf{h}_{[l]} \in [0,1]^q$, if the sigmoid nonlinear activation function is used. Therefore, if we consider the composite of the intermediate layers as a single nonlinear transformation, it can be said that the top layer performs a *linear* regression, or classification, on the nonlinearly transformed dataset.

Another way to look at what each intermediate layer does is to consider it as a *feature detector* (see, e.g., Haykin, 2009). The j-th hidden unit $h_j^{[l]}$ in the l-th intermediate layer is likely to be active, if the activation in the layer immediately below embeds a feature similar to the associated weights $\left[w_{1,j}^{[l]}, \cdots, w_{p,j}^{[l]}\right]$, where p is the number of units in the layer below. In other words, each unit detects a feature in the activation of the layer below.

It is obvious that there exists more than one computational path from an input vector to a unit in any intermediate layer l ($l > 1$), assuming that there are more than one intermediate layers each consisting of more than one hidden units and most of the weights parameters are not zero. This implies that each intermediate unit *shares* the features detected by the units in the lower (closer to the input layer) layers. See Fig. 3.1(b) for the illustration.

Each intermediate layer is *trainable* in a sense that the associated parameters are fitted to a given training set D by minimizing the following joint cost function:

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} d\left(y^{(n)}, u(\mathbf{x}^{(n)})\right), \tag{3.1}$$

where $d(\cdot, \cdot)$ is a suitably chosen distance function. For a regression task, Euclidean distance may be used, as in Eq. (2.4), and for a classification task, one may use cross-entropy loss, as in Eq. (2.9). Learning the parameters of multiple layers can be efficiently done by backpropagation algorithm (Rumelhart et al., 1986) which will be discussed more in Section 3.4.

The importance of having one or more intermediate layers of nonlinear hidden units between the input and output units has been highlighted by the universal approximator property (Cybenko, 1989; Hornik et al., 1989). It basically states that there exists an MLP with a single hidden layer that can approximate a continuous function whose support is on a hypercube, with an arbitrary small error.

This model is a typical example of a deep neural network that can be characterized by having *multiple layers of trainable feedforward nonlinear hidden units*. This model trivially satisfies the two conditions for being a deep neural network.

3.1.1 Related, but Shallow Neural Networks

Here we explain some models that are closely related to an MLP, but are not considered deep.

Support Vector Machines and Kernel Methods

A support vector machine (SVM) proposed by Cortes and Vapnik (1995) is one of the most widely used neural network models. The SVM is based on two important ideas which are the maximum-margin principle and a kernel method. We briefly describe

these principles and try to find the connection to the MLP here. For more details on SVMs, the readers are asked to refer to (Schölkopf and Smola, 2001).

The maximum-margin principle provides a principled way of regularizing the parameters of a model. For instance, in the case of a classifier separating two classes (see Section 2.1.2), the optimal separating hyperplane is the one that has the maximal margin of separation, where the margin of separation is the distance between the hyperplane and the closest training samples from both classes.

Under this principle, the objective of training a perceptron, assuming -1 and 1 for outputs of the Heaviside function in Eq. (2.5) and each output $y^{(n)} \in \{-1, 1\}$, becomes

$$\min \frac{1}{2}\|\mathbf{w}\|^2 \tag{3.2}$$

subject to

$$y^{(n)}\left(\mathbf{w}^\top \mathbf{x}^{(n)} + b\right) \geq 1, \forall n = 1, \ldots, N$$

Once the optimization is over, the separating hyperplane with the maximum margin can be mathematically described in terms of support vectors, as in Fig. 3.2(a).

A kernel method employed by an SVM does a similar thing to what the intermediate hidden layers of an MLP do. Instead of training a linear classifier directly on raw training samples, consider training a classifier on, possibly nonlinearly, transformed samples. Let us denote the nonlinear vector transformation ϕ. Then, in essence, the classifier is trained not on $D = \left\{\left(\mathbf{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}$, but on $\tilde{D} = \left\{\left(\phi(\mathbf{x}^{(n)}), y^{(n)}\right)\right\}$. It is easy to see the connection to MLPs explained earlier by comparing ϕ to the multiple intermediate layers of hidden units.

However, instead of using an explicit transformation ϕ, the kernel method uses a *kernel* function $k(\cdot, \cdot)$ between two samples. This is justified by looking at a dual form of the objective function in Eq. (3.2) obtained by introducing Lagrangian multipliers $\boldsymbol{\alpha}$. In a dual form, the output y of an unseen, test sample $\mathbf{x}$ is written as

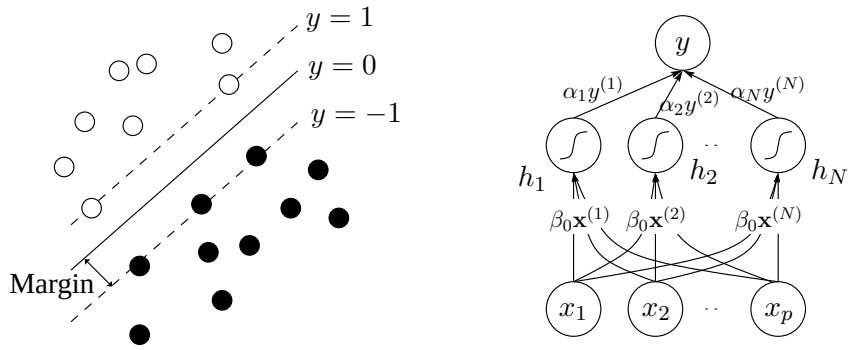
$$y = \sum_{n=1}^{N} \alpha_n y^{(n)} {\mathbf{x}^{(n)}}^\top \mathbf{x} + b,$$

and we may replace the inner product between $\mathbf{x}^{(n)}$ and $\mathbf{x}$ with a positive-definite kernel function $k(\mathbf{x}^{(n)}, \mathbf{x})$.

In fact, it is possible to build an equivalent MLP when a certain type of kernel function is used. For instance, an MLP equivalent to the SVM with the hyperbolic tangent kernel function (see Eq. (2.8))

$$k(\mathbf{x}, \mathbf{x}') = \tanh(\beta_0 \mathbf{x}^\top \mathbf{x}' + \beta_1)$$

can be constructed (see, e.g., Haykin, 2009).



(a) Maximum Margin (b) Equivalent MLP

Figure 3.2. Illustrations of (a) the maximum-margin principle and (b) constructing a multi-layer perceptron equivalent to a given support vector machine. In (a), there are two classes (denoted by black and white circles). A solid line is the maximum-margin separating hyperplane, and the samples on the dashed lines are support vectors. In (b), it should be noticed that this construction is *not* optimal, as any hidden unit not corresponding to a support-vector may be omitted.

An equivalent, but not necessarily optimal, MLP has a single intermediate hidden layer with N hidden units, each having a hyperbolic tangent activation function. The weights $\mathbf{W}$ of the edges connecting the input units to the hidden units are fixed to the training samples scaled by β_0, that is, $\mathbf{W} = \beta_0 \left[\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(N)}\right]$. The biases to all the hidden units are set to β_1. The weights of outgoing edges from the hidden units to the output unit are $\mathbf{U} = \left[\alpha_1 y^{(1)}, \alpha_2 y^{(2)}, \cdots, \alpha_N y^{(N)}\right]^\top$, and the bias to the output unit is b. The structure of this MLP is illustrated in Fig. 3.2(b).

It suggests that a kernel method is one way to extend the simple, linear neural network by adding one intermediate nonlinear hidden layer. However, this is often considered a *shallow* neural network, since the number of intermediate hidden layers is limited to one, and the added layer is usually *not* trainable (Bengio and LeCun, 2007). Because there is only a single intermediate layer, there is no sharing of features detected in the lower layers. Essentially, a kernel method is a principled way to apply a template-matching approach to a machine learning problem.

We do not consider nor discuss about the kernel method, as well as SVMs, any further in this thesis. It should be, however, noted that there has been a few attempts to make it deeper recently by Cho and Saul (2009) and Damianou and Lawrence (2013). Also, there has been a study linking the maximum-margin principle used in SVMs to a learning criterion of multi-layer perceptrons as well as a simpler perceptron (see, e.g., Collobert and Bengio, 2004, and references therein).

Extreme Learning Machines

Another related model is an extreme learning machine (ELM) proposed recently by Huang et al. (2006b). The ELM is, in essence, an MLP with a *single* intermediate layer of nonlinear hidden units.

The main difference between an ELM and an MLP is whether all layers are jointly adapted to minimize the cost function in Eq. (3.1). While the parameters of all layers of an MLP are jointly estimated, in an ELM, only the parameters of the last output layer or the outgoing weights from the penultimate layer to the output layer, are adapted.

This is equivalent to using shallow supervised neural networks, presented in Section 2.1 with a transformed training set $\tilde{D} = \left\{ \left(\phi(\mathbf{x}^{(n)}), y^{(n)} \right) \right\}$. The transformation $\phi(\mathbf{x})$ corresponds to the activations of the intermediate layer. Interestingly, in an ELM, the parameters of the transformation ϕ are not estimated, but *sampled* from a random distribution.

One of the underlying reasons why an ELM *works* at all is the Cover's separability theorem (Cover, 1965; Haykin, 2009) stating that samples in a classification problem are more likely to become linearly separable when the input is nonlinearly transformed to higher-dimensional space. Under this theorem, an ELM can be considered as a two-step model that first nonlinearly projects an input sample $\mathbf{x}$ into a higher-dimensional space $\phi(\mathbf{x})$ ($q > p$) to make it, in probability, more likely to be linearly separable and then performs classification/regression in the new space.

Since it is relatively easy and computationally efficient to estimate the parameters of the shallow neural networks, the most obvious advantage of using an ELM over an MLP is *speed*. Furthermore, the cost function then becomes a convex function, which prevents any problem arising from the existence of many, potentially infinite local minima, again unlike an MLP. These reasons have led to the popularity of an ELM recently (Huang et al., 2011).

Similarly to the SVM we discussed right before, an ELM is not a deep neural network. Firstly, it is not clear whether stacking any more intermediate layers with randomly sampled parameters would improve the performance. If we naively consider the Cover's theorem, there is no theoretical or empirical motivation for using multiple layers of gradually increasing dimensionality compared to just using a large number of hidden units in a single intermediate layer. Thus, it does not satisfy the first condition. Secondly, the parameters between the input and intermediate hidden layers are *not* trained, further violating the second condition.

3.2 Deep Autoencoders

As was the case with MLPs, a linear autoencoder can have more modeling power by employing multiple nonlinear intermediate layers symmetrically in the encoder and decoder. The units corresponding to the hidden variables in Eq. (2.11) may also be replaced with nonlinear units instead of the linear units originally used in the

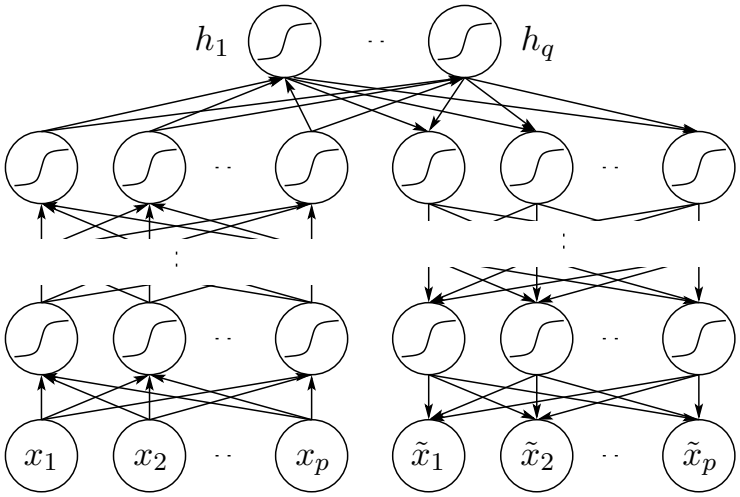


Figure 3.3. Illustration of a deep autoencoder. The left- and right- halves correspond to the encoder and decoder, respectively. Although each hidden node is drawn to have a sigmoid nonlinear activation function, a different activation function may be used.

linear autoencoder. We call this a *deep* autoencoder, and it is a typical example of an unsupervised deep neural network.

When there are $L-1$ intermediate layers of hidden units both in the encoder and decoder, the encoder becomes

$$\mathbf{h} = f(\mathbf{x}) = f_{[L-1]} \circ f_{[L-2]} \circ \cdots \circ f_{[1]}(\mathbf{x}) \tag{3.3}$$

and the decoder is

$$\tilde{\mathbf{x}} = g(\mathbf{x}) = g_{[1]} \circ g_{[2]} \circ \cdots \circ g_{[L-1]}(\mathbf{h}), \tag{3.4}$$

where $f_{[l]}$ and $g_{[l]}$ are the encoding and decoding nonlinear modules at the l-th layer. They are defined by

$$f_{[l]}(\mathbf{s}_{[l-1]}) = \phi_{[l]}\left(\mathbf{W}_{[l]}^{\top}\mathbf{s}_{[l-1]} + \mathbf{b}_{[l]}\right)$$

and

$$g_{[l]}(\mathbf{s}_{[l+1]}) = \varphi_{[l]}\left(\mathbf{U}_{[l]}\mathbf{s}_{[l+1]} + \mathbf{c}_{[l]}\right),$$

where $\mathbf{W}_{[l]}$, $\mathbf{U}_{[l]}$, $\mathbf{b}_{[l]}$ and $\mathbf{c}_{[l]}$ are parameters of the l-th hidden module, and $\phi_{[l]}$ and $\varphi_{[l]}$ are component-wise nonlinearities used in the encoder and decoder, respectively. f here should not be confused with the unknown generating function f in Eq. (2.11). We may simply write $f = f_{[L-1]} \circ \cdots \circ f_{[1]}$ and $g = g_{[1]} \circ \cdots \circ g_{[L-1]}$. See Fig. 3.3 for the illustration.

The parameters of a deep autoencoder can be found by minimizing the difference between the original input $\mathbf{x}^{(n)}$ and the reconstructed input $\tilde{\mathbf{x}}^{(n)}$ for all N training samples, as in Eq. (2.14). Of course, the difference may be measured by any suitable

distance metric such as, for instance, a squared Euclidean distance or a cross-entropy loss in the case of binary input.

We may safely call this a *deep* autoencoder, as this network satisfies the conditions of being a deep neural network. The network can be further extended by employing more intermediate hidden layers, and each and every layer is trainable.

This way of extending a linear autoencoder by adding multiple intermediate layers of hidden units with a bottleneck layer has been proposed by, for instance, Oja (1991) and Kramer (1991). In these early works, it was usual to use a bottleneck layer with less units to perform dimensionality reduction or data compression.

3.2.1 Recognition and Generation

A deep autoencoder is nothing but a plain MLP if we transformed a training set D consisting of only inputs

$$D = \left\{ \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)} \right\}$$

to another training set $\tilde{D}$ such that the label of each training sample is the training sample itself:

$$\tilde{D} = \left\{ \left(\mathbf{x}^{(1)}, \mathbf{x}^{(1)} \right), \ldots, \left(\mathbf{x}^{(N)}, \mathbf{x}^{(N)} \right) \right\}.$$

However, it becomes more interesting when we look at the deep autoencoder as a sequential composition of recognition and generation of a set of visible units.

Let us restate the underlying unsupervised model in Eq. (2.11):

$$\mathbf{x} = g(\mathbf{h}) + \epsilon.$$

This model specifies the generation of each sample $\mathbf{x}$ given the corresponding latent representation. If we further constraint the model to be explicitly parameterized, we get

$$\mathbf{x} = g(\mathbf{h} \mid \boldsymbol{\theta}_g) + \epsilon,$$

where $\boldsymbol{\theta}_g$ is a set of parameters. Furthermore, let us assume that $\mathbf{x}$ and $\mathbf{h}$ are binary vectors such that each component of them is either 0 or 1. Instead of f in the original equation (2.11), we used g for making the connection with the autoencoder more clearly.

In this section, we consider a probabilistic perspective from which a deep autoencoder can be viewed. From this perspective, we assume that the training samples were generated from the set of latent variables in a bottleneck layer. Then, the encoder is expected to approximately infer the states of the latent variables given a sample in the visible layer, and the decoder to generate a sample from the inferred latent variables.

3.2.2 Variational Lower Bound and Autoencoder

Since we will now view this model in a probabilistic framework, let us define a prior distribution over $\mathbf{h}$ by Bernoulli distribution with parameterized probability masses γ and $1-\gamma$:

$$p(\mathbf{h}) = \prod_{j=1}^{q} \gamma^{h_j}(1-\gamma)^{1-h_j} \tag{3.5}$$

We assume now that there exists a parameterized deterministic nonlinear mapping g from $\mathbf{h}$ to $\mathbf{x}$ such that the conditional distribution of $\mathbf{x}$ given $\mathbf{h}$ is

$$p(\mathbf{x} \mid \mathbf{h}) = \prod_{i=1}^{p} \tilde{x}_i^{x_i}(1-\tilde{x}_i)^{1-x_i}, \tag{3.6}$$

where $\tilde{\mathbf{x}} = g(\mathbf{h} \mid \boldsymbol{\theta}_g)$. These fully describe the probabilistic model

$$p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}_g) = p(\mathbf{x} \mid \mathbf{h})p(\mathbf{h}). \tag{3.7}$$

Let us further assume that there is a deterministic nonlinear mapping f, parameterized by $\boldsymbol{\theta}_f$ that approximates the factorized posterior distribution over $\mathbf{h}$ given $\mathbf{x}$:

$$Q(\mathbf{h}) = \prod_{j=1}^{q} q(h_j), \tag{3.8}$$

where $\mathbf{h} = f(\mathbf{x} \mid \boldsymbol{\theta}_f)$ and $q(h_j = 1) = \mu_j$. For now, we assume that $\boldsymbol{\theta}_f$ is fixed.

Since the exact evaluation of the marginal log-likelihood is intractable, the parameters $\boldsymbol{\theta}_g$ of the decoder then can be found by maximizing the lower bound of the marginal log-likelihood $\log p(D = \{\mathbf{x}^{(n)}\}_{n=1}^{N} \mid \boldsymbol{\theta}_g)$:

$$\log p(D \mid \boldsymbol{\theta}_g) \geq \mathbb{E}_{Q(\mathbf{h})}\left[\log p\left(D, H = \left\{\mathbf{h}^{(n)}\right\}_{n=1}^{N} \mid \boldsymbol{\theta}_g\right)\right] + \text{const},$$

where

$$\log p(D, H \mid \boldsymbol{\theta}_g) = \sum_{n=1}^{N}\sum_{i=1}^{p} x_i^{(n)} \log \tilde{x}_i^{(n)} + (1-x_i^{(n)}) \log(1-\tilde{x}_i^{(n)}) + \text{const}.$$

and the equality holds only when $Q(\mathbf{h})$ is the true posterior distribution. See Section 2.3.2 to see how the lower bound is derived.

It is easy to see that this corresponds to learning the parameters of the decoder of an autoencoder by minimizing the cross-entropy loss in Eq. (2.9). The only difference is that the bottleneck layer consists of stochastic units where their activations are sampled rather than decided deterministically.

This suggests that the decoder g of the deep autoencoder (see Eq. (3.4)) *generates* a visible sample starting from its latent representation, following the model given

in Eq. (3.7). On the other hand, the encoder f *recognizes* a visible sample by encoding it into a latent representation, which is in the probabilistic framework equivalent to inferring the posterior distribution over the latent variables $\mathbf{h}$. However, it should be understood that the encoder f does not perform the exact inference, but only the approximate inference assuming the fully factorized posterior distribution Q in Eq. (3.8).

If we drop the binary constraint of $\mathbf{x}$ off and instead assume that $\mathbf{x}$ follows a multivariate Gaussian distribution with a diagonal covariance matrix, we get the marginal log-likelihood that corresponds to the negative sum of squared reconstruction error (see Eq. (2.14)).

However, this interpretation does neither provide an intuitive way of learning the parameters $\boldsymbol{\theta}_f$ of the encoder f nor tell how latent representations be designed. Vincent et al. (2010) argued that minimizing the reconstruction error in an autoencoder trained by, for instance, backpropagation is equivalent to maximizing the lower bound of the mutual information between the input $\mathbf{x}$ and latent representation $\mathbf{h}$.

Combining these two approaches, in summary a deep autoencoder *recognizes* an input sample $\mathbf{x}$ with a latent representation $\mathbf{h}$ such that the mutual information between them is maximal, and *generates* a reconstructed sample $\tilde{\mathbf{x}}$ from the latent representation $\mathbf{h}$ so that the reconstruction error between the original input $\mathbf{x}$ and the reconstructed sample $\tilde{\mathbf{x}}$ is minimal.

3.2.3 Sigmoid Belief Network and Stochastic Autoencoder

A sigmoid belief network proposed by Neal (1992) consists of a single layer of stochastic binary visible units $\mathbf{x}$ and $L > 0$ layers of stochastic binary hidden units $\mathbf{h}^{[l]}$ such that a sample $\mathbf{x}$ is generated starting from the activations of the L-th layer hidden units sampled from

$$p\left(h_j^{[L]} = 1\right) = \phi\left(b_j^{[L]}\right)$$

by iteratively sampling from

$$p\left(h_j^{[l]} = 1 \,\middle|\, h_1^{[l+1]}, h_2^{[l+1]}, \dots\right) = \phi\left(\sum_k w_{jk}^{[l]} h_k^{[l+1]} + b_j^{[l]}\right), \tag{3.9}$$

where ϕ is a sigmoid function (see Eq. (2.7)).

Obviously, the probability of each component x_i given the activations of the units in the layer one above is

$$p\left(x_i = 1 \,\middle|\, \mathbf{h}^{[1]}\right) = \phi\left(\sum_j w_{ij} h_j^{[1]} + b_j\right). \tag{3.10}$$

Note that this description assumes that there are no intra-layer edges. This is not necessary, but makes it much easier to understand the generative process the network is modeling.

Once the sigmoid belief network has been trained, one important task is to *infer* the states of the hidden units given a novel sample. However, it is not trivial to do so due to the nonlinear nature of multiple layers of the hidden units.

Hinton et al. (1995) proposed a wake-sleep algorithm, which is based on the principle of minimum description length (see, e.g., Grünwald, 2007, and references therein), that learns simultaneously the *generative* parameters $\boldsymbol{\theta}_+$, used in Eqs. (3.9)–(3.10), and the *recognition* parameters $\boldsymbol{\theta}_-$, used for approximate inference of the posterior distribution over the hidden units. Since it is intractable to compute the posterior distribution exactly, the wake-sleep algorithm instead maximizes the lower bound $\mathcal{L}_Q(\boldsymbol{\theta})$ of the true marginal log-likelihood $\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \sum_{\mathbf{h}} p(\mathbf{x}^{(n)}, \mathbf{h} \mid \boldsymbol{\theta})$ (see Eqs. (2.24)–(2.26)).

Let us look at a single update step given a single data sample $\mathbf{x}$. In the *wake* stage, samples of the hidden units are collected from the approximate posterior distribution Q parameterized by the recognition parameters $\boldsymbol{\theta}_-$. With the fixed samples $h_j^{[l]}$'s from all the hidden layers, the conditional expectation $p_j^{[l]}$ of each hidden unit is computed using Eq. (3.9) with the generative parameters $\boldsymbol{\theta}_+$. Then, we update each generative weight $w_{ij}^{[l]}$ by

$$w_{ij}^{[l]} \leftarrow w_{ij}^{[l]} + \eta(h_i^{[l]} - p_i^{[l]})h_j^{[l+1]}. \tag{3.11}$$

In the *sleep* stage, the process is reversed. Samples of the hidden units are collected starting from the top layer in a top-down manner using the generative parameters $\boldsymbol{\theta}_+$. This time, the conditional expectation $p_j^{(l)}$ is computed using the recognition parameters $\boldsymbol{\theta}_-$. With these, each recognition weight $u_{ij}^{(l)}$ is updated by

$$u_{ij}^{(l)} \leftarrow u_{ij}^{(l)} + \eta s_i^{(l)}(s_j^{(l+1)} - p_k^{(l+1)}). \tag{3.12}$$

It is easy to spot the similarity between the deep autoencoder and the sigmoid belief network trained with the wake-sleep algorithm (see Fig. 3.4 for an example). Both of them maintain two sets of parameters; encoding and decoding parameters of the autoencoder, and recognition and generation parameters of the sigmoid belief network. The encoder and the recognition parameters are used to *infer* the states of the hidden units given a sample. The decoder and the generative parameters *generate* a sample given the states of the hidden units.

However, unlike the encoder in the autoencoder which computes the activation of each hidden unit *deterministically*, the recognition of the sigmoid belief network computes the activation *probability* of each hidden unit. An actual activation needs

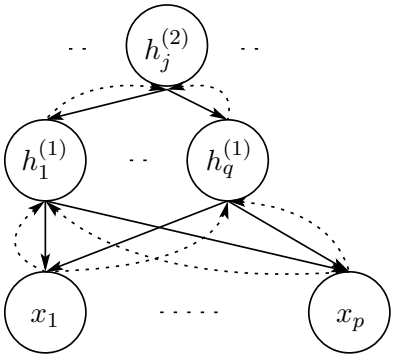


Figure 3.4. Illustration of a sigmoid belief network having two hidden layers. The solid lines indicate the generation path starting from the top hidden layer and are parameterized by the generation parameters. The dashed curves correspond to the recognition paths starting from the bottom, visible layer, parameterized by the recognition parameters.

to be sampled. The same difference exists in the process of generation. Hence, we may consider the recognition and generation passes of the sigmoid belief network as a *stochastic* deep autoencoder, however, trained with an objective function other than the reconstruction error. We will discuss more about this difference in connection to the up-down algorithm proposed to train a deep belief network (Hinton et al., 2006) later in Section 5.3.2.

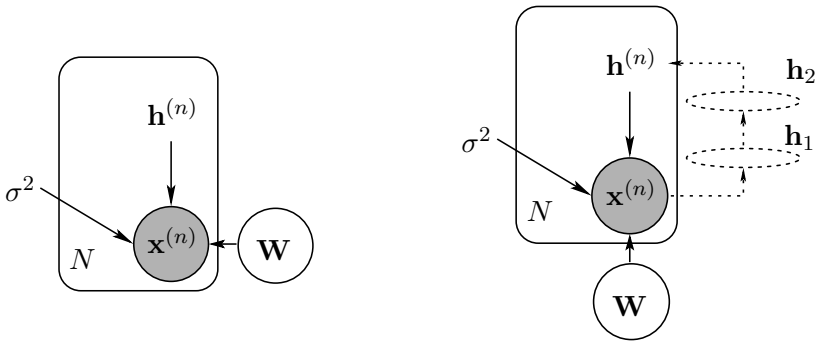
A deep belief network (DBN) (Hinton et al., 2006) extends the sigmoid belief network by adding a layer of undirected network on its top. For the remaining lower layers, the DBN keeps, as did the sigmoid belief network, two separate sets of parameters; recognition and generative parameters (Hinton et al., 2006). Except for the top two layers which are connected with undirected edges, it is clear that the encoder and decoder of a deep autoencoder correspond to the deterministic version of the recognition and generation weights of the DBN.

More on the deep belief network will be presented in Section 4.5.2.

3.2.4 Gaussian Process Latent Variable Model

Under this probabilistic approach, it is possible to use only a part of a deep autoencoder together with another probabilistic model. For instance, any latent variable model may use the encoder of a deep autoencoder to perform a fast inference of hidden variables. Here, we will briefly describe one such model, called a Gaussian process latent variable model with a back-constraint.

A Gaussian process latent variable model (GP-LVM) proposed by Lawrence (2004) is a dual formulation of probabilistic PCA introduced earlier in Section 2.3.2. Whereas the probabilistic PCA tries to maximize the log-likelihood with respect to the weights after marginalizing out the hidden variables $\mathbf{h}$, the basic version of GP-LVM maximizes the log-likelihood with respect to the hidden variables and hyper-parameters



(a) GP-LVM (b) With Back-Constraint

Figure 3.5. Illustrations of a Gaussian process latent-variable model (a) with and (b) without back-constraint. Note that the differences from the probabilistic principal component in Fig. 2.4(b) that in GP-LVM, $\mathbf{h}^{(n)}$ is *not* a random variable whereas $\mathbf{W}$ is. In (b), the back-constraint by a feedforward neural network with multiple hidden layers $\mathbf{h}_1$ and $\mathbf{h}_2$, or the encoder of a deep autoencoder, is drawn with dashed lines.

after marginalizing out the weights.

Instead of putting a prior distribution over $\mathbf{h}$, GP-LVM puts a prior distribution over $\mathbf{W}$ such that

$$p(\mathbf{W}) = \prod_{i=1}^{p} \mathcal{N}(\mathbf{w}_i \mid \mathbf{0}, \alpha^{-1}\mathbf{I}),$$

where $\mathcal{N}(\mathbf{w} \mid \mathbf{m}, \mathbf{S})$ is a probability density of $\mathbf{x}$ under a multivariate Gaussian distribution with mean $\mathbf{m}$ and covariance $\mathbf{S}$. Then, the marginal log-likelihood, after marginalizing out the weights, becomes

$$\mathcal{L} = -\frac{qN}{2}\log 2\pi - \frac{q}{2}\log|\mathbf{K}| - \frac{1}{2}\mathrm{tr}(\mathbf{K}^{-1}\mathbf{X}\mathbf{X}^\top), \tag{3.13}$$

where $\mathbf{H} = \left[\mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(N)}\right]^\top$ and $\mathbf{K} = \alpha\mathbf{H}\mathbf{H}^\top + \sigma^2\mathbf{I}$. The illustration of this probabilistic model is shown in Fig. 3.5(a). It is usual to replace $\mathbf{H}\mathbf{H}^\top$ in $\mathbf{K}$ with a nonlinear kernel matrix with hyper-parameters to construct a nonlinear GP-LVM. However, unless the linear kernel matrix is used, it is difficult to find an analytical solution for both the hyper-parameters and the hidden representations, and one must resort to an iterative optimization algorithm.

Once the hyper-parameters and the hidden representations $\mathbf{H}$ of the training samples were learned by optimization, it is easy to find a projection of a novel hidden representation on the input space by the fact that

$$p(\mathbf{X} \mid \mathbf{H}, \sigma^2) = \frac{1}{(2\pi)^{\frac{qN}{2}}|\mathbf{K}|^{\frac{q}{2}}} \exp\left\{-\frac{1}{2}\mathrm{tr}(\mathbf{K}^{-1}\mathbf{H}\mathbf{H}^\top)\right\}.$$

However, it is not trivial to find the hidden representation given a novel sample. One has to, again, resort to using an iterative optimization method, which usually is done simultaneously while the hyper-parameters are estimated. Furthermore, since the initial optimization of the marginal log-likelihood involves only the mapping from

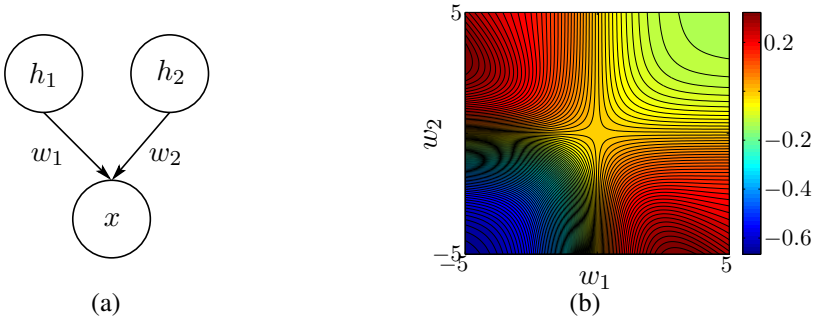


Figure 3.6. (a) A simple sigmoid belief network with two hidden units and a single visible unit. (b) The log-ratio between $p(h_2 = 1 \mid x = 1, h_1 = 1)$ and $p(h_2 = 1 \mid x = 1)$.

the hidden space to the input space, distances among the training samples in the input space are not well preserved in the hidden space.

Hence, Lawrence and Quiñonero Candela (2006) proposed to optimize the marginal log-likelihood in Eq. (3.13) with respect to a nonlinear mapping from the input space to the hidden space, instead of the hidden representations directly. The nonlinear mapping is called a back-constraint, and any preferably nonlinear function whose partial derivatives with respect to its parameters can be computed can be used. See Fig. 3.5(b) for the relationship between the GP-LVM and the back-constraint.

One of the obvious choices is the encoder part of a deep autoencoder. As the back-constraint can be considered to find a mean of the posterior distribution of hidden variables given an input, we can naturally see the encoder as doing a recognition/inference.

3.2.5 Explaining Away, Sparse Coding and Sparse Autoencoder

Let us now consider an autoencoder from the probabilistic framework considered in Section 3.2.1 together with the discussion of a sigmoid belief network in Section 3.2.3.

As originally discussed by Hinton et al. (1995), the approximate posterior distribution, or the distribution computed by the encoder of an autoencoder, is a factorized distribution. That is, the hidden units in the intermediate layer are mutually independent given the states of the visible units below. This is beneficial in the sense that the number of parameters required to express the (conditional) probability distribution over the hidden units in a single layer is reduced to $q - 1$, where q is the number of the hidden units. On the other hand, if they are *not* independent from each other, it will require up to an exponential number of parameters.

This approximation, however, severely prevents possible competitions among the hidden units. For instance, the effect of *explaining away* (see, e.g., Wellman and Henrion, 1993) cannot be observed in the factorial approximate posterior distribution.

As an example, let us consider a sigmoid belief network consisting of only two

hidden units and a single visible unit in Fig. 3.6(a). The two hidden units h_1 and h_2 are a priori mutually independent while each is equally likely to be either 0 or 1. The conditional probability of the visible unit x is defined by

$$p(x = 1 \mid h_1, h_2) = \phi(w_1 h_1 + w_2 h_3),$$

where ϕ is a logistic sigmoid function.

Under this model, the factorial assumption in the approximate posterior states that

$$p(h_2 = 1 \mid x = 1, h_1 = 1) = p(h_2 = 1 \mid x = 1, h_1 = 0), \tag{3.14}$$

since h_2 and h_1 are independent from each other conditioned on x.

The conditional probability of h_2 being 1 given $x = 1$ and $h_1 = 1$ is

$$p(h_2 = 1 \mid x = 1, h_1 = 1) = \frac{\phi(w_1 + w_2)}{\phi(w_1) + \phi(w_1 + w_2)}, \tag{3.15}$$

and that given $x = 1$ and $h_1 = 0$ is

$$p(h_2 = 1 \mid x = 1, h_1 = 0) = \frac{\phi(w_2)}{0.5 + \phi(w_1 + w_2)}. \tag{3.16}$$

These are unlikely to be identical, unless $w_1 = 0$, which means that h_1 and x are effectively disconnected.

It is also interesting to see how the state of h_1 affects the state of h_2. For instance, we can compare Eq. (3.15) against the conditional probability of $h_2 = 1$ given $x = 1$ after marginalizing out h_1. Fig. 3.6(b) shows

$$\log \frac{p(h_2 = 1 \mid x = 1, h_1 = 1)}{p(h_2 = 1 \mid x = 1)},$$

where

$$p(h_2 = 1 \mid x = 1) = \frac{\phi(w_2) + \phi(w_1 + w_2)}{\phi(0) + \phi(w_1) + \phi(w_2) + \phi(w_1 + w_2)}.$$

From this figure, we can see that the fact that h_1 is known to have caused $x = 1$ ($x = 1, h_1 = 1$) decreases the conditional probability of h_2 being the cause of $x = 1$, when both w_1 and w_2 are larger than zero. When the signs of the weights are difference, the opposite happens.

Intuitively speaking, if h_1, which is likely to have triggered an observed event ($w_1 > 0$), has been found to be true ($h_1 = 1$), it is unlikely that h_2, which is as well likely to have triggered the event ($w_2 > 0$), has happened also. Furthermore, if h_1 is known to decrease the likelihood of the observed event happening ($w_2 < 0$) happened, h_2 which is known to increase the likelihood of the event ($w_2 > 0$), is more likely to have triggered the event. This same phenomenon, called explaining away, happens when we consider the posterior probability of h_1 when h_2 was observed to be 1.

The factorial approximation of posterior distribution of the recognition process of sigmoid belief network, or of the encoder of an autoencoder, however, is unable to incorporate this competition between hidden units.

Sparse Coding and Predictive Sparse Decomposition

Sparse coding (see, e.g., Olshausen and Field, 1996) is another linear generative method which aims to learn a latent variable model in Eq. (2.11). There are two important characteristics that distinguish sparse coding from other methods such as PCA based on a linear generative model.

Firstly, sparse coding typically assumes that there are more hidden units (q) than visible units (p). It is usual to use $q = c \times p$ with the constant $c \geq 2$. Secondly, it requires that the number of nonzero components of $\mathbf{h}$ is strictly below a certain level ρ.

Given a set of training samples $D = \left\{\mathbf{x}^{(1)}, \dots \mathbf{x}^{(N)}\right\}$ sparse coding aims to find a set of weights $\mathbf{W}$ and a set of *sparse code*s $\left\{\mathbf{h}^{(1)}, \dots, \mathbf{h}^{(N)}\right\}$ by minimizing the following cost function:

$$\sum_{n=1}^{N} \left\| \mathbf{x}^{(n)} - \mathbf{W}\mathbf{h}^{(n)} \right\|_2^2 \tag{3.17}$$

subject to

$$\|\mathbf{h}^{(n)}\|_0 \leq L, \forall n = 1, \dots, N, \tag{3.18}$$

where $L = \rho q$ is a predefined level of sparsity. Similarly, we may rewrite the cost function and the constraint as

$$\|\mathbf{h}^{(n)}\|_0, \forall n = 1, \dots, N, \tag{3.19}$$

subject to

$$\left\| \mathbf{x}^{(n)} - \mathbf{W}\mathbf{h}^{(n)} \right\|_2^2 \leq \epsilon, \forall n = 1, \dots, N,$$

where ϵ is a bound on the reconstruction error.

Since it is often difficult to minimize the cost function in Eq. (3.19) directly, it is usual to relax the problem by minimizing the L_2 reconstruction error while regularizing the L_1-norm of sparse codes, that is,

$$\underset{\mathbf{W}, \left\{\mathbf{h}^{(n)}\right\}_{n=1}^{N}}{\arg\min} \sum_{n=1}^{N} \left\| \mathbf{x}^{(n)} - \mathbf{W}\mathbf{h}^{(n)} \right\|_2^2 + \lambda \sum_{n=1}^{N} \|\mathbf{h}^{(n)}\|_1. \tag{3.20}$$

Unlike an autoencoder which has an explicit encoder that outputs a hidden representation of a given input, in the framework of sparse coding, the sparse code must be found via optimization, for instance by an orthogonal matching pursuit (OMP) algorithm (Davis et al., 1994).

Most of those algorithms basically start from an all-zero code and sequentially search for a hidden unit that minimizes a certain criterion such as the reconstruction error. The search continues until the number of non-zero components reaches the

predefined level L. This sequential process, in effect, avoids the problem of the factorial assumption made by the encoder of an autoencoder described earlier.

For instance, let us consider the OMP. At each step of the algorithm, let us assume that there are two hidden units h_1 and h_2 that have the corresponding weight vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ whose inner products with a residual vector $\mathbf{r}$ are large, compared to other hidden units. From a generative modeling perspective, h_1 and h_2 generate a similar pattern in the visible units.

Assuming that $\langle \mathbf{w}_1, \mathbf{r} \rangle = \langle \mathbf{w}_2, \mathbf{r} \rangle + \epsilon$ where ϵ is a very small positive constant, the OMP will choose h_1 instead of h_2. Once $\mathbf{w}_1$ is included in the chosen bases, it is unlikely that at any step later h_2, and correspondingly its basis $\mathbf{w}_2$, will be chosen as the $\mathbf{r}$ afterward is almost orthogonal to $\mathbf{w}_2$. In essence, when h_1 was chosen, it *explained away* h_2.

Albeit this preferable property of sparse coding, the computational cost of inferring the states of hidden units is much higher compared to autoencoders. Hence, there have been attempts to combine sparse coding together with a parameterized encoder of an autoencoder to construct a model that can implement at least approximately both *explaining away* and *fast inference* (see, e.g., Kavukcuoglu et al., 2010; Gregor and LeCun, 2010).

In the case of (Kavukcuoglu et al., 2010), a regular sparse coding cost function in Eq. (3.20) is augmented with another regularization term that penalizes the difference between the sparse code $\left\{\mathbf{h}^{(n)}\right\}_{n=1}^{N}$ and the predicted sparse code computed by a parameterized nonlinear encoder f. This model, called predictive sparse decomposition (PSD), solves the following problem:

$$\underset{\mathbf{W}, \left\{\mathbf{h}^{(n)}, \boldsymbol{\theta}_f\right\}_{n=1}^{N}}{\arg\min} \sum_{n=1}^{N} \left\| \mathbf{x}^{(n)} - \mathbf{W}\mathbf{h}^{(n)} \right\|_2^2 + \lambda \sum_{n=1}^{N} \|\mathbf{h}^{(n)}\|_1 + \alpha \sum_{n=1}^{N} \|\mathbf{h}^{(n)} - f(\mathbf{x}^{(n)} \mid \boldsymbol{\theta}_f)\|_2^2,$$

where $\boldsymbol{\theta}_f$ is a set of parameters that define the encoder f, and α is a constant that balances between the encoder f and the optimal hidden states.

The underlying principle of the PSD is similar to that of the GP-LVM with a back-constraint described earlier. In an original formulation of sparse coding, decoding is straightforward and computationally cheaper, while encoding is not. Hence, the computationally expensive encoding step is approximated by a nonlinear parametric, potentially multi-layered encoder.

Sparse Autoencoder and Explicit Sparsification

Let us consider an autoencoder with only a single intermediate layer of sigmoid hidden units. In this case, we have already discussed that its encoder can be considered

as performing an approximate inference of the factorial posterior distribution of the hidden units given a state of the visible units. In the approximate factorial posterior distribution, each hidden unit follows a Bernoulli distribution with its mean computed by the encoder such that

$$p(h_j = 1 \mid \mathbf{x}, \boldsymbol{\theta}) = \phi\left(\sum_{i=1}^{p} w_{ij} x_i + c_j\right),$$

for all $j = 1, \ldots, q$.

Under this interpretation, it is possible for us to regularize the autoencoder such that on average over training samples, each hidden unit is unlikely to be active. Equivalently, the probability of each hidden unit being active should on average be low.

The autoencoder regularized to have a low average hidden activation probability is called a *sparse* autoencoder. One of the most widely used regularization term was proposed by Lee et al. (2008):

$$\Omega(\boldsymbol{\theta}, D) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{2} \left| \frac{1}{q} \sum_{j=1}^{q} \hat{h}_j^{(n)} - \rho \right|^2, \tag{3.21}$$

where ρ is a target hidden activation probability and

$$\hat{h}_j^{(n)} = p(h_j = 1 \mid \mathbf{x}^{(n)}, \boldsymbol{\theta}).$$

Instead of the squared Euclidean distance, one may use the Kullback-Leibler (KL) distance such that

$$\Omega(\boldsymbol{\theta}, D) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{q} \sum_{j=1}^{q} \left(\rho \log \frac{\rho}{\hat{h}_j^{(n)}} + (1-\rho) \log \frac{1-\rho}{1-\hat{h}_j^{(n)}} \right).$$

There are other possibilities to learn a sparse autoencoder, for instance, by a sparse encoding symmetric machine (SESM) proposed by Ranzato et al. (2008). However, here, we stick to the regularization based on Eq. (3.21).

If we consider any model that has hidden units, we find that either encoding (autoencoders) or inference (probabilistic PCA and sparse coding) is equivalent to a (nonlinear) mapping f that has its domain $\mathbb{P} \subseteq \mathbb{R}^p$ and range $\mathbb{Q} \subseteq \mathbb{R}^q$. $\mathbb{P}$ is naturally defined by the set of training samples, while $\mathbb{Q}$ is defined by a regularization term. For instance, the range of the encoder of a regular PCA, which does not have any regularization or constraint, is unrestricted at all and equals to $\mathbb{R}^q$.

However the inference procedure of sparse coding, for instance, according to the constraint in Eq. (3.18), maps an input $\mathbf{x} \in \mathbb{P}$ to $\mathbb{Q}$ which includes only those points $\mathbf{h}$ that satisfy

$$\|\mathbf{h}\|_0 \leq L,$$

for some constant $L > 0$. Similarly, the encoder of a sparse autoencoder trained with the sparsity regularization in Eq. (3.21) has the (approximate) range $\mathbb{Q} \subseteq \mathbb{R}^q$ such

that

$$\mathbb{Q} \approx \left\{ \mathbf{h} = f(\mathbf{x}) \,\middle|\, \mathbf{x} \in \mathbb{P}, \|\mathbb{E}_{\mathbf{x}\in\mathbb{P}}\left[h_j\right] - \rho\|_2^2 = 0 \right\} \tag{3.22}$$

with the amount of error controlled by the regularization constant.

In the case of sparse autoencoder, mapping $f(\mathbf{x})$ of any sample $\mathbf{x}$ that is close to one of the training samples will fall in $\mathbb{Q}$. In other words, the average activation of $f(\mathbf{x})$ will be around the predefined ρ. If the average activation of $f(\mathbf{x})$ is either much smaller or much larger than ρ, it can be suspected that the sample $\mathbf{x}$ is neither of the same type as training samples nor corrupted with high level of noise.

This leads to the idea of explicit sparsification proposed in Publication VIII. It is claimed that the encoded state of hidden units of a sparse autoencoder should not be used as it is. Rather, it must be checked whether $\mathbf{h} = f(\tilde{\mathbf{x}})$ belongs to $\mathbb{Q}$, and if not, $\mathbf{h}$ needs to be projected on or nearby $\mathbb{Q}$ by an explicit sparsification.

An explicit sparsification R is defined by

$$R(\mathbf{h}) = \arg\min_{\mathbf{q}\in\mathbb{Q}} d(\mathbf{h} - \mathbf{q}), \tag{3.23}$$

where $d(\cdot, \cdot)$ is a suitable distance metric.

One simple way to explicitly sparsify $f(\mathbf{x})$ is to use a *simple sparsification* which effectively sets small components to zero by

$$\mathbf{h} \leftarrow \max\left(\mathbf{h} - \max\left(\frac{1}{q}\|\mathbf{h}\|_1 - (1-\bar{\rho}), 0\right), 0\right), \tag{3.24}$$

where $\max$ applies to each component. It is obvious to see that $\bar{\rho}$ which defines a target sparsity should be set to one minus the target hidden activation probability ρ in Eq. (3.21).

This simple approach was empirically shown to be effective when a sparse autoencoder trained on clean samples was applied to novel samples which were corrupted with high level of noise in Publication VIII. The two tasks considered in Publication VIII were image denoising and classifying highly corrupted test samples. In both cases, more robust performance could be achieved if the activations of the units in the bottleneck were treated with the proposed simple sparsification.

3.3 Manifold Assumption and Regularized Autoencoders

The manifold[1] assumption (Chapelle et al., 2006) states that *the (high-dimensional) data lie (roughly) on a low-dimensional manifold.*

Informally, it says that a (small) neighborhood or chart $U_{\mathbf{x}}$ surrounding each data sample $\mathbf{x} \in \mathbb{R}^p$ which lies on a manifold $\mathcal{M} \subset \mathbb{R}^p$ has a bijection φ onto an open

[1] For more details on manifolds, we refer any interested reader to (Absil et al., 2008).

subset of a d-dimensional Euclidean space $\mathbb{R}^d$, where $d \ll p$. Any other point $\mathbf{x}'$ in the neighborhood $U_{\mathbf{x}}$ can be reached from $\mathbf{x}$ by

$$\mathbf{x}' = \varphi^{-1}\left(\varphi(\mathbf{x}) + \epsilon\right),$$

where $\epsilon \in \mathbb{R}^d$. Thus, the degree of freedom is d which is smaller than p.

Intuitively, this says that the number of local variations in the above case q, allowed for a sample to remain valid on the manifold is smaller than the dimensionality p of the original space to which it belongs to. Furthermore, any other variation pushes the sample outside the manifold, making it invalid, or noisy sample of data.

Under this assumption, it is desirable to transform the coordinate system of an original space into one that describes the variations allowed on the manifold only. In other words, we want the transformation to *capture* and *disentangle* the hidden, underlying factors of variations (Bengio, 2009) while ignoring any possible variation in the original space that moves away from the manifold. These transformed coordinates hopefully will improve the performance of a target task using another model. For instance, Bengio et al. (2013b) recently conjectured and provided an empirical evidence that this disentangling, or transformation, results in better mixing of MCMC chains as well as the improvement in classification tasks.

One prominent example that wholeheartedly implements this manifold assumption is principal component analysis (PCA) discussed earlier in Section 2.2.1. The PCA assumes that the manifold on which training samples lie is *linear*. The linear manifold is described by few largest principal components which correspond to the directions of maximal variance (see, e.g., Bishop, 2006). Any change along the directions of small variances, or few last principal components, is mainly considered meaningless noise. However, the linear nature of PCA is highly restrictive in a sense that in many problems most interesting factors of variations are unlikely to be linear, which is one of the reasons that motivated introducing deep neural networks with nonlinear hidden units.

In fact, an MLP with multiple intermediate hidden layers can be seen as capturing the data manifold and performing classification on the captured manifold. As discussed earlier in Section 3.1, each pair of two consecutive non-output layers nonlinearly transforms the coordinates of the input from the lower layer. If each of them can gradually capture and disentangle the factors of variations along the manifold on which training samples lie, we can expect that the top two layers which perform the actual target task, either classification or regression, will benefit from it.

Unless we are lucky, however, we cannot expect that this type of transformation that captures the data manifold happens when we estimate the parameters of an MLP. It might happen that minimizing the cost function in Eq. (3.1) ends up in the (local)

solution that corresponds to the situation where intermediate hidden layers gradually capture the data manifold, but it is not likely nor guaranteed.

Instead, there is another approach that gradually builds up a sequence of transformations, starting from the raw data. This incremental way of transforming coordinates is known widely as a greedy layer-wise pretraining (Hinton and Salakhutdinov, 2006). This will be discussed more in Chapter 5, and in the remainder of this section we will look at two variants of autoencoders that are known to capture the data manifold. These two models have been recently shown to be good candidates for gradually capturing the data manifold.

3.3.1 Denoising Autoencoder and Explicit Noise Injection

Instead of our original aim of training an autoencoder, we may decide to train it to *denoise* a noisy sample so that

$$\mathbf{x} \approx g(f(\tilde{\mathbf{x}})),$$

where $\mathbf{x}$ and $\tilde{\mathbf{x}}$ are clean and noisy versions of the same sample, and f and g are the encoder and decoder of the autoencoder, respectively.

Naturally, this can be done trivially by modifying the cost function in Eq. (2.14) so that noise is explicitly injected before the encoder is applied to training samples. When the cost function is modified accordingly, we call the resulting autoencoder a *denoising* autoencoder (Vincent et al., 2010).

A denoising autoencoder is constructed to have in many cases a single intermediate hidden layer. The parameters are estimated by minimizing the following cost function:

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} \left\| \mathbf{x}^{(n)} - g\left(f(\kappa(\mathbf{x}^{(n)}))\right) \right\|_2^2, \tag{3.25}$$

where f and g are respectively the encoder and decoder defined by Eqs. (3.3) and (3.4). κ is a stochastic operator that adds random noise to the input[2].

The denoising autoencoder finds the coordinate system of the data manifold. Minimizing Eq. (3.25) effectively corresponds to *ignoring* any direction of variation injected by κ. This is equivalent to maintaining only those directions that are implicitly found by the training samples (Vincent et al., 2010). From this, we may say that the

[2] $\kappa(\mathbf{x})$ may be designed to corrupt the input using the combination of multiple types of noise. In (Vincent et al., 2010), the following types of noise were proposed:

1. Additive white Gaussian: add white Gaussian noise to each component
2. Masking: randomly force some components to 0
3. Salt-and-Pepper noise: randomly force some components to either the maximum or minimum allowed value

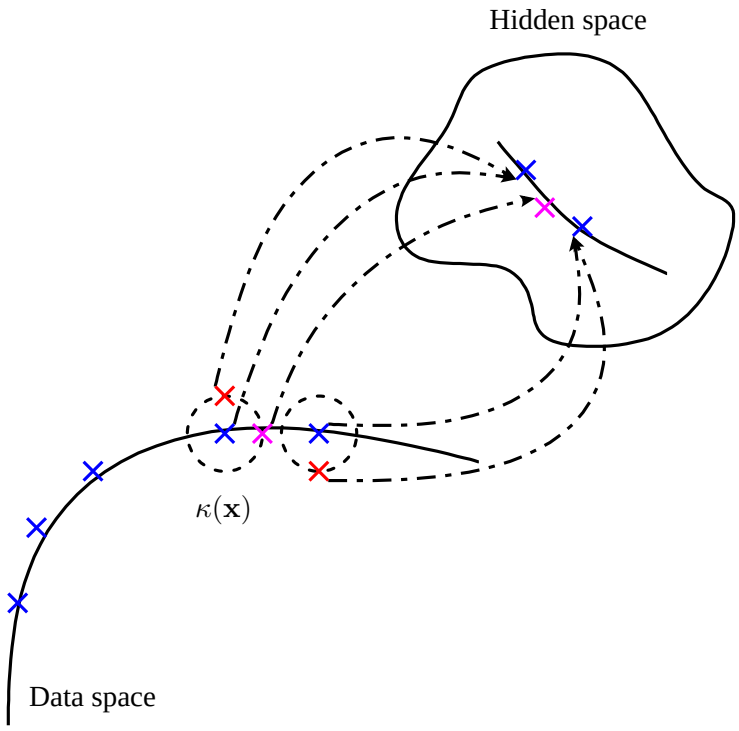


Figure 3.7. Illustration of how a denoising autoencoder learns a data manifold. The solid curves are the manifolds on which the training samples in the data space and their corresponding representations in the hidden space lie. The blue points (×) correspond to both training samples in the data space and their hidden representations in the hidden space ($\mathbf{x}$, $\mathbf{x}'$ and $f(\mathbf{x})$, $f(\mathbf{x}')$ in the text). The dashed circle around a training sample shows the amount of error explicitly injected by κ. The red points (×) are examples of the neighboring points of the training samples that are *not* on the data manifold, while the magenta point (×) lies on the manifold between two training samples ($\bar{\mathbf{x}}$ and $f(\bar{\mathbf{x}})$ in the text). The deviations of both the red points from their original training samples are ignored in the encoder, while that of the magenta point is preserved.

denoising autoencoder learns the data manifold, since the data manifold is by definition a set of small neighborhoods (charts) that encode those directions of variations.

Here let us in more detail try to understand at least informally how the denoising autoencoder internally *learns* the data manifold. We will consider a denoising autoencoder with a single intermediate hidden layer only.

Let $\mathbf{x}$ and $\mathbf{x}'$ be two nearby real-valued training samples, and $\mathbf{h} = f(\mathbf{x})$ and $\mathbf{h}' = f(\mathbf{x}')$ are corresponding hidden states encoded by a denoising autoencoder. We assume that κ was chosen to add a white Gaussian noise.

If we consider only $\mathbf{x}$, because we trained the denoising autoencoder by minimizing Eq. (3.25), any point $\mathbf{x} + \epsilon$ in a small area surrounding $\mathbf{x}$, defined by κ, will map almost exactly to $\mathbf{h}$. This applies to $\mathbf{x}'$ as well.

However, we should pay attention to a point $\bar{\mathbf{x}}$ in an *overlapping* area. For instance, a middle point $\bar{\mathbf{x}} = \frac{\mathbf{x}+\mathbf{x}'}{2}$ between $\mathbf{x}$ and $\mathbf{x}'$ will neither be reconstructed exactly to $\mathbf{x}$ nor $\mathbf{x}'$. Instead, it will be reconstructed to be a point between $\mathbf{x}$ and $\mathbf{x}'$, and because the decoder is linear, the hidden state of $\bar{\mathbf{x}}$ will also lie between $\mathbf{h}$ and $\mathbf{h}'$. See Fig. 3.7 for illustration.

In short, the change in a hidden representation encoded by the denoising autoencoder corresponds *only* to the change on the manifold on which training samples lie. Any other small change in directions moving outside the manifold will be ignored. In this way, a denoising autoencoder learns the data manifold.

This, however, does not mean that every possible state of the hidden units encodes a point in the data manifold. It is not well defined nor known how any point far from the manifold will be encoded. Fortunately, this may not be a big problem when we deal with a task where both training and test samples were obtained from a single distribution whose probability mass is mostly concentrated along the manifold. This will eliminate any need of encoding a sample far away from the manifold.

As can be expected from this argument, it has been shown by, for instance, Vincent et al. (2010), and later by other researchers (see, e.g., Bengio, 2009, and references therein), that the denoising autoencoder tends to learn a *better* representation that is more suitable for a further machine learning task such as classification than the one extracted by an ordinary autoencoder. Therefore, this model has been used widely to pretrain an MLP layer-wise.

Explicit Noise Injection for Classification

Let us now discuss briefly another aspect of injecting random noise explicitly to training samples while estimating parameters.

It is usual that we know *a priori* that the data distribution from which the training samples as well as the test samples are sampled is smooth with respect to each point in the state space. In supervised learning, this translates to that a point $\mathbf{x}$ which is not necessarily in a training set but in the neighborhood of a training sample $\mathbf{x}^{(n)}$, is highly likely to belong to the same class $y^{(n)}$. In unsupervised learning, it means that a point $\mathbf{x}$, again in the neighborhood of a training sample $\mathbf{x}^{(n)}$, is likely to have a similar probability assigned.

If we go one step further, we may say that there is a certain invariance with respect to some transformation, potentially nonlinear and stochastic, κ such that $\kappa(\mathbf{x})$ is similar, or has a similar property of $\mathbf{x}$. The property shared by those two points may be their classes or their probabilities in the data distribution.

An extremely easy and obvious way to incorporate this prior knowledge is to use corrupted or transformed versions of training samples $\left\{\mathbf{x}^{(n)}\right\}_{n=1}^{N}$ when estimating the parameters of a model (see, e.g., Bishop, 2006, Chapter 5.5).

For instance, in the case of an MLP the aim of the neural network is to find a nonlinear mapping from an input to its corresponding output. The above prior knowledge, however, suggests that rather than trying to optimize the network to find the mapping from the raw training set, the network needs to be optimized to find the mapping from

a transformed input $\kappa(\mathbf{x}^{(n)})$ to its output $y^{(n)}$. Then, the cost function original in the form of Eq. (3.1), is replaced by

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} d\left(y^{(n)}, u(\kappa(\mathbf{x}^{(n)}))\right), \tag{3.26}$$

where $u(\cdot)$ is the output of the MLP.

If we assume that κ stochastically corrupts the input, thus making a noisy version of it, the trained MLP becomes more *robust* to noise in the input. In other words, the input corrupted by some level of noise will nevertheless be mapped to its desired output which is the output expected from the clean input.

It is natural to use the stochastic gradient method described in Section 2.5 to minimize Eq. (3.26). At each iteration, a randomly chosen subset of the training set is stochastically corrupted by κ before the steepest descent direction is computed.

In the case where κ simply adds uncorrelated white Gaussian noise to an input, Bishop (1995) showed that minimizing Eq. (3.26) is equivalent to regularizing the sensitivity of an output with respect to the input such that the cost function to be minimized becomes

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} d\left(y^{(n)}, u(\mathbf{x}^{(n)})\right) + \frac{\lambda}{2} \sum_{i=1}^{p} \sum_{j=1}^{q} \left(\frac{\partial u(\mathbf{x}^{(n)})}{\partial \mathbf{x}^{(n)}}\right)^2, \tag{3.27}$$

where the regularization constant λ is determined by the amount of noise injected by κ in Eq. (3.26).

3.3.2 Contractive Autoencoder

The informal discussion in Section 3.3.1 on how the denoising autoencoder learns the data manifold, arrives at the conclusion that the key in doing so is the invariance of hidden activations to any direction of change moving outside the manifold. Or, it may be said that any infinitesimal change to a point in the data space should not change its corresponding encoded point in the hidden space. In other words, the norm of the Jacobian matrix J_f should be minimized. The Jacobian matrix $J_f \in \mathbb{R}^{q \times p}$ with respect to the encoder f is defined by

$$J_f = \begin{bmatrix} \frac{\partial \hat{h}_1}{\partial x_1} & \cdots & \frac{\partial \hat{h}_1}{\partial x_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{h}_q}{\partial x_1} & \cdots & \frac{\partial \hat{h}_q}{\partial x_p} \end{bmatrix}, \tag{3.28}$$

where $\hat{h}_j$ is the j-th component of $f(\mathbf{x})$.

A denoising autoencoder achieves this by injecting predefined levels of noise to training samples. However, this does not directly minimize J_f, as we have seen

previously in Eq. (3.27), it rather penalizes $J_{g \circ f}$ which is

$$J_{g \circ f} = \begin{bmatrix} \frac{\partial \tilde{x}_1}{\partial x_1} & \cdots & \frac{\partial \tilde{x}_1}{\partial x_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial \tilde{x}_q}{\partial x_1} & \cdots & \frac{\partial \tilde{x}_q}{\partial x_p} \end{bmatrix},$$

where $\tilde{x}_j$ is the j-th component of the reconstruction.

Hence, Rifai et al. (2011b) proposed to directly regularize the L_2-norm of J_f by adding the following regularization term to the cost function based on reconstruction error

$$\Omega(D, \boldsymbol{\theta}) = \sum_{\mathbf{x} \in D} \sum_{i=1}^{p} \sum_{j=1}^{q} \left(\frac{\partial \hat{h}_j}{\partial x_i} \right)^2. \tag{3.29}$$

Furthermore, in (Rifai et al., 2011a), they proposed to regularize, in addition to the Jacobian, the Hessian H_f approximated using the finite-difference method by modifying Eq. (3.29) to

$$\Omega(D, \boldsymbol{\theta}) = \sum_{\mathbf{x} \in D} \sum_{i=1}^{p} \sum_{j=1}^{q} \left(\frac{\partial \hat{h}_j}{\partial x_i} \right)^2 + \gamma \sum_{i=1}^{p} \sum_{j=1}^{q} \mathbb{E}_{\epsilon} \left[\left(\frac{\partial \hat{h}_j}{\partial x_i}(x_i) - \frac{\partial \hat{h}_j}{\partial x_i}(x_i + \epsilon) \right)^2 \right],$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

This idea of utilizing the Jacobian matrix with respect to each training sample suggests us to further use the Jacobian matrix to investigate the local neighborhood, or chart, centered on the training sample. The leading singular vectors or principal components of the Jacobian matrix are the directions parallel to the data manifold, while the minor singular vectors indicate those directions that are close to perpendicular to the manifold.

If the manifold assumption at the beginning of this section indeed holds true and the contractive regularization in Eq. (3.29) enables the autoencoder to capture this manifold, only few leading singular vectors will have large corresponding singular values while most others will be close to zero. In other words, the neighborhood centered on each training sample will have only a few degrees of freedom which correspond to the leading singular vectors.

Rifai et al. (2011b) empirically showed that this is indeed the case with the autoencoders trained with the contractive regularization (Eq. (3.29)) as well as the denoising autoencoders. The phenomenon was most visible when the contractive regularization was used, which suggests that directly regularizing the Jacobian matrix may help capturing the data manifold more effectively and efficiently.

A similar idea was applied to training restricted Boltzmann machines (see Section 4.4.2) in Publication IV.

3.4 Backpropagation for Feedforward Neural Networks

Before finishing our discussion on deep feedforward neural networks, in this section, we briefly describe in this section an efficient algorithm that computes the gradient of a cost function with respect to each parameter of a deep feedforward neural network. The algorithm, called backpropagation, was proposed by Rumelhart et al. (1986).

In order to compute the gradient, we first perform a *forward* pass. For each sample $\mathbf{x}$, we recursively compute the activation of the j-th hidden unit $h_j^{[l]}$ at the l-th hidden layer by

$$h_j^{[l]} = \phi_j^{[l]} \left(\sum_{k=1}^{q_{[l-1]}} h_k^{[l-1]} w_{kj}^{[l]} \right),$$

where the activation of the j-th hidden unit in the first hidden layer is

$$h_j^{[1]} = \phi_j^{[1]} \left(\sum_{i=1}^{p} x_i w_{ij}^{[1]} \right)$$

and the activation of the j-th linear output unit is

$$u_j(\mathbf{x}) = \phi_j^o \left(\sum_{k=1}^{q_{[L]}} h_k^{[L]} u_{kj} \right).$$

Note that we used a different notation for the nonlinear function of each unit, other than visible ones, to emphasize that the algorithm works in a general feedforward neural network. For simplicity, we have omitted biases without loss of generality.

After the forward pass, we begin the *backward* pass, where a local gradient δ is computed per each unit. For each output unit, where a desired output y_j is available, the local gradient is

$$\delta_j = \frac{\partial J}{\partial u_j}$$

which is simply a difference between the predicted and desired values:

$$\delta_j = y_j - u_j(\mathbf{x}),$$

assuming that the output units are linear ($\phi_j^o(x) = x$) and the cost function J is defined based on the squared Euclidean distance between the true and predicted outputs. The same computation holds when the cross-entropy loss is used together with the sigmoid output units.

For a hidden unit in a subsequent intermediate layers, we compute the local gradients recursively using the backward pass by

$$\delta_j^{[l-1]} = \left(h_j^{[l-1]} \right)' \sum_{k=1}^{q_{[l]}} w_{jk}^{[l]} \delta_j^{[l]}$$

until l reaches 2, where $\left(h_j^{[l-1]} \right)'$ is the partial derivative of $h_j^{[l-1]}$ with respect to the input to the unit. For instance, if the nonlinear function used by the unit $\phi_j^{[l-1]}$ is a

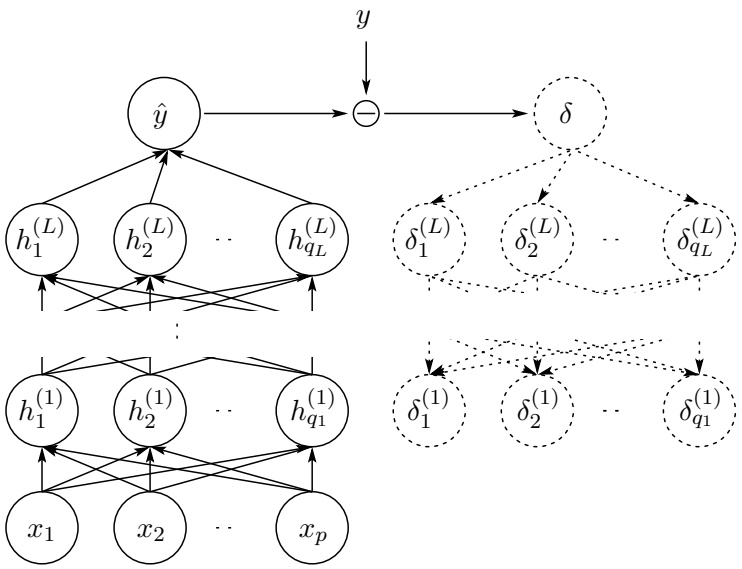


Figure 3.8. An illustration of the forward and backward passes. Starting from the input layer on left, the activation of each unit is iteratively computed up until the output $\hat{y}$ is predicted. The difference δ between the prediction $\hat{y}$ and true output y is computed, and the local gradients $\delta_j^{(l)}$ are iteratively computed in the backward pass. The left and right parts of the figure respectively correspond to the forward and backward passes.

logistic sigmoid function, then

$$\left(h_j^{[l-1]}\right)' = h_j^{[l-1]}\left(1 - h_j^{[l-1]}\right).$$

Once all the inputs and local gradients are computed, the top-level and intermediate-level weight parameters u_{kj} and $w_{ij}^{[l]}$ are respectively updated by

$$\begin{aligned} u_{kj} &\leftarrow u_{kj} - \eta\left(\delta_j h_k^{[L]}\right) \\ w_{ij}^{[l]} &\leftarrow w_{ij}^{[l]} - \eta\left(\delta_j^{[l+1]} h_i^{[l-1]}\right), \forall l \geq 2 \\ w_{ij}^{[1]} &\leftarrow w_{ij}^{[1]} - \eta\left(\delta_j^{[2]} x_i\right), \end{aligned} \tag{3.30}$$

where η is a learning rate. See Fig. 3.8 for the illustration of the forward and backward passes.

The gradient of each weight parameter relies only on the input and local gradient of the adjacent vertices. This implies that the backpropagation is *not* limited to training a feedforward neural network, but may also be applied to a recurrent neural network with a slight modification (see Section 4.2.1 to see how a recurrent neural network can be *unfolded* over time to become an infinitely-deep feedforward neural network).

Here we showed how to compute the gradient using a single sample, but extending it to multiple samples is straightforward by taking an average of gradients computed from a number of samples at each update. It is usual to use the gradient computed by the backpropagation to perform stochastic gradient descent on the cost function using a subset of training samples at a time (see Section 2.5). One often refers to this approach as *stochastic backpropagation*.

3.4.1 How to Make Lower Layers Useful

However, it has been noticed by many researchers that it is not easy to train deep neural networks using the plain stochastic backpropagation (see, e.g., Bengio and LeCun, 2007, and references therein). Especially, neural networks with more than two intermediate layers of hidden units usually result in a worse generalization performance[3] than those with only one or two intermediate layers (Bengio et al., 2007).

One important hypothesis explaining the underlying reasons for this phenomenon was proposed by Bengio et al. (2007). According to this hypothesis, this difficulty may come from the fact that the parameters of the lower layers (closer to a layer with input units) are poorly estimated since *the top three layers, consisting of the output layer and two last hidden layers, are capable enough of learning a given training set almost perfectly*. They essentially act as a deep neural network with a single intermediate hidden layer, which already has a universal approximator property, using the activation of the hidden units in the layer immediately below as an input. However, this capability of the top two layers does not translate to the performance of the neural network on unseen samples.

In other words, training an MLP with backpropagation will simply adapt the parameters of the top layers to fit the training set as well as possible. The algorithm in general will *not* necessarily prefer a solution that aims to force the lower intermediate layers, which act as *feature detectors* according to our discussion in Section 3.1, to detect important features. A simple experiment by Bengio et al. (2007) showed further that even when the top layers are *not* powerful enough to minimize the cost function almost perfectly, the plain stochastic backpropagation will fail to make the lower intermediate layers any more useful.

In an extreme case, it might be that the top two layers, an output and the last hidden layers, will be enough to minimize the cost function almost perfectly. In this case, it is possible that the network effectively becomes an extreme learning machine discussed in Section 3.1.1, since the weights of lower layers do not change from their randomly initialized values.

In order to avoid this problem, an approach that incrementally adds intermediate layers starting from the visible layer has been suggested by, for instance, Fahlman and Lebiere (1990) and Lengellé and Denæux (1996). Cascade correlation algorithm (Fahlman and Lebiere, 1990) starts from a logistic regression network without any intermediate hidden layer, and then incrementally trains and adds intermediate layers while keeping all the existing connections as long as the cost function decreases by

[3] By *generalization performance*, we refer to the performance of a trained model on an unseen, novel sample.

doing so. Lengellé and Denæux (1996) similarly add new intermediate layers, or units, until the objective function based on the class separability does not improve. However, these approaches based on supervised criteria have not been widely adapted as they did not show much improvement.

Hinton and Salakhutdinov (2006) proposed a method called greedy layer-wise pretraining which pretrains each pair of two consecutive layers starting from the bottom two layers, as if it were an restricted Boltzmann machine (see Section 4.4.2). Similarly to the earlier attempts, this approach allows stacking multiple layers. However, each pair is trained in an *unsupervised* manner such that no output information is used during the layer-wise pretraining. The success of this approach started, so called, the *second neural network renaissance* (Schmidhuber et al., 2011). We will discuss in more detail this approach in Chapter 5.

However, the hypothesis by Bengio et al. (2007) is not the only explanation available. Martens (2010) pointed out that another potential factor that makes it difficult to estimate the parameters of lower layers of a deep MLP, or a deep autoencoder by a plain backpropagation is the existence of a pathological curvature of the cost function, not the powerfulness of the top few layers.

Martens (2010) claimed that the directions of the cost function that correspond to the parameters in the lower layers have lower curvature compared to those corresponding to the parameters in the top few layers. In this scenario, the plain backpropagation, relying solely on the first-order information, is unable to make rapid, if any, progress in those directions with low curvatures, leaving the parameters of the lower layers mostly unchanged.

The Hessian-free optimization, or truncated Newton method with subsampled Hessian algorithms, proposed by Martens (2010) and Byrd et al. (2011) independently, overcomes this problem by utilizing the Hessian of the cost function. Specifically in the cases of deep autoencoders and recurrent neural networks, Martens (2010) and Sutskever et al. (2011) empirically showed that the Hessian-free optimization can indeed train very deep neural networks without much difficulty. For detailed discussion on implementing the Hessian-free optimization for training deep neural networks, we refer any interested reader to (Martens and Sutskever, 2012).

In a similar sense, Raiko et al. (2012) proposed instead of a novel optimization algorithm to linearly transform each unit in a deep neural network such that the mean and the first-derivative of the output of each unit are close to zero. In order to make the neural network invariant under such a transformation, shortcut connections skipping one or few intermediate layers were introduced. Raiko et al. (2012) informally explained and empirically showed that the proposed *linear* transformation makes the Fisher information matrix closer to diagonal, thus pushing the steepest descent gradi-

ent closer to the natural gradient direction which is known to be efficient in estimating the parameters of an MLP (Amari, 1998).

Glorot and Bengio (2010) extensively tested the plain stochastic backpropagation in training deep MLPs. They were able to provide some practical advice on the choice of nonlinear functions of hidden units as well as how to initialize the parameters to make learning avoid being stuck in a poor local optimum or a plateau.

However, one must be careful about making any definite conclusion on using the plain backpropagation. The underlying factor of the difficulty might simply be a lack of patience, and extremely longer training, potentially assisted by the recent advances in utilizing GPUs (Raina et al., 2009), might help avoiding the aforementioned problem.

Recently, for instance, Ciresan et al. (2012b) showed that a deep MLP trained with the plain stochastic backpropagation can achieve the state-of-the-art result on handwritten digits classification *without* any of the methods described earlier. Many recent papers from the same group (see, e.g. Ciresan et al., 2012c,d,a) presented deep (convolutional) neural networks achieving the state-of-the-art performances on various tasks using just the plain stochastic backpropagation.

4. Boltzmann Machines with Hidden Units

In Section 2.3.2, a fully-visible Boltzmann machine (BM) was introduced as a stochastic extension of the Hopfield network which learns an underlying statistics of training samples. These models are not considered *deep*, since it is not possible to extend them without any additional hidden units. Hence, in this chapter we discuss BMs that have hidden units in addition to the usual visible units.

In this chapter, we start with a general description of a BM with hidden units. Then, we give a brief explanation why any recurrent neural network with hidden units may be considered a deep neural network (see, e.g., Bengio et al., 2013). This will provide a basis on which any Boltzmann machine, which is one particular type of recurrent neural networks, with hidden units is considered *deep*. Afterwards, we discuss more about BMs with hidden units in depth.

4.1 Fully-Connected Boltzmann Machine

Here we generalize the fully-visible Boltzmann machine described in Section 2.3.2 to a general, fully-connected Boltzmann machine (Ackley et al., 1985). A fully-connected Boltzmann machine has, in addition to a set of visible stochastic units, a separate set of *hidden* units[1].

Regardless of whether it is visible or hidden, each unit x_i is binary having either 0 or 1 as its state and has a bias b_i. Each pair x_i and x_j is connected by an undirected edge with its weight w_{ij}, or one may say that there are two directed edges from x_i to x_j and from x_j to x_i with a symmetric weight w_{ij}.

Let us denote the weights of the edges connecting between the visible and hidden units, of those among the visible units, and of those among the hidden units by $\mathbf{W} \in \mathbb{R}^{p\times q}$, $\mathbf{U} \in \mathbb{R}^{p\times p}$ and $\mathbf{V} \in \mathbb{R}^{q\times q}$, respectively. Then, as we did with both the Hopfield network and the fully-visible Boltzmann machine previous, we define the

[1] Whenever it is clear that a referred unit is stochastic, we will drop the term *stochastic*.

energy of the network by

$$-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \mathbf{b}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{h} + \mathbf{x}^\top \mathbf{W} \mathbf{h} + \frac{1}{2} \mathbf{x}^\top \mathbf{U} \mathbf{x} + \frac{1}{2} \mathbf{h}^\top \mathbf{V} \mathbf{h}, \tag{4.1}$$

where $\mathbf{b}$ and $\mathbf{c}$ are vectors of the biases to the visible and hidden units, respectively. $\boldsymbol{\theta}$ indicates a set of all parameters including $\mathbf{b}$, $\mathbf{c}$, $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$. Naturally, we define a probability of a state $\left[\mathbf{x}^\top, \mathbf{h}^\top\right]^\top$ with the Boltzmann distribution by

$$p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left\{-E\left(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}\right)\right\}, \tag{4.2}$$

where again $Z(\boldsymbol{\theta})$ is a normalization constant that requires the summation over an exponential number of terms with respect to the number of all units.

See Fig. 4.1 for the illustration of this Boltzmann machine.

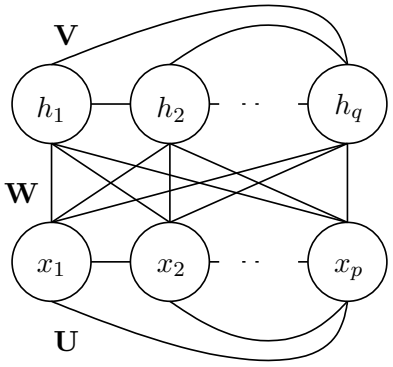


Figure 4.1. Fully-Connected Boltzmann machine.

When it comes to estimating parameters, however, the marginal log-likelihood needs to be maximized instead. The marginal log-likelihood is computed by marginalizing out the hidden units such that

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \log \sum_{\mathbf{h}} \exp\left(-E\left(\mathbf{x}^{(n)}, \mathbf{h} \mid \boldsymbol{\theta}\right)\right) - \log Z(\boldsymbol{\theta}). \tag{4.3}$$

The parameters can be estimated by maximizing Eq. (4.3) using, for instance, the stochastic gradient algorithm. This algorithm will take a small step in the direction of the steepest ascent direction in the parameter space. The steepest direction is computed for each parameter by the partial derivative of the marginal log-likelihood with respect to it. The partial derivative with respect to a parameter θ is

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta} \propto \left\langle \frac{\partial\left(-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})\right)}{\partial \theta} \right\rangle_{p(\mathbf{h}|\mathbf{x}, \boldsymbol{\theta}) p_D(\mathbf{x})} - \left\langle \frac{\partial\left(-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})\right)}{\partial \theta} \right\rangle_{p(\mathbf{x}, \mathbf{h}|\boldsymbol{\theta})}. \tag{4.4}$$

The first term computes the expectation of the partial derivative of the negative energy with respect to the parameter under the posterior distribution of the hidden units with the visible units fixed to training samples from the empirical, or data, distribution. The second term is the expectation of the same quantity under the model distribution represented by the Boltzmann machine. It is obvious that computing these terms exactly is intractable. Both terms require evaluating the partial derivative of the negative energy exponentially many times.

Hence, we must resort to estimating them with various techniques instead of trying to compute them exactly. The first choice is to use a technique of sampling to collect

samples from those intractable distributions to estimate the statistics. The other possibility is to approximate, for instance, the true posterior distribution over the hidden units given the state of the visible units with a simpler distribution. In Section 4.3, we briefly describe the underlying ideas of these two approaches and how they can be applied to estimating parameters of a Boltzmann machine.

There is yet another formulation of training Boltzmann machines that leads to exactly the same update rules in Eq. (4.4). This formulation is based on minimizing the negative KL-divergence between the data distribution and the model distribution, which is defined by

$$-\mathrm{KL}(P_0 \| P_\infty) = -\sum_{\mathbf{x}} \log \frac{P_0(\mathbf{x})}{P_\infty(\mathbf{x})} P_0(\mathbf{x}) \approx \langle \log P_\infty(\mathbf{x}) \rangle_{P_0} - \langle \log P_0(\mathbf{x}) \rangle_{P_0} , \tag{4.5}$$

where the shorthand notations P_0 and P_∞ are, respectively, defined by

$$\begin{aligned} P_0 =& p\left(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta}\right) p_D\left(\mathbf{x}\right) \\ P_\infty =& p\left(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}\right) . \end{aligned}$$

A reason for this notation is related to interpolating between the data and model distributions, which will be made more clear in Section 4.4.2. The last term, corresponding to the entropy of the data distribution, does not depend on the parameters $\boldsymbol{\theta}$ and can be safely ignored.

It is easy to see that Eq. (4.5) is equivalent to Eq. (4.3), since

$$\langle \log P_\infty(\mathbf{x}) \rangle_{P_0} \approx \frac{1}{N} \sum_{n=1}^{N} \log \sum_{\mathbf{h}} p(\mathbf{x}^{(n)}, \mathbf{h} \mid \boldsymbol{\theta})$$

which is exactly the marginal log-likelihood of Eq. (4.3). This fact will become useful when we describe an efficient learning procedure, called minimizing contrastive divergence (Hinton, 2002), later for a structurally-restricted family of Boltzmann machines.

4.1.1 Transformation Invariance and Enhanced Gradient

It is interesting to notice that an addition of any constant to the energy function of a Boltzmann machine does *not* alter the probability distribution it defines over the state space. This can be easily verified by the fact that the new normalization constant $\tilde{Z}(\boldsymbol{\theta})$ after adding a constant to the energy is just the original normalization constant $Z(\boldsymbol{\theta})$ multiplied by the exponential function of the added constant, assuming that the

constant is independent of the states of both $\mathbf{x}$ and $\mathbf{h}$:

$$\begin{aligned}\tilde{Z}(\boldsymbol{\theta}) &= \sum_{\mathbf{x}} \sum_{\mathbf{h}} \exp\left\{-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) + C\right\} \\ &= \exp\left\{C\right\} \sum_{\mathbf{x}} \sum_{\mathbf{h}} \exp\left\{-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})\right\} \\ &= \exp\left\{C\right\} Z(\boldsymbol{\theta}).\end{aligned}$$

With this property in mind, we may now investigate a case where the states of some units are transformed. To make equations uncluttered, in this section, we do not distinguish between visible and hidden units, and use $\mathbf{x}$ to denote all units. Also, instead of $p + q$ units, we will use p solely to indicate the number of units.

A bit-flipping transformation $\tilde{x}_i$ of unit $x_i \in \{0, 1\}$ is defined by an indicator variable $f_i \in \{0, 1\}$ such that

$$\tilde{x}_i = x_i^{1-f_i} \left(1 - x_i\right)^{f_i}.$$

In other words, the i-th unit of a Boltzmann machine is *flipped*, if f_i is set to 1. Given the states $\mathbf{x}$ of the units, the bit-flipping transformation by $\mathbf{f}$ results in $\tilde{\mathbf{x}}$.

Let us assume that we are given a Boltzmann machine $\mathcal{B}(0)$ with the parameters $\boldsymbol{\theta}$. It was shown by Cho et al. (2013) that there exists a Boltzmann machine $\mathcal{B}(\mathbf{f})$ parameterized by $\tilde{\boldsymbol{\theta}}$ that assigns the probability to the state $\tilde{\mathbf{x}}$ transformed by $\mathbf{f}$ which is equivalent to the probability assigned by the Boltzmann machine $\mathcal{B}(0)$ to the original state $\mathbf{x}$. In other words,

$$E(\tilde{\mathbf{x}} \mid \tilde{\boldsymbol{\theta}}) = E(\mathbf{x} \mid \boldsymbol{\theta}) + C,$$

with the following re-parameterization:

$$\tilde{w}_{ij} = (-1)^{f_i + f_j} w_{ij}, \tag{4.6}$$

$$\tilde{b}_i = (-1)^{f_i} \left(b_i + \sum_j f_j w_{ij}\right). \tag{4.7}$$

A direct implication of this transformation-invariance property is that learning parameters by maximizing the log-likelihood in Eq. (4.3) is dependent on the data representation. In other words, completely different behaviors will be observed when training a Boltzmann machine on the same data, however, with different representations. Especially, in Publication I it is empirically shown that certain representations of the same dataset are favored over other representations. For instance, a sparse data set can easily be learned by a Boltzmann machine with the stochastic gradient method, while its inverted form, a dense version of the same data, cannot be learned as easily.

This phenomenon becomes more clear, if we understand that there are exponentially many possible update directions each time with respect to the number of units.

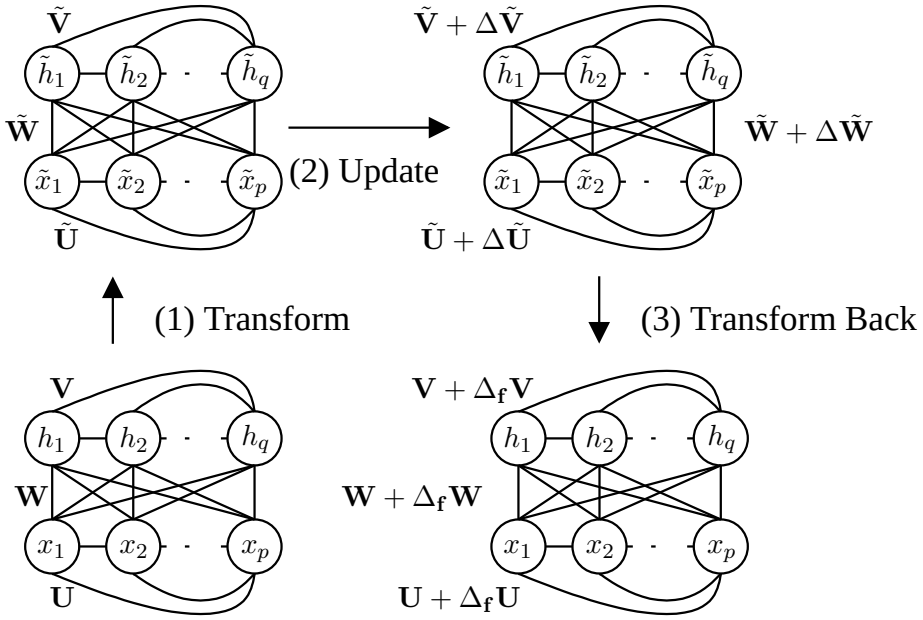


Figure 4.2. Illustration of the three-step update procedure consisting of (1) transforming, (2) updating and (3) transforming back. $\Delta_{\mathbf{f}}\boldsymbol{\theta}$ denotes the change in $\boldsymbol{\theta}$ dependent on the transformation $\mathbf{f}$.

In fact, if a transformation is chosen to be other than the bit-flipping transformation, there are potentially infinitely many update directions available, as will be discussed later.

With a given bit-flipping transformation $\mathbf{f}$, we first *transform* the Boltzmann machine according to Eqs. (4.6)–(4.7). The transformed model is *updated* by one step of the stochastic gradient method, and then, is *transformed back*. A simple illustration of how this three-step update is performed is given in Fig. 4.2.

Interestingly, this approach results in a valid steepest-ascent update rule for each parameter that *depends on* the transformation $\mathbf{f}$:

$$w_{ij} \leftarrow w_{ij} + \eta \left[\mathrm{Cov}_{\mathrm{d}}(x_i, x_j) - \mathrm{Cov}_{\mathrm{m}}(x_i, x_j) + \right. \\ \left. \left(\langle x_i \rangle_{\mathrm{dm}} - f_i \right) \nabla b_j + \left(\langle x_j \rangle_{\mathrm{dm}} - f_j \right) \nabla b_i \right], \tag{4.8}$$

$$b_i \leftarrow b_i + \eta \left(\nabla b_i - \sum_j f_j \nabla_{\mathbf{f}} w_{ij} \right), \tag{4.9}$$

where we used the following shorthand notations

$$\nabla_{\mathbf{f}} w_{ij} = \langle x_i x_j \rangle_{\mathrm{d}} - \langle x_i x_j \rangle_{\mathrm{m}} - f_i \nabla b_j - f_j \nabla b_i,$$
$$\langle x_i \rangle_{\mathrm{dm}} = \frac{1}{2} \left(\langle x_i \rangle_{\mathrm{d}} + \langle x_i \rangle_{\mathrm{m}} \right),$$

and the covariance between two units with respect to the distribution P is defined by

$$\mathrm{Cov}_P \left(x_i, h_j \right) = \langle x_i h_j \rangle_P - \langle x_i \rangle_P \langle h_j \rangle_P .$$

As defined earlier in Section 2.3.2, d and m correspond to the data and model distributions, respectively.

All these 2^p update directions, in the case of the bit-flipping transformation, are valid so that they will increase the log-likelihood. However, it may not be the case that all such directions will lead to the same solution, considering that the log-likelihood function in Eq. (4.3) is a highly non-concave function. For instance, one direction may lead to the solution that is superior, in terms of log-probabilities of test samples, to those solutions reached by other update directions.

To alleviate this problem, the author of this thesis together with two co-authors, in Publication I and Publication II, proposed to use the weighted sum of all those directions. We define the weight $\gamma_{\mathbf{f}}$ of each gradient update dependent on the transformation $\mathbf{f}$ by

$$\gamma_{\mathbf{f}} = \prod_{i=1}^{p} \langle x_i \rangle_{\text{dm}}^{f_i} \left(1 - \langle x_i \rangle_{\text{dm}}\right)^{1-f_i},$$

where

$$\langle x_i \rangle_{\text{dm}} = \frac{1}{2} \left(\langle x_i \rangle_{\text{d}} + \langle x_i \rangle_{\text{m}} \right).$$

The weighted sum of exponentially many transformation-dependent update directions, called the enhanced gradient, is then

$$\nabla_e w_{ij} = \text{Cov}_{\text{d}}(x_i, x_j) - \text{Cov}_{\text{m}}(x_i, x_j), \tag{4.10}$$

$$\nabla_e b_i = \nabla b_i - \sum_j \langle x_j \rangle_{\text{dm}} \nabla_e w_{ij}, \tag{4.11}$$

where we use ∇_e to distinguish the enhanced gradient from the conventional gradient. It must be noticed that the enhanced gradient is *not* dependent on the transformation $\mathbf{f}$, making it invariant to the bit-flipping transformation.

Empirical evidence has been provided in Publication I that the enhanced gradient is more robust to the choice of hyper-parameters, such as a learning rate and its scheduling, and extracts more discriminative features, when used for training an restricted Boltzmann machine. A restricted Boltzmann machine, which is a structurally-restricted variant of a Boltzmann machine, will be discussed in detail later.

The bit-flipping transformation introduced in Publication I and Publication II have inspired others since. For instance, Montavon and Müller (2012) showed that centering each unit helps avoiding a difficulty in training a deep Boltzmann machine which is another structurally-restricted variant of a Boltzmann machine. Instead of the bit-flipping transformation, they used a shifting transformation such that

$$\tilde{x}_i = x_i - \beta_i.$$

Cho et al. (2011), which was recently published as Publication VI, also proposed independently the same transformation for Gaussian visible units.

With the shifting transformation $\boldsymbol{\beta}$, one can construct the equivalent Boltzmann machine with the following parameters

$$\begin{aligned}\tilde{w}_{ij} &= w_{ij} \\ \tilde{b}_i &= b_i + \sum_j w_{ij}\beta_j.\end{aligned}$$

However, in this case, there are *infinitely* many possible update directions at a time, and one needs to choose a single update direction, where

$$\beta_i = \langle x_i \rangle_{\text{dm}}$$

was proposed in Publication VI, and Montavon and Müller (2012) proposed to use

$$\beta_i = \langle x_i \rangle_{\text{d}} .$$

A similar idea was also proposed by Tang and Sutskever (2011), but they applied the transformation only to visible units.

4.2 Boltzmann Machines with Hidden Units are Deep

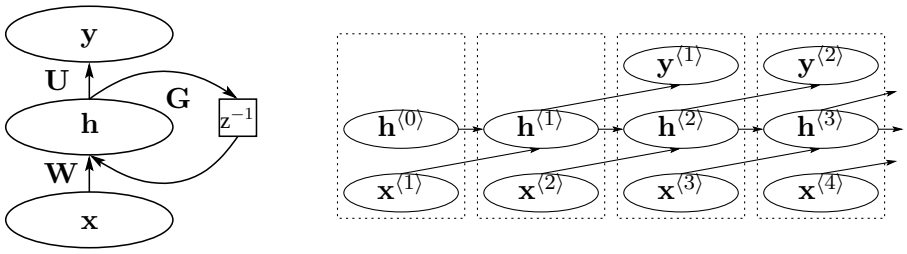
Before discussing further about Boltzmann machines, in this section, we first provide an intuitive explanation as to why Boltzmann machines, especially with hidden units, are considered deep. This is done by showing that a recurrent neural network, after being unfolded over time, is deep and an equivalent recurrent neural network can be constructed for each Boltzmann machine.

A recurrent neural network consists of input, output and hidden units as usual with any other types of neural networks considered so far in this thesis. It, however, differs from, for instance, a multilayer perceptron introduced earlier that there exist *feedback* edges. Hence, a state $\left[\mathbf{x}_{\langle t \rangle}^\top, \mathbf{h}_{\langle t \rangle}^\top\right]^\top$ of the network at time t does not only depend on the given input $\mathbf{x}_{\langle t \rangle}$, but it also depends on the state $\left[\mathbf{x}_{\langle t-1 \rangle}^\top, \mathbf{h}_{\langle t-1 \rangle}^\top\right]^\top$ of the all units at the previous time step $t-1$. More comprehensive discussions on general recurrent neural network and recent advances in their learning algorithms can be found in (Bengio et al., 2013; Sutskever, 2013; Haykin, 2009, and references therein).

In this section, we first discuss why a recurrent neural network with hidden units is *deep*. Then, we describe how a Boltzmann machine with hidden units may be considered a recurrent neural network. Based on these observations, we conclude that Boltzmann machines with hidden units are *deep* neural networks.

4.2.1 Recurrent Neural Networks with Hidden Units are Deep

Let us start with a simplified recurrent neural network, at each time t, consisting of an input layer, a single nonlinear hidden layer and a output layer. Each hidden



(a) Recurrent Neural Network (b) Unfolded over Time

Figure 4.3. (a) A simple recurrent neural network. z^{-1} is a unit delay operator. (b) The same recurrent neural network has been unfolded over time. Each layer is grouped by a rectangle with dashed lines.

layer receives signal from the hidden layer at the previous time $t-1$. A set $\boldsymbol{\theta}$ of parameters consists of weights from the input layer to the hidden layer ($\mathbf{W}$), those from the hidden layer to the output layer ($\mathbf{U}$) and those from the previous hidden layer to the current hidden layer ($\mathbf{G}$). Every unit has its own bias term. See Fig. 4.3(a) for the illustration of this network.

The goal of this network is to learn to predict a sequence of labels given a sequence of input vectors. It is also possible to train it to predict the next input given the sequence of previous inputs. For instance, Sutskever et al. (2011) showed that a recurrent neural network was able to learn and generate an arbitrary sequence of text once the network was trained to predict the next input.

Now, let us unfold the described recurrent neural network in time, starting from $t = 0$ assuming that that the activations of the input and hidden units are fixed to zero initially. We will try to construct an infinite-depth multilayer perceptron (MLP) from the unfolded recurrent neural network.

If we denote the input, output and hidden layers of the recurrent neural network at time t by $\mathbf{x}^{\langle t \rangle}$, $\mathbf{y}^{\langle t \rangle}$ and $\mathbf{h}^{\langle t \rangle}$, the l-th layer of the infinite-depth MLP is constructed by concatenating $\mathbf{x}^{\langle t \rangle}$, $\mathbf{y}^{\langle t-2 \rangle}$ and $\mathbf{h}^{\langle t-1 \rangle}$. Then, each consecutive intermediate layers t and $t+1$ are connected by a sparse weight matrix, where the connections from $\mathbf{y}^{\langle t-2 \rangle}$ to $\mathbf{h}^{\langle t \rangle}$ and from $\mathbf{h}^{\langle t-1 \rangle}$ to $\mathbf{x}^{\langle t+1 \rangle}$ are *not* present, but both $\mathbf{h}^{\langle t-1 \rangle}$ to $\mathbf{h}^{\langle t \rangle}$ ($\mathbf{G}$) and to $\mathbf{y}^{\langle t-1 \rangle}$ ($\mathbf{U}$) and $\mathbf{x}^{\langle t \rangle}$ to $\mathbf{h}^{\langle t \rangle}$ ($\mathbf{W}$) are. In this MLP, unlike how MLPs were described in Section 3.1 earlier, not only the bottom-most and top-most layers, but all intermediate layers also have input and output units, and this network shares the weights parameters of each layer. See Fig. 4.3(b) for the structure of this unfolded network.

This unfolded recurrent neural network satisfies, like a plain MLP, the two conditions described in Section 2.4 in a non-trivial way.

First, the network can be easily extended, just like an MLP, by adding one or more hidden layers. However, in the case of a recurrent neural network, this addition grows the size of each layer instead of the number of layers, when viewed as the

time-unfolded network. This is an obvious consequence from the fact that the time-unfolded network already has infinitely many layers.

The second condition is trivially satisfied as each layer is parameterized. One can modify the backpropagation algorithm (see Section 3.4) used for training conventional MLPs to train all the parameters.

Hence, we consider any recurrent neural network with hidden units a *deep* neural network.

4.2.2 Boltzmann Machines are Recurrent Neural Networks

Similarly, let us consider a fully-connected Boltzmann machine (BM) trained on a training set consisting of vectors of samples augmented by their labels. Each label is considered to be transformed into a binary vector using a 1-of-K coding. That is, the visible units are divided into those $\mathbf{x}$ that correspond to input components and those $\mathbf{y}$ that correspond to label components[2]. Furthermore, let us assume that there is no edge between the sets of the input and label units and also no edge inside those sets. Then, we may rewrite the energy function of the BM in Eq. (4.1) into

$$-E(\mathbf{x}, \mathbf{y}, \mathbf{h} \mid \boldsymbol{\theta}) = \mathbf{a}^\top \mathbf{x} + \mathbf{b}^\top \mathbf{y} + \mathbf{c}^\top \mathbf{h} + \mathbf{x}^\top \mathbf{W} \mathbf{h} + \mathbf{y}^\top \mathbf{U} + \frac{1}{2} \mathbf{h}^\top \mathbf{V} \mathbf{h}. \quad (4.12)$$

Once the BM is trained, we can obtain a sequence $\left(\mathbf{y}_{\langle 0 \rangle}, \mathbf{y}_{\langle 1 \rangle}, \ldots, \mathbf{y}_{\langle \infty \rangle}\right)$ of vectors of the states of label units given a visible sample by simulating the Boltzmann machine. To do so, at each time t, we need to have the previous states of the hidden units $\mathbf{h}_{\langle t-1 \rangle}$ and label units $\mathbf{y}_{\langle t-1 \rangle}$.

Given these previous states, we first compute the states of the hidden units at time t by replacing *one* unit. The state of the unit h_{j_t} can be sampled from

$$p(h_{j_t} = 1 \mid \mathbf{x}, \mathbf{y}_{\langle t-1 \rangle}, \mathbf{h}_{\langle t-1 \rangle}, \boldsymbol{\theta}) = \phi \left(\sum_i w_{ij_t} x_i + \sum_l u_{lj_t} y'_l + \sum_{k \neq j_t} v_{kj_t} h'_k + b_{j_t} \right),$$

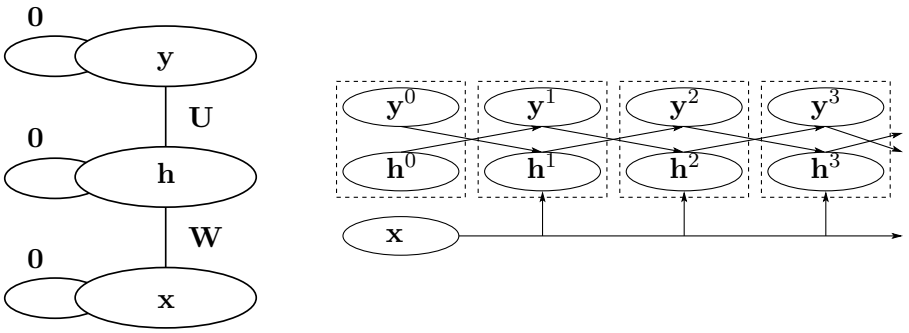
where y'_l and h'_k are the l-th component of $\mathbf{y}_{\langle t-1 \rangle}$ and the k-th component of $\mathbf{h}_{\langle t-1 \rangle}$, respectively. Subsequently with the new hidden state $\mathbf{h}_{\langle t \rangle}$, we replace a *single* unit y_{l_t} to get the new label state $\mathbf{y}_{\langle t \rangle}$ similarly by sampling from

$$p(y_{l_t} = 1 \mid \mathbf{x}, \mathbf{y}_{\langle t-1 \rangle}, \mathbf{h}_{\langle t \rangle}, \boldsymbol{\theta}) = \phi \left(\sum_i w_{ij_t} x_i + \sum_{k \neq l_t} u_{kl_t} y'_k + \sum_j v_{lj} h'_j + a_{l_t} \right).$$

The indices j_t and l_t may be chosen arbitrarily at each time t.

It is easy to see that this way of viewing the simulation of the BM is equivalent to performing a feedforward pass through time in a recurrent neural network. One

[2] A practical description of how a Boltzmann machine trained on samples augmented with their labels will be presented in more detail in Section 5.2.1. Here, this model is only used to illustrate how an equivalent recurrent neural network can be built for a Boltzmann machine.



(a) Original Boltzmann Machine (b) Equivalent Recurrent Neural Network

Figure 4.4. (a) A Boltzmann machine with some edges fixed to 0 having visible units that correspond to a label. (b) The equivalent recurrent neural network unfolded over time. Each layer is grouped by a rectangle with dashed lines.

difference to the recurrent neural network described in Section 4.2.1 is that the visible units are fixed to a sample over time.

Also, it is obvious that there are multiple computation paths from the visible units via both the hidden and label units to the label units at some time $t > 1$. Furthermore, all the parameters in the BM, or in the equivalent recurrent neural network, are trainable in the sense that they are estimated by maximizing the marginal log-likelihood in Eq. (4.3). Based on these observations, we safely consider Boltzmann machines with hidden units *deep*.

In Fig. 4.4, an illustration of a Boltzmann machine with label units and its equivalent time-unfolded recurrent neural network is shown. Note that for simplicity in the figure we further assumed that there is no edge among hidden units.

4.3 Estimating Statistics and Parameters of Boltzmann Machines

In order to compute the gradient of the marginal log-likelihood of a fully-connected Boltzmann machine in Eq. (4.3), we must be able to compute the statistics under the following intractable distributions[3]:

1. $p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta})$: the posterior distribution of $\mathbf{h}$ with the fixed state of $\mathbf{x}$
2. $p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})$: the joint distribution modeled by the Boltzmann machine

In this section, we briefly discuss two approaches that can be used to compute the statistics under these distributions approximately.

[3]Here we say that a distribution is *intractable*, when there is no explicit analytical form of the probability mass/density function available, or the statistics of the distribution cannot be computed analytically.

4.3.1 Markov Chain Monte Carlo Methods for Boltzmann Machines

When one considers estimating the expectation of a function f over a distribution $\tilde{p}$, the most obvious approach one can think of is to collect a large number of samples from the distribution and use them to compute f, such that

$$\mathbb{E}_{\tilde{p}(\mathbf{x})}\left[f(\mathbf{x})\right] = \sum_{\mathbf{x}} f(\mathbf{x})\tilde{p}(\mathbf{x}) \approx \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{x}_{\langle t\rangle}),$$

where $\left\{\mathbf{x}_{\langle t\rangle}\right\}_{t=1}^{T}$ is a set of T samples collected from $\tilde{p}$. This effectively reduces the problem of computing the statistics of an intractable distribution to the problem of collecting samples from the distribution.

A Markov Chain Monte Carlo (MCMC) method is a general framework on which sequential sampling can be built to collect samples from a distribution (see, e.g., Neal, 1993; Mackay, 2002, for comprehensive review on using MCMC sampling in probabilistic models). This method is based on the idea that sampling from a target distribution is equivalent to simulating a *Markov chain* which has the target distribution as its unique stationary distribution.

A Markov chain consists of a set of states, for instance, all possible states of the units of a Boltzmann machines $\{\mathbf{x}\}$, and a transition probability $\mathcal{T}(\tilde{\mathbf{x}} \mid \mathbf{x})$ of the new state $\tilde{\mathbf{x}}$ from the current state $\mathbf{x}$ which only depends on the current state. Given a probability distribution $p^{\langle t\rangle}$ over the states at time t, the probability distribution $p^{\langle t+1\rangle}$ can be fully specified by

$$p^{\langle t+1\rangle}(\mathbf{x}) = \sum_{\tilde{\mathbf{x}}} p^{\langle t\rangle}(\tilde{\mathbf{x}})\mathcal{T}(\mathbf{x} \mid \tilde{\mathbf{x}}).$$

In order to be used as a sampler of a target distribution $\tilde{p}$, the stationary distribution $p^{\langle\infty\rangle}$ of this chain at the limit of $t \to \infty$ must coincide with the target distribution $\tilde{p}$, that is,

$$p^{\langle t\rangle}(\mathbf{x}) = \tilde{p}(\mathbf{x}) \text{ as } t \to \infty.$$

The stationary distribution, or equivalently the target distribution, must be invariant with respect to the chain. In other words, any further simulation of the Markov chain does not alter the distribution of the states once the stationary distribution has been reached. That is,

$$p^{\langle\infty\rangle}(\mathbf{x}) = \sum_{\tilde{\mathbf{x}}} p^{\langle\infty\rangle}(\tilde{\mathbf{x}})\mathcal{T}(\mathbf{x} \mid \tilde{\mathbf{x}}).$$

Due to this property of invariance, the stationary distribution is often referred to as the *equilibrium distribution*.

This condition of invariance is usually met by designing a transition probability, or operator, $\mathcal{T}$ to satisfy the detailed balance:

$$\mathcal{T}(\mathbf{x}_0 \mid \mathbf{x}_1)\tilde{p}(\mathbf{x}_1) = \mathcal{T}(\mathbf{x}_1 \mid \mathbf{x}_0)\tilde{p}(\mathbf{x}_0),$$

for arbitrary $\mathbf{x}_0$ and $\mathbf{x}_1$. Satisfying this implies that the target distribution $\tilde{p}$ is invariant under the Markov chain.

Furthermore, the chain must be ergodic, which implies that the stationary distribution is reachable regardless of the initial distribution $p^{\langle 0 \rangle}$:

$$p^{\langle t \rangle}(\mathbf{x}) = \tilde{p}(\mathbf{x}) \text{ as } t \to \infty, \forall p^{\langle 0 \rangle}. \tag{4.13}$$

Intuitively, it will be impossible to use the chain to collect samples from a target distribution, if the simulation of the chain converges to one or more distributions that are *not* the target distribution, depending on the initial distribution.

Using this property, we can use a Markov chain to collect samples from a target distribution. We start with an initial distribution which usually gives to a single state the whole probability ($= 1$) and all the others zero probability. Because the chain is ergodic, the simulation of the chain will eventually end up in the stationary distribution which is equivalent to the target distribution. From there on, we record the sequence of states the simulation visits. The invariance of the distribution on the chain ensures that the state transitions, or simulation, will not move away from the stationary distribution. The recorded list of states will be the samples from the target distribution, albeit not necessarily independent samples.

One representative example of MCMC methods is Metropolis-Hastings method (Hastings, 1970). In the Metropolis-Hastings method, we assume that the target distribution is computable up to the normalization constant, and introduce a proposal distribution Q from which we can readily and efficiently sample.

Let us define a transition probability $\mathcal{T}$ as

$$\mathcal{T}(\tilde{\mathbf{x}} \mid \mathbf{x}) = Q(\tilde{\mathbf{x}} \mid \mathbf{x})\alpha(\tilde{\mathbf{x}}, \mathbf{x}),$$

where the acceptance probability $\alpha(\tilde{\mathbf{x}}, \mathbf{x})$ is defined as

$$\alpha(\tilde{\mathbf{x}}, \mathbf{x}) = \min\left(1, \frac{p^*(\tilde{\mathbf{x}})q(\mathbf{x} \mid \tilde{\mathbf{x}})}{p^*(\mathbf{x})q(\tilde{\mathbf{x}} \mid \mathbf{x})}\right).$$

We used p^* to denote an unnormalized probability such that $\tilde{p}(\mathbf{x}) = \frac{p^*(\mathbf{x})}{\int p^*(\mathbf{x}')\mathrm{d}\mathbf{x}'}$.

With this, we can iteratively sample from the target distribution $\tilde{p}$ by

(Proposal) Sample $\mathbf{x}'$ from $Q(\mathbf{x} \mid \mathbf{x}_{\langle t \rangle})$

(Acceptance)

(Accept) Set $\mathbf{x}_{\langle t+1 \rangle} = \mathbf{x}'$ with the probability $\alpha(\mathbf{x}', \mathbf{x}_{\langle t \rangle})$

(Reject) Otherwise, set $\mathbf{x}_{\langle t+1 \rangle} = \mathbf{x}_{\langle t \rangle}$.

With some mild assumptions on the proposal distribution Q, the distribution of $\mathbf{x}_{\langle t \rangle}$ will converge to the target distribution $\tilde{p}(\mathbf{x})$ using the Metropolis-Hastings method.

Gibbs Sampling

One particular form of the Metropolis-Hastings method is Gibbs sampling in which we are more interested, in the case of Boltzmann machines. Gibbs sampling can be derived by using the conditional distribution of a single component k given the state of all other components, denoted by $-k$, as the proposal distribution Q. That is,

$$Q(\tilde{\mathbf{x}} \mid \mathbf{x}) = p(\mathbf{x}_k \mid \mathbf{x}_{-k}),$$

where $\tilde{\mathbf{x}}_{-k} = \mathbf{x}_{-k}$. This choice makes the acceptance probability α always one so that every sample is accepted.

With this proposal distribution and the property of accepting always, the Gibbs sampling can be implemented simply by repeatedly collecting a new sample from

$$\mathbf{x}' = \left[x_1, x_2, \cdots, x'_k, \cdots, x_p\right],$$

where x'_k is sampled by

$$x'_k \sim x_k \mid x_1, \cdots, x_{k-1}, x_{k+1}, \cdots, x_p,$$

and p is the dimensionality of $\mathbf{x}$, while changing k in a predefined schedule. It is usual to simply cycle k through all the components sequentially.

Because the conditional probability of each unit in a Boltzmann machine (Eq. (2.31)) is well defined and easily evaluated, it is natural to use the Gibbs sampling to evaluate the statistics of the distributions $p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta})$ and $p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})$.

However, this sampling-based approach, especially with the Gibbs sampling, may not be practical due to several reasons. Firstly, it is usually difficult, if not impossible, to determine the convergence of the Markov chain, which leaves collecting as many samples as possible the only option. Secondly, the Gibbs sampling or any other Metropolis-Hastings method with a local proposal distribution in which each consecutive sampling steps can only make a local change, will require a large amount of steps to explore the whole distribution, especially when there are multiple modes in it.

The latter one is especially true when one tries to sample from the joint distribution $p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})$. The joint distribution, and consequently the marginal distribution of $\sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})$, is after all optimized to have multiple modes that correspond to, for instance, different classes or clusters of a given training set. Unless the training set consists of samples from a unimodal distribution, the joint distribution $p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})$ that models the training set well enough will always have more than one modes.

Other than Gibbs Sampling: Parallel Tempering

The major problem of the Gibbs sampling is that it often fails to explore modes lying far away or isolated in the distribution. In many cases, the Gibbs sampling chain is stuck at one mode and is unable to escape from it. In other words, the convergence of the Gibbs chain to the stationary distribution might take too long, or infinitely long.

In order to prevent this problem, several approaches have been proposed to improve the mixing property of the Gibbs sampling in collecting samples from Boltzmann machines. Salakhutdinov (2009) proposed to use tempered transition (Neal, 1994), while a similar, but not identical sampling method based on tempered chains called parallel tempering was proposed in Publication III and independently by Desjardins et al. (2010b,a). Instead of introducing a new sampling scheme, Tieleman and Hinton (2009) described a method of using *fast* parameters, especially, in the case of training restricted Boltzmann machines which will be discussed later.

Here, we will mainly discuss one of those proposed approaches, called *parallel tempering*, which was empirically found to improve the generative performance.

Parallel tempering was introduced by Swendsen and Wang (1986) under the name of a replica Monte Carlo simulation applied to an Ising model which is equivalent to the fully-visible Boltzmann machine (see Section 2.3.2). Geyer (1991) later presented applying parallel chaining of MCMC sampling based on the speed of mixing of samples across parallel chains to the maximum likelihood estimator.

Let us first introduce an inverse temperature β which was assumed to be fixed to 1 earlier for a Boltzmann machine. The joint probability of a Boltzmann machine of the inverse temperature β is defined by

$$p_\beta(\mathbf{x}, \mathbf{h}) = p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}_\beta) = \frac{1}{Z(\boldsymbol{\theta}_\beta)} \exp\left\{-\beta E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})\right\}.$$

Let us call the distribution represented by the Boltzmann machine with the inverse temperature β a *tempered distribution* with β.

Parallel tempering can be understood as a composite of two transition operators, if we only consider two tempered distributions with $\beta = 1$ (model distribution) and $\beta = \beta_q < 1$ (proposal distribution) for now. Since the concatenation of valid MCMC transitions satisfies the properties of a valid MCMC method, parallel tempering as a composite of two operators is a valid MCMC method.

The first transition operator, which performs a single-step Gibbs sampling on the model distribution, is applied to the current sample $\mathbf{x}$ and results in a new sample $\mathbf{x}'$. Then the second transition operator performs a single step of the Metropolis-Hastings sampling from the new sample $\mathbf{x}'$.

In the second transition operator, the proposal distribution is the tempered distribution with the inverse temperature β_q. The sample $\mathbf{x}''$ from the proposal distribution

will be obtained by a yet another Gibbs sampling on the tempered distribution. Then, $\mathbf{x}''$ is accepted with the probability

$$p_{\text{swap}}(\mathbf{x}', \mathbf{x}'') = \min\left(1, \frac{p_1^*(\mathbf{x}'')p_{\beta_q}^*(\mathbf{x}')}{p_1^*(\mathbf{x}')p_{\beta_q}^*(\mathbf{x}'')}\right), \tag{4.14}$$

where p^* indicates that it is an *unnormalized* probability. If accepted, we keep $\mathbf{x}''$ as the new sample and otherwise, $\mathbf{x}'$ is kept.

The reason why the acceptance probability in Eq. (4.14) was denoted a *swap* probability is that once we switch the roles between the model distribution and the proposal distribution, we can see that this probability, in essence, decides whether the samples from the two distributions be swapped or not.

Let us now assume that we have a series of $N + 1$ tempered transitions such that $\beta_0 = 0 < \cdots < 1 = \beta_N$. Then we can apply the above transition operator to each consecutive pairs of the tempered distribution. That is, the sampling from the proposal distribution is replaced by the same transition operator applied to the pair of tempered distributions immediately below. This *chaining* continues down to the bottom pair.

Since the top tempered distribution with the inverse temperature $\beta = 1$ corresponds to the model distribution, the samples that stay on it are the ones from the model distribution. We can use them to compute the statistics of the model distribution of a Boltzmann machine. When we use the parallel tempering with the stochastic approximation procedure (see Section 4.3.3), we apply the above transition operator a few times at each update and use the samples from the top chain to compute the gradient while maintaining the samples of the other chains as well.

When the inverse temperature is 0, the tempered distribution is completely flat, meaning that every state is assigned the same probability $\frac{1}{2^{p+q}}$. Under this distribution, one can draw an exact sample from the stationary distribution without even running any MCMC sampling chain. One can simply draw the state of each unit from a Bernoulli random variable with its mean 0.5.

In other words, the tempered distributions with low β's tend to be smooth overall, and the Gibbs sampling on those distributions is less prone to being trapped in a single, or only few, modes out of all modes in the distributions. As β approaches 1 (model distribution), it becomes more difficult for the plain Gibbs sampling to explore the whole state space efficiently.

In this regard, the obvious advantage of the parallel tempering when compared to the Gibbs sampling is that the parallel tempering can avoid being trapped in a single mode. This is possible, since the samples from the Gibbs sampling chains in the tempered distributions with smaller β's are less prone to becoming highly correlated with each other. A sample in a lower chain, that is far away from the mode the current

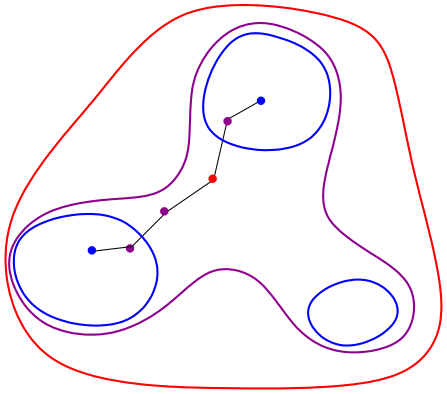


Figure 4.5. Illustration of how PT sampling could avoid being trapped in a single mode. The red, purple, and blue curves and dots indicate distributions and the samples from the distributions with the high, medium, and cold temperatures, respectively. Each black line indicates a single sampling step. Reprinted from (Cho, 2011).

sample at the top chain is trapped in, may be swapped to become a new sample at the top chain, and this helps preventing having a sequence of highly correlated samples.

This behavior of exploring other modes easily is illustrated in Fig. 4.5.

In Publication III, it is empirically shown that if the same number of Gibbs steps is allowed, using parallel tempering to compute the statistics of the model distribution results in a better generative model compared to the plain Gibbs sampling, when an RBM was trained.

Similarly, the tempered transition and the coupled adaptive simulated tempering (Salakhutdinov, 2010) are all based on using the tempered distributions. All these methods are superior to the plain Gibbs sampling in a sense that the whole state space can be explored more easily.

4.3.2 Variational Approximation: Mean-Field Approach

A variational approximation is another way of computing the intractable statistics of a probability distribution such as the posterior distribution of a Boltzmann machine over its hidden units. Here we discuss how the variational approximation, which has already been discussed briefly earlier in Section 3.2.2, can be applied to computing the statistics of a Boltzmann machine.

Let us start by restating how the marginal log-likelihood in general can be decomposed into two terms, which was presented in Eqs. (2.24)–(2.25):

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= \mathcal{L}_Q(\boldsymbol{\theta}) + \mathrm{KL}(Q \| P) \\ &\geq \mathbb{E}_Q \left[\log p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) \right] + \mathcal{H}(Q) \end{aligned} \tag{4.15}$$

Q and P are respectively an arbitrary distribution over hidden variables and the posterior distribution over the hidden variables given the states of visible, or observed, variables.

This same decomposition applies to the marginal log-likelihood of a Boltzmann machine with hidden units presented in Eq. (4.3). In other words, we can also in the case of Boltzmann machines maximize the lower bound $\mathcal{L}_Q(\boldsymbol{\theta})$ instead of maximizing the original marginal log-likelihood directly.

Let us assume that we use a fully factorized distribution $Q(\mathbf{h} \mid \boldsymbol{\theta}_Q)$ parameterized by $\boldsymbol{\theta}_Q$. By considering that each hidden unit can have either 0 or 1, we can use the following factorized Q proposed by Salakhutdinov (2009):

$$Q(\mathbf{h} \mid \boldsymbol{\theta}_Q) = \prod_{j=1}^{q} q(h_j), \tag{4.16}$$

where $q(h_j = 1) = \mu_j$ and μ_j's are the parameters of Q. This approach of using a fully factorized approximate posterior is often called a mean-field approximation.

By plugging Eq. (4.16) and Eq. (4.2) into Eq. (4.15), we can rewrite the lower bound $\mathcal{L}_Q(\boldsymbol{\theta})$ as

$$\begin{aligned}\mathcal{L}_Q(\boldsymbol{\theta}) = &\sum_{i=1}^{p} b_i x_i + \sum_{j=1}^{q} c_j \mu_j + \sum_{i=1}^{p} x_i \mu_j w_{ij} + \\ &\sum_{i=1}^{p} \sum_{j=i}^{p} x_i x_j u_{ij} + \sum_{i=1}^{q} \sum_{j=i}^{p} \mu_i \mu_j v_{ij} - \log Z(\boldsymbol{\theta}) \\ &- \sum_{j=1}^{p} \left(\mu_j \log \mu_j + (1 - \mu_j) \log(1 - \mu_j)\right). \end{aligned} \tag{4.17}$$

By maximizing this with respect to Q, or its parameters $\boldsymbol{\theta}_Q$, we can minimize the difference between Q and the true posterior distribution.

Maximization can be done simply by taking the partial derivative of $\mathcal{L}_Q$ with respect to each variational parameter μ_j and updating it according to the computed direction. By setting the partial derivative of $\mathcal{L}_Q$ with respect to μ_j to zero, we get the following fixed-point iteration:

$$\mu_j \leftarrow \phi\left(\sum_{i=1}^{p} w_{ij} x_i + \sum_{k=1, k \neq j}^{q} v_{kj} h_k + c_j\right), \tag{4.18}$$

where ϕ is a logistic sigmoid function.

Once $\boldsymbol{\theta}_Q$ converges after running the fixed-point iterations several times, we may use it not only for estimating the parameters as a part of a problem of maximizing the lower bound of the marginal log-likelihood, but also as an approximate posterior distribution of a given sample. For instance, one can use the variational parameters $\boldsymbol{\theta}_Q$ of each sample as a feature vector for another model.

Despite its advantages, such as easy parallelization and easy-to-check convergence, there is an important limitation in this approach. The limitation comes from the unimodality of the fully factorized form of the approximate posterior distribution Q.

Because all hidden units were assumed to be independent from each other, Q can only have a single mode. If the distribution approximated by Q has more than one mode, due to the property of the KL-divergence and the order[4] of Q and the true distribution P, the approximate distribution Q will tend to approximate one of those multiple modes in the true distribution P (see Murphy, 2012, Section 21.2.2 for more details).

This especially limits using the variational approximation for estimating the joint distribution $p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})$. As discussed earlier, it is highly likely that $p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})$ will be highly multimodal as learning continues, and the statistics estimated by the approximate distribution will *not* reflect the true statistics well. Furthermore, in the context of Boltzmann machines, this approximation will not work for the joint distribution since the parameters estimated by the gradient-based update will increase the KL-divergence between the approximate distribution and the true joint distribution due to the minus sign in front of $\log Z(\boldsymbol{\theta})$ in Eq. (4.17).

Hence, it is usual to use this variational approximation for estimating the statistics under the posterior distribution $p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta})$, while an approach based on MCMC methods is used to estimate the statistics under the joint distribution.

4.3.3 Stochastic Approximation Procedure for Boltzmann Machines

Using a naive stochastic gradient method described in Section 2.5, we can find the set of parameters that maximizes the marginal log-likelihood (4.3) or the variational lower bound (4.17) of a Boltzmann machine. One can simply repeat the following update to each parameter:

$$\begin{aligned}\theta^{\langle t+1 \rangle} &= \theta^{\langle t \rangle} + \eta_{\langle t \rangle} \left(\left\langle h(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}^{\langle t \rangle}) \right\rangle_{\mathrm{d}} - \left\langle h(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}^{\langle t \rangle}) \right\rangle_{\mathrm{m}} \right) \\ &= \theta^{\langle t \rangle} + \eta_{\langle t \rangle} \left(H_0 - H_\infty \right), \end{aligned} \tag{4.19}$$

where $\theta^{\langle t \rangle}$ and $\eta_{\langle t \rangle}$ are the parameter value and the learning rate at time t, and

$$h(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \frac{\partial \left(-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) \right)}{\partial \theta}.$$

$\eta_{\langle t \rangle}$ should decrease over time while satisfying Eqs. (2.37)–(2.38). Note that we used the following shorthand notations for simplicity:

$$\begin{aligned} H_0 &= \left\langle h(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}^{\langle t \rangle}) \right\rangle_{\mathrm{d}} \\ H_\infty &= \left\langle h(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}^{\langle t \rangle}) \right\rangle_{\mathrm{m}}. \end{aligned}$$

The first term H_0 can be computed quite efficiently by the variational approximation with a fixed number of training samples randomly collected from the training

[4]Note that the order of Q and P matters when their KL-divergence is computed, as the KL-divergence is *not* a symmetric measure.

set. Let $\left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\right\}$ be a set of randomly chosen samples from the training set, and let $\boldsymbol{\mu}^{(n)}$ be the variational parameters obtained by iteratively applying Eq. (4.18) to all hidden units conditioned on $\mathbf{x}^{(n)}$. Then,

$$H_0 \approx \frac{1}{N} \sum_{n=1}^{N} h\left(\mathbf{x}^{(n)}, \boldsymbol{\mu}^{(n)} \middle| \boldsymbol{\theta}^{\langle t \rangle}\right).$$

However, the problem is with the second term H_∞ which requires running a Gibbs sampling chain until the convergence. For instance, let us assume that we collected a finite number N_0 of samples $\left\{(\mathbf{x}^{(1)}, \mathbf{h}^{(1)}), \ldots, (\mathbf{x}^{(N_0)}, \mathbf{h}^{(N_0)})\right\}$ from the model distribution using the Gibbs sampling. Then,

$$H_\infty \approx \frac{1}{N_0} \sum_{n=1}^{N_0} h\left(\mathbf{x}^{(n)}, \mathbf{h}^{(n)} \middle| \boldsymbol{\theta}^{\langle t \rangle}\right).$$

The obvious problem is that it is difficult to choose or determine N_0. Furthermore, N_0 might be determined too large to be of any practical use.

A computationally efficient method to overcome this problem was proposed by Younes (1988). This algorithm, sometimes called *stochastic approximation procedure* (Salakhutdinov, 2009), does not run the Gibbs sampling chain, starting from random states until the convergence at each update.

Let $X^{\langle t \rangle} = \left\{\left(\mathbf{x}^{(1)}_{\langle t \rangle}, \mathbf{h}^{(1)}_{\langle t \rangle}\right), \ldots, \left(\mathbf{x}^{(N_0)}_{\langle t \rangle}, \mathbf{h}^{(N_0)}_{\langle t \rangle}\right)\right\}$ be a set of states of visible and hidden units. At time $t = 0$, $X^{\langle 0 \rangle}$ is initialized with random samples, or a randomly chosen subset of the training set. Then at each time t before updating parameters by Eq. (4.19), we obtain $X^{\langle t+1 \rangle}$ by applying the following transition to each sample a few times:

$$\left(\mathbf{x}^{(n)}_{\langle t+1 \rangle}, \mathbf{h}^{(n)}_{\langle t+1 \rangle}\right) \sim \mathcal{T}_{\boldsymbol{\theta}^{\langle t \rangle}}\left(\mathbf{x}, \mathbf{h} \middle| \mathbf{x}^{(n)}_{\langle t \rangle}, \mathbf{h}^{(n)}_{\langle t \rangle}\right),$$

where $\mathcal{T}_{\boldsymbol{\theta}^{\langle t \rangle}}$ is the transition probability of the Gibbs sampling on the Boltzmann machine parameterized by $\boldsymbol{\theta}^{\langle t \rangle}$. With the new set $X^{\langle t+1 \rangle}$ of samples, we compute H_∞ by

$$H_\infty \approx \frac{1}{N_0} \sum_{(\mathbf{x}, \mathbf{h}) \in X^{\langle t+1 \rangle}} h\left(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}^{\langle t \rangle}\right).$$

Simply put, this approach does not wait for the Gibbs sampling chain to converge to the equilibrium distribution. Rather, it performs only a few Gibbs sampling steps starting from the samples used during the last update, and use the new samples to compute the second term H_∞ of the gradient. This algorithm arises from the fact that if the parameters converge slowly to, for instance, $\boldsymbol{\theta}^*$, then $X^{\langle t \rangle}$ will converge to the equilibrium distribution of the Boltzmann machine parameterized by $\boldsymbol{\theta}^*$ in the limit of $t \to \infty$.

This approach was proposed independently for training a restricted Boltzmann machine by Tieleman (2008). Tieleman (2008) called this approach *persistent contrastive divergence* based on the similarity between this approach and an approach of minimizing contrastive divergence (see Section 4.4.2).

Although this approach is only a special case of a stochastic gradient method, we refer to this algorithm as a *stochastic approximation procedure* in order to distinguish it from a method that uses a randomly sampled subset of training samples to compute a gradient.

4.4 Structurally-restricted Boltzmann Machines

Beside the intractability of computing the statistics of the distributions modeled by a fully-connected Boltzmann machine exactly, the approximate methods introduced before, such as MCMC methods and variational approximations, are still computationally very expensive. Especially when it comes to using MCMC methods, especially the Gibbs sampling, the full connectivity of Boltzmann machines prevents an efficient, parallel sampling procedure.

In this section, we first describe how the Boltzmann machine can be interpreted as a Markov random field. This interpretation allows us to examine the underlying reason of the difficulty in parallelizing Gibbs sampling in a fully-connected Boltzmann machine. Furthermore, it sheds light on the direction in which the structural restriction will be applied.

Based on this interpretation, we introduce two structurally restricted variants of Boltzmann machines that have become widely used recently. The first model, called a restricted Boltzmann machine, simplifies the connectivity of units such that no pair of units of the same type is connected. This allows an extremely efficient and exact computation of the posterior probability of the hidden units, avoiding any need for the variational approximation. Furthermore, this bipartite structure allows an easy implementation of parallel Gibbs sampling.

The other model is called a deep Boltzmann machine. It relaxes the structural restriction of the restricted Boltzmann machine by allowing multiple layers of hidden units, instead of just a single one. Again, each pair of layers is fully connected, while no pair of units in the same layer is connected.

4.4.1 Markov Random Field and Conditional Independence

A Markov random field (MRF) is an undirected graphical model that consists of multiple random variables and undirected edges connecting some pairs of the random

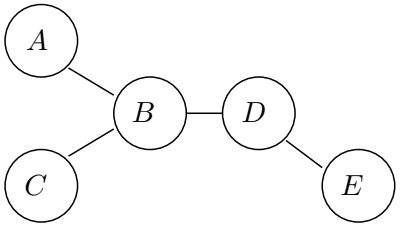


Figure 4.6. An example Markov random field with five random varaibles.

variables (see, e.g., Kindermann et al., 1980). A Boltzmann machine is a special case of MRFs. See Fig. 4.6 for one example of an MRF.

An MRF is constructed from a set of random variables as vertices $V = \{x_1, \ldots, x_p\}$ and a set of undirected edges connecting those vertices. The probability of a state $\mathbf{V}$ is defined by

$$p(\mathbf{V}) = \frac{1}{Z} \prod_{c \in C} \varphi_c \left(\mathbf{V}_c \right),$$

where Z is a normalization constant and C is a set of all possible cliques[5]. $\varphi_c \left(\mathbf{V}_c \right)$ is a positive potential function assigned to a clique c. It is usual that the unnormalized probability ($p^*(\mathbf{V}) = Zp(\mathbf{V})$) is easy to compute, while it is intractable to compute Z exactly.

It is straightforward to see that a Boltzmann machine is a special case of MRFs. Each variable of the Boltzmann machine corresponds to a vertex, and edges between all pairs indicate that it is a complete, undirected graph. A potential function of all cliques of two vertices is

$$\varphi_{ij} = \exp \left\{ x_i x_j w_{ij} \right\},$$

and that of all cliques of a single vertex is

$$\varphi_i = \exp \left\{ x_i b_i \right\}.$$

All other cliques are assigned a constant potential ($= 1$).

In an MRF, two variables A and B are *conditionally independent* from each other, if there is at least one variable observed in each and every path between A and B. That is,

$$p(A \mid \mathbf{X}_{\text{obs}}, B) = p(A \mid \mathbf{X}_{\text{obs}}),$$

if there is no path between A and B without any observed variable. $\mathbf{X}_{\text{obs}}$ is the state of all observed variables.

In this sense, we define a *Markov blanket* of a random variable as a set of all immediate neighboring variables. If all the variables in the Markov blanket of another

[5] A *clique* is a complete subgraph, where all vertices in the subgraph are fully connected to each other (see, e.g., Bondy and Murty, 2008).

variable were observed, the variable is independent from all other variables conditioned on the variables in the Markov blanket.

In the example MRF in Fig. 4.6, there are five random variables; A, B, C, D and E. Let us consider the variable A, first. When D is observed, A is conditionally independent from E since D the observed in the only path A–B–D–E between them. However, A and C are mutually dependent, as the path A–B–C has no observed variable.

In this example, the Markov blanket of D consists of B and E. Whenever both B and E are observed, D is conditionally independent from all other variables, in this case A and C, regardless of the connectivity in the graph.

This conditional independence property provides a means to parallelize a sampling procedure by Gibbs sampling. Let us, for instance, assume that we run a Gibbs sampler to collect samples from the example MRF in Fig. 4.6. One way is to grab a sample from one variable at a time, sequentially. However, if we consider the fact that A, E and C are conditionally independent from each other when B and D are observed, we can use the following schedule for the Gibbs sampler repeatedly:

1. Sample from $p(A \mid B)$, $p(C \mid B)$ and $p(D \mid B, E)$ in parallel
2. Sample from $p(B \mid A, C, D)$ and $p(E \mid D)$ in parallel

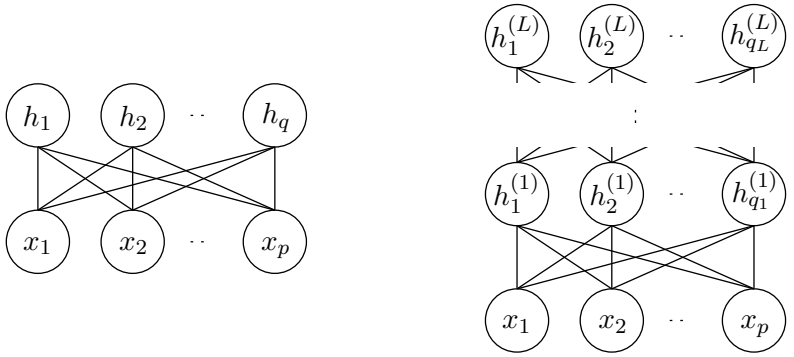
This will greatly speed up the sampling process, assuming that parallel computation is easily accessible and implementable.

Now it is easy to see why it is difficult to evaluate statistics of a fully-connected Boltzmann machine by collecting samples. When the Markov blanket of each unit consists of all other units, sampling must be done for each unit sequentially. This has led to attempts to overcome this problem by imposing structural restrictions to a Boltzmann machine using the property of conditional independence of an MRF. Some of these attempts will be described in this section.

4.4.2 Restricted Boltzmann Machines

A restricted Boltzmann machine (RBM) proposed by Smolensky (1986) is a variant of a Boltzmann machine that has a bipartite structure such that each visible unit is connected to all hidden units and each hidden unit to all visible units, but there are no edges between the same type of units (see Fig. 4.7(a) for illustration). In other words, we put constraints in the energy of a Boltzmann machine (4.1) so that $\mathbf{U} = \mathbf{0}$ and $\mathbf{V} = \mathbf{0}$. Then, the energy is simplified into

$$-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \mathbf{b}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{h} + \mathbf{x}^\top \mathbf{W} \mathbf{h}. \tag{4.20}$$



(a) Restricted Boltzmann Machine (b) Deep Boltzmann Machine

Figure 4.7. Illustrations of restricted and deep Boltzmann machines.

Contrary to the fully-connected Boltzmann machine, the Markov blanket of each unit consists only of the units of an opposite type. For instance, the Markov blanket of a visible unit x_i is a set of all hidden units, and that of a hidden unit h_j is a set of all visible units. Simply put, the hidden units are conditionally independent given a state of the visible units, and vice versa.

This conditional independence has two positive consequences in computing the statistics required for learning the parameters of an RBM. Firstly, the Gibbs sampling can be implemented efficiently by parallelizing the sampling procedure. The states of the units in each type of layer, either a visible or hidden layer, can be sampled in parallel given the state of the units in the other type of layer. Essentially, samples are collected repeatedly from the following distributions alternately:

$$p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta}) = \prod_{i=1}^{p} \nu_i^{x_i} (1 - \nu_i^{1-x_i}), \tag{4.21}$$

$$p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta}) = \prod_{j=1}^{q} \mu_j^{h_j} (1 - \mu_j^{1-h_j}), \tag{4.22}$$

where

$$\nu_i = p(x_i = 1 \mid \mathbf{h}, \boldsymbol{\theta}) = \phi\left(\sum_{j=1}^{q} w_{ij} h_j + b_i\right),$$

$$\mu_j = p(h_j = 1 \mid \mathbf{x}, \boldsymbol{\theta}) = \phi\left(\sum_{i=1}^{p} w_{ij} x_i + c_j\right).$$

Effectively, in just two parallelized steps, the state of each unit is replaced by a new sample. See Fig. 4.8 for illustration of the Gibbs sampling in the case of RBMs.

Furthermore, it is easy to see that we can efficiently compute the posterior distribution over the hidden units exactly, since the posterior distribution is fully factorized. The exact posterior distribution $p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta})$ is given in Eq. (4.22). This enables us to exactly compute the first term of the gradient in Eq. (4.4):

$$\left\langle \frac{\partial \left(-E(\mathbf{x}^{(n)}, \mathbf{h} \mid \boldsymbol{\theta})\right)}{\partial \theta} \right\rangle_{\mathrm{d}} = \frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial - E(\mathbf{x}^{(n)}, \boldsymbol{\mu}^{(n)} \mid \boldsymbol{\theta})}{\partial \theta} \right), \tag{4.23}$$

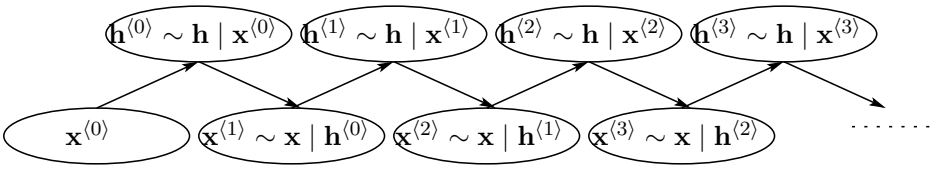


Figure 4.8. An illustration of performing a block Gibbs sampling on a restricted Boltzmann machine. $\mathbf{x}_0$ is initialized to a random binary vector.

where $\boldsymbol{\mu}^{(n)} = \left[\mu_1^{(n)}, \ldots, \mu_q^{(n)}\right]^\top$ and $\mu_j^{(n)} = p(h_j = 1 \mid \mathbf{x}^{(n)}, \boldsymbol{\theta})$. We used a shorthand notation d to denote the data distribution which was replaced with the N training samples $\left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\right\}$.

However, this property of an RBM that enables parallelized Gibbs sampling and exact computation of posterior distribution over hidden units do *not* fully avoid the difficulty in computing the second term of the gradient. Firstly, this efficient parallelized implementation does not overcome the fundamental weakness of Gibbs sampling that it is easy to get trapped in a single mode (see Section 4.3.1). Secondly, a sample gradient at each time computed using samples from Gibbs sampling tends to have high variance, which easily leads to unstable learning.

In the remainder of this section, we describe an efficient learning algorithm for an RBM based on the principle of minimizing contrastive divergence, proposed by Hinton (2002).

Product of Experts

A product of experts (PoE) (Hinton, 2002) is a model that combines multiple tractable probabilistic models, or experts, by multiplying their contributions and normalization the product to sum up to one. The probability assigned to a single input vector $\mathbf{x}$ can then be written as

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \frac{\prod_{j=1}^q \varphi_j(\mathbf{x} \mid \boldsymbol{\theta}_j)}{\sum_{\mathbf{x}'} \prod_{j=1}^q \varphi_j(\mathbf{x}' \mid \boldsymbol{\theta}_j)}, \tag{4.24}$$

where φ_j is the positive contribution of the j-th expert parameterized by $\boldsymbol{\theta}_j$, and the denominator is the normalization constant. It is not necessary for each φ_j to be a valid probability, since their product will be normalized afterward.

By replacing $\varphi_j(\mathbf{x} \mid \boldsymbol{\theta}_j)$ with $\tilde{\varphi}_j(\mathbf{x} \mid \boldsymbol{\theta}_j) = \varphi_j(\mathbf{x} \mid \boldsymbol{\theta}_j) - 1$, we may rewrite Eq. (4.24) as

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{j=1}^q \left[1 + \tilde{\varphi}_j(\mathbf{x} \mid \boldsymbol{\theta}_j)\right], \tag{4.25}$$

where we used $Z(\boldsymbol{\theta})$ for the normalization constant.

If we assume that there is a *binary* hidden variable associated with each expert, Eq. (4.25) can be written as a marginal probability of the joint probability of both the

visible and hidden variables:

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \sum_{\mathbf{h}} \frac{1}{Z(\boldsymbol{\theta})} \prod_{j=1}^{q} \tilde{\varphi}_j(\mathbf{x} \mid \boldsymbol{\theta}_j)^{h_j}. \tag{4.26}$$

Furthermore, one can clearly see that the posterior probability of $\mathbf{h}$ given the state of $\mathbf{x}$ is factorized so that each h_j is independent from all other hidden variables. The posterior probability of h_j is given by

$$p(h_j = 1 \mid \mathbf{x}, \boldsymbol{\theta}_j) = \frac{\tilde{\varphi}_j(\mathbf{x} \mid \boldsymbol{\theta}_j)}{1 + \tilde{\varphi}_j(\mathbf{x} \mid \boldsymbol{\theta}_j)}. \tag{4.27}$$

If it is further assumed that the choice of φ_j's makes the conditional probability of $\mathbf{x}$ factorized, as was the case with RBMs, efficient parallelized Gibbs sampling can be implemented for the PoE. One example of a contribution φ_j or $\tilde{\varphi}_j$, is

$$\tilde{\varphi}_j(\mathbf{x} \mid \boldsymbol{\theta}_j) = \exp\left\{ c_j + \sum_{i=1}^{p} w_{ij} x_i \right\},$$

which leads to the factorized conditional probability

$$\begin{aligned} p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta}) &= \frac{1}{Z(\boldsymbol{\theta})} \exp\left\{ \sum_{j=1}^{q} h_j c_j + \sum_{i=1}^{p} \sum_{j=1}^{q} w_{ij} x_i h_j \right\} \\ &= \frac{1}{Z'(\boldsymbol{\theta})} \prod_{i=1}^{p} \exp\left\{ \sum_{j=1}^{q} w_{ij} h_j \right\}^{x_i}, \end{aligned} \tag{4.28}$$

where $Z'(\boldsymbol{\theta}) = Z(\boldsymbol{\theta}) \exp\left\{ \sum_j h_j c_j \right\}$ is a new normalization constant.

This model corresponds in fact exactly to the RBM (Freund and Haussler, 1994), under certain constraints, such as (1) one of the experts is always on to act as a visible bias in an RBM and (2) $\mathbf{x} \in \{0, 1\}^p$. Compare Eqs. (4.27) and (4.28) against Eqs. (4.22) and (4.21). An RBM is a special case of the PoE, and a learning algorithm for PoEs can be immediately applied to an RBM.

Another example of PoE models that results in a factorized conditional distribution is the exponential family harmonium proposed by Welling et al. (2005). The exponential family harmonium assumes that the prior distributions of visible variables $\mathbf{x}$ and hidden variables $\mathbf{h}$ belong to the exponential family. This implies that both conditional and posterior distributions as well as marginal distributions are in the form of the exponential family distributions.

Interpolation between Data and Model Distributions and Contrastive Divergence

Consider now a series of distributions that are defined by running a Gibbs sampler for a finite k steps under a PoE with a factorized conditional distribution for visible variables.

A distribution P_0 is defined to be the data distribution D from which a set of training samples was sampled. Since the goal of the model is to model this distribution as well as possible, the form of this distribution is assumed to be unknown *a priori*.

Let $X_k = \left\{\mathbf{x}_k^{(1)}, \ldots, \mathbf{x}_k^{(N)}\right\}$ be a set of samples from P_k. We may then define a set of samples X_{k+1}. First, we collect hidden samples $H_k = \left\{\mathbf{h}_k^{(1)}, \ldots, \mathbf{h}_k^{(N)}\right\}$, where each $\mathbf{h}_k^{(n)}$ is a sample from the posterior distribution $p(\mathbf{h} \mid \mathbf{x} = \mathbf{x}_k^{(n)}, \boldsymbol{\theta})$. Given H_k, we collect samples or their respective means by the conditional distribution such that $X_{k+1} = \left\{\mathbf{x}_{k+1}^{(1)}, \ldots, \mathbf{x}_{k+1}^{(N)}\right\}$, where

$$\mathbf{x}_{k+1}^{(n)} \sim \mathbf{x} \left| \mathbf{h} = \mathbf{h}_k^{(n)}, \boldsymbol{\theta}. \right.$$

We call the distribution, represented by the samples in X_{k+1}, P_{k+1}.

Starting from the distribution P_0, P_k in the limit of $k \to \infty$ converges to the model distribution which is the stationary distribution of the Markov Chain defined by the Gibbs sampler used (see Section 4.3.1). It does not matter at all that we start the chain from the training samples, rather than randomly chosen states, assuming that the Gibbs chain is ergodic (see Eq. (4.13) for its definition).

Under these definitions, we may write the objective of learning in RBMs, and respectively PoEs with a factorized conditional distribution of visible variables, by $\mathrm{KL}(P_0 \| P_\infty)$. There P_∞ is the model distribution that can be described by the samples collected by running a Gibbs sampler long time starting from the set of training samples.

With these interpolated distributions, Hinton (2002) proposed a learning algorithm that minimizes a contrastive divergence, instead of maximizing the log-likelihood. The k-step contrastive divergence (CD) is defined as a difference between $\mathrm{KL}(P_0 \| P_\infty)$ and $\mathrm{KL}(P_k \| P_\infty)$, and minimizing it is equivalent to maximizing the log-likelihood in the infinite limit of k, since $\mathrm{KL}(P_\infty \| P_\infty) = 0$.

The gradient of the k-step CD is, then,

$$\frac{\partial \mathrm{CD}_k}{\partial \theta} \propto \left\langle \frac{\partial \left(-E(\mathbf{x}^{(n)}, \mathbf{h} \mid \boldsymbol{\theta})\right)}{\partial \theta} \right\rangle_{P_0} - \left\langle \frac{\partial \left(-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})\right)}{\partial \theta} \right\rangle_{P_k}, \tag{4.29}$$

where $\mathrm{CD}_k = \mathrm{KL}(P_0 \| P_\infty) - \mathrm{KL}(P_k \| P_\infty)$. This approach is directly applicable to the enhanced gradient described in Section 4.1.1 by replacing m with P_k in Eqs. (4.10)–(4.11).

This method has a huge computational advantage over maximizing the log-likelihood exactly in a naive way. As one can immediately see, computing Eq. (4.29) requires neither running the Gibbs sampler for indefinite time nor checking the convergence of the sampling chain. Empirically, minimizing CD was found to learn a good model even with k as small as 1[6].

Although minimizing CD has been used successfully in practice since it was introduced, Carreira-Perpiñán and Hinton (2005); Bengio and Delalleau (2009) showed

[6]From here on, we will occasionally refer to the learning algorithm that minimizes CD as CD learning, when there is no ambiguity.

that it is a biased estimate of the true log-likelihood with a finite k. This is contrary to the stochastic approximation procedure described in Section 4.3.3, which is guaranteed to converge to the right maximum-likelihood estimate under some mild assumptions. However, minimizing CD is preferable in many cases, as much smaller learning rate needs to be used with the stochastic approximation procedure, especially using a plain Gibbs sampling (see, e.g., (Hinton, 2012)).

For detailed justification on minimizing CD, we refer any interested reader to (Bengio and Delalleau, 2009).

4.4.3 Deep Boltzmann Machines

Deep Boltzmann machine (DBM) was proposed by Salakhutdinov and Hinton (2009a) as a relaxed version of an RBM. A DBM simply stacks multiple additional layers of hidden units on the layer of hidden units of an RBM. As was the case with an RBM, consecutive layers are fully connected, while there is no edge among the units in one layer. See Fig. 4.7(b) for an example structure of a DBM.

The energy function is defined as

$$-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \mathbf{b}^\top \mathbf{x} + \mathbf{c}_{[1]}^\top \mathbf{h}_{[1]} + \mathbf{x}^\top \mathbf{W} \mathbf{h}_{[1]} + \sum_{l=2}^{L} \left(\mathbf{c}_{[l]}^\top \mathbf{h}_{[l]} + \mathbf{h}_{[l-1]}^\top \mathbf{U}_{[l-1]} \mathbf{h}_{[l]} \right), \tag{4.30}$$

where L is the number of hidden layers. The state and biases of the hidden units at the l-th hidden layer and the weight matrix between the l-th and $(l+1)$-th layers are respectively defined by

$$\mathbf{h}_{[l]} = \left[h_1^{[l]}, \ldots, h_{q_l}^{[l]} \right]^\top, \mathbf{c}_{[l]} = \left[c_1^{[l]}, \ldots, c_{q_l}^{[l]} \right]^\top, \mathbf{U}_{[l]} = \left[u_{ij}^{[l]} \right],$$

where q_l is the number of the units in the l-th layer and $\mathbf{U}_{[l]} \in \mathbb{R}^{q_l \times q_{l+1}}$.

Given a set D of training samples, a DBM can also be trained by maximizing the marginal log-likelihood in Eq. (4.3). However, unlike an RBM the posterior distribution over the hidden units is not factorized, and the variational approximation described in Section 4.3.2 needs to be used to compute the statistics of the data distribution.

The statistics of the model distribution can again be estimated by using the Gibbs sampling. The layered structure of a DBM makes it easy to parallelize the Gibbs sampling procedure. Let us denote the state of the odd-numbered and even-numbered hidden layers by $\mathbf{h}_+$ and $\mathbf{h}_-$, respectively:

$$\mathbf{h}_+ = \left[\mathbf{h}_{[1]}^\top, \mathbf{h}_{[3]}^\top, \ldots \right]^\top, \mathbf{h}_- = \left[\mathbf{h}_{[2]}^\top, \mathbf{h}_{[4]}^\top \ldots \right]$$

We can collect samples by repeating the following steps:

1. Sample $\mathbf{h}_+$ in parallel, conditioned on $\mathbf{x}$ and $\mathbf{h}_-$
2. Sample $\{\mathbf{x}, \mathbf{h}_-\}$ in parallel, conditioned on $\mathbf{h}_+$.

This procedure can similarly be used to estimate the statistics of the data distribution by the variational approximation. This is due to the fact that the fixed-point update rules in Eq. (4.18) of variational parameters $\boldsymbol{\mu}^{[l]}$ in a single layer are mutually independent given the variational parameters in the adjacent layers. Based on this, we can perform the variational fixed-point updates of the variational parameters of a DBM efficiently using the following steps:

1. Compute $\boldsymbol{\mu}_+$ in parallel using $\mathbf{x}$ and $\boldsymbol{\mu}_-$
2. Compute $\boldsymbol{\mu}_-$ in parallel using $\boldsymbol{\mu}_+$

We used $\boldsymbol{\mu}_+$ and $\boldsymbol{\mu}_-$ to denote the variational parameters of the odd-numbered and even-numbered hidden layers, respectively.

It has been noticed by many researchers, for instance Salakhutdinov and Hinton (2009a) and Desjardins et al. (2012) as well as the author in Publication VI and Publication VII, that training a DBM starting from randomly initialized parameters is not trivial.

In order to alleviate this difficulty, Salakhutdinov and Hinton (2009a, 2012a) proposed an algorithm that initializes the parameters of a DBM by pretraining each layer separately starting from the bottom layer. In Publication VII yet another pretraining algorithm was proposed that utilizes a deep autoencoder to obtain a set of variational parameters for the hidden units in the even-numbered hidden layers. We will discuss those pretraining algorithms later in Section 5.3.3.

There have been other approaches to avoid this difficulty. Montavon and Müller (2012) suggested that a shifting transformation that centers each unit, described briefly in Section 4.1.1, helps training a DBM directly from a set of randomly initialized parameters at least when the size of the DBM is relatively small. On top of the shifting transformation, Desjardins et al. (2013) proposed a metric-free natural gradient method that utilizes the idea of natural gradient together with the subsampled Hessian algorithm.

4.5 Boltzmann Machines and Autoencoders

In this section, we try to describe the connections between the Boltzmann machines and the autoencoders which we have discussed earlier in Section 2.2.1 and Section 3.2.

We begin by relating an RBM with an autoencoder that has only a single hidden layer. This can be done by considering the correspondence between the learning algorithms for the two models. We will discuss this from the perspectives of contrastive divergence learning and score matching (Hyvärinen, 2005). These connections were respectively discovered and presented by Bengio and Delalleau (2009), Swersky et al. (2011) and Vincent (2011).

We continue on to describing a deep belief network (Hinton et al., 2006) which is an extension of the sigmoid belief network described earlier in Section 3.2.3 as one of the related approaches of deep autoencoders.

4.5.1 Restricted Boltzmann Machines and Autoencoders

Let us start with rather shallow models that have only a single hidden layer consisting of nonlinear units, which are restricted Boltzmann machines and autoencoders with a single intermediate hidden layer.

We briefly discuss two different ways to relate RBMs to shallow autoencoders here. However, it should be noticed that these are *not* the only ways (see, e.g., Ranzato et al., 2007a, for another interpretation that unifies RBMs and autoencoders).

Contrastive Divergence and Reconstruction Error

In (Bengio and Delalleau, 2009), learning parameters of an RBM by minimizing contrastive divergence was justified by considering an expansion of the gradient of the marginal log-likelihood of an RBM in Eq. (4.23).

As we defined the interpolated distributions between the data and model distributions in Section 4.4.2, let us here consider a potentially infinite sequence $\left\{ \left(\mathbf{x}^{\langle 0 \rangle}, \mathbf{h}^{\langle 0 \rangle} \right), \left(\mathbf{x}^{\langle 1 \rangle}, \mathbf{h}^{\langle 1 \rangle} \right), \dots \right\}$, where

$$\mathbf{x}^{\langle n \rangle} \sim \mathbf{x} \mid \mathbf{h}^{\langle n-1 \rangle}, \boldsymbol{\theta}$$

and

$$\mathbf{h}^{\langle n \rangle} \sim \mathbf{h} \mid \mathbf{x}^{\langle n \rangle}, \boldsymbol{\theta}.$$

$\mathbf{x}^{\langle 0 \rangle}$ denotes a training sample.

In this case, Bengio and Delalleau (2009) showed that the gradient of the marginal log-likelihood, considering only a single sample for now, can be expanded by

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta} = \sum_{s=1}^{t-1} \left(\mathbb{E}\left[\frac{\partial \log p(\mathbf{x}^{\langle s \rangle} \mid \mathbf{h}^{\langle s \rangle})}{\partial \theta} \right] + \mathbb{E}\left[\frac{\partial \log p(\mathbf{h}^{\langle s \rangle} \mid \mathbf{x}^{\langle s+1 \rangle})}{\partial \theta} \right] \right) + \mathbb{E}\left[\frac{\partial \log p(\mathbf{x}^{\langle t \rangle})}{\partial \theta} \right], \tag{4.31}$$

where the terms inside the summation and the last term converge to zero as t grows.

The expectations are computed over the data distribution. Dependency on $\boldsymbol{\theta}$ is omitted to make the expression less cluttered.

When only the first two terms of the expansion in Eq. (4.31) are considered, we get the update direction that minimizes the contrastive divergence with $k = 1$. The remaining terms determine the bias in the resulting model.

Bengio and Delalleau (2009) went one more step to investigate the case of truncating even earlier. In that case, the truncated expression becomes

$$\sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{x}^{\langle 0 \rangle}) \frac{\partial \log p(\mathbf{x}^{\langle 1 \rangle} \mid \mathbf{h})}{\partial \theta}.$$

If we use a mean-field approximation to replace each component h_j of $\mathbf{h}$ inside $\log p(\mathbf{x}^{\langle 1 \rangle} \mid \mathbf{h})$ with its posterior mean $\hat{h}_j = \mathbb{E}\left[h_j \mid \mathbf{x}^{\langle 0 \rangle}\right]$, we get

$$\frac{\partial \log p(\mathbf{x}^{\langle 1 \rangle} \mid \hat{\mathbf{h}})}{\partial \theta}. \tag{4.32}$$

Let us rewrite it by adopting our usual notation using Eq. (2.10) such that we assume to be given a set of training samples as

$$D = \left\{\mathbf{x}^{(n)}\right\}_{n=1}^{N}.$$

Eq. (4.32) becomes

$$\sum_{n=1}^{N} \frac{\partial \log p(\mathbf{x}^{(n)} \mid \hat{\mathbf{h}}^{(n)})}{\partial \theta}.$$

It is straightforward to see that this is the gradient of the cross-entropy loss of a single-layer autoencoder with a single hidden layer of sigmoid hidden units.

This result says that minimizing contrastive divergence is an extreme *stochastic approximation* of maximizing the marginal log-likelihood, while minimizing the reconstruction error or cross-entropy corresponds to a *deterministic approximation*. Since the latter truncated one more term from the expansion, the latter obviously results in a more biased estimation, if we look at the resulting model as an RBM.

However, the objective function of an autoencoder, which is a negative reconstruction error, can be exactly and efficiently computed, while computing the marginal log-likelihood of an RBM is intractable. This makes it easier to learn parameters by minimizing the reconstruction error compared to either minimizing the contrastive divergence or maximizing the marginal log-likelihood directly.

Gaussian Restricted Boltzmann Machines, Score Matching and Autoencoders

Let us define an RBM that can handle continuous visible units by replacing binary visible units with Gaussian units. The energy function of this RBM, called a Gaussian-Bernoulli RBM (GRBM, Hinton and Salakhutdinov, 2006) is then,

$$-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = -\sum_{i=1}^{p} \frac{(x_i - b_i)^2}{2\sigma_i^2} + \sum_{j=1}^{q} h_j c_j + \sum_{i=1}^{p} \sum_{j=1}^{q} \frac{x_i}{\sigma_i^2} h_j w_{ij}. \tag{4.33}$$

The conditional distributions of the visible and hidden units are defined by

$$p(x_i = x|\mathbf{h}) = \mathcal{N}\left(x \,\middle|\, b_i + \sum_j h_j W_{ij}\ ,\ \sigma_i^2\right) \tag{4.34}$$

and

$$p(h_j = 1|\mathbf{x}) = \phi\left(c_j + \sum_i W_{ij}\frac{x_i}{\sigma_i^2}\right), \tag{4.35}$$

where ϕ is again a sigmoid function.

In other words, each visible unit conditionally follows a Gaussian distribution whose mean is determined by the weighted sum of the states of the hidden units. The conditional probability of each hidden unit being one is determined by the weighted sum of the states of the visible units *scaled* by their variances[7]. It should be noticed that despite the change of the visible units' types, there is essentially no change in its learning algorithm compared to the original RBM with binary visible units.

In this case, an alternative learning algorithm called *score matching* proposed by Hyvärinen (2005), can be used instead of the stochastic approximation procedure or CD learning. Score matching can be used to estimate the parameters of a model whose *differentiable* unnormalized probability can be tractably computed exactly, while computing its normalization constant is intractable. This requirement reminds us of a PoE described earlier in Section 4.4.2 which is a family of models that includes a GRBM.

We start by writing a GRBM in the form of PoE:

$$\begin{aligned}\log p(\mathbf{x} \mid \boldsymbol{\theta}) &= -\frac{(b_i - x_i)^2}{2\sigma_i^2} + \sum_{j=1}^{q} \log\left(1 + \exp\left\{\sum_{i=1}^{p}\frac{x_i}{\sigma_i^2}w_{ij} + c_j\right\}\right) - \log Z(\boldsymbol{\theta}) \\ &= \log p^*(\mathbf{x} \mid \boldsymbol{\theta}) - \log Z(\boldsymbol{\theta}).\end{aligned} \tag{4.36}$$

[7]The energy function of a GRBM was originally proposed by Hinton and Salakhutdinov (2006) to be

$$-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = -\sum_{i=1}^{p}\frac{(x_i - b_i)^2}{2\sigma_i^2} + \sum_{j=1}^{q} h_j c_j + \sum_{i=1}^{p}\sum_{j=1}^{q}\frac{x_i}{\sigma_i}h_j w_{ij},$$

which results in a conditional probability of a visible unit

$$p(x_i = v|\mathbf{h}) = \mathcal{N}\left(x \,\middle|\, b_i + \sigma_i \sum_j h_j W_{ij}\ ,\ \sigma_i^2\right).$$

In order to avoid the standard deviation σ_i affecting the mean, it was proposed in Publication V that the energy function be modified so that σ_i does not influence the conditional mean of the visible unit. In Publication V, it was claimed that this formulation makes estimating σ_i's easier, and hence we use the modified definition of the energy function here.

Then, the score function of the GRBM is

$$\begin{aligned}\psi_i(\mathbf{x} \mid \boldsymbol{\theta}) &= \frac{\partial \log p^*(\mathbf{x} \mid \boldsymbol{\theta})}{\partial x_i} \\ &= \frac{b_i - x_i}{\sigma_i^2} + \sum_{j=1}^{q} \hat{h}_j \frac{w_{ij}}{\sigma_i^2}. \end{aligned} \tag{4.37}$$

Furthermore, we will need

$$\frac{\partial \psi_i(\mathbf{x} \mid \boldsymbol{\theta})}{\partial x_i} = -\frac{1}{\sigma_i^2} + \sum_{j=1}^{q} \hat{h}_j(1 - \hat{h}_j) \frac{w_{ij}^2}{\sigma_i^2}, \tag{4.38}$$

where $\hat{h}_j = \phi\left(\sum_{i=1}^{p} \frac{x_i}{\sigma_i^2} w_{ij} + c_j\right)$ is the activation probability of the j-th hidden units.

Given a set D of N training samples, the score matching then tries to minimize the following cost function instead of maximizing the marginal log-likelihood:

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{p} \left[\frac{\partial \psi_i(\mathbf{x}^{(n)} \mid \boldsymbol{\theta})}{\partial x_i} + \frac{1}{2} \psi_i(\mathbf{x} \mid \boldsymbol{\theta})^2 \right] \tag{4.39}$$

If we assume that the bias b_i and the conditional variance σ_i to each visible unit are respectively fixed to 0 and 1[8], the cost function in Eq. (4.39) can be rewritten as

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \left[\frac{1}{2} \left(\sum_{i=1}^{p} \sum_{j=1}^{q} \hat{h}_j w_{ij} - x_i \right)^2 + \sum_{i=1}^{p} \sum_{j=1}^{q} h_j(1 - h_j) w_{ij}^2 \right] + C,$$

where C is a constant that does not depend on $\boldsymbol{\theta}$.

It is easy to see that the first term inside the summation is the reconstruction error of a single-layer autoencoder with a tied set of weights $\mathbf{W}$ (see Eq. (2.14)). The other term can be considered a regularization term.

Hence, this leads us to understand that learning the parameters of a GRBM by score matching is equivalent to learning the parameters of a single-layer autoencoder that has a hidden layer having nonlinear sigmoidal units by minimizing the reconstruction error with a regularization (Swersky et al., 2011). Furthermore, this connection gives a justification on using a tied set of weights for the encoder and decoder of an autoencoder (Vincent, 2011).

Additionally to an ordinary autoencoder, Vincent (2011) showed a connection between a denoising autoencoder (see Section 3.3.1) and a modified form of GRBM, again via score matching.

[8] Fixing b_i to 0 can be indirectly achieved by normalizing each component of training samples to have a zero-mean, and it has been an usual practice to ignore σ_i by forcing it to 1 (see, e.g., Hinton, 2012).

4.5.2 Deep Belief Network

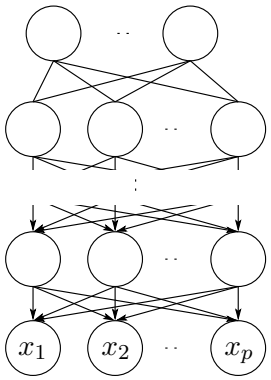


Figure 4.9. An illustration of a deep belief network. Note that the top two layers are connected to each other by *undirected* edges.

In Section 3.2.3, we presented the sigmoid belief network in a relation with a deep autoencoder and subsequently briefly described the deep belief network. Here, we present another aspect of the DBN as a combination of a sigmoid belief network and an RBM.

A deep belief network (DBN), proposed by Hinton et al. (2006), is a hybrid model that has both directed and undirected edges. The top two hidden layers $\mathbf{h}^{(L-1)}$ and $\mathbf{h}^{(L)}$ are connected to each other (without any intra-layer edges) by undirected edges, while all subsequent pairs of layers below are connected with the directed downward edges. Hence, one might say that the top two layers generatively model the *prior* distribution of $\mathbf{h}^{(L-1)}$, and all the other hidden layers below model the conditional distribution that generates the states of the units in the layer immediately below. See Fig. 4.9 for the illustration.

In this case, as we did with Boltzmann machines previously, we may define the energy as

$$-E(\mathbf{x}, \mathbf{h}^{[1]}, \dots, \mathbf{h}^{[L]} \mid \boldsymbol{\theta}) = \log p(\mathbf{x} \mid \mathbf{h}^{[1]}, \boldsymbol{\theta}) + \sum_{l=1}^{L-2} \log p(\mathbf{h}^{[l]} \mid \mathbf{h}^{[l+1]}, \boldsymbol{\theta}) + \log p(\mathbf{h}^{[L-1]}, \mathbf{h}^{[L]}, \boldsymbol{\theta}) \tag{4.40}$$

where the last term denotes the top two layers connected by the undirected edges. The conditional distribution between the consecutive intermediate hidden layers as well as between the visible and first hidden layers is defined by Eqs. (3.9) and (3.10), respectively.

Again, similarly to the Boltzmann machine, we can define the joint probability of all units using a Boltzmann distribution:

$$p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left\{-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})\right\},$$

where we simply used $\mathbf{h}$ to denote the units in all hidden layers.

A DBN maintains two sets of parameters as well as the parameters for the top two layers. The first two sets $\boldsymbol{\theta}_-$ and $\boldsymbol{\theta}_+$ correspond to the recognition and generation parameters of a sigmoid belief network, and the last one does to the parameters of an RBM.

Using this hybrid architecture Hinton et al. (2006) proposed an efficient learning algorithm that consists of two stages. In the first stage, each pair of layers, starting from the bottom, is *pretrain*ed as if it were an RBM.

Specifically, the first pair of layers $\mathbf{x}$ and $\mathbf{h}^{[1]}$ is trained as an RBM to model a given set of training samples $D = \left\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\right\}$. Once the training is over, we can efficiently and exactly compute the posterior distribution over $\mathbf{h}^{(1)}$ given each training sample $Q(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)})$ and collect samples from these distributions. We call the distribution from which those samples were collected an *aggregate posterior* distribution

$$\tilde{Q}\left(\mathbf{h}^{[1]}\right) = \frac{1}{N}\sum_{n=1}^{N} Q\left(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)}\right)$$

Let us denote the set of the samples from this distribution by D_1.

Then, the next pair of layers $\mathbf{h}^{[1]}$ and $\mathbf{h}^{[2]}$ are pretrained to model $\tilde{Q}\left(\mathbf{h}^{[1]}\right)$ using the set D_1. From this RBM we can again collect a set of samples D_2 from the next aggregate posterior distribution $\tilde{Q}\left(\mathbf{h}^{[2]}\right)$. We continue this process up until we pretrain the last pair of layers $\mathbf{h}^{[L-1]}$ and $\mathbf{h}^{[L]}$. Let us use $\boldsymbol{\theta}_0^{[l]}$ to indicate the parameters of an RBM consisting of the l-th and $(l+1)$-th layers, learned during the first stage.

The first stage corresponds to a layer-wise pretraining. Instead of jointly optimizing all the layers at once, during the first stage each pair of two consecutive layers is optimized one by one starting from the bottom pair consisting of the visible and first hidden layers. This approach of adding more hidden layers in a DBN has been shown to guarantee to improve the model (Hinton et al., 2006), and will be discussed more in Section 5.3.2.

The second stage highly resembles the wake-sleep algorithm used to estimate the parameters of a sigmoid belief network from Section 3.2.3. The learning algorithm, called an up-down algorithm, starts by initializing the weights to those estimated during the first stage. The recognition and generation parameters will be identical, since the RBMs trained in the first stage use a tied set of weights for both inference and generation. The parameters between the top two layers will be $\boldsymbol{\theta}_0^{[L-1]}$.

The *up*-pass of the up-down algorithm corresponds to the *wake*-stage, and given a training sample, the samples are collected from the approximate posterior distributions over the hidden units using the recognition weights up until the penultimate layer. The generation parameters of the intermediate hidden layers are updated using Eq. (3.11), while the parameters of the top two layers are updated using the learning rule of an RBM in Eq. (4.23).

At the *down*-pass, unlike the *sleep*-stage, one does not attempt to collect model samples starting from scratch. Rather, Gibbs sampling is run starting from the samples of the penultimate layer collected during the up-pass for several iterations, which reminds us of minimizing contrastive divergence (see Section 4.4.2). From the sam-

ples gathered by the several-step Gibbs sampling, the generation parameters are used to generate samples of the subsequent intermediate layers down until the visible layer. The recognition parameters are then updated using Eq. (3.12). Unlike in the up-pass, the parameters of the top two layers are *not* adjusted in the down-pass.

This model can thus be thought as combining a deep autoencoder having stochastic hidden units together with a top-level RBM. The stochastic deep autoencoder models both the conditional distribution of a layer given the state of the upper layer and the approximate posterior distribution of a layer given the state of the lower layer. The prior distribution of the penultimate hidden layer is learned by the top-level RBM.

Deep Energy Model

Before ending this section, we will briefly introduce a deep energy model (DEM) proposed by Ngiam et al. (2011) as an alternative formulation of combining a deep autoencoder with an RBM. A DEM extends an RBM by introducing a nonlinear encoder with multiple layers of *deterministic* hidden units. According to Ngiam et al. (2011), this potentially avoids the difficulty introduced by having stochastic hidden units in a deep belief network.

Let us consider a GRBM from Section 4.5.1, however, assuming that $\sigma_i = 1$ for all $i = 1, \ldots, p$. The energy function of a DEM is a modified form of Eq. (4.33) such that

$$-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \sum_{i=1}^{p} x_i b_i + \sum_{j=1}^{q} h_j c_j + \sum_{i=1}^{p'} \sum_{j=1}^{q} f_i(\mathbf{x} \mid \boldsymbol{\theta}_f) h_j w_{ij},$$

where $f_i(\mathbf{x} \mid \boldsymbol{\theta}_f)$ is the i-th component of an output of a nonlinear encoder $f : \mathbb{R}^p \to \mathbb{R}^{p'}$ parameterized by $\boldsymbol{\theta}_f$.

The parameters of both an RBM and an encoder can be estimated by maximizing the marginal log-likelihood as usual with an RBM. Ngiam et al. (2011) used the stochastic approximation procedure with hybrid Monte Carlo (Neal, 1993) to estimate the statistics of the model distribution.

5. Unsupervised Neural Networks as the First Step

So far in this thesis, we have considered the unsupervised and supervised neural networks separately. In fact, our main focus was on the unsupervised neural networks rather than the supervised ones. We spent most of time discussing autoencoders and Boltzmann machines both of which aim to learn the distribution of training samples and are not specifically engineered for other tasks.

In Section 3.4.1, we briefly discussed so called layer-wise pretraining where shallow unsupervised neural networks were used to initialize the parameters of a deep multi-layer perceptron (MLP). Notably a stack of, for instance, restricted Boltzmann machines used for initializing an MLP was trained without any prior knowledge that it will be used for classification.

This suggests that it may be possible or even beneficial to utilize unsupervised neural networks as the first step to train a more sophisticated model that aims to perform a potentially different target task. For instance, an unsupervised neural network such as a denoising autoencoder or contractive autoencoder may be used to transform the coordinate system of input samples into one that is more useful for supervised learning tasks. A restricted Boltzmann machine may be trained to facilitate training more sophisticated deep Boltzmann machines, or it can learn the joint distribution of input and output and use it to compute the conditional distribution of output given a new input.

In this chapter, we discuss some of the approaches proposed recently to incorporate the power of generative modeling provided by unsupervised neural networks to improve the performance of another, often more complex neural network.

5.1 Incremental Transformation: Layer-Wise Pretraining

Let us consider a classification task. We already discussed earlier in Sections 2.1.2 and 2.3.1 that a simple perceptron without any intermediate nonlinear hidden layer can perform this task perfectly only if training samples are *linearly separable*. Other-

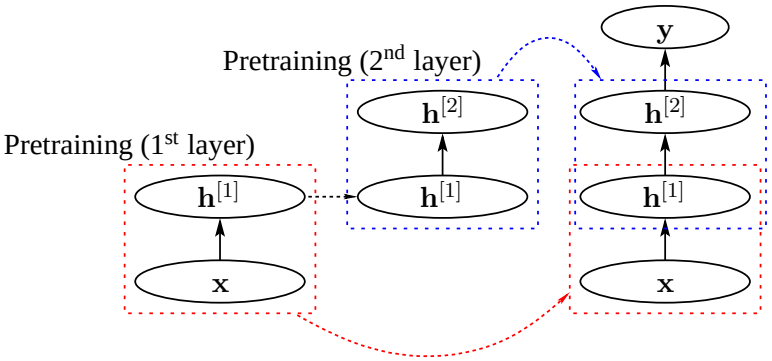


Figure 5.1. Illustration of the layer-wise pretraining of a multi-layer perceptron. The dashed directed lines indicate *copying* of either pretrained models or the activations of the hidden units of the pretrained models.

wise, this simple neural network fails to do so. However, even if the training set has a nonlinear separating hyperplane, an MLP which is essentially a perceptron with one or more intermediate nonlinear hidden layers can be trained to classify the samples, assuming that the MLP is large enough.

The feedforward computation starting from the bottom, visible layer fixed to an input $\mathbf{x}$ gradually performs nonlinear transformations up until the last hidden layer. Let $\mathbf{h} = f(\mathbf{x})$ be the nonlinearly transformed representation of $\mathbf{x}$ by the feedforward pass of an MLP. Then, the last part of the MLP essentially does a perceptron-like *linear* classification on $\mathbf{h}$.

In other words, the encoder f computed by the MLP attempts to transform each sample $\mathbf{x}^{(n)}$ into a new point $\mathbf{h}^{(n)}$ in a new coordinate system such that the set of these new points $\tilde{D} = \left\{(\mathbf{h}^{(n)}, y^{(n)})\right\}_{n=1}^{N}$ is as linearly separable as possible. It does not matter whether the original training set $D = \left\{(\mathbf{x}^{(n)}, y^{(n)})\right\}_{n=1}^{N}$ is linearly separable or not.

This process f is carried out in an incremental fashion. Each subsequent intermediate layer tries to stretch out the entangled samples, and as the samples pass through multiple stages of stretching-out, they become more evenly distributed and hence hopefully linearly separable, or better suited for other machine learning models.

This suggests a possibility of incrementally building a sequence of intermediate layers that gradually extract better and better representations of data[1]. This is contrary to how the parameters of an MLP were estimated. For an MLP, we first fix the structure of the model and jointly optimize all layers, while what is being suggested here is the other way around. We start with one visible and one hidden layer to learn a *transformation* that extracts a somewhat better representation. Repeatedly, using the representation from the previous stage, we train another model with one visible

[1]We say the *representation is better* when better *generalization* performance on another machine learning task, for instance classification, can be achieved using the representation compared to using the raw representation (Bengio et al., 2007).

and one hidden layer to learn a still somewhat better representation. See Fig. 5.1 for the illustration.

In fact, this approach of incrementally learning multiple layers of representations constitutes one of the most important principles in the field called *deep learning* (see, e.g., Bengio, 2009).

5.1.1 Basic Building Blocks: Autoencoder and Boltzmann Machines

The most straightforward way to implement this incremental feature learning is to consider each consecutive pair of layers of an MLP separately.

Each pair consists of a lower layer of visible units $\mathbf{x} \in \mathbb{R}^p$ and an upper layer of nonlinear hidden units $\mathbf{h} \in \mathbb{R}^q$, and they are connected by directed edges from the lower to upper layers. A hidden activation, or hidden state, is computed by

$$\mathbf{h} = \phi\left(\mathbf{W}^\top \mathbf{x} + \mathbf{c}\right),$$

where ϕ is a component-wise nonlinearity. This reminds us of an encoder of an autoencoder (see Eq. (3.3)) or the conditional probability of hidden units of a restricted Boltzmann machine (RBM, see Eq. (4.22)).

Hence, it is natural to estimate the parameters $\mathbf{W}$ and $\mathbf{c}$ as if a pair of those two layers would form either an autoencoder with a single hidden layer or a restricted Boltzmann machine. We start from the bottom by training the visible layer and the first hidden layer as if they formed an RBM. Once the RBM is trained, we compute the posterior distribution of hidden units and consider them as a new set of training samples for an upper pair of layers consisting of the first and second hidden layers. We repeat this step until the last two hidden layers were trained as yet another RBM. This same procedure can be done using either ordinary or regularized autoencoder instead of the RBMs.

An ordinary autoencoder (see, e.g., Bengio et al., 2007, and Section 3.2), a regularized autoencoder (see, e.g., Ranzato et al., 2008, and Section 3.2.5) as well as sparse coding (see, e.g., Raina et al., 2007, and Section 3.2.5) and a restricted Boltzmann machine (see, e.g., Hinton and Salakhutdinov, 2006, and Section 4.4.2) as well as a sparse restricted Boltzmann machine (see, e.g., Lee et al., 2008) have been used widely in this way to perform incremental feature learning. The denoising autoencoder (Vincent et al., 2010) and contractive autoencoder (Rifai et al., 2011b) discussed in Section 3.3 have recently been shown to be effective in this approach.

Once the sequence of these models is obtained it is possible, though not necessary, to *finetune* them all together for a specific target task. For instance, an MLP can be initialized by stacking the learned sequence and be trained continuing from the *pretrained* parameters using backpropagation (see Section 3.4). A deep autoencoder

with more than one intermediate layer can also benefit from this approach (Hinton and Salakhutdinov, 2006). In this context, the approach of incremental feature learning is often referred to as a *layer-wise pretraining*. This layer-wise pretraining has been successfully used to train a deep neural network that has been known to be difficult to train well starting from randomly initialized parameters.

5.2 Unsupervised Neural Networks for Discriminative Task

Throughout this thesis, we have mostly concentrated on *unsupervised* neural networks. There are obvious tasks to which these unsupervised neural networks can trivially be applied.

For instance, Burger et al. (2012), Xie et al. (2012) and Cho (2013) (Publication IX) recently showed that (denoising) autoencoder and restricted/deep Boltzmann machines can denoise large, corrupted images. The performance of these neural networks was shown to be comparable to, or sometimes better than, conventional methods of image denoising such as BM3D (Dabov et al., 2007) or K-SVD (Portilla et al., 2003).

A deep autoencoder initialized by a deep belief network was shown to excel at extracting low-dimensional binary codes for documents (Salakhutdinov and Hinton, 2009b). Also, Salakhutdinov et al. (2007) showed that an RBM can be successfully used for collaborative filtering.

However, obviously, these unsupervised models cannot be used directly for performing any supervised task. This is clear as none of these models have at their disposal known outputs of training samples.

As in the layer-wise pretraining discussed earlier in this chapter, however, the unsupervised neural networks can be used to improve the discriminative performance of supervised models. In the layer-wise pretraining, recursively stacking shallow unsupervised neural networks was shown to extract better representations that are more suitable for classification, which may be improved further by finetuning the stack with backpropagation.

In the remainder of this section, we introduce other approaches than the previously discussed layer-wise pretraining that aim to improve the discriminative performance by utilizing unsupervised neural networks. These approaches may use a separate unsupervised neural network, or combine a supervised and an unsupervised neural network.

5.2.1 Discriminative RBM and DBN

The most straightforward way to perform discriminative tasks with an unsupervised neural network is to model the joint probability distribution of both the input and output $p(\mathbf{x}, y)$. Once the network is trained, one can utilize the joint distribution to perform a prediction on a new sample $\mathbf{x}$.

Regardless of whether the task is classification or regression, the best output $\hat{y}$ given a new sample $\mathbf{x}^*$ can be found by, for instance, the maximum a posteriori (MAP):

$$\hat{y} = \arg\max_y p^*(\mathbf{x}^*, y), \tag{5.1}$$

where we have used the unnormalized probability p^* to emphasize that there is no need to compute the potentially intractable normalization constant.

Alternatively, one may be interested in computing the expected value of the output

$$\hat{y} = \frac{1}{Z} \sum_{y \in \mathbf{Y}} y p^*(\mathbf{x}^*, y), \tag{5.2}$$

where Z and $\mathbf{Y}$ are the normalization constant and the set of all possible values for y, respectively. However, in the latter case, the normalization constant which is often computational intractable has to be computed or estimated, which makes it less practical. Hence, in this section, we only focus on the MAP solution for the output y.

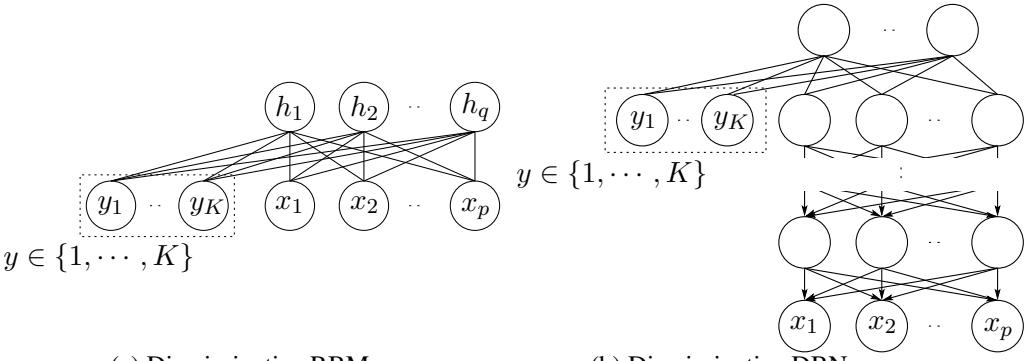
If y can only have a finite number of possible outcomes (classification), we can simply evaluate $p(\mathbf{x}^*, y)$ for all possible y's and choose y with the largest value. Otherwise, it is possible to optimize $p(\mathbf{x}^*, y)$ with respect to y to compute the best possible y, although it may only find a local mode if $p(\mathbf{x}^*, y)$ has more than one modes.

Let us consider using a restricted Boltzmann machine (RBM, Section 4.4.2) for classification.

First, given a training set $D = \left\{\left(\mathbf{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^N$, we turn each output $y^{(n)} \in \{1, 2, \ldots, q\}$ into a q-dimensional vector $\mathbf{y}^{(n)}$ whose $y^{(n)}$-th component is one and all other components are zero. With the transformed output vectors, we create a new training set $\tilde{D} = \left\{\left[(\mathbf{x}^{(n)})^\top (\mathbf{y}^{(n)})^\top\right]^\top\right\}_{n=1}^N$ by concatenating $\mathbf{x}^{(n)}$ and $\mathbf{y}^{(n)}$ for each n.

Then we train an RBM with the transformed set $\tilde{D}$ (see Fig. 5.2(a)) either using, for instance, the stochastic approximation procedure (see Section 4.3.3) or by minimizing contrastive divergence (see Section 4.4.2). Recalling that the unnormalized probability of $\left[\mathbf{x}^\top, \mathbf{y}^\top\right]$ after marginalizing out the hidden units can be efficiently and exactly computed (see Section 4.4.2), we can predict the label of a new sample by Eq. (5.2).

Larochelle and Bengio (2008) proposed a *discriminative* objective function for training this kind of an RBM. The proposed objective function maximizes instead



(a) Discriminative RBM (b) Discriminative DBN

Figure 5.2. Illustrations of a discriminative restricted Boltzmann machine and discriminative deep belief network. Note that the 1-of-K coding is used for the output label y which may take K discrete values.

of the log-likelihood the conditional log-likelihood

$$\mathcal{L}_d(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{y}^{(n)} \mid \mathbf{x}^{(n)}, \boldsymbol{\theta}).$$

Furthermore, they showed that a better classification performance can be achieved by maximizing the weighted sum of the log-likelihood and the conditional log-likelihood together.

A similar idea was also presented earlier for a deep belief network (DBN) by Hinton et al. (2006). Instead of augmenting the visible layer with a transformed label $\mathbf{y}$, they augmented the penultimate layer. The augmented units corresponding to $\mathbf{y}$ are only connected to the top layer with undirected edges. See Fig. 5.2(b) for illustration.

This model can be trained by the procedure described in Section 4.5.2, however with a slight modification. Firstly, during the first stage of layer-wise pretraining, we augment the posterior distribution of the penultimate layer with the labels of the training samples. During the second stage, where the up-down algorithm is used, the Gibbs sampling steps between the top two layers start from the samples from the (approximate) posterior distribution attached with the labels of the samples in a minibatch.

Once the training is over, we can classify a new sample $\mathbf{x}^*$ easily by first obtaining the approximate (fully factorized) posterior means of the penultimate layer $\boldsymbol{\mu}^*$ and computing the unnormalized probabilities of the combination of $\boldsymbol{\mu}^*$ and all possible label states $\mathbf{y}$. The one that gives the largest unnormalized probability is chosen as a prediction $\hat{\mathbf{y}}$.

Surprisingly, both of these approaches which perform both generative $p(\mathbf{x}, y)$ and discriminative $p(y \mid \mathbf{x})$ modeling achieve the classification performance comparable to or often better than the models which were trained purely to perform discriminative modeling (Hinton et al., 2006; Larochelle and Bengio, 2008).

5.2.2 Deep Boltzmann Machine to Initialize an MLP

It is straightforward to initialize a multi-layer perceptron (MLP) with a deep belief network (DBN) as well as a restricted Boltzmann machine (RBM). Once the parameters of those unsupervised neural networks are estimated, we can directly use them as initial parameters of an MLP. This corresponds to the layer-wise pretraining scheme discussed in Section 5.1. However, when it comes to a deep Boltzmann machine (DBM), one must take into account the nature of each layer receiving both bottom-up and top-down signals.

A naive way of utilizing a DBM for a discriminative task in this case is to forget transforming it into an MLP, and simply use the approximate posterior means of hidden units as features (see, e.g., Montavon et al. (2012) and Publication VII). In other words, for each sample $\mathbf{x}$ we compute the variational parameters $\boldsymbol{\mu}$ by maximizing the variational lower bound in Eq. (5.9) with respect to them. Then the obtained variational parameters are used instead of the original sample. However, it is often obvious that a better discriminative performance is achieved when the model is specifically *finetuned* to optimize it.

Salakhutdinov and Hinton (2009a) proposed that the structure of an MLP be modified to simulate the top-down signal in a DBM. Given a DBM with L hidden layers $\left\{\mathbf{h}^{[l]}\right\}_{l=1}^{L}$ and a single visible layer $\mathbf{x}$, let us construct an MLP with L intermediate hidden layers $\left\{\tilde{\mathbf{h}}^{[l]}\right\}_{l=1}^{L}$ and a single output layer $\tilde{\mathbf{y}}$ and a single visible layer $\tilde{\mathbf{x}}$. The main goal of this construction is to make sure that a single forward pass results in the states of the units in the penultimate layer $\mathbf{h}^{[L]}$ of the MLP being identical to the mean-field approximation $\boldsymbol{\mu}^{[L]}$ of them.

The fixed point of the variational parameters of the first hidden layer that locally maximizes the variational lower bound in Eq. (5.9) is

$$\boldsymbol{\mu}^{[1]} = \phi\left(\mathbf{W}^\top \mathbf{x} + \mathbf{U}^{[1]}\boldsymbol{\mu}^{[2]}\right),$$

where ϕ is a component-wise logistic sigmoid function. Then, if we let the visible layer of the MLP to be $\tilde{\mathbf{x}} = \left[\mathbf{x}^\top, \boldsymbol{\mu}^{[2]\top}\right]^\top$ and connect $\mathbf{x}$ with the first hidden layer of the MLP by $\mathbf{W}$ and $\boldsymbol{\mu}^{[2]}$ by $\mathbf{U}^{[1]\top}$, a single forward pass will result in the activation of the first hidden layer of the MLP $\tilde{\mathbf{h}}^{[1]}$ to be exactly $\boldsymbol{\mu}^{[1]}$.

This applies similarly to all intermediate hidden layers of the DBM. For any l-th layer of the DBM, where $l < L - 1$, we constrain the l-th hidden layer of the MLP by appending $\boldsymbol{\mu}^{[l]}$ with $\boldsymbol{\mu}^{[l+2]}$. By connecting them to $\tilde{\mathbf{h}}^{[l+1]}$ with the corresponding weights from the DBM, we can ensure that the activation of the $(l+1)$-th hidden layer of the MLP will be initially identical to $\boldsymbol{\mu}^{[l+1]}$.

Since the last hidden layer of the DBM only receives the bottom-up signal, there will be no need to construct the last hidden layer in this way. Simply it is enough to

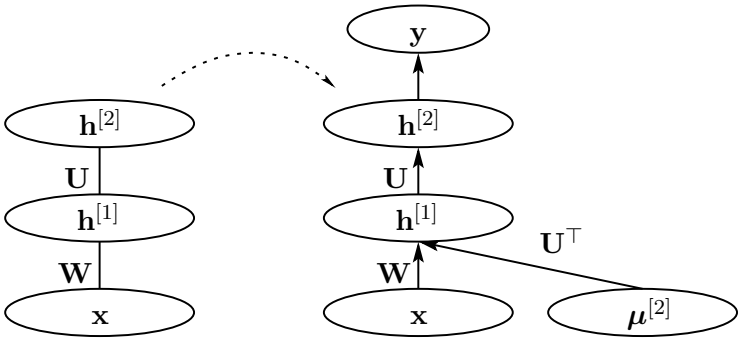


Figure 5.3. A deep Boltzmann machine with two hidden layers, on the left, is transformed to initialize a multi-layer perceptron on the right. $\boldsymbol{\mu}^{[2]}$ is a vector of the variational parameters of the second hidden layer of the DBM.

connect $\tilde{\mathbf{h}}^{[L]}$ with $\tilde{\mathbf{h}}^{[L-1]}$ by $\mathbf{U}^{[L-1]}$.

This way of constructing an MLP (see Fig. 5.3) guarantees that the activation of the last hidden layer of the MLP after a forward pass with the initialized weights will coincide with their variational parameters. From there on, we can use the backpropagation (see Section 3.4) to further finetune the model.

For instance, in (Salakhutdinov and Hinton, 2009a) and (Hinton et al., 2012), this way of initializing an MLP with a DBM was shown to improve the performance on handwritten digits as well as 3-D object recognition tasks.

5.3 Pretraining Generative Models

So far in this chapter we have described how unsupervised neural networks can be used to improve the performance of supervised tasks. In this section we will discuss how a simpler unsupervised neural network can help training more complex, deeper unsupervised neural networks with some theoretical guarantees.

Earlier in this chapter we have looked at the incremental feature learning from the perspective of *incremental transformation* that nonlinearly transforms an input into a better representation. Once the features are obtained by the incremental transformation, they are fed to another machine learning model, or the last pair of layers in an MLP, to perform a task whose objective was not necessarily utilized when learning the transformation. However, there is another perspective from which the incremental transformation can be viewed.

In Section 4.5.2, we described the learning algorithm for training a deep belief network (DBN). It is obvious now to see that the first stage of the learning algorithm does almost exactly what the incremental feature learning introduced earlier in this chapter does. Starting from a single restricted Boltzmann machine (RBM) trained on original training samples, we repeatedly stack an RBM trained on the aggregate posterior distribution of the previous RBM on top. However, the ultimate goal of this

almost the same procedure was very different. The goal of this procedure in training a DBN was to build a good *generative* model that learns the data distribution well, rather than to nonlinearly transform the input space so that another machine learning task will benefit from the new, better representation.

Ultimately it is a question of what or how much stacking another restricted Boltzmann machine on top of the existing deep neural network improves the performance. Furthermore, one might ask if another pretraining scheme, other than incrementally stacking shallow models, is possible.

In the remainder of this section, we first describe how an RBM can be viewed as an infinitely deep sigmoid belief network with tied weights (Hinton et al., 2006). Based on this observation we describe in more detail how stacking RBMs improves the generative performance of a DBN according to the argument given in (Hinton et al., 2006; Salakhutdinov and Hinton, 2012b). We then continue on to discuss how this scheme can be extended to initializing the parameters of a deep Boltzmann machine (DBM) based on (Salakhutdinov and Hinton, 2012b,a). At the end of the section, another pretraining scheme for DBMs proposed in Publication VII, utilizing such directed deep neural networks as deep autoencoders and DBNs is introduced.

5.3.1 Infinitely Deep Sigmoid Belief Network with Tied Weights

The most straightforward way to obtain samples from a sigmoid belief network is to sample from the conditional distribution of each layer given the states of the layer immediately above, starting from the top layer. An unbiased sample can be easily obtained from the top layer, since the prior distribution of the units in the top layer is factorized. Simply, for each variable in the top layer, we flip a coin with a probability decided by $\phi(b_i)$ where b_i is the bias and ϕ is a logistic sigmoid function. This way of gathering samples in a directed network is often known as *ancestral sampling* (see, e.g., Bishop, 2006; Murphy, 2012).

Let us construct a sigmoid belief network with infinitely many layers, given a set of parameters of an RBM. The bottom layer corresponds to the visible layer of the RBM, and there are directed edges from the layer above which corresponds to the hidden layer of the RBM. The weights of the edges are set to those of the RBM. Again, on top of the second layer, another layer that corresponds to the visible layer of the RBM is connected with the downward directed edges with their weights fixed to those of the RBM. We repeat this step infinitely, and we get the infinitely deep sigmoid belief network with the tied weights[2].

Let us perform ancestral sampling from a layer, very far up from the first layer

[2]For simplicity, we omit biases, but they can be considered a part of the weights.

denoted $\mathbf{x}^{[-L]}$, in the infinite limit of L that corresponds to the visible layer, starting from a random state. Next, we will sample from the conditional distribution of a layer immediately below $\mathbf{h}^{[-L]}$ that corresponds to the hidden layer of the RBM. Subsequently, we will repeatedly sample from the conditional distributions of $\mathbf{x}^{[-L+1]}$, $\mathbf{h}^{[-L+1]}$, $\mathbf{h}^{[-L+2]}$, $\mathbf{h}^{[-L+2]}$, $\ldots$, $\mathbf{h}^{[-1]}$, and $\mathbf{x}^{[0]}$, where $\mathbf{x}^{[0]}$ is the visible layer (See Fig. 5.4).

It is easy to see that this is exactly what Gibbs sampling does to get samples from the model distribution of an RBM. If this procedure starts from a layer far enough from the visible layer, the Gibbs chain will reach the equilibrium distribution by the time samples from the bottom layers were collected.

This means that if we instead of a random state perform the same sampling procedure while fixing all those layers that correspond to the visible layer of the RBM to a given sample, the samples of the other layers will represent the true posterior distribution which is factorized equivalently to the posterior distribution over the hidden units of the RBM. In other words, in this infinitely deep sigmoid belief network, we can sample *exactly* from the posterior distribution over the hidden units, because the weights are tied.

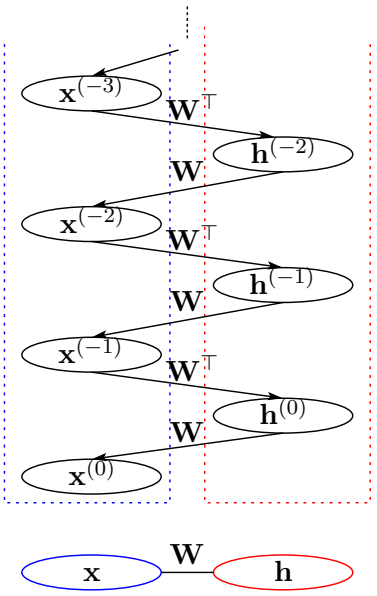


Figure 5.4. The ancestral sampling on an infinitely deep sigmoid belief network with tied weights is equivalent to the block Gibbs sampling on a restricted Boltzmann machine.

5.3.2 Deep Belief Network: Replacing a Prior with a Better Prior

Here, we take a more detailed look at the first stage of the learning algorithm of a DBN discussed in Section 4.5.2. The first stage consists of iteratively training an RBM on the samples collected from the posterior distribution of the hidden units of an RBM immediately below. Let us first consider a DBN with two layers of hidden units of which the second hidden layer has as many units as there are visible units.

The first RBM trained on a training set D with N samples learns

$$p(\mathbf{x} \mid \boldsymbol{\theta}_1) = \sum_{\mathbf{h}^{[1]}} p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_1) p(\mathbf{x} \mid \mathbf{h}^{[1]}, \boldsymbol{\theta}_1),$$

where $p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_1)$ is a *prior* distribution of the hidden units parameterized with $\boldsymbol{\theta}_1$

defined by

$$p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_1) = \sum_{\mathbf{x}} p(\mathbf{h}^{[1]}, \mathbf{x} \mid \boldsymbol{\theta}_1).$$

We used the superscript $[1]$ to indicate that the hidden units are in the first hidden layer of a DBN.

An interesting observation can be made here. The same set of the parameters $\boldsymbol{\theta}_1$ is used to model both the conditional distribution $p(\mathbf{x} \mid \mathbf{h}^{[1]})$ and the prior distribution $p(\mathbf{h}^{[1]})$. From this observation, it is natural to consider *replacing* the prior of $\mathbf{h}^{[1]}$ such that it is not anymore constrained to be modeled with the same parameters $\boldsymbol{\theta}_1$. In other words, we would like to replace $p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_1)$ with

$$p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_2) = \sum_{\mathbf{h}^{[2]}} p(\mathbf{h}^{[1]}, \mathbf{h}^{[2]} \mid \boldsymbol{\theta}_2).$$

First, we need to recall that the infinitely deep sigmoid belief network with *tied* weights is equivalent to an RBM from Section 5.3.1. Then, since the DBN considered here has as many units in $\mathbf{h}^{[2]}$ as there are in the visible layer $\mathbf{x}$, we get the same model even after replacing the prior, if we fix $\boldsymbol{\theta}_2$ to $\boldsymbol{\theta}_1$[3].

Let us write the variational lower bound of the marginal log-likelihood, as described in Eq. (2.24)–(2.26), for the DBN with two hidden layers:

$$\begin{aligned}\sum_{n=1}^{N} \log p(\mathbf{x}^{(n)} \mid \boldsymbol{\theta}) &\geq \sum_{n=1}^{N} \left(\mathbb{E}_{Q(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)})} \left[\log p(\mathbf{x}^{(n)}, \mathbf{h}^{[1]} \mid \boldsymbol{\theta}_1) \right] + \mathcal{H}(Q) \right) \\ &= \sum_{n=1}^{N} \left(\mathbb{E}_{Q(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)})} \left[\log p(\mathbf{x}^{(n)} \mid \mathbf{h}^{[1]}, \boldsymbol{\theta}_1) \right] + \mathcal{H}(Q) \right) \\ &\quad + \sum_{n=1}^{N} \mathbb{E}_{Q(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)})} \left[\log \sum_{\mathbf{h}^{[2]}} p(\mathbf{h}^{[1]}, \mathbf{h}^{[2]} \mid \boldsymbol{\theta}_2) \right], \quad (5.3)\end{aligned}$$

where $\mathcal{H}(Q)$ is the entropy functional of Q. This bound holds for any $Q(\mathbf{h}^{[1]} \mid \mathbf{x})$, but when $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_2$, we can use the true posterior $p(\mathbf{h}^{[1]} \mid \mathbf{x})$ to make the bound equal to the marginal log-likelihood.

In Eq. (5.3), we can see that only the last term is dependent on $\boldsymbol{\theta}_2$. This means that if we increase the last term which corresponds to training another RBM on the aggregate posterior[4] by using the stochastic approximation method, we can improve the bound, ignoring any possible stochastic fluctuation. This, however, will move Q away from the true posterior distribution, as $\boldsymbol{\theta}_2$ is now different from $\boldsymbol{\theta}_1$, which makes the bound less tight.

Once $\boldsymbol{\theta}_2$ is estimated, we can recursively perform this procedure to replace the prior distribution of $\mathbf{h}^{[2]}$, and so on. Each replacement will result in an improved bound,

[3] If we only consider the weights, assuming that the weights include the biases, $\mathbf{W}_{[2]} = \mathbf{W}_{[1]}^{\top}$. However, without loss of generality, we simply state that $\boldsymbol{\theta}_2 = \boldsymbol{\theta}_1$.

[4] The aggregate posterior was described in Section 4.5.2 as a mixture of N posterior distributions.

making the generative model better and better each time. The approximate posterior distribution Q, however, becomes less and less tight each time, and the second stage, called the up-down algorithm (Hinton et al., 2006), is needed to jointly estimate both the recognition and generation parameters of all layers.

This guarantee of improving the variational lower bound *only* holds when

1. The weights are initialized to be identical to those of the lower layer,
2. Samples from the aggregate posterior are used, and
3. RBMs are trained to maximize the log-likelihood.

However, in practice, most of these conditions are violated. The weights of the newly added RBM are often randomly initialized, and the probabilities rather than samples of the lower hidden units are used. Furthermore, in most cases due to the computational reason, RBMs are trained by minimizing the contrastive divergence.

Stochastic Units vs. Deterministic Units

Let us discuss slightly more about using the probabilities instead of actual samples as training samples for training an upper RBM. This can be considered as using a different approximate posterior distribution Q.

In the original proper pretraining procedure, the approximate posterior distribution

$$Q(\mathbf{h}) = \prod_{l=1}^{L} q_{\boldsymbol{\mu}^{[l]}}\left(\mathbf{h}^{[l]}\right)$$

is defined to be factorized only *layer-wise*. In other words, the hidden units are not mutually independent across layers, but only mutually independent inside each layer given the sampled activations of the units in the lower layer.

Because of this we obtain the variational parameters $\boldsymbol{\mu}^{[l]}$ iteratively, starting from the first hidden layer by

$$\boldsymbol{\mu}^{[l]} = \phi\left(\mathbf{W}_{[l-1]}^{\top}\tilde{\mathbf{h}}^{[l-1]} + \mathbf{b}_{[l]}\right), \tag{5.4}$$

where $\tilde{\mathbf{h}}^{[l-1]}$ is a vector consisting of elements sampled by

$$\tilde{h}_k^{[l-1]} \sim q\left(h_k^{[l-1]} = 1 \mid \mu_k^{[l-1]}\right).$$

Here ϕ is a sigmoid function as usual. This is equivalent to performing a feedforward pass on the encoder of a *stochastic* autoencoder (see Section 3.2.3).

We can design another approximate posterior distribution $\tilde{Q}$ such that the variational parameters $\boldsymbol{\mu}^{[l]}$ are obtained without actual sampling of hidden units. This new posterior distribution will correspond to using probability values instead of sampled activations to train an upper RBM.

Again, the approximate posterior distribution is defined by

$$\tilde{Q}(\mathbf{h}) = \prod_{l=1}^{L} \tilde{q}_{\tilde{\boldsymbol{\mu}}^{[l]}} \left(\mathbf{h}^{[l]} \right)$$

However, in this case we assume a *fully*-factorized distribution.

To cope with this assumption, the procedure for computing the variational parameters $\tilde{\boldsymbol{\mu}}^{[l])}$ should be modified accordingly. The variational parameters of the l-th layer are now computed recursively using the following formula:

$$\tilde{\boldsymbol{\mu}}^{[l]} = \phi \left(\mathbf{W}_{[l-1]}^{\top} \tilde{\boldsymbol{\mu}}^{[l-1]} + \mathbf{b}_{[l]} \right). \tag{5.5}$$

One can immediately see that this is equivalent to the encoder part of a *deterministic* autoencoder. However, the decoder part still differs from an autoencoder in the sense that it propagates down sampled activations, not the real-valued probabilities. Roughly put, this difference can be understood so that the decoder of the autoencoder approximates the generation path of a deep belief network by using again a fully factorized distribution. Table 5.1 summarizes these differences.

Exact ⟶ Approximate	DBN	DBN-FF	AE
Posterior	Layer-wise Factorial	Fully Factorial	Fully Factorial
Conditional	Layer-wise Factorial	Layer-wise Factorial	Fully Factorial

Table 5.1. The forms of approximate/exact distributions used by a deep belief network (DBN), a deep belief network pretrained by a stack of RBMs with probabilities used for training upper RBMs (DBN-FF), and a deep autoencoder (DAE).

The latter choice of an approximate posterior distribution has two obvious advantages. Firstly, the computation is easier, since no sampling is required. Also, the approximated variational parameters are less noisy, since no stochastic mechanism is involved. However, this choice nullifies the guarantee discussed earlier in Section 5.3.2.

Based on this and the similarity between the fully factorized approximate posterior and the encoder of a deterministic autoencoder, we informally conclude that in terms of generative modeling of data autoencoders may lag behind a deep belief network of the same structure. However, in practice where features extracted by these models are more interesting, the fully factorized approximate posterior is often used to train even a deep belief network. Furthermore, since the network can be further finetuned generatively by the up-down algorithm (see Section 4.5.2) later on, the importance of using a layer-wise factorized approximate posterior distribution using pretraining diminishes.

5.3.3 Deep Boltzmann Machine

In the paper where a deep Boltzmann machine was proposed, Salakhutdinov and Hinton (2009a) noticed that it is difficult to estimate the parameters well. Hence, they proposed a layer-wise pretraining scheme for deep Boltzmann machines that facilitates estimating its parameters (Salakhutdinov and Hinton, 2012b). The proposed layer-wise pretraining scheme was further improved in (Salakhutdinov and Hinton, 2012a) to better utilize the undirected nature of the connectivity in a deep Boltzmann machine.

Firstly, we must realize that that all the edges in a DBM are *undirected*. This is a critical difference from a DBN, when we consider the conditional distribution of a single intermediate hidden layer $\mathbf{h}^{[l]}$. In a DBN, this is simply conditioned on the layer immediately above $\mathbf{h}^{[l+1]}$, whereas the conditional distribution under a DBM is conditioned on both the layers immediately *above* and *below* such that

$$p(h_j^{[l]} = 1 \mid \mathbf{h}^{[l-1]}, \mathbf{h}^{[l+1]}, \boldsymbol{\theta}) = \phi\left(\sum_{k=1}^{q_{l-1}} h_k^{[l-1]} w_{kj}^{[l-1]} + \sum_{i=1}^{q_{l+1}} h_i^{[l+1]} w_{ji}^{[l]} + c_j^{[l]}\right),$$

where we follow the notations from Section 4.4.3.

Simply put, DBMs must be pretrained while taking into account that each hidden unit receives signal from both upper and lower layers. On the other hand, no such considerations are needed when pretraining a DBN. If the same pretraining method for DBNs is used, the conditional distribution of each hidden unit will be too peaked and saturated to prevent any further finetuning (jointly optimizing the whole layers) to improve the model.

Salakhutdinov and Hinton (2009a) proposed to modify the structure of RBMs to cope with this difference. For the bottom two layers, an RBM is modified to have two copies of visible units with tied weights such that the additional set of visible units supplies signal that compensates for the lack of signal from the second hidden layer. Similarly, an RBM that consists of the top two layers has two copies of hidden units. For any pair of intermediate hidden layers, an RBM is constructed to have two copies of both visible and hidden units. See Fig. 5.5 for illustration.

Recently, Salakhutdinov and Hinton (2012b) were able to show that the variational lower bound is guaranteed to increase by adding the top hidden layer using the proposed pretraining scheme. Although their proof only applies to the top layer, it is worth discussing it in relation to the pretraining scheme for DBNs.

We can rewrite the variational lower bound of a DBN having two hidden layers,

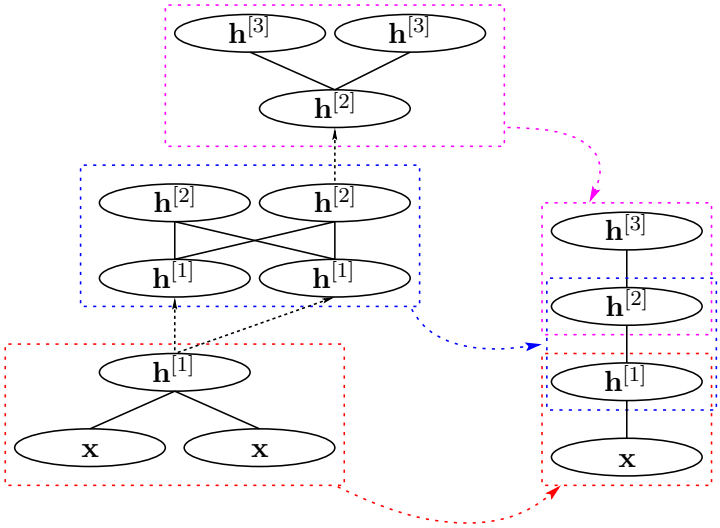


Figure 5.5. Illustration of the layer-wise pretraining of a deep Boltzmann machine. The dashed directed lines indicate *copying* of either pretrained models or the activations of the hidden units of the pretrained models.

given in Eq. (5.3) as

$$\begin{aligned}\sum_{n=1}^{N} \log p(\mathbf{x}^{(n)} \mid \boldsymbol{\theta}) \geq \sum_{n=1}^{N} \Bigg(& \mathbb{E}_{Q(\mathbf{h}^{[1]}|\mathbf{x}^{(n)})} \left[\log p(\mathbf{x}^{(n)} \mid \mathbf{h}^{[1]}, \boldsymbol{\theta}_1) \right] \\ & + \mathbb{E}_{Q(\mathbf{h}^{[1]}|\mathbf{x}^{(n)})} \left[\log \frac{p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_2)}{Q(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)})} \right] \Bigg) \\ = \sum_{n=1}^{N} \Bigg(& \mathbb{E}_{Q(\mathbf{h}^{[1]}|\mathbf{x}^{(n)})} \left[\log p(\mathbf{x}^{(n)} \mid \mathbf{h}^{[1]}, \boldsymbol{\theta}_1) \right] \\ & - \mathrm{KL} \left(Q(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)}) \| p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_2) \right) \Bigg) . \end{aligned} \tag{5.6}$$

From this, it is clear that replacing the prior of the first hidden layer to improve the lower bound is equivalent to estimating $\boldsymbol{\theta}_2$ such that the new prior distribution $p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_2)$ becomes closer to the aggregate posterior, since the KL-divergence between two distributions is non-negative and becomes zero only when they are identical.

If we train the first RBM having two copies of visible units, following the pretraining algorithm of a *DBM*, we can compute the marginal or prior distribution over $\mathbf{h}^{[1]}$ by using the fact that by considering $\mathbf{h}^{[1]}$ as a visible layer, the whole model is simply a product-of-expert (PoE) model (see Section 4.4.2) with a hidden layer consisting of $\mathbf{x}$ and $\mathbf{h}^{[2]}$. Note that we used $\mathbf{h}^{[2]}$ to denote the second copy of the visible units. The prior distribution of $\mathbf{h}^{[1]}$ is then

$$p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_1) = \frac{1}{Z(\boldsymbol{\theta}_1)} \left(\sum_{\mathbf{x}} p(\mathbf{h}^{[1]}, \mathbf{x} \mid \boldsymbol{\theta}_1) \right) \left(\sum_{\mathbf{h}^{[2]}} p(\mathbf{h}^{[1]}, \mathbf{h}^{[2]} \mid \boldsymbol{\theta}_1) \right) . \tag{5.7}$$

Since both $\mathbf{x}$ and $\mathbf{h}^{[2]}$ are free variables in Eq. (5.7), we can improve the variational lower bound in Eq. (5.6) by replacing it with an RBM having two copies of hidden

units with tied parameters $\boldsymbol{\theta}_2$. If we start by fixing $\boldsymbol{\theta}_2 = \boldsymbol{\theta}_1$ and follow the steepest gradient direction with the stochastic gradient method, the KL-divergence between the aggregate posterior and the new prior distribution will be guaranteed to decrease, or stay identical at least, which amounts to improving the lower bound. The new prior distribution can be similarly written as

$$p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_2) = \frac{1}{Z(\boldsymbol{\theta}_2)} \left(\sum_{\mathbf{x}} p(\mathbf{h}^{[1]}, \mathbf{x} \mid \boldsymbol{\theta}_2) \right) \left(\sum_{\mathbf{h}^{[2]}} p(\mathbf{h}^{[1]}, \mathbf{h}^{[2]} \mid \boldsymbol{\theta}_2) \right). \tag{5.8}$$

With these two RBMs, we can form a DBM with two hidden layers, by replacing the half of the original prior distribution in Eq. (5.7) with the half of the new distribution in Eq. (5.8) such that

$$p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_1, \boldsymbol{\theta}_2) = \frac{1}{Z(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)} \left(\sum_{\mathbf{x}} p(\mathbf{h}^{[1]}, \mathbf{x} \mid \boldsymbol{\theta}_1) \right) \left(\sum_{\mathbf{h}^{[2]}} p(\mathbf{h}^{[1]}, \mathbf{h}^{[2]} \mid \boldsymbol{\theta}_2) \right),$$

which is guaranteed to have a smaller KL-divergence than the aggregate posterior (Hinton, 2002).

Hence, adding one more layer on top of an existing RBM is guaranteed to improve the variational lower bound, if the proposed pretraining method is used. However, this procedure does not extend trivially to the intermediate hidden layers.

This mathematical justification suggests that the pretraining method for a DBM differs from that for a DBN in a sense that only a *half* of a prior distribution is replaced by the upper layer. Conversely, it states that the existing layer is still used to model the remaining half of the prior distribution, whereas in a DBN the lower layer only concentrated on modeling the conditional generative distribution given the state of units in the upper layer.

In fact, this way of justifying the pretraining algorithm allows to extend the algorithm so that the amount of modeling the prior distribution is distributed *unevenly*. For instance, Salakhutdinov and Hinton (2012a) proposed to use *three* copies of visible and hidden units, respectively, when training the bottom and top RBMs. This is equivalent to replacing two thirds of the prior distribution by the top RBM. They were able to show that better generative models could be learned by DBMs with this approach.

Two-Stage Pretraining: From Autoencoders To Deep Boltzmann Machine

This layer-wise approach is not the only pretraining method available for DBMs. In Publication VII, another approach that utilizes a deep belief network or a deep autoencoder is proposed.

Let us look at the variational lower bound of the marginal probability of $\mathbf{x}$ under a DBM using a fully factorized approximate posterior distribution

$Q(\mathbf{h}) = \prod_{l=1}^{L} \prod_{j=1}^{q_l} q(h_j^{[l]})$, where $q(h_j^{[l]} = 1) = \mu_j^{[l]}$:

$$\log p(\mathbf{x} \mid \boldsymbol{\theta}) \geq -E(\mathbf{x}, \boldsymbol{\mu}) + \mathcal{H}(Q) - \log Z(\boldsymbol{\theta}), \tag{5.9}$$

where $\boldsymbol{\mu}$ is a vector of all variational parameters.

The parameters of a DBM can be estimated by plugging the EM algorithm (see Section 2.3.2) in the stochastic approximation procedure (see Section 4.3.3). At each update step, we first perform the E-step of the EM algorithm, which is equivalent to maximizing Eq. (5.9) with respect to the variational parameters $\boldsymbol{\mu}$. Then, with the fixed $\boldsymbol{\mu}$, we compute the stochastic gradient and update the parameters, which corresponds to the M-step.

The gradient of the marginal log-likelihood of a Boltzmann machine tries to match two distinct distributions. One distribution is the model distribution characterized by a mixture of products of visible samples and the corresponding posterior distributions over hidden units. The other distribution, the data distribution, is a mixture of training samples and the corresponding posterior distribution. The gradient drives the BM by pushing the model distribution closer to the data distribution.

At the beginning of learning, as parameters were initialized to have small magnitude, the variational posterior distribution of hidden units in the deep layers is effectively *random* in a sense that each hidden unit is likely to be 0 or 1 with equal probabilities. This leads to almost zero gradient with respect to the parameters in the deep layers, since the (variational) posterior distributions under the both data and model distributions match already. Then, it is unlikely that the stochastic gradient method will make the deep hidden layers useful. In other words, it is likely that the hidden units in upper layers will stay random even after many updates, which was noticed in Publication VI.

Hence, in Publication VII it was claimed that it may be important to have a sensible variational posterior distribution to start with. To obtain a sensible variational posterior distribution, the two-stage pretraining algorithm proposed in the same paper uses a deep directed neural network, such as a deep belief network or a deep autoencoder, that has the similar structure as the target DBM.

This way of *borrowing* an approximate posterior distribution from another model is informally justified from the fact that the lower bound in Eq. (5.9) holds for *any* distribution Q. Thus, we can maximize the variational lower bound by updating the parameters using the stochastic gradient method, with any fixed Q. This amounts to moving the true posterior distribution to approximately match the arbitrary approximate posterior distribution.

Once the true posterior distribution is close enough to the fixed Q, the variational lower bound can be further maximized using the standard EM approach. In other

words, after some updates we can *free* the variational posterior distribution Q and let it be estimated by maximizing Eq. (5.9) with respect to the variational parameters $\boldsymbol{\mu}$.

Since the units $\mathbf{h}_+$ in the odd-numbered hidden layers can be explicitly summed out, one only needs to borrow the approximate posterior distribution only for those units in the even-numbered hidden layers. This significantly reduces the computational time required to train a deep, directed neural network from which the approximate posterior is borrowed.

Let us rewrite Eq. (5.9) to marginalize out the units in the odd-numbered hidden layers:

$$\mathbb{E}_{p_D(\mathbf{x})}\left[\log p(\mathbf{x}^{(n)} \mid \boldsymbol{\theta})\right] \geq \mathbb{E}_{p_D(\mathbf{x})Q(\mathbf{h}_-|\mathbf{x})}\left[\log \sum_{\mathbf{h}_+} \exp\left\{-E(\mathbf{x}^{(n)}, \mathbf{h}_-, \mathbf{h}_+)\right\}\right] + \mathcal{H}(Q) - \log Z(\boldsymbol{\theta}), \tag{5.10}$$

where $p_D(\mathbf{x})$ is the data distribution which is approximated by the set of training samples.

It becomes immediately apparent that the first term of the lower bound in Eq. (5.10) is equivalent to the marginal log-likelihood of an RBM having visible units $[\mathbf{x}, \mathbf{h}_-]$ and hidden units $\mathbf{h}_+$ with the data distribution $p_D(\mathbf{x})Q(\mathbf{h}_- \mid \mathbf{x})$. Hence, with the fixed Q, we can maximize the lower bound efficiently using all the available techniques, such as minimizing contrastive divergence (see Section 4.4.2), the enhanced gradient (see Section 4.1.1) and advanced MCMC sampling methods (see Section 4.3.1), for training RBMs. This stage is referred to as the *second* stage in the two-stage algorithm.

The arbitrary approximate posterior distribution Q is found in the first stage. For instance, consider having a deep belief network that has a visible layer corresponding to $\mathbf{x}$, and a number of hidden layers corresponding to the even-numbered hidden layers $\mathbf{h}_-$ of the DBM. Then we can perform a layer-wise pretraining described earlier in this chapter, to estimate the *recognition* parameters that represent the approximate posterior distribution of the DBN.

We may also use a deep autoencoder that consists of the input and output layers corresponding to $\mathbf{x}$ and two copies of hidden layers that correspond to $\mathbf{h}_-$. Once trained, either with or without a layer-wise pretraining, we may use its encoder part to approximate the posterior distribution over the hidden units.

In summary, the two-stage algorithm borrows an approximate posterior distribution from another model trained in the first stage, and maximizes the variational lower bound with the variational posterior fixed to the borrowed posterior distribution[5], as

[5]We acknowledge here that a similar idea of borrowing an approximate posterior distribution

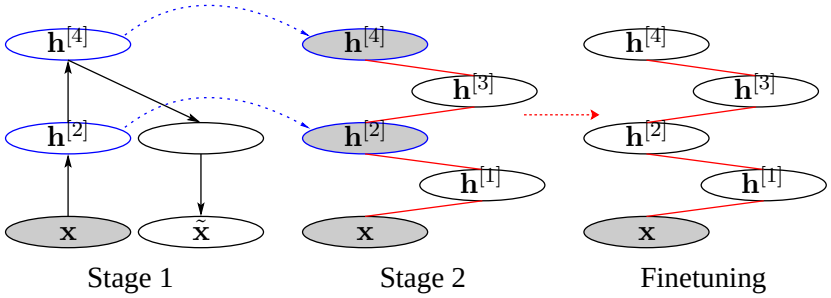


Figure 5.6. Illustration of the layer-wise pretraining of a deep Boltzmann machine. The dashed directed lines indicate *copying* of either pretrained models or the activations of the hidden units of the pretrained models. In this figure, a deep autoencoder is used to learn an arbitrary approximate posterior in the first stage. The red-colored edges indicate that the weights parameters learned in the second stage are used as initial values when finetuning the DBM. Note that the parameters learned in the first stage are discarded immediately after the first stage.

if the DBM were an RBM. See Fig. 5.6 for illustration.

This approach was empirically shown to be very effective at learning a good generative model using a DBM in Publication VII. Furthermore, DBMs with Gaussian visible units trained using this approach were found to be good at denoising images corrupted with high level of noise in Publication IX.

from another model was used for a variational Bayesian nonlinear blind source separation method by Honkela et al. (2004).

6. Discussion

Lately deep neural networks have shown remarkable performances in various machine learning tasks. They include[1], but are not limited to, speech recognition (see, e.g., Hinton et al., 2012; Dahl et al., 2012) and large-scale object recognition (see, e.g., Krizhevsky et al., 2012; Hinton et al., 2012) as well as natural language processing (see, e.g., Socher et al., 2011). In these cases, deep neural networks were able to outperform the conventional models and algorithms significantly.

Based on these advances in academic research, deep neural networks have rapidly found their ways into commercial use through companies such as Google, Microsoft and Apple. For instance, Google Goggles[2] uses a deep belief network (see Section 4.5.2) in production[3]. Microsoft has replaced the existing speech recognition algorithm based on Gaussian mixtures with one based on deep neural networks (Deng et al., 2013). Furthermore, the voice assistance feature of Apple's iPhone, called Siri, as well as Google's Street View are also known to utilize deep neural networks[4].

These recent successes of deep neural networks in both academic research and commercial applications may be attributed to several recent breakthroughs. Layerwise pretraining of a multi-layer perceptron proposed by (Hinton and Salakhutdinov, 2006; Bengio et al., 2007; Ranzato et al., 2007b) showed that a clever way of initializing parameters can easily overcome the difficulties of optimizing a large, deep neural network. Furthermore, this new pretraining scheme was found to be a useful way to incorporate a large amount of unlabeled data in addition to a small number of labeled data. This led to a success of deep neural networks, for example, in unsupervised and

[1] For a more comprehensive list, see, for instance, (Bengio et al., 2013a).

[2] http://www.google.fi/mobile/goggles/

[3] See the invited talk *Machine Learning in Google Goggles* by Hartmut Neven at the International Conference on Machine Learning 2011: http://techtalks.tv/talks/machine-learning-in-google-goggles/54457/

[4] See the article *Scientists See Promise in Deep-Learning Programs* by John Markoff featured in the New York Times on 23 November 2012:
http://www.nytimes.com/2012/11/24/science/scientists-see-advances-in-deep-learning-a-part-of-artificial-intelligence.html

transfer learning tasks (Guyon et al., 2011; Mesnil et al., 2012; Raina et al., 2007, see, e.g.,). Beside the layer-wise pretraining as well as unsupervised learning, several modifications to multi-layer perceptrons, such as novel nonlinear hidden units (see, e.g., Nair and Hinton, 2010; Glorot et al., 2011; Goodfellow et al., 2013) and dropout regularization (Hinton et al., 2012), have further pushed the performance boundary of deep neural networks.

Despite these recent breakthroughs that brought in the recent surge of popularity[5] of deep neural networks, the core principles of building deep neural networks have evolved rather gradually over more than 50 years since Rosenblatt (1958) proposed a perceptron. Over those years many seemingly distant classes of neural networks as well as other machine learning models have been proposed and later found to be closely related (see, e.g., Haykin, 2009). In due course many concepts, mainly from probabilistic approaches and semi-supervised learning, have been absorbed by the field of neural networks to form the solid basic principles of deep neural networks.

The main text of this thesis has been written to show how different perspectives and concepts of machine learning interact with each other to form basic principles of deep neural networks.

In this last chapter, the author briefly summarizes the main contents and continues on to discussing potential future research directions. At the end of this chapter, some important models, concepts as well as practical matters that have not been discussed in this thesis are briefly explained.

6.1 Summary

In this thesis, the author has aimed to reveal relationships among different neural networks as well as other machine learning methods. Mainly, two classes of deep neural networks, namely autoencoders and Boltzmann machines, were discussed in detail and found to be related to each other. The underlying principles of those two models were found to be useful in understanding layer-wise pretraining of a deep multi-layer perceptron.

The thesis began with simple, linear neural networks. Linear regression and perceptrons were first described in terms of neural networks, and later the equivalent models were formulated from probabilistic perspective. Similarly, a linear autoencoder was shown to be equivalent to principal component analysis as well as its probabilistic

[5] As an example of the increasing popularity of deep neural networks and the field of deep learning, MIT Technological Review selected deep learning as one of the ten breakthrough technologies in 2013:
http://www.technologyreview.com/featuredstory/513696/deep-learning/

variant.

The author went on to describing their deeper, nonlinear versions which are a multi-layer perceptron in the case of supervised models, and a deep autoencoder in the case of unsupervised models. Especially, the thesis focused mainly on autoencoders. The autoencoder was interpreted as performing approximate inference and generation in a probabilistic latent variable model. Furthermore, the author explained a geometric interpretation of the states of hidden units encoded by a variant of autoencoders based on the manifold assumption which constitutes a central part of semi-supervised learning.

Independently from autoencoders, the thesis discussed Boltzmann machines extensively. Starting from Hopfield networks which may be seen as a deterministic variant of Boltzmann machines, the author described in detail estimating statistics and learning parameters of Boltzmann machine as well as their structurally restricted variants such as restricted Boltzmann machines (RBM) and deep Boltzmann machines. Furthermore, a justification for viewing Boltzmann machines with hidden units as deep neural networks was provided in connection to recurrent neural networks.

These two seemingly separate classes of neural networks, autoencoder and Boltzmann machines, were shown to be closely related to each other. Especially, an autoencoder with a single hidden layer was described to be an approximation of an RBM. The author introduced recent studies showing their relatedness as well as equivalence. Additionally, a deep belief network was described as a combination of an RBM and a deep autoencoder with stochastic units.

At the end of the thesis, the author has explained recently introduced method of pretraining a deep multi-layer perceptron. This method of pretraining, called layer-wise pretraining, is analyzed from two different viewpoints. Firstly, the fact that some variants of autoencoders capture the data manifold was used to view the layer-wise pretraining as a way of incrementally extracting more useful features. Secondly, the author explained how stacking another layer on top of the existing neural network in the layer-wise pretraining can guarantee the improvement in the variational lower bound of the marginal log-likelihood.

6.2 Deep Neural Networks Beyond Latent Variable Models

Two different perspectives from which deep neural networks can be understood have been discussed in this thesis. One is based on the idea of incrementally capturing data manifold through stacking multiple layers of nonlinear hidden units. The other considers a feedforward computation of a deep neural network as performing an approximate inference of the posterior distribution over hidden units in higher layer.

The latter perspective essentially divides a deep neural network into two distinct parts. The first part is a latent variable model that models the distribution of training samples without labels, and then the second part makes a decision based on the inferred posterior distribution given a sample, on which class the sample belongs to. For example, a forward pass computation up until the penultimate layer of a multi-layer perceptron (MLP, see Section 3.1) corresponds to performing an approximate inference, and the computation from the penultimate layer to the output layer to decision-making.

In this framework of a latent variable model, the most exact way of performing a classification or a decision-making is to marginalize out hidden variables $\mathbf{h}$ to obtain the conditional distribution of missing variables $\mathbf{x}_m$ given the states of observed variables $\mathbf{x}_o$:

$$p(\mathbf{x}_m \mid \mathbf{x}_o) = \sum_{\mathbf{h}} p(\mathbf{x}_m \mid \mathbf{h}) p(\mathbf{h} \mid \mathbf{x}_o).$$

However, since this exact marginalization is often computationally intractable, a deep neural network replaces this by

$$p(\mathbf{x}_m \mid \mathbf{x}_o) \approx p(\mathbf{x}_m \mid \mathbf{h}) Q(\mathbf{h} \mid \mathbf{x}_o) \approx \tilde{p}(\mathbf{x}_m \mid \mathbf{x}_o) = p\left(\mathbf{x}_m \mid \mathbf{h} = \mathbb{E}_Q\left[\mathbf{h} \mid \mathbf{x}_o\right]\right)$$

using a parametric nonlinear function that computes $\mathbb{E}_{\tilde{p}}\left[\mathbf{x}_m \mid \mathbf{x}_o\right]$ in a single sweep.

Assuming a simple, unimodal distribution $p(\mathbf{x}_m \mid \mathbf{h})$, one consequence of this approximation is that the approximate predictive distribution $\tilde{p}(\mathbf{x}_m \mid \mathbf{x}_o)$ loses most of information in the true predictive distribution $p(\mathbf{x}_m \mid \mathbf{x}_o)$. It is natural since $\tilde{p}(\mathbf{x}_m \mid \mathbf{x}_o)$ is unimodal, while the true predictive distribution has many probabilistic modes.

The inherent limitations of this approximate approach employed by deep neural networks are obvious[6]. There is no guarantee that the parametric form employed by a deep neural network of an approximate posterior distribution Q is good enough to make the above approximation close to the exact marginalization. Furthermore, a usual method of fitting the variational parameters of Q to minimize the Kullback-Leibler divergence between Q and the true posterior distribution $p(\mathbf{h} \mid \mathbf{x}_o)$ tends to find only a single mode of the true posterior distribution, but we usually cannot tell how representative the found mode is. If the true posterior distribution is highly multi-modal, this approximation based on an arbitrary mode will be generally poor. Lastly, it is not obvious how a deep neural network can cope with a more flexible setting where the observed components are not fixed a priori[7].

[6] Note that the discussion in this section has been highly motivated and influenced by Section 5 of (Bengio, 2013).

[7] In a classical setting of, for instance, classification, we know in advance that label components are not going to be observed but all other components are.

A way has been proposed to overcome each of these limitations. For instance, one may bypass the problem of marginalization or approximate inference by directly mapping from an input $\mathbf{x}_o$ to the distribution of $\mathbf{x}_m$ given $\mathbf{x}_o$ which may have been learned by another model such as a restricted Boltzmann machine (Mnih et al., 2011). In this way, a deep neural network can learn to approximate a true predictive distribution $p(\mathbf{x}_m \mid \mathbf{x}_o)$ without going through an extra step of approximation in the middle. However, if the ultimate goal is to make a decision that maximizes the predictive distribution, we still need to be able to evaluate either approximately or exactly the multi-modal predictive distribution quickly and well, which brings us back to one of the limitations of the current approach based on the approximate inference of hidden variables.

A Boltzmann machine provides a principled way to overcome the problem of having to use an approximate inference of the posterior distribution over hidden variables. Instead of an variational approximation, one may perform an (asymptotically) exact inference utilizing Markov chain Monte Carlo (MCMC) sampling such that

$$p(\mathbf{x}_m \mid \mathbf{x}_o) \approx \frac{1}{T} \sum_{t=1}^{T} p(\mathbf{x}_m \mid \mathbf{h}^{\langle t \rangle}),$$

where $\mathbf{h}^{\langle t \rangle}$ is the t-th sample from $p(\mathbf{h} \mid \mathbf{x}_o)$. In fact, it is natural with a Boltzmann machine to consider any combination of observed components and missing components. Therefore this approach is tempting, but as the size of a model grows and the number of modes in the predictive distribution increases, it becomes impractical to use an MCMC sampling for making a rapid decision.

Hence, we want to have a radically new neural network that keeps the best of the two types of deep neural networks discussed throughout this thesis. A fast and efficient computation of feedforward neural networks (see Chapter 3) is required for a rapid decision-making, while most information and structure contained in a complex multi-modal predictive distribution must be maintained, just like a Boltzmann machine is able to learn a multi-modal distribution (see Chapter 4). To the author's current knowledge, there is no such neural network at the moment[8]. It is, hence, left for future to build such a neural network that combines these two very different characteristics.

The ultimate goal of deep neural networks and the field of deep learning will be to build a large deep neural network that can learn

$$f(\mathbf{x}_o \mid \boldsymbol{\theta}) = \arg\max_{\mathbf{x}_m} p(\mathbf{x}_m \mid \mathbf{x}_o),$$

where $\boldsymbol{\theta}$ denotes a set of parameters, and the indices of observed and missing components are not fixed a priori. A deep neural network that computes this function

[8] A few recent works are showing some promising directions using stochastic neural networks Bengio and Thibodeau-Laufer (see, e.g., 2013); Tang and Salakhutdinov (see, e.g., 2013).

will have to be powerful and flexible enough to consider all different possibilities or modes in the predictive distribution in a single sweep of the network in feedforward manner.

6.3 Matters Which Have Not Been Discussed

Despite the original intention of describing as much detailed as possible the basics of deep neural networks, some important models and concepts have been left out from the main text. The author would like to make brief notes on a few of them here to provide any interested reader further references.

The remainder of this section starts by providing a list of the previous works that relate independent component analysis as well as general factor analysis to the models discussed previously. Additionally, some references that attempted to make these models deeper are presented.

The models described so far in this thesis are fairly powerful in the sense that many of them have the universal approximator property. In Section 3.1, it was stated that a multi-layer perceptron has the universal approximator property, but no results about the other models were presented nor described. Thus, a list of previous research results which showed and proved the universal approximator property of the models discussed earlier in this dissertation is provided.

Subsequently, we discuss how a Boltzmann machine can be evaluated. Unlike most feedforward neural networks, the exact computation of the objective function in training a Boltzmann machine cannot be done tractably. This is due to both the intractability of the normalization constant and the difficulty of marginalizing out hidden units. Hence, the author discusses some approaches that can approximately evaluate Boltzmann machines.

Throughout this thesis, there was no discussion on selecting hyper-parameters that are necessary for training a deep neural network. Section 6.3.4 briefly describes a problem of hyper-parameter selection in one particular setting of pretraining and fine-tuning a multi-layer perceptron, and introduces recently proposed hyper-parameter optimization approaches based on Bayesian optimization.

This thesis is directed towards deep neural networks, but it must be reminded that not all models nor research directions have been covered. For instance, the entire text was written assuming that all training samples are independent and identically distributed, ignoring any temporal dependence. However, a recurrent neural network described in Section 4.2.1 is capable of learning temporal dependencies between samples.

Furthermore, it was implicitly assumed that a target task is permutation invariant,

meaning that the structure of a deep neural network is independent from the order of the components of an input vector. This ensures that the models discussed in this thesis are generally applicable to any data without prior knowledge. However, this may not be an optimal approach especially in a task which exhibits clear spatial structures among input components. One such case involves handling images, and at the end of this section we briefly discuss an approach based on convolutional neural networks that specifically aims at handling images.

6.3.1 Independent Component Analysis and Factor Analysis

One important model which belongs to a family of linear generative models is independent component analysis (ICA, see, e.g., Hyvärinen et al., 2001, and references therein). This model is closely related to many models we have discussed. For instance, ICA formulated using the information-theoretic approach by Bell and Sejnowski (1995) is equivalent to the maximum likelihood solution of sparse coding (see Section 3.2.5) in the limit of no noise when the numbers of inputs and sources are same (Olshausen and Field, 1997). Furthermore, by replacing the orthogonality constraint with the minimal reconstruction regularization, it was shown by Le et al. (2011a) that a linear autoencoder (see Section 2.2.1) with a soft-sparsity regularization on hidden activations is equivalent to ICA. Similarly, an approach that extracts principal components can be extended to extract independent components by employing certain nonlinear activation functions for hidden units (see, e.g., Oja, 1997; Hyvärinen et al., 2001). ICA is further related to a restricted Boltzmann machine (see Section 4.4.2) via an energy-based model proposed by Teh et al. (2003).

There have been approaches to extend basic ICA which assumes a single layer of sources, or hidden units, to have more than one hidden layers. For instance, Lappalainen and Honkela (2000) proposed a nonlinear generative model where the visible variables are generated through multiple layers of nonlinear hidden units starting from the mutually independent top hidden units, or sources. They called this model Bayesian nonlinear independent component analysis. This qualifies as a deep neural network by satisfying the first two conditions provided in Section 2.4.

When put into a probabilistic framework, most of these generative models, including principal component analysis, ICA and sparse coding, with directed edges are special cases of factor analysis with certain assumptions. Factor analysis assumes that the observation has been generated from a number of factors, or hidden units, via a certain mapping. In basic factor analysis the mapping is assumed to be linear, and the factors follow Gaussian distribution (see, e.g., Bishop, 2006, and references therein).

Similarly to ICA, factor analysis has also been extended into a nonlinear model

with potentially a *deep* structure. Raiko (2001) and Raiko et al. (2007), for instance, describe a hierarchical nonlinear factor analysis method which starts from a set of factors at top and generates sequentially a layer of stochastic hidden units until the visible layer of observed variables. This hierarchical model can be considered as a generalization of the sigmoid belief network (see Section 3.2.3) such that hidden units are not restricted to be binary. A similar model which replaces binary units of the sigmoid belief network with Gaussian units with squashing nonlinearity functions applied was proposed by Frey and Hinton (1999).

6.3.2 Universal Approximator Property

In Section 3.1, it was mentioned that a multi-layer perceptron (MLP) with a single hidden layer has the universal approximator property provided that there are enough hidden units. It is natural to question whether any other model discussed in this dissertation has the same property.

Support vector machines with certain kernel functions have the universal approximator property (Hammer and Gersmann, 2003). Furthermore, an extreme learning machine was shown to have the same property (Huang et al., 2006a). Deep autoencoders are obviously universal approximators in the sense that they can reconstruct any input sample arbitrary well, as each of them is equivalent to an MLP.

Similarly, universal approximator property can be defined for unsupervised models such that a model has the universal approximator property if any distribution can be modeled arbitrary well with respect to a chosen divergence criterion by the model.

Le Roux and Bengio (2008) proved that a restricted Boltzmann machine (RBM, see Section 4.4.2) has this property when any binary distribution is considered. This naturally extends to the fact that a deep Boltzmann machine (DBM, see Section 4.4.3) and a fully-connected Boltzmann machine are both universal approximator, since they are more general models of an RBM. Later, the same authors in (Le Roux and Bengio, 2010) showed that a deep belief network (see Section 4.5.2) has the same property on any binary distribution with the upper-bound on the number of units in each hidden layer, albeit requiring exponentially many hidden layers with respect to the input dimensionality.

In Publication VI, it was shown that an equivalent DBM with Gaussian visible units (GDBM) can be constructed for any mixture of Gaussians (MoG, see, e.g., Bishop, 2006). Since MoGs are universal approximators, so are GDBMs. However, this argument does not apply to RBMs with Gaussian visible units (GRBM, see Section 4.5.1), since not all MoGs can be modeled by GRBMs.

6.3.3 Evaluating Boltzmann Machines

One important reason that makes training a Boltzmann machine more difficult than, for instance, an autoencoder is that the exact computation of the cost function is intractable. This is an obvious consequence from having an intractable normalization constant in its formulation (see Eqs. (4.2)–(4.3)). Also, except for restricted Boltzmann machines (RBM), it is intractable to marginalize hidden units.

Salakhutdinov (2008) proposed to first estimate the normalization constant using annealed importance sampling (Neal, 1998) and then to use the estimated constant for computing the variational lower bound (see Eq. (4.17)) of either training or test samples. The average of multiple runs of annealed importance sampling can compute an unbiased estimate of the normalization constant. Empirical evidence (Salakhutdinov, 2008) showed that the variance of the estimates is sufficiently small even with only a small number of runs. Furthermore, the variational lower bound turned out to be surprisingly tight in the case of deep Boltzmann machines (Salakhutdinov and Hinton, 2012b).

Based on the idea of the annealed importance sampling and parallel tempering (see Section 4.3.1), Desjardins et al. (2011) introduced a method of tracking the normalization constant while training a restricted Boltzmann machine. Also, in Publication II and Publication VI an adaptive learning rate which estimates the local change in the log-likelihood, or its variational lower bound up to the constant multiple, was proposed based on the idea of the annealed importance sampling.

In the case of RBMs, a crude approximation of the cost function can be computed based on the connection between the RBM and the autoencoder. Especially, when the RBM is trained by minimizing the contrastive divergence (see Section 4.4.2), the reconstruction error (see Section 4.5.1) may be used to monitor the learning progress (Hinton, 2012). However, as pointed out in (Hinton, 2012), the reconstruction error is *not* an absolute measure of the performance of an RBM.

6.3.4 Hyper-Parameter Optimization

One might have noticed throughout this thesis that there are quite many hyper-parameters involved in training these deep neural networks. Typically, training an unsupervised neural network with a single hidden layer involves, for instance in the case of a denoising autoencoder, nine hyper-parameters which are:

1. The number of hidden units
2. Noise variance
3. Drop ratio

4. Initial learning rate
5. Learning rate scheduling
6. Momentum
7. Weight decay constant
8. Target sparsity
9. Sparsity regularization constant

Hence, if we use a denoising autoencoder to pretrain a deep multi-layer perceptron (MLP), this number of the hyper-parameters is multiplied by the number of hidden layers in the MLP. Furthermore, there are 6 more hyper-parameters that need to be further tuned for finetuning by backpropagation, which are

1. Noise variance
2. Drop ratio
3. Initial learning rate
4. Learning rate scheduling
5. Momentum
6. Weight-decay

In total, if we have to train a deep MLP using layer-wise pretraining by denoising autoencoders, we need to choose $9L + 6$ hyper-parameters correctly.

When the number of hyper-parameters is low, it is usual to use the grid-search to find the best set. It is, however, less preferable in the case of deep neural networks, since the number of candidate points on the grid grows exponentially with respect to the number of hyper-parameters which grows linearly with respect to the number of hidden layers.

This is highly problematic considering that training even a single deep neural network is computationally expensive. One cannot allow training exponentially many deep neural networks only for selecting a set of hyper-parameters. Hence, a hyper-parameter optimization method for deep neural networks must be able to find only a *small* set of *good* candidate points in the hyper-parameter space.

Based on this motivation, two recent studies (Bergstra et al., 2011; Snoek et al., 2012) explored the idea of utilizing Bayesian optimization (see, e.g., Brochu et al., 2010, and references therein). The Bayesian optimization simultaneously models the posterior distribution over a function $h(\Psi)$ that maps from a set Ψ of hyper-parameters to the performance of a model trained using Ψ and explores the state-space of Ψ to find the maximum of the unknown function h.

As both of these approaches have shown promising empirical result, we finish this

section by recommending any of these two approaches for choosing hyper-parameters of deep neural networks. We do not go any further into the details of these algorithms as it is out of the scope of this thesis.

6.3.5 Exploiting Spatial Structure: Local Receptive Fields

When we have prior knowledge about the structure of the data, it may be possible to exploit it to obtain a better representation. For instance, a two-dimensional image has a local structure that is not maintained once the pixels in the image are shuffled. Similarly, a segment of speech has a temporal structure that easily breaks down when the speech samples are shuffled across time.

In (Coates et al., 2011), an image classification framework based on a single level of incremental feature learning was described in detail. The framework consists of three stages; (1) training a shallow neural network on randomly selected small image patches from training images, (2) extracting features from each image using the trained neural network, and (3) training a (linear) classifier on the extracted features. The first two stages correspond to a single step of the already described incremental feature learning.

The underlying idea in the first two stages is based on the convolutional neural networks (see, e.g., LeCun et al., 1998a; Lee et al., 2009) where a hidden unit is connected only to a small neighborhood, or local patch, in the input image. A set of hidden units is trained on a large set of fixed-size patches of training images, rather than the whole images. Then, each image is *scanned* with the trained neural network to extract features. Each hidden unit of the model used in this stage is often referred to as a *local receptive field*. Usually after this stage the sets of features from nearby patches are *pooled* to form a subsampled set of features.

Coates et al. (2011) empirically compared using different neural networks for the first two stages. They compared sparse autoencoders (see Section 3.2.5), restricted Boltzmann machines with the sparsity regularization, K-means clustering and Gaussian mixtures. In most cases they considered, they were able to achieve the state-of-the-art performance.

This approach of convolution and pooling can be further used as a single stage in the incremental feature learning. In fact, if we started our discussion on the incremental feature learning rather from the convolutional neural network (see, e.g., LeCun et al., 1998a) than from the fully-connected MLP, we would have arrived at the idea of incremental feature learning where each stage consists of *convolution*, *contrast normaliziation*, and *pooling* (see, e.g., LeCun et al., 2010, and references therein). Lee et al. (2009) showed that this way of incrementally stacking convolutional layers enables the neural network, specifically a convolutional deep belief network in their

work, to learn hierarchical part-based representations of data.

As the aim of this thesis, however, was not on the specific tasks of image or audio recognition which are known to benefit heavily from this convolutional structure, we have not discussed it any further. For more details on learning local receptive fields and using them for image classification, we refer readers to (Coates, 2012). We further suggest (Krizhevsky et al., 2012) and (Ciresan et al., 2012d) for the latest advances in using convolutional neural networks for image classification.

Bibliography

P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, 2008.

D. H. Ackley, G. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

S.-I. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2): 251–276, 1998.

P. Baldi and K. Hornik. Neural networks and principal component analysis: learning from examples without local minima. *Neural Networks*, 2(1):53–58, Jan. 1989.

D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

A. J. Bell and T. J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, Nov. 1995.

Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

Y. Bengio. Deep learning of representations: Looking forward. *arXiv:*1305.0445 [cs.LG], May 2013.

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

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

Y. Bengio and É. Thibodeau-Laufer. Deep generative stochastic networks trainable by backprop. *arXiv:*1306.1091 [cs.LG], June 2013.

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, Cambridge, MA, 2007.

Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. Advances in optimizing recurrent networks. In *Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2013)*, May 2013. To appear.

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013a.

Y. Bengio, G. Mesnil, Y. Dauphin, and S. Rifai. Better mixing via deep representations. In *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 552–560. JMLR W&CP, June 2013b.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. 2011.

C. M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, 7(1):108–116, Jan. 1995.

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

A. Bondy and U. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2008.

L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 161–168. MIT Press, Cambridge, MA, 2008.

L. Bottou and Y. LeCun. Large scale online learning. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv:*`1012.2599 [cs.LG]`, Dec. 2010.

D. Broomhead and D. Lowe. *Radial Basis Functions, Multi-variable Functional Interpolation and Adaptive Networks*. RSRE memorandum / Royal Signals and Radar Establishment. Royals Signals & Radar Establishment, 1988.

H. Burger, C. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2012)*, pages 2392–2399, June 2012.

R. H. Byrd, G. M. Chin, W. Neveitt, and J. Nocedal. On the use of stochastic Hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011.

M. A. Carreira-Perpiñán and G. Hinton. On contrastive divergence learning. In R. G. Cowell and Z. Ghahramani, editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, Jan 6-8, 2005, Savannah Hotel, Barbados*, pages 33–40. Society for Artificial Intelligence and Statistics, 2005.

O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

K. Cho. Improved Learning Algorithms for Restricted Boltzmann Machines. Master's thesis, Aalto University School of Science, 2011.

K. Cho. Boltzmann machines and denoising autoencoders for image denoising. *arXiv:*`1301.3468 [stat.ML]`, Jan. 2013.

K. Cho, T. Raiko, and A. Ilin. Gaussian-Bernoulli deep Boltzmann machine. In *NIPS 2011 Workshop on Deep Learning and Unsupervised Feature Learning*, Sierra Nevada, Spain, Dec. 2011.

K. Cho, T. Raiko, and A. Ilin. Enhanced gradient for training restricted Boltzmann machines. *Neural Computation*, 25(3):805–831, Mar. 2013.

Y. Cho and L. Saul. Kernel methods for deep learning. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 342–350. 2009.

B. A. Cipra. An introduction to the Ising model. *American Mathematics Monthly*, 94(10): 937–959, Dec. 1987.

D. Ciresan, A. Giusti, luca Maria Gambardella, and J. Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2852–2860. 2012a.

D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Deep big multilayer perceptrons for digit recognition. In G. Montavon, G. Orr, and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 581–598. Springer Berlin Heidelberg, 2012b.

D. C. Ciresan, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012c.

D. C. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2012)*, pages 3642–3649, June 2012d.

A. Coates. *Demystifying Unsupervised Feature Learning*. PhD thesis, Stanford University, 2012.

A. Coates, H. Lee, and A. Ng. An analysis of single-layer networks in unsupervised feature learning. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *JMLR Workshop and Conference Proceedings*, pages 215–223. JMLR W&CP, 2011.

R. Collobert and S. Bengio. Links between perceptrons, MLPs and SVMs. In *Proceedings of the 21st International Conference on Machine learning (ICML 2004)*, July 2004.

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

T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14 (3):326–334, 1965.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, Dec. 1989.

K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-D Transform-Domain collaborative filtering. *IEEE Transactions on Image Processing*, 16 (8):2080–2095, Aug. 2007.

G. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, Jan. 2012.

A. C. Damianou and N. D. Lawrence. Deep Gaussian processes. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2013)*, volume 31 of *JMLR Workshop and Conference Proceedings*, pages 207–215. JMLR W&CP, Apr. 2013.

G. M. Davis, S. G. Mallat, and Z. Zhang. Adaptive time-frequency decompositions. *Optical Engineering*, 33(7):2183–2191, 1994.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams, Y. Gong, and A. Acero. Recent advances in deep learning for speech research at Microsoft. In *Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2013)*, May 2013.

G. Desjardins, A. Courville, and Y. Bengio. Adaptive parallel tempering for stochastic maximum likelihood learning of RBMs. In *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning*, 2010a.

G. Desjardins, A. Courville, Y. Bengio, P. Vincent, and O. Delalleau. Parallel tempering for training of restricted Boltzmann machines. In Y.-W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9 of *JMLR Workshop and Conference Proceedings*, pages 145–152. JMLR W&CP, 2010b.

G. Desjardins, A. Courville, and Y. Bengio. On tracking the partition function. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2501–2509. 2011.

G. Desjardins, A. Courville, and Y. Bengio. On training deep Boltzmann machines. *arXiv:*`1203.4416 [cs.NE]`, Mar. 2012.

G. Desjardins, R. Pascanu, A. Courville, and Y. Bengio. Metric-free natural gradient for joint-training of Boltzmann machines. In *Proceedings of the First International Conference on Learning Representations (ICLR 2013)*, May 2013.

S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

R. Fletcher. *Practical Methods of Optimization*. Wiley-Interscience, New York, NY, USA, 2nd edition, 1987.

Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. Technical report, Santa Cruz, CA, USA, 1994.

B. J. Frey and G. Hinton. Variational learning in nonlinear Gaussian belief networks. *Neural Computation*, 11(1):193–213, 1999.

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

C. J. Geyer. Markov chain Monte Carlo maximum likelihood. In *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, pages 156–163, 1991.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9 of *JMLR Workshop and Conference Proceedings*, pages 249–256. JMLR W&CP, May 2010.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, volume 15 of *JMLR Workshop and Conference Proceedings*, pages 315–323. JMLR W&CP, Apr. 2011.

G. H. Golub and C. F. van Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)*. The Johns Hopkins University Press, 3rd edition, Oct. 1996.

I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout Networks. *arXiv:*`1302.4389 [stat.ML]`, Feb. 2013.

K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th Internation Conference on Machine Learning (ICML 2010)*, pages 399–406, Haifa, Israel, June 2010.

P. D. Grünwald. *The Minimum Description Length Principle*. MIT Press Books. The MIT Press, Apr. 2007.

I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Aha. Unsupervised and transfer learning challenge. In *Proceedings of the 2011 International Joint Conference on Neural Networks (IJCNN 2011)*, pages 793–800, 2011.

B. Hammer and K. Gersmann. A note on the universal approximation capability of support vector machines. *Neural Processing Letters*, 17:43–53, 2003.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, Apr. 1970.

S. Haykin. *Neural Networks and Learning Machines*. Pearson Education, 3rd edition, 2009.

D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York, June 1949.

G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, Aug. 2002.

G. Hinton. A practical guide to training restricted Boltzmann machines. In G. Montavon, G. B. Orr, and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer Berlin Heidelberg, 2012.

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

G. Hinton, P. Dayan, B. J. Frey, and R. Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1158–1161, 1995.

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

G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, 29(6):82–97, 2012.

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

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

A. Honkela, S. Harmeling, L. Lundqvist, and H. Valpola. Using kernel PCA for initialisation of variational Bayesian nonlinear blind source separation method. In C. Puntonet and A. Prieto, editors, *Proceedings of Independent Component Analysis and Blind Signal Separation (ICA 2004)*, volume 3195 of *Lecture Notes in Computer Science*, pages 790–797. Springer, 2004.

J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, Jan. 1989.

G.-B. Huang, L. Chen, and C.-K. Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, 2006a.

G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1—3):489–501, 2006b.

G.-B. Huang, D. Wang, and Y. Lan. Extreme learning machines: a survey. *International Journal of Machine Learning and Cybernetics*, 2:107–122, 2011.

A. Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709, Dec. 2005.

A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. Wiley-Interscience, May 2001.

K. Kavukcuoglu, M. Ranzato, and Y. LeCun. Fast inference in sparse coding algorithms with applications to object recognition. *arXiv:*1010.3467 [cs.CV], Oct. 2010.

R. Kindermann, J. Snell, and A. M. Society. *Markov Random Fields and Their Applications*. Contemporary mathematics. American Mathematical Society, 1980.

T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1106–1114. 2012.

S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:49–86, 1951.

L. D. Landau and E. M. Lifshitz. *Statistical Physics, Third Edition, Part 1: Volume 5 (Course of Theoretical Physics, Volume 5)*. Butterworth-Heinemann, 3rd edition, Jan. 1980.

H. Lappalainen and A. Honkela. Bayesian non-linear independent component analysis by multi-layer perceptrons. In M. Girolami, editor, *Advances in Independent Component Analysis*, Perspectives in Neural Computing, pages 93–121. Springer London, 2000.

H. Larochelle and Y. Bengio. Classification using discriminative restricted Boltzmann machines. In *Proceedings of the 25th International Conference on Machine learning (ICML 2008)*, pages 536–543, New York, NY, USA, 2008. ACM.

N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

N. D. Lawrence and J. Quiñonero Candela. Local distance preservation in the GP-LVM through back constraints. In *Proceedings of the 23rd International Conference on Machine learning (ICML 2006)*, pages 513–520, New York, NY, USA, 2006. ACM.

Q. Le, A. Karpenko, J. Ngiam, and A. Ng. ICA with reconstruction cost for efficient overcomplete feature learning. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1017–1025. 2011a.

Q. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Ng. On optimization methods for deep learning. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 265–272, New York, NY, USA, June 2011b. ACM.

N. Le Roux and Y. Bengio. Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation*, 20:1631–1649, June 2008.

N. Le Roux and Y. Bengio. Deep belief networks are compact universal approximators. *Neural Computation*, 22(8):2192–2207, Aug. 2010.

N. Le Roux, P.-A. Manzagol, and Y. Bengio. Topmoumoute online natural gradient algorithm. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 849–856. MIT Press, Cambridge, MA, 2008.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998a.

Y. LeCun, L. Bottou, G. Orr, and K. R. Müller. Efficient BackProp. In G. Orr and K. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 5–50. Springer Verlag, 1998b.

Y. LeCun, K. Kavukvuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS 2010)*, June 2010.

H. Lee, C. Ekanadham, and A. Ng. Sparse deep belief net model for visual area V2. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 873–880. MIT Press, Cambridge, MA, 2008.

H. Lee, R. Grosse, R. Ranganath, and A. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*, pages 609–616, New York, NY, USA, 2009. ACM.

R. Lengellé and T. Denæux. Training MLPs layer by layer using an objective function for internal representations. *Neural Networks*, 9(1):83–97, Jan. 1996.

D. J. C. Mackay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, 1st edition, June 2002.

J. Martens. Deep learning via Hessian-free optimization. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th Internation Conference on Machine Learning (ICML 2010)*, pages 735–742, Haifa, Israel, June 2010.

J. Martens and I. Sutskever. Training deep and recurrent networks with Hessian-free optimization. In G. Montavon, G. Orr, and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 479–535. Springer Berlin Heidelberg, 2012.

G. Mesnil, Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, P. Vincent, A. Courville, and J. Bergstra. Unsupervised and transfer learning challenge: a deep learning approach. In I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, editors, *Proceedings of the Unsupervised and Transfer Learning Challenge and Workshop*, volume 27 of *JMLR Workshop and Conference Proceedings*, pages 97–110. JMLR W&CP, 2012.

M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

V. Mnih, H. Larochelle, and G. Hinton. Conditional restricted Boltzmann machines for structured output prediction. In F. G. Cozman and A. Pfeffer, editors, *Proceedings of the 27th International Conference on Uncertainty in Artificial Intelligence (UAI 2011)*, pages 514–522, July 2011.

G. Montavon and K.-R. Müller. Deep Boltzmann machines and the centering trick. In G. Montavon, G. Orr, and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 621–637. Springer Berlin Heidelberg, 2012.

G. Montavon, M. L. Braun, and K.-R. Müller. Deep Boltzmann machines as feed-forward hierarchies. In *Proceedings of the Fifteenth Internation Conference on Artificial Intelligence and Statistics (AISTATS 2012)*, volume 22 of *JMLR Workshop and Conference Proceedings*, pages 798–804. JMLR W&CP, Apr. 2012.

K. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning Series. Mit Press, 2012.

V. Nair and G. Hinton. Rectified linear units improve restricted Boltzmann machines. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th Internation Conference on Machine Learning (ICML 2010)*, pages 807–814, 2010.

R. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113, July 1992.

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

R. Neal. Sampling from multimodal distributions using tempered transitions. *Statistics and Computing*, 6:353–366, 1994.

R. Neal. Annealed importance sampling. *Statistics and Computing*, 11:125–139, 1998.

R. Neal and G. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, *Learning in graphical models*, pages 355–368. MIT Press, Cambridge, MA, USA, 1999.

J. Ngiam, Z. Chen, P. W. Koh, and A. Ng. Learning deep energy models. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 1105–1112, New York, NY, USA, June 2011. ACM.

E. Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273, 1982.

E. Oja. Data compression, feature extraction, and autoassociation in feedforward neural networks. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, volume 1, pages 737–745. Elsevier Science Publishers B.V., North-Holland, 1991.

E. Oja. The nonlinear PCA learning rule in independent component analysis. *Neurocomputing*, 17(1):25–45, 1997.

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

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

J. Portilla, V. Strela, M. Wainwright, and E. Simoncelli. Image denoising using scale mixtures of Gaussians in the wavelet domain. *IEEE Transactions on Image Processing*, 12(11): 1338–1351, Nov. 2003.

T. Raiko. Hierarchical Nonlinear Factor Analysis. Master's thesis, Aalto University School of Science, 2001.

T. Raiko, H. Valpola, M. Harva, and J. Karhunen. Building blocks for variational Bayesian learning of latent variable models. *Journal of Machine Learning Research*, 8:155–201, May 2007.

T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In *Proceedings of the Fifteenth Internation Conference on Artificial Intelligence and Statistics (AISTATS 2012)*, volume 22 of *JMLR Workshop and Conference Proceedings*, pages 924–932. JMLR W&CP, Apr. 2012.

R. Raina, A. Battle, H. Lee, B. Packer, and A. Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th International Conference on Machine learning (ICML 2007)*, pages 759–766, New York, NY, USA, 2007. ACM.

R. Raina, A. Madhavan, and A. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*, pages 873–880, New York, NY, USA, 2009. ACM.

M. Ranzato, Y.-L. Boureau, S. Chopra, and Y. LeCun. A unified energy-based framework for unsupervised learning. In *Proceedings of the Tenth Conference on AI and Statistics (AISTATS 2007)*, volume 2 of *JMLR Workshop and Conference Proceedings*, pages 860–867. JMLR W&CP, 2007a.

M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1137–1144. MIT Press, Cambridge, MA, 2007b.

M. Ranzato, Y.-L. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1185–1192. MIT Press, Cambridge, MA, 2008.

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

S. Rifai, G. Mesnil, P. Vincent, X. Muller, Y. Bengio, Y. Dauphin, and X. Glorot. Higher order contractive auto-encoder. In D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 6912 of *Lecture Notes in Computer Science*, pages 645–660. Springer Berlin Heidelberg, 2011a.

S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 833–840, New York, NY, USA, June 2011b. ACM.

H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, Nov. 1958.

S. Roweis. EM algorithms for PCA and SPCA. In *Advances in Neural Information Processing Systems 10*, pages 626–632, Cambridge, MA, USA, 1998. MIT Press.

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

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

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.

R. Salakhutdinov. Learning deep Boltzmann machines using adaptive MCMC. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pages 943–950, Haifa, Israel, June 2010.

R. Salakhutdinov and G. Hinton. Deep Boltzmann machines. In *Proceedings of the Twelfth Internation Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, volume 5 of *JMLR Workshop and Conference Proceedings*, pages 448–455. JMLR W&CP, 2009a.

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

R. Salakhutdinov and G. Hinton. A better way to pretrain deep Boltzmann machines. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2456–2464. 2012a.

R. Salakhutdinov and G. Hinton. An effcient learning procedure for deep Boltzmann machines. *Neural Computation*, 24:1967–2006, 2012b.

R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning (ICML 2007)*, pages 791–798, New York, NY, USA, 2007. ACM.

J. Schmidhuber, D. Cireşan, U. Meier, J. Masci, and A. Graves. On fast deep nets for AGI vision. In J. Schmidhuber, K. R. Thórisson, and M. Looks, editors, *Artificial General Intelligence*, volume 6830 of *Lecture Notes in Computer Science*, pages 243–246. Springer Berlin Heidelberg, 2011.

B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

N. N. Schraudolph, J. Yu, and S. Günter. A stochastic quasi-Newton method for online convex optimization. In M. Meila and X. Shen, editors, *Proceedings of the Eleventh International Conference Artificial Intelligence and Statistics (AISTATS 2007)*, volume 2 of *JMLR Workshop and Conference Proceedings*, pages 436–443. JMLR W&CP, 2007.

P. Smolensky. Information processing in dynamical systems: foundations of harmony theory. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*, pages 194–281. MIT Press, Cambridge, MA, USA, 1986.

J. Snoek, H. Larochelle, and R. Adams. Practical Bayesian optimization of machine learning algorithms. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2960–2968. 2012.

R. Socher, C. C. Lin, A. Ng, and C. Manning. Parsing natural scenes and natural language with recursive neural networks. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 129–136, New York, NY, USA, 2011. ACM.

I. Sutskever. *Training Recurrent Neural Networks*. PhD thesis, University of Toronto, 2013.

I. Sutskever, J. Martens, and G. Hinton. Generating text with recurrent neural networks. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 1017–1024, New York, NY, USA, June 2011. ACM.

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

K. Swersky, M. Ranzato, D. Buchman, B. Marlin, and N. de Freitas. On autoencoders and score matching for energy based models. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 1201–1208, New York, NY, USA, June 2011. ACM.

Y. Tang and R. Salakhutdinov. A new learning algorithm for stochastic feedforward neural networks. In *ICML 2013 Workshop on Challenges in Representation Learning*, Atlanta, Georgia, June 2013.

Y. Tang and I. Sutskever. Data normalization in the learning of restricted Boltzmann machines. Technical Report UTML-TR-11-2, Department of Computer Science, University of Toronto, 2011.

Y.-W. Teh, M. Welling, S. Osindero, and G. Hinton. Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4:1235–1260, Dec. 2003.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th Internation Conference on Machine Learning (ICML 2008)*, pages 1064–1071, New York, NY, USA, 2008. ACM.

T. Tieleman and G. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*, pages 1033–1040, New York, NY, USA, 2009. ACM.

M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 61:611–622, 1999.

D. S. Touretzky and D. A. Pomerleau. What is hidden in the hidden layers? *Byte*, 14: 227–233, 1989.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

P. Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, July 2011.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, Dec. 2010.

M. Welling, M. Rosen-Zvi, and G. Hinton. Exponential family harmoniums with an application to information retrieval. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1481–1488. MIT Press, Cambridge, MA, 2005.

M. P. Wellman and M. Henrion. Explaining 'explaining away'. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(3):287–292, Mar. 1993.

J. Xie, L. Xu, and E. Chen. Image denoising and inpainting with deep neural networks. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 350–358. 2012.

L. Younes. Estimation and annealing for Gibbsian fields. *Annales de l'institut Henri Poincaré (B) Probabilités et Statistiques*, 24(2):269–294, 1988.