# Preface

This dissertation summarizes the work I have carried out as a doctoral student at the Department of Information and Computer Science, Aalto University School of Science under the supervision of Prof. Juha Karhunen, Prof. Tapani Raiko and Dr. Alexander Ilin between 2011 and early 2014, while being generously funded by the Finnish Doctoral Programme in Computational Sciences (FICS). None of these had been possible without enormous support and help from my supervisors, the department and the Aalto University. Although I cannot express my gratitude fully in words, let me try: Thank you!

During these years I was a part of a group which started as a group on *Bayesian Modeling* led by Prof. Karhunen, but recently become a group on *Deep Learning and Bayesian Modeling* co-led by Prof. Karhunen and Prof. Raiko. I would like to thank all the current members of the group: Prof. Karhunen, Prof. Raiko, Dr. Ilin, Mathias Berglund and Jaakko Luttinen.

I have spent most of my doctoral years at the Department of Information and Computer Science and have been lucky to have collaborated and discussed with researchers from other groups on interesting topics. I thank Xi Chen, Konstantinos Georgatzis (University of Edinburgh), Mark van Heeswijk, Sami Keronen, Dr. Amaury *Momo* Lendasse, Dr. Kalle Palomäki, Dr. Nima Reyhani (Valo Research and Trading), Dusan Sovilj, Tommi Suvitaival and Seppo Virtanen (of course, not in the order of preference, but in the alphabetical order). Unfortunately, due to the space restriction I cannot list all the colleagues, but I would like to thank all the others from the department as well. *Kiitos!*

I was warmly invited by Prof. Yoshua Bengio to Laboratoire d'Informatique des Systèmes Adaptatifs (LISA) at the Université de Montréal for six months (Aug. 2013 – Jan. 2014). I first must thank FICS for kindly funding the research visit so that I had no worry about daily survival. The visit at the LISA was fun and productive! Although I would like to list all of the members of the LISA to show my appreciation during my visit, I can only list a few: Guillaume Allain, Frederic Bastien, Prof.

Bengio, Prof. Aaron Courville, Yann Dauphin, Guillaume Desjardins (Google DeepMind), Ian Goodfellow, Caglar Gulcehre, Pascal Lamblin, Mehdi Mirza, Razvan Pascanu, David Warde-Farley and Li Yao (again, in the alphabetical order). Remember, it is Yoshua, not me, who recruited so many students. *Merci!*

Outside my comfort zones, I would like to thank Prof. Sven Behnke (University of Bonn, Germany), Prof. Hal Daumé III (University of Maryland), Dr. Guido Montúfar (Max Planck Institute for Mathematics in the Sciences, Germany), Dr. Andreas Müller (Amazon), Hannes Schulz (University of Bonn) and Prof. Holger Schwenk (Université du Maine, France) (again, in the alphabetical order).

I express my gratitude to Prof. Nando de Freitas of the University of Oxford, the opponent in my defense. I would like to thank the pre-examiners of the dissertation; Prof. Hugo Larochelle of the University of Sherbrooke, Canada and Dr. James Bergstra of the University of Waterloo, Canada for their valuable and thorough comments on the dissertation.

I have spent half of my twenties in Finland from Summer, 2009 to Spring, 2014. Those five years have been delightful and exciting both academically and personally. Living and studying in Finland have impacted me so significantly and positively that I cannot imagine myself without these five years. I thank all the people I have met in Finland and the country in general for having given me this enormous opportunity. Without any surprise, I must express my gratitude to ***Alko*** for properly regulating the sales of alcoholic beverages in Finland.

Again, I cannot list all the friends I have met here in Finland, but let me try to thank at least a few: Byungjin Cho (and his wife), Eunah Cho, Sungin Cho (and his girlfriend), Dong Uk *Terry* Lee, Wonjae Kim, Inseop *Leo* Lee, Seunghoe Roh, Marika Pasanen (and her boyfriend), Zaur Izzadust, Alexander Grigorievsky (and his wife), David Padilla, Yu Shen, Roberto Calandra, Dexter He and Anni Rautanen (and her boyfriend and family) (this time, in a random order). *Kiitos!*

I thank my parents for their enormous support. I thank *and* congratulate my little brother who married a beautiful woman who recently gave a birth to a beautiful baby.

Lastly but certainly not least, my gratitude and love goes to *Y*. Her encouragement and love have kept me and my research sane throughout my doctoral years.

Espoo, February 17, 2014,

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)*, Pages 106–113, 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)*, Pages 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)*, Pages 611–618, 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)*, Pages 6729–6733, 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 | Multilayer 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, $\eta$, $\lambda$ and $\sigma$ 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 $\phi$, a stochastic noise operator $\kappa$, an encoder function $f$, and a decoder function $g$.

A component-wise nonlinear activation function $\phi$ is used for different types of activation functions depending on the context. For instance, $\phi$ 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. There should not be any 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 the data distribution and the 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 $\mathrm{d}$, $p_D$ or $P_0$, and the model distribution by either $\mathrm{m}$, $p$ or $P_\infty$. 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 the 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 introduction of layer-wise pretraining proposed in 2006 by Hinton and Salakhutdinov (2006); Bengio et al. (2007); Ranzato et al. (2007b), research on artificial neural networks began as early as 1958 when Rosenblatt (1958) presented the first 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), multilayer 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 cost function used to train a linear autoencoder with a single hidden layer corresponds exactly to that of PCA. PCA can be further generalized to nonlinear PCA through, for instance, an autoencoder with multiple nonlinear hidden layers (Kramer, 1991; Oja, 1991).

Due to the recent popularity of deep learning, two of the most widely studied artificial neural networks 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 multilayer perceptrons.

Thus, this thesis starts from some of the earlier ideas in the artificial neural networks and arrives at those two currently popular models. In due course, the author will 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 avoids 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 on 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 the author's contributions is given in Section 1.3.

### 1.2.1 Shallow Neural Networks

In Chapter 2, the author gives 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, although many often consider a neural network having a single layer of hidden units shallow as well. 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 which do not have any output unit are considered shallow when either there are no hidden units or there are only linear hidden units. A Hopfield network is one example having no hidden units, and a linear autoencoder, or equivalently principal component analysis, is an example 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, the 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 discusses some conditions that are often satisfied by a neural network to be considered deep in Section 2.4.

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 multilayer 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.

An algorithm called backpropagation for efficiently computing the gradient of the cost function of a feedforward neural network with respect to the parameters is presented in Section 3.4. The computed gradient is often used by the stochastic gradient method to estimate the parameters.

After a brief description of backpropagation, the section further discusses the difficulty of training deep feedforward neural networks 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 Boltzmann machines are recurrent neural networks and that any recurrent neural network with nonlinear hidden units is deep.

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 con-

ditional 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 multilayer perceptron 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 multilayer 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 multilayer 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. multilayer perceptrons, autoencoders and Boltzmann machines, which are the 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.

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 multilayer 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 points 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 the 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 rate and 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 the 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 chapter, we discuss some conditions that distinguish deep neural networks from the simple neural networks introduced in the earlier part of this chapter.

[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 $\mathbf{x}$:

$$y = \kappa(f(\mathbf{x})). \tag{2.2}$$

where $\kappa(\cdot)$ is a stochastic operator that randomly corrupts the input. 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 $y$ is a continuous variable, 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 $x$.

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

[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.

possible. If we assume Gaussian noise $\epsilon$, 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 $\Omega$ and $\lambda$ are the regularization term and its strength. Regularization is a method for controlling the complexity of a model to prevent the 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). For instance, $\hat{\mathbf{w}}$ is obtained by multiplying $\mathbf{y} = \left[y^{(1)}, y^{(2)}, \ldots, y^{(N)}\right]^\top$ to a pseudo-inverse of $\mathbf{X}^\top = \left[\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\right]^\top$.

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

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

It is, however, usual with other regularization terms 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 $y$, this network can be extended to predict a multi-dimensional output. Simply, the network will require as many output units as 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 the weights corresponding to each output simultaneously.

This simple linear neural network is highly restrictive in the sense that it can only approximate, or simulate, a *linear* function arbitrary well. When the unknown function $f$ is not linear, this network will most likely fail 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 $\phi$ 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:

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

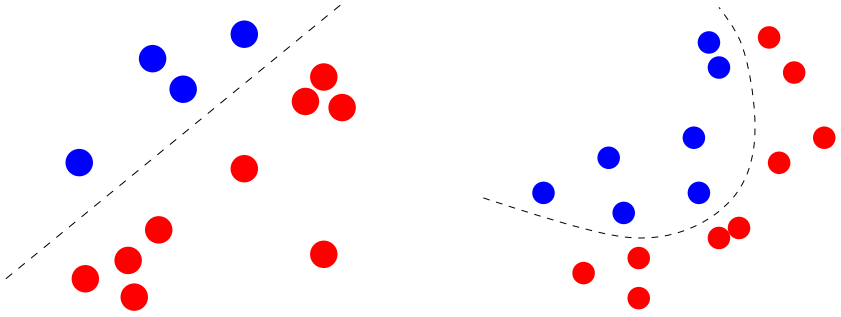
This formula allows us to perform a binary classification, where each sample is either classified as *negative* (0) or *positive* (1).

The illustration of a perceptron in Fig. 2.1 (b) shows that the perceptron is identical to the linear regression network except that the activation function of the output is a nonlinear step function.

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 (Minsky and Papert, 1969). Linear separability means that there exists a linear hyperplane that separates $\mathbf{x}^{(n)}$ that belongs to the positive class from those that belong to the negative class (see Fig. 2.2). With a correct set of



(a) Linearly separable (b) Nonlinearly separable

**Figure 2.2.** (a) Samples are linearly separable. (b) They are separable, but not linearly.

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 was proposed to estimate the set of weights 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 possible to use any other nonlinear saturating function whose range is limited from above and below 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 not separable with a linear 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 classification tasks where classes are not linear separable, such as 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, unsupervised learning considers a case where there is no target value. In this case, 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

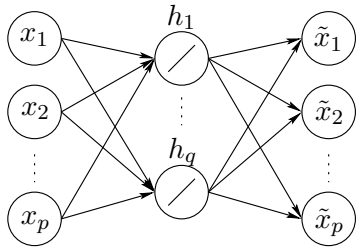
$$\mathbf{x} = f(\mathbf{h}). \tag{2.11}$$

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).

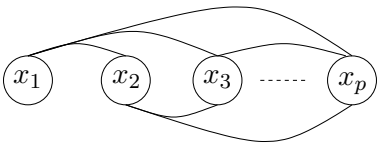
This is, however, not the only way to formulate an unsupervised model. Another way is to build a model that learns direct relationships among the input components $x_1, \ldots, x_p$. This does not require any hidden variable, but still learns an (unknown) structure of the model.

### 2.2.1 Linear Autoencoder and Principal Component Analysis

In this section, we look at the case where hidden variables are assumed to have linearly generated training samples. In this case, it is desirable for us to learn not only an unknown function $f$, but also another function $g$ that is an (approximate) inverse function of $f$. Opposite to $f$, $g$ recognizes a given sample by finding a corresponding




(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.

state of the hidden variables.[3]

Let us construct 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.[4] 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*.[5] 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

[3]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.

[4]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.

[5]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[6] 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 we explicitly constrain 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 binary 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

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

[6] 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 $\eta$ 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 $\phi$ 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.

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 the sense that only direct correlations among visible units are modeled. In other words, the network can only learn 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 model 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 relationship to neural networks.

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

### 2.3.1 Supervised Model

Here we consider 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 the observation of $y$ is corrupted by additive noise $\epsilon$ which 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 $\epsilon$ parameterized[7] by $\boldsymbol{\theta}$, that is, $p(y \mid \mathbf{x}, \boldsymbol{\theta})$.

The prediction of the output $\hat{y}$ given a new sample $\mathbf{x}$ can be computed from the

[7]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.

conditional distribution $p(y \mid \mathbf{x}, \tilde{\boldsymbol{\theta}})$ with the estimated parameters $\tilde{\boldsymbol{\theta}}$. It is typical to use 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 built by assuming that the noise $\epsilon$ follows a Gaussian distribution with zero mean and its variance is 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.

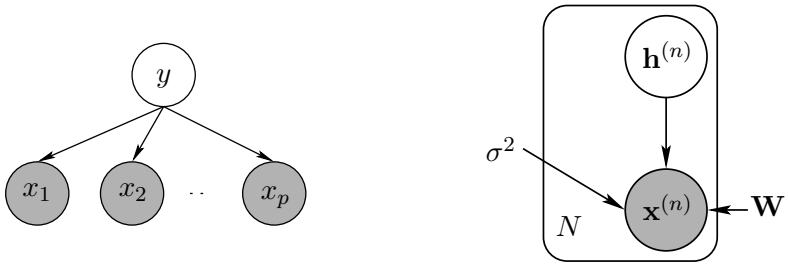
A regularization term can be inserted by considering the parameters as random variables. When each weight parameter $w_i$ is given a prior distribution, the log-posterior distribution $\log p(\mathbf{w} \mid \mathbf{x}, y)$ of the weights can be written, using Bayes' rule[8], as

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

[8]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 $X$. 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

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

Maximizing this posterior 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[9] $y$, 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

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

[9] A *label* of a sample tells to which *class* the sample belongs. Often 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

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

The posterior probability above takes the 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*

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 $\epsilon$ follows a zero-mean Gaussian distribution with its variance $\sigma^2$. Furthermore, we may assume that the training samples $\mathbf{x}^{(n)}$ are centered so that their mean is zero.[10]

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 $\sigma^2$ by maximizing the marginal log-likelihood, given by

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

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 $\lambda_i$ are 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.[11] Furthermore, in (Tipping and Bishop, 1999),

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

[11] 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.

In this section, however, we are more interested in solving PCA using the 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 estimate 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

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \mathcal{L}^{(n)}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \mathcal{L}_{Q^{(n)}}(\boldsymbol{\theta}) + \mathrm{KL}(Q^{(n)} \| P^{(n)}), \tag{2.24}$$

where

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

and

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

and $P^{(n)}$ and $Q^{(n)}$ denote the true posterior distribution $p(\mathbf{h} \mid \mathbf{x}^{(n)}, \boldsymbol{\theta})$ 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 $\mathbf{x}^{(n)}$, and the second term is a KL divergence between a distribution $Q^{(n)}$ and the posterior distribution over $\mathbf{h}$. $\mathcal{H}(Q)$ is an entropy functional of the distribution $Q$.

$\mathcal{L}_{Q^{(n)}}(\boldsymbol{\theta})$ is a lower bound of $\mathcal{L}^{(n)}(\boldsymbol{\theta})$ for all $n$ in Eq. (2.24), since the KL-divergence $\mathrm{KL}(Q^{(n)} \| P^{(n)})$ is always non-negative. From this, we can see that $\sum_{n=1}^{N} \mathcal{L}_{Q^{(n)}}(\boldsymbol{\theta})$ is a lower bound of the marginal log-likelihood $\mathcal{L}(\boldsymbol{\theta})$. The lower bound is equal to the marginal log-likelihood when an arbitrary distribution $Q^{(n)}$ is identical to the posterior distribution ($\mathrm{KL}(Q^{(n)} \| P^{(n)}) = 0$) for all the training samples.

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

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

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

$$\mathbb{E}\left[\mathbf{h}^{(n)}\right] = \tilde{\mathbf{W}}^\top \mathbf{x}^{(n)} \tag{2.27}$$
$$\mathbb{E}\left[\mathbf{h}^{(n)} \mathbf{h}^{(n)\top}\right] = \mathbb{E}\left[\mathbf{h}^{(n)}\right] \mathbb{E}\left[\mathbf{h}^{(n)}\right]^\top. \tag{2.28}$$

The mean of each hidden unit in Eq. (2.27) 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 $\sigma^2$ 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 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[12], 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 distri-*

[12] 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).

*bution*[13] (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 probabilities sum up to one. Although it is possible to have hidden units that do not correspond to any component in the input of a Boltzmann machine, we do not consider them in this section.

The conditional probability of a unit $x_i$ given the state 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 $\phi$ is a sigmoid function. This equation is equivalent to 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

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

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.[14]

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).

[13] 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 $E(\mathbf{x})$.

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

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 the 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 does 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. This, however, does not fully characterize the deep neural networks we are interested in.

In essence, we often 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 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, *the data decide how many layers a deep neural network needs*.[15]

One important consequence of the first condition is that since the units in an added layer use the activations of the units in the existing lower layers to compute their own activations, the activations of the units in the lower layers are *shared*. In other words,

[15] Yoshua Bengio, personal communication

there exist more than one computational path from the input layer to a unit in the added, upper layers.

## 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 the 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 the average of losses over training samples such that

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

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

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

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

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 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 procedure 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 iterations in Eq. (2.36) converge 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 a 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) 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).

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 an online approximation to the natural gradient. Schraudolph et al. (2007) proposed an online variant of the popular Quasi-Newton method. Also, Le et al. (2011a) compared the performances of various second-order optimization methods in training deep neural networks against the stochastic gradient descent.

We will 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:

**Multilayer 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 the 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.

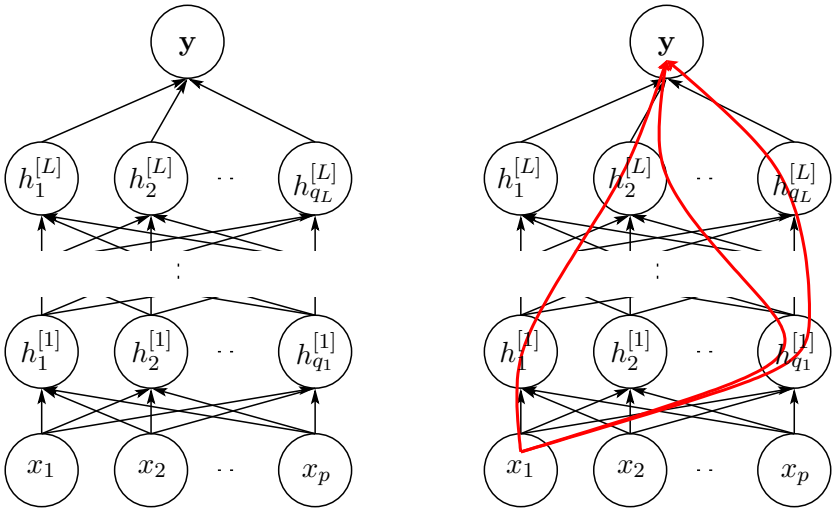
The linear regression network, perceptron, and linear autoencoder, introduced in the previous chapter, are shallow realizations of feedforward neural networks. If we group, starting from the input units, each set of the disjoint units as a layer, then one can evaluate the activations, or states, of the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

Here we discuss in more details two types of deep feedforward neural networks: a multilayer 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 Multilayer Perceptron

Let us extend the linear regression network and the 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 *multilayer perceptron* (MLP) (Rosenblatt, 1962). 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 training samples. Assuming linear output units (see Eq. (2.3)), the output of an MLP with




(a) Multilayer Perceptron (b) Multiple Computational Paths

**Figure 3.1.** Illustrations of (a) a multilayer 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.

$L$ intermediate hidden layers is

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

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. $\phi_{[l]}$ is a component-wise nonlinear function at the $l$-th layer, 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

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

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 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 composition 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.

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, there exists more than one computational path from an input vector to a unit in any intermediate layer $l$ ($l > 1$), 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 an illustration.

Each intermediate layer is *trainable* in the 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, the Euclidean distance may be used, as in Eq. (2.4), and for a classification task, one may use the cross-entropy loss, as in Eq. (2.9). The parameters can be efficiently learned using the stochastic gradient method with the gradient computed by the 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 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.

A multilayer perceptron described in this section is a typical example of a deep neural network that can be characterized by having *multiple layers of trainable feedforward nonlinear hidden units*.

### 3.1.1 Related, but Shallow Neural Networks

Here we explain two 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 kernel methods. We briefly describe these principles and find the connection to the MLP here. For more details on SVMs, the readers are referred to (Schölkopf and Smola, 2001).

The maximum-margin principle provides a way of selecting an optimal separating

hyperplane among multiple possible separating hyperplanes. 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 $1$ as the 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 for 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 $\phi$. Then, in essence, the classifier is trained not on $D = \left\{\left(\mathbf{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}$, but on $\tilde{D} = \left\{\left(\phi(\mathbf{x}^{(n)}), y^{(n)}\right)\right\}$. The connection to MLPs explained earlier can be found by comparing $\phi$ to the multiple intermediate layers of hidden units.

The kernel method uses a *kernel* function $k(\cdot, \cdot)$ between two samples, instead of using an explicit transformation $\phi$. 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

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

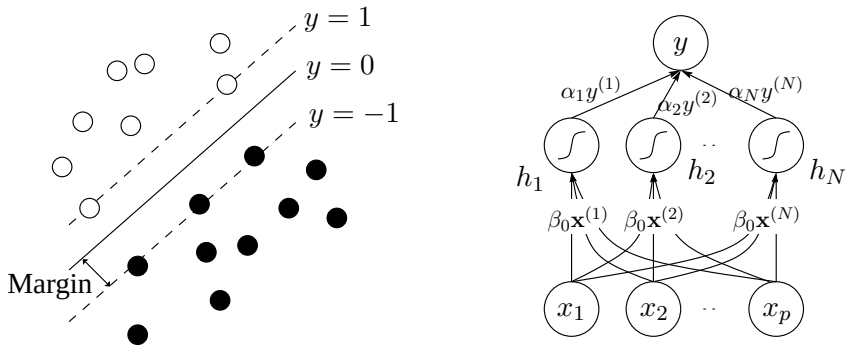
where $\phi$ is the Heaviside function, and we may replace the inner product between $\mathbf{x}^{(n)}$ and $\mathbf{x}$ with a positive-definite kernel function $k(\mathbf{x}^{(n)}, \mathbf{x})$.

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

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

can be constructed.

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




(a) Maximum Margin (b) Equivalent MLP

**Figure 3.2.** Illustrations of (a) the maximum-margin principle and (b) constructing a multilayer 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.

weights $\mathbf{W}$ of the edges connecting the input units to the hidden units are fixed to the training samples scaled by $\beta_0$, 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 $\beta_1$. 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).

We do not consider nor discuss kernel methods, 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 multilayer perceptrons as well as a simpler perceptron (see, e.g., Collobert and Bengio, 2004).

*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.[1]

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

[1] A similar idea of learning only the parameters of the last output layer has also been proposed and used for recurrent neural networks (see, e.g., LukošEvičIus and Jaeger, 2009).

$\phi(\mathbf{x})$ corresponds to the activations of the intermediate layer. Interestingly, in an ELM, the parameters of the transformation $\phi$ 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) stating that samples in a classification problem are more likely to become linearly separable when the input is nonlinearly transformed to a 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 ELM recently (Huang et al., 2011).

## 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 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

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

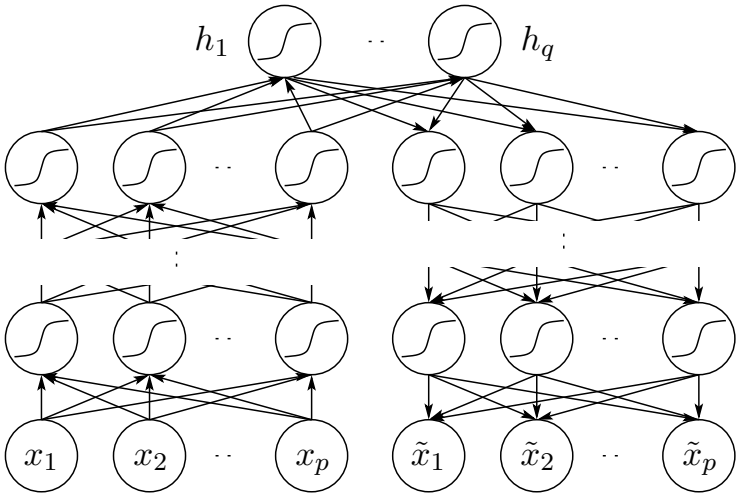



**Figure 3.3.** Illustration of a deep autoencoder. The left- and right- halves correspond to the encoder and decoder, respectively. Although each hidden node is drawn to have a sigmoid nonlinear activation function, a different activation function may be used.

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 inputs.

We may call this a *deep* autoencoder, as this 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 $\mathbf{h}$ with the Bernoulli distribution, with parameterized probability masses $\gamma$ 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 $\boldsymbol{\mu} = f(\mathbf{x} \mid \boldsymbol{\theta}_f)$ 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.

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 exact inference, but only approximate inference assuming the fully factorized posterior distribution $Q$ in Eq. (3.8), without trying to reduce the Kullback-Leibler divergence between $Q$ and the true posterior distribution.

If we drop the binary constraint of $\mathbf{x}$ 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)).

This interpretation, however, does neither provide an intuitive way of learning the parameters $\boldsymbol{\theta}_f$ of the encoder $f$ nor tell us how latent representations should be designed. Vincent et al. (2010) argued that minimizing the reconstruction error in an autoencoder trained by, for instance, stochastic gradient descent 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 $\phi$ is a sigmoid function (see Eq. (2.7)).

The probability of each component $x_i$ given the activations of the units in the layer 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. Inference may be performed by generating a set of samples from the posterior distribution over binary hidden units (Neal, 1992) or by approximating the posterior distribution with a simpler distribution (see, e.g., Saul et al., 1996; Jordan et al., 1999). Both approaches may easily become computational inefficient to be used in practice.

Instead, Hinton et al. (1995) proposed a wake-sleep algorithm, which is based on the principle of minimum description length (Rissanen, 1978), 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

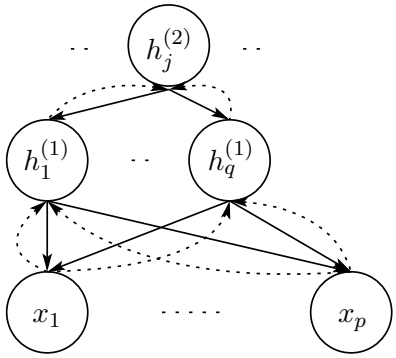



**Figure 3.4.** Illustration of a sigmoid belief network having two hidden layers. The solid lines indicate the generation path starting from the top hidden layer and are parameterized by the generation parameters. The dashed curves correspond to the recognition paths starting from the bottom, visible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

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

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

We can notice 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.

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 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 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.

### 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 back-constraints.

The 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 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

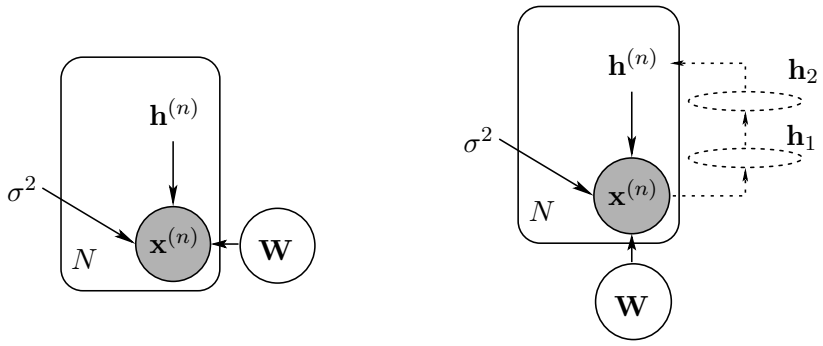
$$\mathcal{L} = -\frac{qN}{2}\log 2\pi - \frac{q}{2}\log|\mathbf{K}| - \frac{1}{2}\text{tr}(\mathbf{K}^{-1}\mathbf{X}\mathbf{X}^\top), \tag{3.13}$$

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, a novel hidden representation can be projected on the input space by

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

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




(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.

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 possible choice is the encoder part of a deep autoencoder. As the back-constraint can be considered to find the mean of the posterior distribution of hidden variables given an input, we can 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.

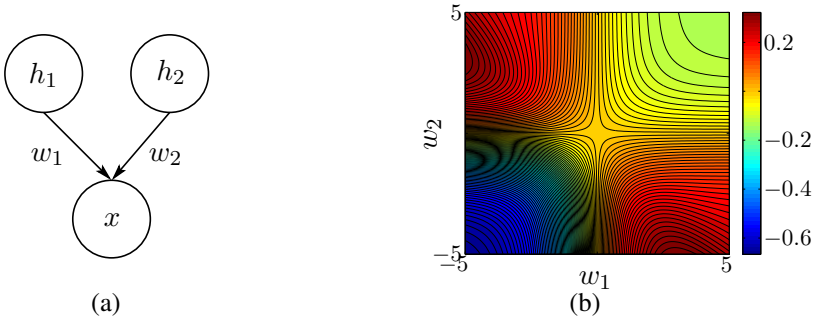



**Figure 3.6.** (a) A simple sigmoid belief network with two hidden units and a single visible unit. (b) The log-ratio between $p(h_2 = 1 \mid x = 1, h_1 = 1)$ and $p(h_2 = 1 \mid x = 1)$.

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 $\phi$ 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 different, 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 $h_1$ which is known to decrease the likelihood of the observed event happening ($w_1 < 0$) 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 $\mathbf{h}$ be strictly below a certain level $\rho$.

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

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

where $\tau = \rho q$ 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 $\epsilon$ 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 start from an all-zero code and sequentially search for a hidden unit that satisfies a certain criterion such as the reduction in 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 $\epsilon$ 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* $h_2$.

Irrespective of 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), the 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 $\alpha$ 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):

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

where $\rho$ 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 defined by the set of training samples, while $\mathbb{Q}$ is defined by regularizing the average hidden activation probability. For instance, the range of the encoder of a regular PCA, which does not have any regularization or constraint, is unrestricted and equals to $\mathbb{R}^q$.

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

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

for some constant $\tau > 0$. 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

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

with the amount of error controlled by the regularization constant.

In the case of a 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 $\rho$. If the average activation of $f(\mathbf{x})$ is either much smaller or much larger than $\rho$, it can be suspected that the sample $\mathbf{x}$ is either not the same type as training samples or 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. $\bar{\rho}$ which defines a target sparsity should be set to one minus the target hidden activation probability $\rho$ 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[2]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 $\varphi$ onto an open 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 $p$.

This says that the number of local variations in the above case $q$, allowed for a sample to remain 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 a noisy sample of the 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 empirical evidence that this disentangling, or transformation, results in better mixing of MCMC chains as well as an improvement in classification tasks.

One prominent example that implements this manifold assumption is principal component analysis (PCA), discussed earlier in Section 2.2.1. PCA assumes that the manifold on which training samples lie is *linear*. The linear manifold is described by the 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 as meaningless noise.

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

However, the linear nature of PCA is highly restrictive, in the 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 will be discovered 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, that encourages the hidden layers of an MLP to disentangle the factors of variations along the data manifold. 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.

This can be done 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). $\kappa$ is a stochastic operator that adds random noise to the input.[3]

The denoising autoencoder finds a coordinate system of the data manifold. Minimizing Eq. (3.25) effectively corresponds to *ignoring* any direction of variation injected by $\kappa$. 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 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 $\kappa$ 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 $\kappa$, will map almost exactly to $\mathbf{h}$. This applies to $\mathbf{x}'$ as well.

A point $\bar{\mathbf{x}}$ in an *overlapping* area needs a closer observation. 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 on the data manifold. It is not well defined nor known to what extent points far from the manifold will be encoded. Fortunately, this may not be a big problem when

[3] $\kappa(\mathbf{x})$ 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

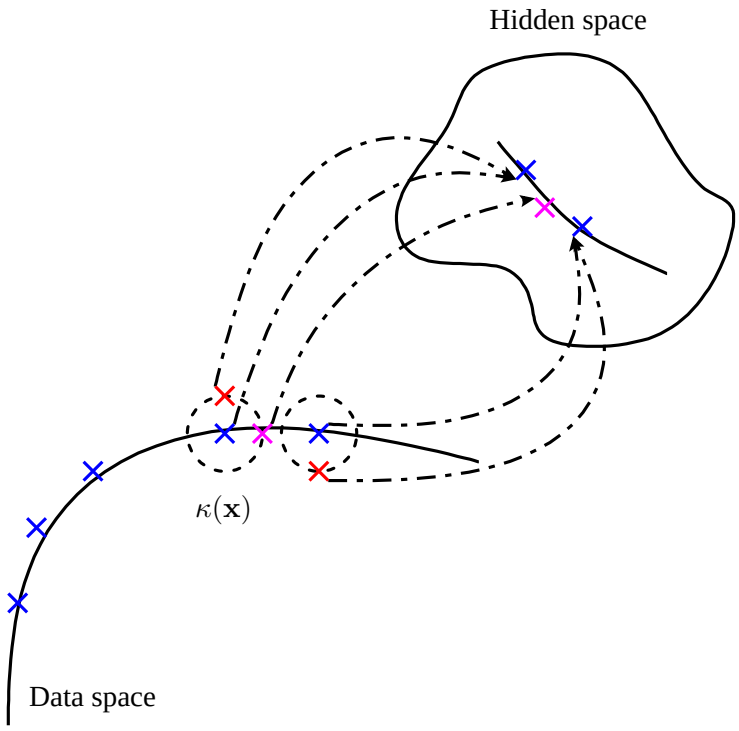



**Figure 3.7.** Illustration of how a denoising autoencoder learns a data manifold. The solid curves are the manifolds on which the training samples in the data space and their corresponding representations in the hidden space lie. The blue points (×) correspond to both training samples in the data space 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 $\kappa$. 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.

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) 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 means 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.

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

One 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 originally 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 $\kappa$ 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.

The stochastic gradient method described in Section 2.5 can be used to minimize Eq. (3.26). At each iteration, a randomly chosen subset of the training set is stochastically corrupted by $\kappa$ before the steepest descent direction is computed.

In the case where $\kappa$ simply adds uncorrelated white Gaussian noise to an input, Bishop (1995) showed that minimizing Eq. (3.26) is related 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 $\lambda$ is determined by the amount of noise injected by $\kappa$ 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 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 as

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

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

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

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, Rifai et al. (2011a) 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 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 a large corresponding singular value,

while the 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 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 for multilayer perceptrons 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 $\delta$ 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 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, we can compute the partial derivatives with respect to the top-level and intermediate-level weight parameters $u_{kj}$ and $w_{ij}^{[l]}$ by

$$\begin{aligned}
\frac{\partial J}{\partial u_{kj}} &= \delta_j h_k^{[L]} \\
\frac{\partial J}{\partial w_{ij}^{[l]}} &= \delta_j^{[l+1]} h_i^{[l-1]}, \forall l \geq 2 \\
\frac{\partial J}{\partial w_{ij}^{[1]}} &= \delta_j^{[2]} x_i.
\end{aligned}$$

See Fig. 3.8 for the illustration of the forward and backward passes.

Here we showed how to compute the gradient using a single sample only, but it is possible to extend it to the case of multiple samples 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).

### 3.4.1 How to Make Lower Layers Useful

It has been observed that neural networks with more than two intermediate hidden layers trained using plain stochastic gradient descent often exhibit worse generalization performance[4] than neural networks with only one or two intermediate layers (see, e.g. Bengio and LeCun, 2007). This is surprising, as we expect a larger model with more parameters to have a higher modeling capacity and thus result in better performance. In this section, we discuss some of the hypotheses explaining this surprising phenomenon.

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

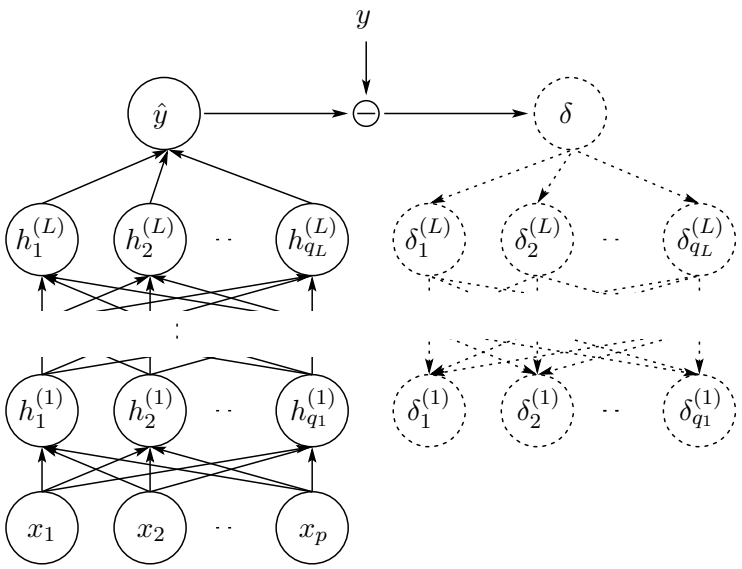



**Figure 3.8.** An illustration of the forward and backward passes. Starting from the input layer on left, the activation of each unit is iteratively computed up until the output $\hat{y}$ is predicted. The difference $\delta$ between the prediction $\hat{y}$ and true output $y$ is computed, and the 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.

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 not well utilized 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 stochastic gradient descent 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 gradient descent will fail to make the lower intermediate layers any more useful.

In an extreme case, it might be that the top two layers, the 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). The 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 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 adopted 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, where no output information is used during the layer-wise pretraining. The success of this approach started a so called *second neural network renaissance* (Schmidhuber et al., 2011). We will discuss in more detail this approach in Chapter 5.

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 stochastic gradient descent 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 gradient descent, 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 a 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 gradient 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 plain stochastic gradient descent 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 avoid being stuck in a poor local optimum or a plateau.

One must be careful about making any definite conclusion on using a plain gradient descent approach. 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 gradient descent can achieve state-of-the-art results 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 plain stochastic gradient descent.

# 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 the 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 as 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 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 previously, we define the negative

[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 the set of all parameters including $\mathbf{b}$, $\mathbf{c}$, $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$. From this, 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 units.

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

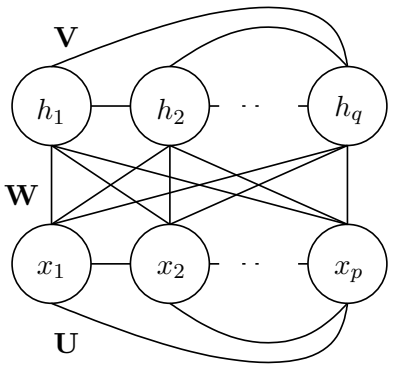



**Figure 4.1.** Fully-Connected Boltzmann machine.

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:

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

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 $\theta$ 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. Unfortunately, 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 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

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

where the shorthand notations $P_0$ and $P_\infty$ 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.

Eq. (4.5) is equivalent to Eq. (4.3), since

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

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).

### 4.1.1 Transformation Invariance and Enhanced Gradient

It is interesting to notice that the 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 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 $\mathbf{h}$:

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

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.

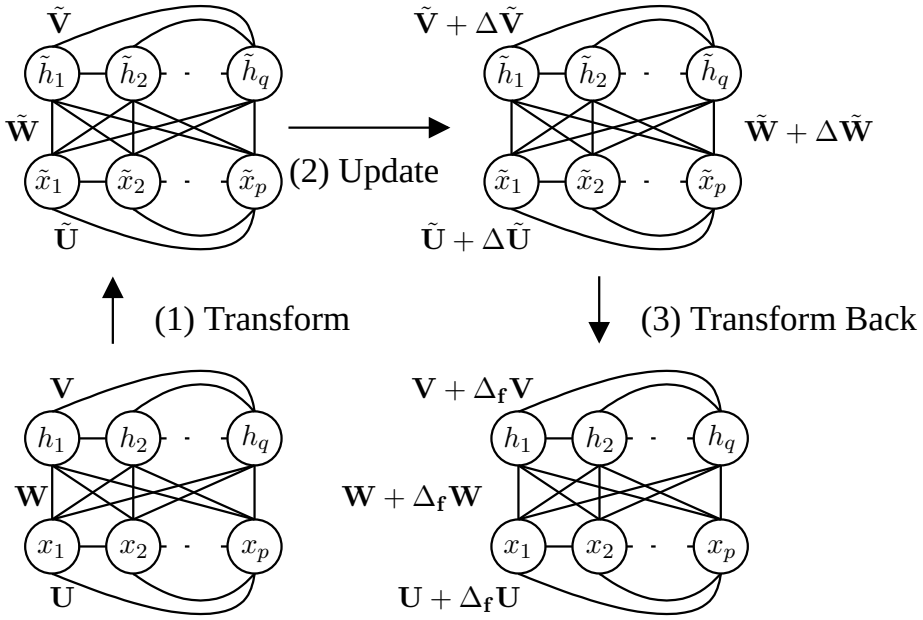



**Figure 4.2.** Illustration of the three-step update procedure consisting of (1) transforming, (2) updating and (3) transforming back. $\Delta_{\mathbf{f}}\boldsymbol{\theta}$ denotes the change in $\boldsymbol{\theta}$ dependent on the transformation $\mathbf{f}$.

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.

We first *transform* the Boltzmann machine with a given bit-flipping transformation $\mathbf{f}$, 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 $\nabla_e$ 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 a 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.

One can construct the equivalent Boltzmann machine given the shifting transformation $\boldsymbol{\beta}$ with the following parameters

$$\tilde{w}_{ij} = w_{ij},$$
$$\tilde{b}_i = b_i + \sum_j w_{ij}\beta_j.$$

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

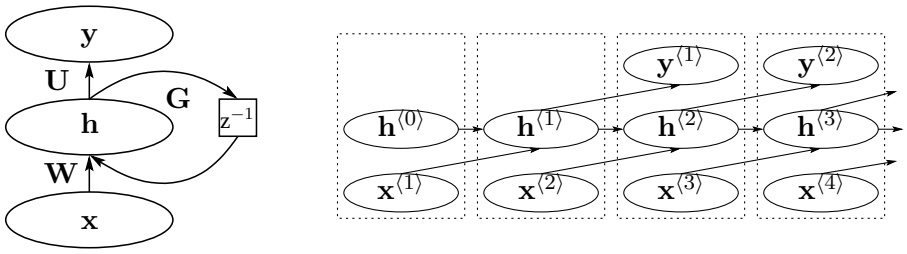
Before discussing Boltzmann machines further, in this section, we first provide an intuitive explanation as to why Boltzmann machines, especially with hidden units, are considered deep. This is done by showing that a recurrent neural network, after being unfolded over time, is deep and an equivalent 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 in 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 units at the previous time step $t-1$. More comprehensive discussions on general recurrent neural networks and recent advances in their learning algorithms can be found in (Bengio et al., 2013; Sutskever, 2013; Graves, 2013).

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 consider a simplified recurrent neural network consisting of an input layer, a single nonlinear hidden layer and a output layer. Each hidden layer receives signal




(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.

from the hidden layer at the previous time $t-1$. The 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 this way.

Now, let us unfold the described recurrent neural network in time, starting from $t=0$, assuming that the activations of the input and hidden units are fixed to zero initially. We will construct a deep 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 is deep, since it satisfies the two conditions described in Section 2.4.

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 potentially 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

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

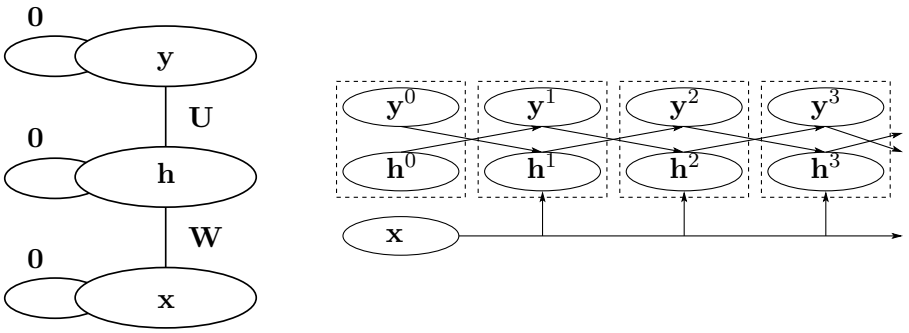
where $y'_l$ and $h'_k$ are the $l$-th component of $\mathbf{y}_{\langle t-1 \rangle}$ and the $k$-th component of $\mathbf{h}_{\langle t-1 \rangle}$, respectively. Subsequently with the new hidden state $\mathbf{h}_{\langle t \rangle}$, we replace a *single* unit $y_{l_t}$ to get the new label state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

This way of viewing the simulation of the BM is equivalent to performing a feedforward pass through time in a recurrent neural network. One difference to the recurrent

[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.

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

Also, 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 can consider Boltzmann machines with hidden units as *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 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 $p^{\langle 0 \rangle}$:

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

Otherwise, it will be impossible to use the chain to collect samples from a target distribution, since the simulation of the chain may converge 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 mass ($= 1$) 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 the 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

$$\mathcal{T}(\tilde{\mathbf{x}} \mid \mathbf{x}) = Q(\tilde{\mathbf{x}} \mid \mathbf{x})\alpha(\tilde{\mathbf{x}}, \mathbf{x}), \tag{4.14}$$

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

$$\alpha(\tilde{\mathbf{x}}, \mathbf{x}) = \min\left(1, \frac{p^*(\tilde{\mathbf{x}})Q(\mathbf{x} \mid \tilde{\mathbf{x}})}{p^*(\mathbf{x})Q(\tilde{\mathbf{x}} \mid \mathbf{x})}\right). \tag{4.15}$$

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

We can sample from the target distribution $\tilde{p}$ With the transition operator in Eq. (4.14), following the procedure described in Alg. 1. The distribution of $\mathbf{x}_{\langle t \rangle}$ will converge to the target distribution $\tilde{p}(\mathbf{x})$ using the Metropolis-Hastings method with some mild assumptions on the proposal distribution $Q$,

**Algorithm 1** Metropolis-Hastings

Initial state $\mathbf{x}_{\langle 0 \rangle}$, the proposal distribution $Q$, the unnormalized distribution $p^*$ and the maximum number $M$ of proposal are required.

Create an empty set of samples $\boldsymbol{X} = \{\}$.

**for** $t = 1, \cdots, M$ **do**

  Sample $\mathbf{x}'$ from $Q(\mathbf{x} \mid \mathbf{x}_{\langle t-1 \rangle})$.

  Compute the acceptance probability $\alpha$ using Eq. (4.15).

  Draw a sample $s$ from the uniform distribution $\mathcal{U}(0, 1)$.

  **if** $s < \alpha$ **then**

    Add $\mathbf{x}'$ to $\boldsymbol{X}$.

    Set $\mathbf{x}_{\langle t \rangle} = \mathbf{x}'$.

  **end if**

**end for**

$\boldsymbol{X}$ is the set of samples collected by 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,

$$Q(\tilde{\mathbf{x}} \mid \mathbf{x}) = p(\mathbf{x}_k \mid \mathbf{x}_{-k}), \tag{4.16}$$

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

With the proposal distribution in Eq. (4.16) 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.

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

This sampling-based approach, especially with the Gibbs sampling, however, 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, 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 issue of requiring a large amount of steps is especially prevalent 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.

*Parallel Tempering*

The major problem of Gibbs sampling is that it often fails to explore isolated modes lying far away 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 Gibbs sampling in collecting samples from Boltzmann machines. Salakhutdinov (2009) proposed to use tempered transitions (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, in the case of training restricted Boltzmann machines.

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 the application of 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 $\beta$ which was assumed to be fixed to 1 earlier for a Boltzmann machine. The joint probability of a Boltzmann machine of the inverse temperature $\beta$ 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 $\beta$ a *tempered distribution* with $\beta$.

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 $\beta_q$. The sample $\mathbf{x}''$ from the proposal distribution will be obtained by yet another Gibbs sampling step on the tempered distribution. Then, $\mathbf{x}''$ is accepted with the probability

$$p_{\text{swap}}(\mathbf{x}', \mathbf{x}'') = \min\left(1, \frac{p_1^*(\mathbf{x}'')p_{\beta_q}^*(\mathbf{x}')}{p_1^*(\mathbf{x}')p_{\beta_q}^*(\mathbf{x}'')}\right), \tag{4.17}$$

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. (4.17) 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, 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

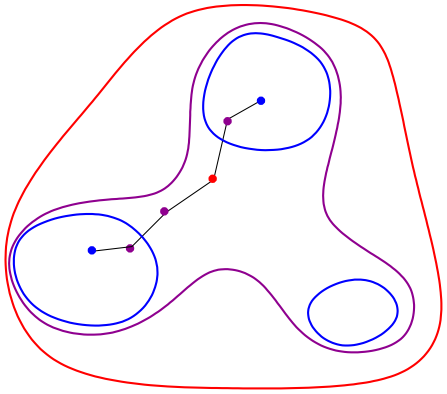


**Figure 4.5.** Illustration of how PT sampling could avoid being trapped in a single mode. The red, purple, and blue curves and dots indicate distributions and the samples from the distributions with the high, medium, and cold temperatures, respectively. Each black line indicates a single sampling step.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

a Bernoulli random variable with its mean $0.5$.

In other words, the tempered distributions with low $\beta$'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 $\beta$ approaches $1$ (model distribution), it becomes more difficult for the plain Gibbs sampling to explore the whole state space efficiently.

In this regard, the most important advantage of parallel tempering when compared to Gibbs sampling is that 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 $\beta$'s are less prone to becoming highly correlated with each other. A sample in a lower chain, that is far away from the mode in which the current sample at the top chain is trapped, 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 tempered distributions. All these methods are superior to the plain Gibbs sampling in the 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 restate how the marginal log-likelihood in general can be decomposed into two terms, which was presented in Eqs. (2.24)–(2.25):

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= \mathcal{L}_Q(\boldsymbol{\theta}) + \mathrm{KL}(Q\|P) \\ &\geq \mathbb{E}_Q\left[\log p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})\right] + \mathcal{H}(Q)\end{aligned} \tag{4.18}$$

$Q$ 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):

$$Q(\mathbf{h} \mid \boldsymbol{\theta}_Q) = \prod_{j=1}^{q} q(h_j), \tag{4.19}$$

where $q(h_j = 1) = \mu_j$ and $\mu_j$'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. (4.19) and Eq. (4.2) into Eq. (4.18), we can rewrite the lower bound $\mathcal{L}_Q(\boldsymbol{\theta})$ as

$$\begin{aligned}\mathcal{L}_Q(\boldsymbol{\theta}) = &\sum_{i=1}^{p} b_i x_i + \sum_{j=1}^{q} c_j \mu_j + \sum_{i=1}^{p} x_i \mu_j w_{ij} + \\ &\sum_{i=1}^{p} \sum_{j=i+1}^{p} x_i x_j u_{ij} + \sum_{i=1}^{q} \sum_{j=i+1}^{q} \mu_i \mu_j v_{ij} - \log Z(\boldsymbol{\theta}) \\ &- \sum_{j=1}^{p} \left(\mu_j \log \mu_j + (1 - \mu_j) \log(1 - \mu_j)\right).\end{aligned} \tag{4.20}$$

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 $\mu_j$ and updating it according to the computed direction. By setting the partial derivative of $\mathcal{L}_Q$ with respect to $\mu_j$ to zero, we get the following fixed-point iteration:

$$\mu_j \leftarrow \phi\left(\sum_{i=1}^{p} w_{ij} x_i + \sum_{k=1, k\neq j}^{q} v_{kj} h_k + c_j\right), \tag{4.21}$$

where $\phi$ 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 $Q$.

$Q$ can only have a single mode, unless $\mu_j = 0.5$ for some $j$, because all hidden units were assumed to be independent from each other. 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. (4.20).

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 (4.20) of a Boltzmann machine. One can simply repeat the following update to each parameter:

$$\begin{aligned}\theta^{\langle t+1 \rangle} &= \theta^{\langle t \rangle} + \eta_{\langle t \rangle} \left( \left\langle h(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}^{\langle t \rangle}) \right\rangle_{\mathrm{d}} - \left\langle h(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}^{\langle t \rangle}) \right\rangle_{\mathrm{m}} \right) \\ &= \theta^{\langle t \rangle} + \eta_{\langle t \rangle} \left( H_0 - H_\infty \right), \end{aligned} \tag{4.22}$$

[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.

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:

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

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 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. (4.21) to all hidden units conditioned on $\mathbf{x}^{(n)}$. Then,

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

The problem is with the second term $H_\infty$ which requires running a Gibbs sampling chain until 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 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 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. (4.22), 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

$H_\infty$ 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 $H_\infty$ 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 such as 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 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 the 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.

From this, we can 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.

[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).

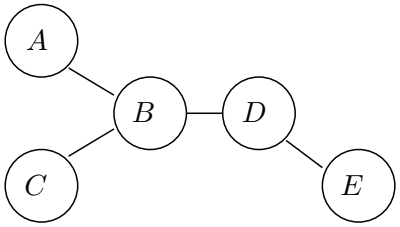



**Figure 4.6.** An example Markov random field with five random varaibles.

In this sense, we define the *Markov blanket* of a random variable as the set of all immediate neighboring variables. If all the variables in the Markov blanket of another variable were observed, the variable would be 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 $A$. When $D$ is observed, $A$ is conditionally independent from $E$ since $D$ is 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.

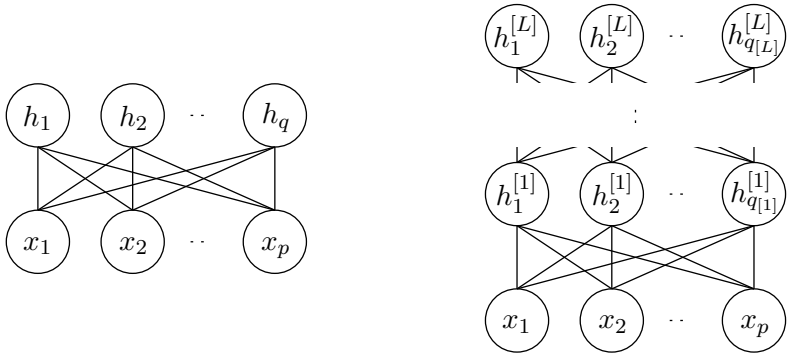
In this example, the Markov blanket of $D$ consists of $B$ and $E$. Whenever both $B$ and $E$ are observed, $D$ is conditionally independent from all other variables, in this case $A$ and $C$, regardless of the connectivity in the graph.

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.

It is unfortunately 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.




(a) Restricted Boltzmann Machine (b) Deep Boltzmann Machine

**Figure 4.7.** Illustrations of restricted and deep Boltzmann machines.

### 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

$$-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \mathbf{b}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{h} + \mathbf{x}^\top \mathbf{W} \mathbf{h}. \tag{4.23}$$

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 the set of all hidden units, and that of a hidden unit $h_j$ is the 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, 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:

$$p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta}) = \prod_{i=1}^{p} \nu_i^{x_i} (1 - \nu_i^{1-x_i}), \tag{4.24}$$

$$p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta}) = \prod_{j=1}^{q} \mu_j^{h_j} (1 - \mu_j^{1-h_j}), \tag{4.25}$$

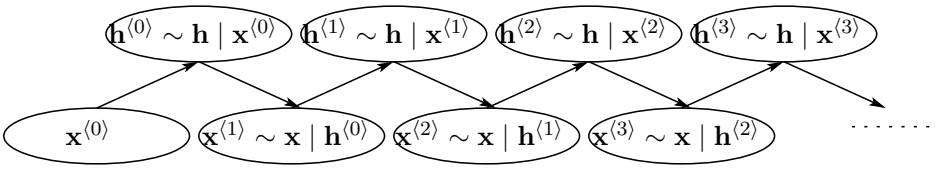



**Figure 4.8.** An illustration of performing a block Gibbs sampling on a restricted Boltzmann machine. $\mathbf{x}_0$ is initialized to a random binary vector.

where

$$\nu_i = p(x_i = 1 \mid \mathbf{h}, \boldsymbol{\theta}) = \phi\left(\sum_{j=1}^{q} w_{ij} h_j + b_i\right),$$
$$\mu_j = p(h_j = 1 \mid \mathbf{x}, \boldsymbol{\theta}) = \phi\left(\sum_{i=1}^{p} w_{ij} x_i + c_j\right).$$

Effectively, in just two parallelized steps, the state of each unit is replaced by a new sample. See Fig. 4.8 for an illustration of Gibbs sampling in the case of RBMs.

Furthermore, 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. (4.25). This enables us to exactly compute the first term of the gradient in Eq. (4.4):

$$\left\langle \frac{\partial\left(-E(\mathbf{x}^{(n)}, \mathbf{h} \mid \boldsymbol{\theta})\right)}{\partial\theta} \right\rangle_{\mathrm{d}} = \frac{1}{N} \sum_{n=1}^{N} \left( \frac{\partial - E(\mathbf{x}^{(n)}, \boldsymbol{\mu}^{(n)} \mid \boldsymbol{\theta})}{\partial\theta} \right), \tag{4.26}$$

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

This property of an RBM that enables parallelized Gibbs sampling and exact computation of posterior distribution over hidden units, however, 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

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \frac{\prod_{j=1}^{q} \varphi_j(\mathbf{x} \mid \boldsymbol{\theta}_j)}{\sum_{\mathbf{x}'} \prod_{j=1}^{q} \varphi_j(\mathbf{x}' \mid \boldsymbol{\theta}_j)}, \tag{4.27}$$

where $\varphi_j$ 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 $\varphi_j$ 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. (4.27) as

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{j=1}^{q} \left[1 + \tilde{\varphi}_j(\mathbf{x} \mid \boldsymbol{\theta}_j)\right], \tag{4.28}$$

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. (4.28) can be written as a marginal probability of the joint probability of both the visible and hidden variables:

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \sum_{\mathbf{h}} \frac{1}{Z(\boldsymbol{\theta})} \prod_{j=1}^{q} \tilde{\varphi}_j(\mathbf{x} \mid \boldsymbol{\theta}_j)^{h_j}. \tag{4.29}$$

Furthermore, one can 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

$$p(h_j = 1 \mid \mathbf{x}, \boldsymbol{\theta}_j) = \frac{\tilde{\varphi}_j(\mathbf{x} \mid \boldsymbol{\theta}_j)}{1 + \tilde{\varphi}_j(\mathbf{x} \mid \boldsymbol{\theta}_j)}. \tag{4.30}$$

If it is further assumed that the choice of $\varphi_j$'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 $\varphi_j$ 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

$$\begin{aligned} p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta}) &= \frac{1}{Z(\boldsymbol{\theta})} \exp\left\{\sum_{j=1}^{q} h_j c_j + \sum_{i=1}^{p} \sum_{j=1}^{q} w_{ij} x_i h_j\right\} \\ &= \frac{1}{Z'(\boldsymbol{\theta})} \prod_{i=1}^{p} \exp\left\{\sum_{j=1}^{q} w_{ij} h_j\right\}^{x_i}, \end{aligned} \tag{4.31}$$

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) $\mathbf{x} \in \{0, 1\}^p$. Comparing Eqs. (4.30) and (4.31)

against Eqs. (4.25) and (4.24), we see that 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 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 $k$ 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 $P_\infty$ is the model distribution that can be described by the samples collected by running a Gibbs sampler for a 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 $\mathrm{KL}(P_k \| P_\infty)$. 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,

$$\frac{\partial \mathrm{CD}_k}{\partial \theta} \approx \left\langle \frac{\partial \left(-E(\mathbf{x}^{(n)}, \mathbf{h} \mid \boldsymbol{\theta})\right)}{\partial \theta} \right\rangle_{P_0} - \left\langle \frac{\partial \left(-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})\right)}{\partial \theta} \right\rangle_{P_k}, \tag{4.32}$$

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. (4.32) 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 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

The 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

$$-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \mathbf{b}^\top \mathbf{x} + \mathbf{c}_{[1]}^\top \mathbf{h}_{[1]} + \mathbf{x}^\top \mathbf{W} \mathbf{h}_{[1]} + \sum_{l=2}^{L} \left( \mathbf{c}_{[l]}^\top \mathbf{h}_{[l]} + \mathbf{h}_{[l-1]}^\top \mathbf{U}_{[l-1]} \mathbf{h}_{[l]} \right), \tag{4.33}$$

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

[6] From here on, we will occasionally refer to the learning algorithm that minimizes CD as CD learning, when there is no ambiguity.

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. (4.21) 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. Instead of maximing the log-likelihood, or its variational lower-bound, (Goodfellow et al., 2013) proposed a learning algorithm that maximizes the multi-prediction objective function.

## 4.5 Boltzmann Machines and Autoencoders

In this section, we 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 as one of the related approaches of deep autoencoders.

### 4.5.1 Restricted Boltzmann Machines and Autoencoders

Here we consider 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. (4.26).

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

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta} = \lim_{t\to\infty} \sum_{s=1}^{t-1} \left( \mathbb{E}\left[\frac{\partial \log p(\mathbf{x}^{\langle s\rangle} \mid \mathbf{h}^{\langle s\rangle})}{\partial \theta}\right] + \mathbb{E}\left[\frac{\partial \log p(\mathbf{h}^{\langle s\rangle} \mid \mathbf{x}^{\langle s+1\rangle})}{\partial \theta}\right]\right) + \mathbb{E}\left[\frac{\partial \log p(\mathbf{x}^{\langle t\rangle})}{\partial \theta}\right], \tag{4.34}$$

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. (4.34) 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

$$\frac{\partial \log p(\mathbf{x}^{\langle 1\rangle} \mid \hat{\mathbf{h}})}{\partial \theta}. \tag{4.35}$$

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. (4.35) becomes

$$\sum_{n=1}^{N} \frac{\partial \log p(\mathbf{x}^{(n)} \mid \hat{\mathbf{h}}^{(n)})}{\partial \theta}.$$

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 results in a more biased estimation, if we look at the resulting model as an RBM.

The objective function of an autoencoder, however, 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,

$$-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = -\sum_{i=1}^{p} \frac{(x_i - b_i)^2}{2\sigma_i^2} + \sum_{j=1}^{q} h_j c_j + \sum_{i=1}^{p} \sum_{j=1}^{q} \frac{x_i}{\sigma_i^2} h_j w_{ij}. \tag{4.36}$$

The conditional distributions of the visible and hidden units are defined by

$$p(x_i = x|\mathbf{h}) = \mathcal{N}\left(x \,\middle|\, b_i + \sum_j h_j W_{ij} \,,\, \sigma_i^2\right) \tag{4.37}$$

and

$$p(h_j = 1|\mathbf{x}) = \phi\left(c_j + \sum_i W_{ij} \frac{x_i}{\sigma_i^2}\right), \tag{4.38}$$

where $\phi$ 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

[7] The energy function of a GRBM was originally proposed by Hinton and Salakhutdinov (2006) to be

$$-E(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = -\sum_{i=1}^{p} \frac{(x_i - b_i)^2}{2\sigma_i^2} + \sum_{j=1}^{q} h_j c_j + \sum_{i=1}^{p} \sum_{j=1}^{q} \frac{x_i}{\sigma_i} h_j w_{ij},$$

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:

$$\begin{aligned}\log p(\mathbf{x} \mid \boldsymbol{\theta}) &= -\frac{(b_i - x_i)^2}{2\sigma_i^2} + \sum_{j=1}^{q} \log\left(1 + \exp\left\{\sum_{i=1}^{p} \frac{x_i}{\sigma_i^2} w_{ij} + c_j\right\}\right) - \log Z(\boldsymbol{\theta}) \\ &= \log p^*(\mathbf{x} \mid \boldsymbol{\theta}) - \log Z(\boldsymbol{\theta}). \end{aligned} \tag{4.39}$$

Then, the score function of the GRBM is

$$\psi_i(\mathbf{x} \mid \boldsymbol{\theta}) = \frac{\partial \log p^*(\mathbf{x} \mid \boldsymbol{\theta})}{\partial x_i} = \frac{b_i - x_i}{\sigma_i^2} + \sum_{j=1}^{q} \hat{h}_j \frac{w_{ij}}{\sigma_i^2}. \tag{4.40}$$

Furthermore, we will need

$$\frac{\partial \psi_i(\mathbf{x} \mid \boldsymbol{\theta})}{\partial x_i} = -\frac{1}{\sigma_i^2} + \sum_{j=1}^{q} \hat{h}_j (1 - \hat{h}_j) \frac{w_{ij}^2}{\sigma_i^4},$$

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, score matching then tries to minimize the following cost function instead of maximizing the marginal log-likelihood:

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{p} \left[\frac{\partial \psi_i(\mathbf{x}^{(n)} \mid \boldsymbol{\theta})}{\partial x_i} + \frac{1}{2} \psi_i(\mathbf{x} \mid \boldsymbol{\theta})^2\right] \tag{4.41}$$

If we assume that the bias $b_i$ and the conditional variance $\sigma_i$ to each visible unit are

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 $\sigma_i$ affecting the mean, it was proposed in Publication V that the energy function be modified so that $\sigma_i$ does not influence the conditional mean of the visible unit. In Publication V, it was claimed that this formulation makes estimating $\sigma_i$'s easier, and hence we use the modified definition of the energy function here.

respectively fixed to 0 and 1[8], the cost function in Eq. (4.41) can be rewritten as

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}\left[\frac{1}{2}\sum_{i=1}^{p}\left(\sum_{j=1}^{q}\hat{h}_j w_{ij} - x_i\right)^2 + \sum_{i=1}^{p}\sum_{j=1}^{q} h_j(1-h_j)w_{ij}^2\right] + C,$$

where $C$ is a constant that does not depend on $\boldsymbol{\theta}$.

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 as 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 a usual practice to ignore $\sigma_i$ by forcing it to 1 (see, e.g., Hinton, 2012).

### 4.5.2 Deep Belief Network

In Section 3.2.3, we presented the sigmoid belief network in relation with a deep autoencoder Here, we describe a deep belief network (DBN) (Hinton et al., 2006) which extends the sigmoid belief network by adding a layer of undirected network on its top.

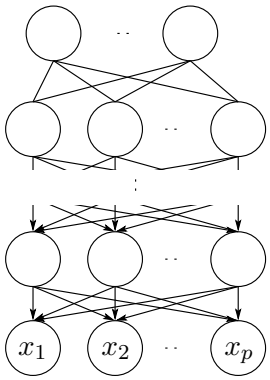



**Figure 4.9.** An illustration of a deep belief network. Note that the top two layers are connected to each other by *undirected* edges.

The 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 of a DBN as

$$-E(\mathbf{x}, \mathbf{h}^{[1]}, \dots, \mathbf{h}^{[L]} \mid \boldsymbol{\theta}) = \log p(\mathbf{x} \mid \mathbf{h}^{[1]}, \boldsymbol{\theta}) + \sum_{l=1}^{L-2} \log p(\mathbf{h}^{[l]} \mid \mathbf{h}^{[l+1]}, \boldsymbol{\theta}) + \log p(\mathbf{h}^{[L-1]}, \mathbf{h}^{[L]}, \boldsymbol{\theta}) \tag{4.42}$$

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. Additionally, there is a separate set of parameters for the parameters of a top-level 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 variational lower-bound of 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 the 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. (4.26).

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. (4.36) 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 unsupervised and supervised neural networks separately. In fact, our main focus was on the unsupervised neural networks rather than the supervised ones. We mostly discussed 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 the so called layer-wise pretraining, where shallow unsupervised neural networks were used to initialize the parameters of a deep multilayer 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 inputs and outputs and use it to compute the conditional distribution of an 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*. Oth-

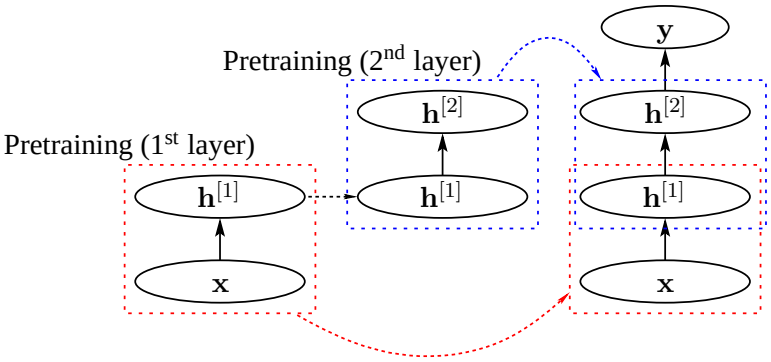



**Figure 5.1.** Illustration of the layer-wise pretraining of a multilayer perceptron. The dashed directed lines indicate *copying* of either pretrained models or the activations of the hidden units of the pretrained models.

erwise, this simple neural network will fail 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 transforms the coordinate system from the immediate lower layer such that the samples become linearly separable after going through the multiple layers of nonlinear transformations.

This suggests a possibility of incrementally building a sequence of intermediate layers that gradually extract better and better representations of data.[1] This is different from 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 different. 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 and

[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).

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 $\phi$ 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. (4.25)).

Hence, we may 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. 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 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.

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 whole stack.

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 using 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 $y$.

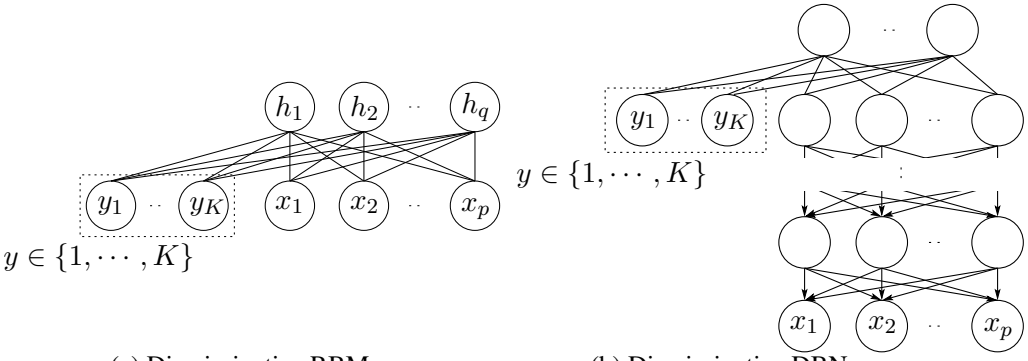
If $y$ can only have a finite number of possible outcomes (classification), we can simply evaluate $p(\mathbf{x}^*, y)$ for all possible $y$'s and choose the $y$ with the largest value. Otherwise, it is possible to optimize $p(\mathbf{x}^*, y)$ with respect to $y$ to compute the best possible $y$, although 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.

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 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 a 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 multilayer 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 about 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 $\left\{\tilde{\mathbf{h}}^{[l]}\right\}_{l=1}^{L}$, 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 $\phi$ 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 $\tilde{\mathbf{h}}^{[l]}$ 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

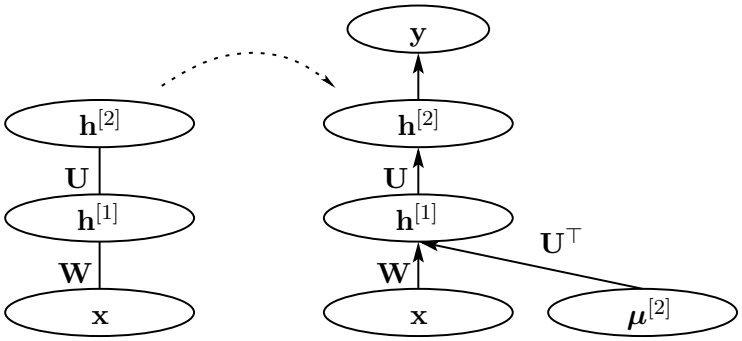



**Figure 5.3.** A deep Boltzmann machine with two hidden layers, on the left, is transformed to initialize a multilayer perceptron on the right. $\boldsymbol{\mu}^{[2]}$ is a vector of the variational parameters of the second hidden layer of the DBM.

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 finetune the model to optimize for classification performance.

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 performance in 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 incremental feature learning from the perspective of an *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 should be noticed 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 identical 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 $\phi$ 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 from 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 whose weights are fixed to those of the RBM. We repeat this step infinitely, and we get the infinitely deep sigmoid belief network with 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).

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 are collected.

This means that if, instead of a random state, we 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.

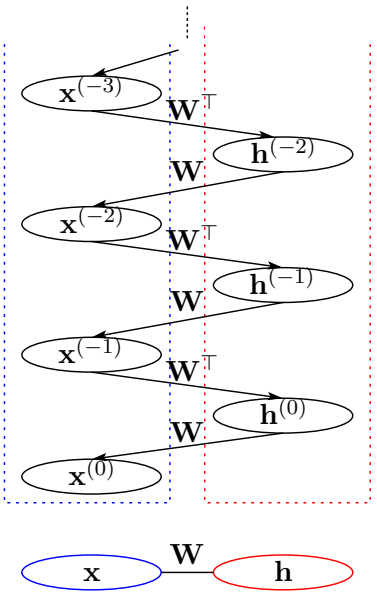



**Figure 5.4.** The ancestral sampling on an infinitely deep sigmoid belief network with tied weights is equivalent to the block Gibbs sampling on a restricted Boltzmann machine.

### 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. 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 where 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 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, we may 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:

$$\begin{aligned}
\sum_{n=1}^{N} \log p(\mathbf{x}^{(n)} \mid \boldsymbol{\theta}) &\geq \sum_{n=1}^{N} \left( \mathbb{E}_{Q(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)})} \left[ \log p(\mathbf{x}^{(n)}, \mathbf{h}^{[1]} \mid \boldsymbol{\theta}_1) \right] + \mathcal{H}(Q) \right) \\
&= \sum_{n=1}^{N} \left( \mathbb{E}_{Q(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)})} \left[ \log p(\mathbf{x}^{(n)} \mid \mathbf{h}^{[1]}, \boldsymbol{\theta}_1) \right] + \mathcal{H}(Q) \right) \\
&\quad + \sum_{n=1}^{N} \mathbb{E}_{Q(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)})} \left[ \log \sum_{\mathbf{h}^{[2]}} p(\mathbf{h}^{[1]}, \mathbf{h}^{[2]} \mid \boldsymbol{\theta}_2) \right], \qquad (5.3)
\end{aligned}$$

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 possibly 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.

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 more about using 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.

We can 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 $\phi$ 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

$$\tilde{Q}(\mathbf{h}) = \prod_{l=1}^{L} \tilde{q}_{\tilde{\boldsymbol{\mu}}^{[l]}} \left( \mathbf{h}^{[l]} \right)$$

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.

| | DBN | DBN-FF | AE |
|---|---|---|---|
| Recognition | Layer-wise Factorial | Fully Factorial | Fully Factorial |
| Generation | 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 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 the 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 all the edges in a DBM are *undirected*. This is a critical difference with 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 a 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 the 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 an 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.

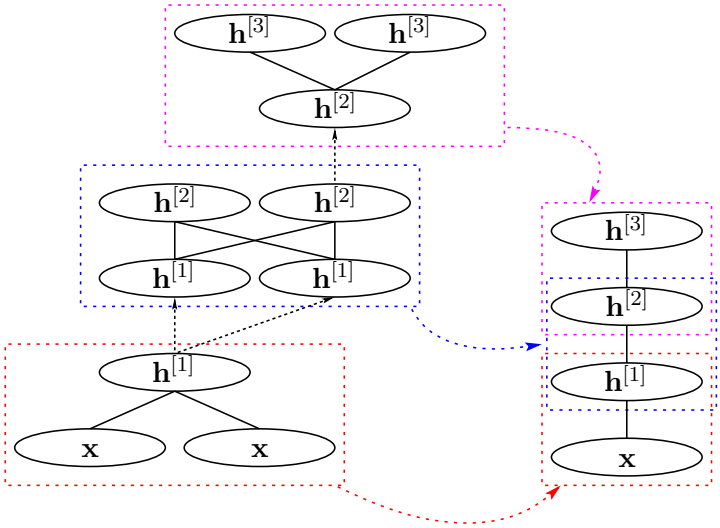



**Figure 5.5.** Illustration of the layer-wise pretraining of a deep Boltzmann machine. The dashed directed lines indicate *copying* either the pretrained models or the activations of the hidden units of the pretrained models.

We can rewrite the variational lower bound of a DBN given in Eq. (5.3) as

$$\begin{aligned}\sum_{n=1}^{N} \log p(\mathbf{x}^{(n)} \mid \boldsymbol{\theta}) &\geq \sum_{n=1}^{N} \left( \mathbb{E}_{Q(\mathbf{h}^{[1]}|\mathbf{x}^{(n)})} \left[ \log p(\mathbf{x}^{(n)} \mid \mathbf{h}^{[1]}, \boldsymbol{\theta}_1) \right] \right. \\ &\qquad \left. + \mathbb{E}_{Q(\mathbf{h}^{[1]}|\mathbf{x}^{(n)})} \left[ \log \frac{p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_2)}{Q(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)})} \right] \right) \\ &= \sum_{n=1}^{N} \left( \mathbb{E}_{Q(\mathbf{h}^{[1]}|\mathbf{x}^{(n)})} \left[ \log p(\mathbf{x}^{(n)} \mid \mathbf{h}^{[1]}, \boldsymbol{\theta}_1) \right] \right. \\ &\qquad \left. - \mathrm{KL}\left( Q(\mathbf{h}^{[1]} \mid \mathbf{x}^{(n)}) \| p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_2) \right) \right). \end{aligned} \tag{5.6}$$

From this, we can see 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 with two copies of visible units ($\mathbf{x}$ and $\mathbf{h}^{[2]}$), 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, 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 in Eq. (5.6). The new prior distribution can be similarly written as

$$p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_2) = \frac{1}{Z(\boldsymbol{\theta}_2)} \left( \sum_{\mathbf{x}} p(\mathbf{h}^{[1]}, \mathbf{x} \mid \boldsymbol{\theta}_2) \right) \left( \sum_{\mathbf{h}^{[2]}} p(\mathbf{h}^{[1]}, \mathbf{h}^{[2]} \mid \boldsymbol{\theta}_2) \right). \quad (5.8)$$

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 from the aggregate posterior $Q(\mathbf{h}^{[1]} \mid \mathbf{x})$ than $p(\mathbf{h}^{[1]} \mid \boldsymbol{\theta}_1)$ in Eq. (5.8) (See Appendix of Salakhutdinov and Hinton, 2012b).

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 that only *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 modeling of 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 the 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 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

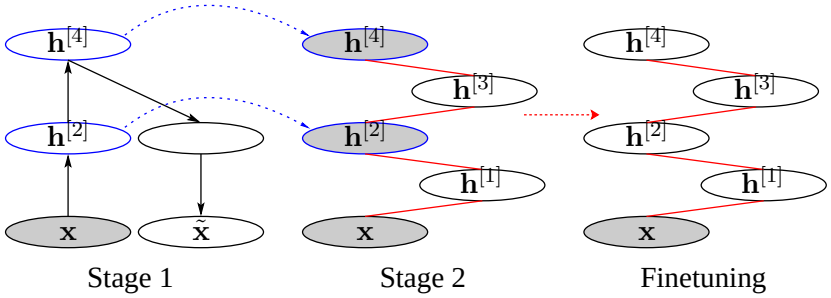



**Figure 5.6.** Illustration of the layer-wise pretraining of a deep Boltzmann machine. The dashed directed lines indicate *copying* of either pretrained models or the activations of the hidden units of the pretrained models. In this figure, a deep autoencoder is used to learn an arbitrary approximate 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 a 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., Goodfellow et al., 2013; 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 way 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] More recently, Google reported that a deep convolutional neural network is able to achieve the performance comparable to that of human operators on recognizing multi-digit numbers from Google Street View images[5] (Goodfellow et al., 2013).

These recent successes of deep neural networks in both academic research and com-

[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

[5] See the article *How Google Cracked House Number Identification in Street View* from the MIT Technology Review:
http://www.technologyreview.com/view/523326/how-google-cracked-house-number-identification-in-street-view/

mercial applications may be attributed to several recent breakthroughs. Layer-wise pretraining of a multilayer 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 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 multilayer 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[6] 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 the 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 the basic principles of deep neural networks.

In this last chapter, the author briefly summarizes the main contents and discusses 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

[6] 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/

detail and found to be related to each other. The underlying principles of those two models were found to be useful in understanding the layer-wise pretraining of a deep multilayer 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 a probabilistic perspective. Similarly, a linear autoencoder was shown to be equivalent to principal component analysis as well as its probabilistic variant.

The author went on to describing their deeper, nonlinear versions which are a multilayer 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 how to estimate the statistics and learn the parameters of Boltzmann machine as well as their structurally restricted variants such as the restricted Boltzmann machines (RBM) and the 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 a recently introduced method of pretraining a deep multilayer 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 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 the 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 layers.

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 multilayer perceptron (MLP, see Section 3.1) would correspond 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 classification or 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 the observed variables $\mathbf{x}_o$:

$$p(\mathbf{x}_m \mid \mathbf{x}_o) = \sum_{\mathbf{h}} p(\mathbf{x}_m \mid \mathbf{h}) p(\mathbf{h} \mid \mathbf{x}_o).$$

A deep neural network, however, replaces this with a parametric nonlinear function that computes

$$\sum_{\mathbf{h}} p(\mathbf{x}_m \mid \mathbf{h}) Q(\mathbf{h} \mid \mathbf{x}_o) \mathbb{E}_Q \left[ p(\mathbf{x}_m \mid \mathbf{h} \right]$$

in a single sweep, since this exact marginalization is often computationally intractable.

One consequence of assuming a simple, unimodal distribution $p(\mathbf{x}_m \mid \mathbf{h})$, is that the approximate predictive distribution $\tilde{p}(\mathbf{x}_m \mid \mathbf{x}_o)$ loses most of the information in the true predictive distribution $p(\mathbf{x}_m \mid \mathbf{x}_o)$. This is due to the unimodality of $\tilde{p}(\mathbf{x}_m \mid \mathbf{x}_o)$ while the true predictive distribution could have many probabilistic modes.

The inherent limitations of this approximate approach employed by deep neural networks are obvious.[7] 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

[7] Note that the discussion in this section has been highly motivated and influenced by Section 5 of (Bengio, 2013).

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 clear how a deep neural network can cope with a more flexible setting where the observed components are not fixed a priori.[8]

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.

If the ultimate goal is to make a decision that maximizes the predictive performance, we still need to be able to evaluate either approximately or exactly the multi-modal predictive distribution quickly and well. This 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 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.[9] It is, hence,

[8] 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.

[9] 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).

left for future research 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 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 a feedforward manner.

## 6.3 Matters Which Have Not Been Discussed

Despite the original intention of describing in much details the basics of deep neural networks, some important models and concepts have been left out from the main text. The author would like to briefly mention a few of them here, to provide any interested reader with some references.

The remainder of this section starts by providing a list of the previous work that relates 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 multilayer 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 finetuning a multilayer 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. A recurrent neural network described in Section 4.2.1 is, however, 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). 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 the 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 layer. 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.

When put into a probabilistic framework, most of these generative models, includ-

ing 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, Chapter 12.2.4).

Similarly to ICA, factor analysis has also been extended into a nonlinear model with potentially *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 the 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 was proposed by Frey and Hinton (1999).

### 6.3.2 Universal Approximator Property

In Section 3.1, it was mentioned that a multilayer 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. 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) and Freund and Haussler (1994) proved that a restricted Boltzmann machine (RBM, see Section 4.4.2) has this property when any binary distribution is considered. This 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 than 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 due to an intractable normalization constant in its formulation (see Eqs. (4.2)–(4.3)). Furthermore, except for restricted Boltzmann machines (RBM), it is intractable to marginalize hidden units.

Salakhutdinov and Murray (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. (4.20)) 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 multiplicative factor, 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 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 de-

noising autoencoder, nine hyper-parameters:

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 multilayer perceptron (MLP), this number of 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:

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 configuration. 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 the 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 using Bayesian optimization (see, e.g., Brochu et al.,

2010). Bayesian optimization simultaneously models the posterior distribution over a function $h(\Psi)$ that maps from a set $\Psi$ of hyper-parameters to the performance of a model trained using $\Psi$ and explores the state-space of $\Psi$ 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 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 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 incremental feature learning. In fact, if we had started our discussion on incremental feature learning from the convolutional neural network (see, e.g., LeCun et al., 1998a)

rather 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). 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, M. Miraz, A. Courville, and Y. Bengio. Multi-prediction deep Boltzmann machines. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 548–556, Dec. 2013.

I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv:*`1302.4389 [stat.ML]`, Feb. 2013.

A. Graves. Generating sequences with recurrent neural networks. *arXiv:*`1308.0850 [cs.NE]`, Aug. 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.

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.

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, Nov. 1999.

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.

M. LukošEvičIus and H. Jaeger. Survey: Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, Aug. 2009.

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.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

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.

F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books, 1962.

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 and I. Murray. On the quantatitive analysis of deep belief networks. In *Proceedings of the 25th International Conference on Machine learning (ICML 2008)*, pages 872–879, New York, NY, USA, 2008. ACM.

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.

L. K. Saul, T. Jaakkola, and M. I. Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

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.

# Publication 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.**

*LETTER* **Communicated by Yoshua Bengio**

# Enhanced Gradient for Training Restricted Boltzmann Machines

**KyungHyun Cho**
*kyunghyun.cho@aalto.fi*
**Tapani Raiko**
*tapani.raiko@aalto.fi*
**Alexander Ilin**
*alexander.ilin@aalto.fi*
*Department of Information and Computer Science, Aalto University School of Science, Espoo, Uusimaa 02150, Finland*

**Restricted Boltzmann machines (RBMs) are often used as building blocks in greedy learning of deep networks. However, training this simple model can be laborious. Traditional learning algorithms often converge only with the right choice of metaparameters that specify, for example, learning rate scheduling and the scale of the initial weights. They are also sensitive to specific data representation. An equivalent RBM can be obtained by flipping some bits and changing the weights and biases accordingly, but traditional learning rules are not invariant to such transformations. Without careful tuning of these training settings, traditional algorithms can easily get stuck or even diverge. In this letter, we present an enhanced gradient that is derived to be invariant to bit-flipping transformations. We experimentally show that the enhanced gradient yields more stable training of RBMs both when used with a fixed learning rate and an adaptive one.**

## 1 Introduction

Deep learning has gained popularity recently as a way for learning complex and large probabilistic models (see, e.g., Bengio, 2009). Especially, deep neural networks such as the deep belief network and the deep Boltzmann machine have been applied to various machine learning tasks with impressive improvements over conventional approaches (Hinton & Salakhutdinov, 2006; Salakhutdinov & Hinton, 2009; Salakhutdinov, 2009; Krizhevsky, 2010; Lee, Grosse, Ranganath, & Ng, 2009).

Deep neural networks are characterized by a large number of layers of neurons and by using layer-wise unsupervised pretraining to learn a probabilistic model for data. A deep neural network is typically constructed by stacking multiple restricted Boltzmann machines (RBM) so that the hidden layer of one RBM becomes the visible layer of another. Layer-wise

pretraining of RBMs then facilitates finding a more accurate model for the data. In cases of performing classification tasks using deep neural networks, various papers (Salakhutdinov & Hinton, 2009; Hinton & Salakhutdinov, 2006; Erhan et al., 2010) have empirically confirmed that such multistage learning works as good as, or in many cases better than, conventional learning methods, such as backpropagation with random initialization. This trend is more apparent when most training samples are unlabeled and only a small number of labeled training samples are available (see, e.g., Ranzato, Huang, Boureau, & LeCun, 2007). It is thus important to have an efficient method for training RBMs.

Unfortunately, training RBMs is known to be difficult. Traditional learning algorithms often converge only with the right choice of metaparameters that specify, for example, learning rate scheduling and the scale of the initial weights.

In this letter, we discuss difficulties of training RBMs using the traditional algorithm and propose a new training algorithm based on a new, enhanced gradient estimate. The new gradient is designed to be invariant to data representation, and it also facilitates learning distinct features by hidden neurons. We show the efficacy of the proposed gradient in experiments with either a fixed or an adaptive learning rate. The preliminary results of this work were presented in our conference paper (Cho, Ilin, & Raiko, 2011) and a technical report (Raiko, Cho, & Ilin, 2011).

## 2 Restricted Boltzmann Machines

**2.1 Model Definition.** The restricted Boltzmann machine is a stochastic neural network with a bipartite structure such that each visible neuron is connected to all the hidden neurons and each hidden neuron is connected to all the visible ones (Smolensky, 1986). The energy and the state probability are defined as

$$\begin{aligned} E(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta}) &= -\mathbf{v}^\top \mathbf{W}\mathbf{h} - \mathbf{b}^\top \mathbf{v} - \mathbf{c}^\top \mathbf{h}, \\ P(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta}) &= \frac{1}{Z(\boldsymbol{\theta})} \exp\{-E(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})\}, \end{aligned} \tag{2.1}$$

where $\mathbf{v}$ and $\mathbf{h}$ are binary column vectors representing the state of the visible and hidden neurons and parameters $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b}, \mathbf{c})$ include weights $\mathbf{W} = [w_{ij}]_{n_v \times n_h}$ and biases $\mathbf{b} = [b_i]_{n_v \times 1}$ and $\mathbf{c} = [c_j]_{n_h \times 1}$. $n_v$ and $n_h$ are the numbers of visible and hidden neurons, respectively. $Z(\boldsymbol{\theta})$ denotes the normalizing constant, which is intractable and can be calculated by summing exponentially many terms.

A useful property of the RBM is that hidden neurons $\mathbf{h}$ are independent of each other given visible neurons $\mathbf{v}$,

$$P(h_j = 1 \mid \mathbf{v}, \boldsymbol{\theta}) = \frac{1}{1 + \exp\left(-\sum_i w_{ij} v_i - c_j\right)}, \tag{2.2}$$

and the same holds for the visible neurons:

$$P(v_i = 1 \mid \mathbf{h}, \boldsymbol{\theta}) = \frac{1}{1 + \exp\left(-\sum_j w_{ij} h_j - b_i\right)}. \tag{2.3}$$

This fact allows for efficient parallel implementation of layer-wise Gibbs sampling when collecting samples from the distribution defined by an RBM.

**2.2 Training Algorithms.** The parameters of an RBM can be learned from data using the standard maximum likelihood estimation. Given a data set $\{\mathbf{v}^{(t)}\}_{t=1}^{N}$, the log likelihood of the parameters is

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{t=1}^{N} \log P(\mathbf{v}^{(t)} | \boldsymbol{\theta}) = \sum_{t=1}^{N} \log \sum_{\mathbf{h}} P(\mathbf{v}^{(t)}, \mathbf{h} \mid \boldsymbol{\theta}),$$

where the samples $\mathbf{v}^{(t)}$s are assumed to be independent of each other and the states $\mathbf{h}$ of the hidden neurons are marginalized out.

The gradient ascent update rules are

$$w_{ij} \leftarrow w_{ij} + \eta_w \nabla w_{ij}, \qquad \nabla w_{ij} = \langle v_i h_j \rangle_{\mathrm{d}} - \langle v_i h_j \rangle_{\mathrm{m}}, \tag{2.4}$$

$$b_i \leftarrow b_i + \eta_b \nabla b_i, \qquad \nabla b_i = \langle v_i \rangle_{\mathrm{d}} - \langle v_i \rangle_{\mathrm{m}}, \tag{2.5}$$

$$c_j \leftarrow c_j + \eta_c \nabla c_j, \qquad \nabla c_j = \langle h_j \rangle_{\mathrm{d}} - \langle h_j \rangle_{\mathrm{m}}, \tag{2.6}$$

where a shorthand notation $\langle \cdot \rangle_{P(\cdot)}$ denotes the expectation computed over probability distribution $P(\cdot)$. $\langle \cdot \rangle_{\mathrm{d}}$ denotes expectation computed over conditional distribution $P(\mathbf{h} \mid \mathbf{v}, \boldsymbol{\theta}) D(\mathbf{v})$ where $D(\mathbf{v})$ is a data distribution and $\langle \cdot \rangle_{\mathrm{m}}$ is expectation computed over the model distribution $P(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})$.

The well-known difficulty of using equations 2.4 to 2.6 is that while the expectations $\langle \cdot \rangle_{\mathrm{d}}$ can easily be calculated using equation 2.2, the exact computation of expectations $\langle \cdot \rangle_{\mathrm{m}}$ is intractable because of the need to compute the normalizing constant.

Conventional learning procedures employ the stochastic gradient ascent method, which uses only a small subset of training data samples, called a minibatch, to compute the gradients at each update.

*2.2.1 Contrastive Divergence.* An efficient method for training RBMs is based on minimizing contrastive divergence (CD; Hinton, 2002). In this approach, the true gradients in equation 2.4 to 2.6 are approximated by replacing expectations $\langle\cdot\rangle_{\mathrm{m}}$ with expectations $\langle\cdot\rangle_{P_n}$ evaluated over the distribution $P_n$ obtained by running $n$ steps of the layer-wise Gibbs sampling, starting from the empirical distribution defined by the training samples. This yields the following update rule for the weights:

$$w_{ij} \leftarrow w_{ij} + \eta_w\left[\langle x_i h_j\rangle_{P_0} - \langle x_i h_j\rangle_{P_n}\right],$$

where $P_0$ denotes distribution $P(\mathbf{h} \mid \mathbf{v}, \boldsymbol{\theta})$ with $\mathbf{v}$ fixed to the training data. In practice, using a short Gibbs sampling chain (e.g., $n = 1$) often yields good performance.

*2.2.2 Approximate Maximum Likelihood.* Minimizing CD is known to be biased for finite $n$ and to provide the maximum likelihood solution only when $n \to \infty$ (Carreira-Perpiñán & Hinton, 2005; Bengio & Delalleau, 2009). This problem is fixed in methods that use the stochastic approximation to the likelihood gradient (Younes, 1989), which yields a different procedure for collecting samples from the model distribution: The main difference in implementation compared to CD is that sampling is not started at the training samples for each minibatch. Existing approaches include persistent contrastive divergence (PCD; Tieleman, 2008), tempered transition (Salakhutdinov, 2009), and parallel tempering (PT; Desjardins, Courville, Bengio, Vincent, & Delalleau, 2010; Cho, Raiko, & Ilin, 2010).

In this letter, we use PCD and PT as representative methods of this kind. PCD is based on plain Gibbs sampling. The basic idea of PT is that multiple chains of Gibbs sampling are run for models with different "temperatures." Chains with higher temperatures correspond to more diffuse distributions and therefore can produce a greater variety of samples. Every now and then, the samples are swapped between the chains, which facilitates better exploration of the state space.

**2.3 Difficulties in Training RBM.** Training RBMs can be difficult in practice. Due to the intractability of the objective function, it is difficult to compare the quality of found solutions or even to know when learning has converged. It has been observed that the training procedure can diverge especially when Gibbs sampling is used to obtain samples from the model distribution (Desjardins, Courville, Bengio, Vincent et al., 2010; Schulz, Müller, & Behnke, 2010; Fischer & Igel, 2010). This diverging behavior

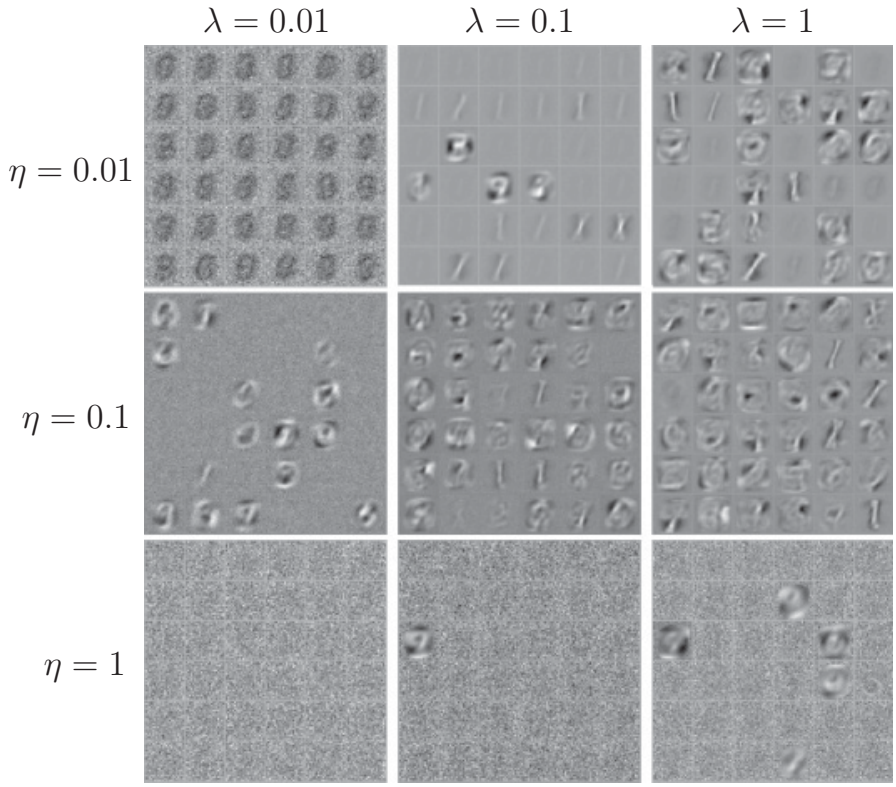



Figure 1: Visualization of the filters learned by RBMs on MNIST with various learning rates $\eta$ and initial weight scaling $\lambda$. Learning was performed for five epochs, each using the traditional gradient with a fixed learning rate.

can be suppressed by using more sophisticated sampling methods, but the likelihood can still fluctuate highly during training unless one uses appropriate learning rate scheduling or adapts the sampler to maintain a certain level of mixing property (Desjardins, Courville, & Bengio, 2010; Desjardins, Courville, Bengio, Vincent et al., 2010; Desjardins, Courville, Bengio, Vincent, & Delalleau, 2009).

*2.3.1 Sensitivity to Metaparameters.* In Figure 1, we demonstrate the sensitivity of the training procedure based on the traditional gradient in equations 2.4 to 2.6 to the scale of weight initialization and the learning rate. In this experiment, RBMs with 36 hidden neurons were trained on the MNIST data set (LeCun, Bottou, Bengio, & Haffner, 1998) with fixed learning rates $\eta$ and PT as the sampling strategy to obtain samples from the model distribution. Weights $w_{ij}$ were initialized with random values drawn from the normal distribution with zero mean and standard deviation $\lambda$. Each visible bias $b_i$ was initialized to $\log \frac{m_i}{1-m_i}$, where $m_i$ is the sample mean of the $i$th pixel in the training data, and all hidden biases $c_j$ were initially set to $-4$. The figure presents the filters learned by RBMs for different combinations of $\lambda$ and $\eta$.

It is clear that the results after a relatively short training period are highly dependent on the choice of metaparameters. Reasonable features are learned by most of the hidden neurons only with a careful selection of the initial weight scale and the learning rate, which was $\eta = 0.1$ and $\lambda = 1$ for the reported experiments. In longer runs, training results generally improve, and the difference between different training scenarios may be less dramatic. Nevertheless, this result suggests that careful selection of the metaparameters is very important for RBMs. The optimal combination of metaparameters is usually found by cross-validation, which may become time-consuming when there are many metaparameters to tune (see, e.g., Bergstra & Bengio, 2012). Hence, it is preferable to have a learning algorithm that is less sensitive to metaparameters.

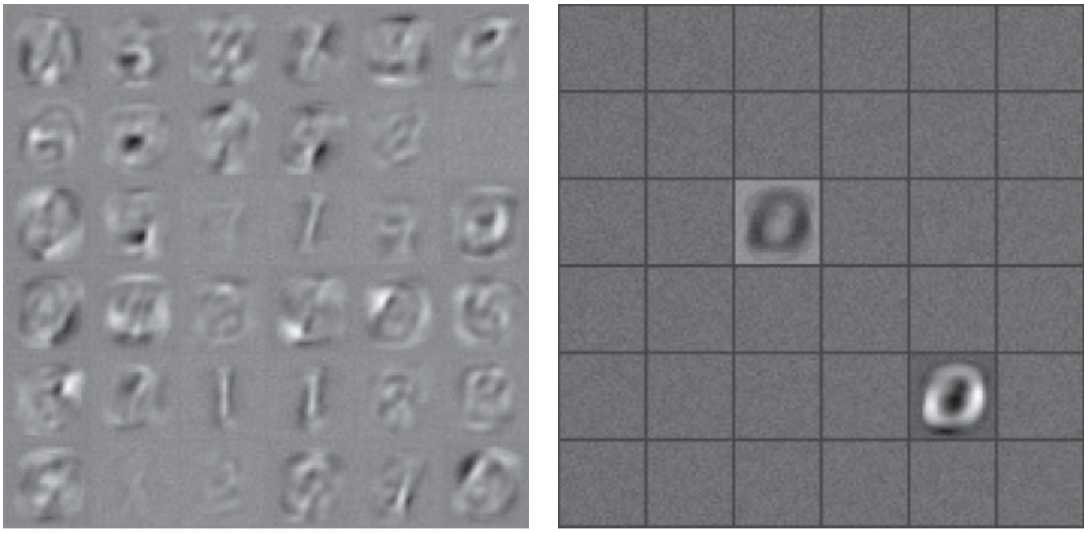
The results obtained with $\eta = 0.01$ illustrate a typical problem in training RBMs when several hidden neurons try to learn global filters that resemble the visible bias term $\mathbf{b}$ (see, e.g., Hinton, 2010). Such hidden neurons are activated for most of the input objects, but they are meaningless because the corresponding weights can be incorporated into biases $b_i$. One can also observe noisy global filters that do not seem to capture any meaningful features, especially in the results obtained with large $\eta$. Such hidden neurons are always inactive and can be removed without affecting the modeling capacity of the RBM. The existence of such hidden neurons is an indicator of poorly trained RBMs.

*2.3.2 Sensitivity to Data Representation.* In Figure 2, we demonstrate that the training procedure based on the traditional gradient in equations 2.4 to 2.6 is also sensitive to data representation. For example, flipping all the bits in the MNIST data set (such that zeros become ones and vice versa) produces an equivalent data set, which we call 1-MNIST. However, training results obtained on MNIST and 1-MNIST can be very different. The RBM trained on 1-MNIST after a relatively short training period does not contain any meaningful features: two hidden neurons model something similar to the visible bias, and the remaining ones are always inactive. In contrast, the RBM trained on MNIST has learned several reasonable features. It suggests that the learning algorithm based on the traditional gradient update is highly sensitive to the data representation.

## 3 Enhanced Gradient

**3.1 Derivation of the Enhanced Gradient.** In this section, we propose a new gradient to modify the update rules 2.4 to 2.6. We first define the covariance between two variables under distribution $P$ as

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



(a) MNIST (b) 1-MNIST

Figure 2: Visualization of the filters learned by RBMs with 36 hidden neurons on MNIST and 1-MNIST using PT sampling after five epochs. Both RBMs were trained with the traditional learning rate and a fixed learning rate 0.1. The initial weights were sampled from the normal distribution with zero mean and standard deviation 0.1.

Then the standard gradient in equation 2.4 can be rewritten as

$$\nabla w_{ij} = \mathrm{Cov}_{\mathrm{d}}(v_i, h_j) - \mathrm{Cov}_{\mathrm{m}}(v_i, h_j) + \langle v_i \rangle_{\mathrm{dm}} \nabla c_j + \langle h_j \rangle_{\mathrm{dm}} \nabla b_i, \tag{3.1}$$

where $\nabla c_j$ and $\nabla b_i$ are the gradients that appear in equations 2.5 and 2.6 and $\langle \cdot \rangle_{\mathrm{dm}} = \frac{1}{2}\langle \cdot \rangle_{\mathrm{d}} + \frac{1}{2}\langle \cdot \rangle_{\mathrm{m}}$ is the average activity of a neuron under the data and model distributions.

One potential problem with the gradient 3.1 is that it contains terms $\nabla c_j$, $\nabla b_i$ that point in the same direction as the gradient with respect to the bias terms. This effect is prominent when there are many neurons that are mainly active, that is, for which $\langle x_k \rangle_{\mathrm{dm}} \approx 1$, where $x_k$ can be either a visible or a hidden neuron. These terms can distract learning of meaningful weights by making many neurons learn features resembling the bias terms, the effect that is visible in Figures 1 and 2. When $\langle x_i \rangle_{\mathrm{dm}} \approx 0$ for most of the neurons, this effect can be negligible, which might explain why learning 1-MNIST is more difficult than MNIST and partially explain why sparse Boltzmann machines (Lee, Ekanadham, & Ng, 2008) have been successful.

Another problem of using gradient 3.1 is that the updated parameter values are different depending on the data representation. This can be

shown by using transformations where some of the binary units are flipped such that zeros become ones and vice versa:

$$\tilde{x}_k = x_k^{1-f_k}(1-x_k)^{f_k}, \qquad f_k \in \{0,1\}.$$

The parameters can then be transformed accordingly to $\tilde{\boldsymbol{\theta}}$:

$$\tilde{w}_{ij} = (-1)^{f_i+f_j} w_{ij}, \tag{3.2}$$

$$\tilde{b}_i = (-1)^{f_i}\left(b_i + \sum_j f_j w_{ij}\right), \tag{3.3}$$

$$\tilde{c}_j = (-1)^{f_j}\left(c_j + \sum_i f_i w_{ij}\right), \tag{3.4}$$

such that the resulting RBM has an equivalent energy function, that is, $E(\tilde{\mathbf{x}} \mid \tilde{\boldsymbol{\theta}}) = E(\mathbf{x} \mid \boldsymbol{\theta}) + \text{const}$ for all $\mathbf{x}$ (see the proof in appendix A).

When a model is transformed, updated, and transformed back, the resulting model depends on the transformations $f_k$:

$$\begin{aligned} w_{ij} &\leftarrow w_{ij} + \eta[\langle v_i h_j\rangle_{\mathrm{d}} - \langle v_i h_j\rangle_{\mathrm{m}} - f_i(\langle h_j\rangle_{\mathrm{d}} - \langle h_j\rangle_{\mathrm{m}}) - f_j(\langle v_i\rangle_{\mathrm{d}} - \langle v_i\rangle_{\mathrm{m}})] \\ &= w_{ij} + \eta[\mathrm{Cov}_{\mathrm{d}}(v_i, h_j) - \mathrm{Cov}_{\mathrm{m}}(v_i, h_j) \\ &\quad + (\langle v_i\rangle_{\mathrm{dm}} - f_i)\nabla c_j + (\langle h_j\rangle_{\mathrm{dm}} - f_j)\nabla b_i] \\ b_i &\leftarrow b_i + \eta\left[\nabla b_i - \sum_j f_j(\nabla w_{ij} - f_i \nabla c_j - f_j \nabla b_i)\right] \\ c_j &\leftarrow c_j + \eta\left[\nabla c_j - \sum_i f_i(\nabla w_{ij} - f_i \nabla c_j - f_j \nabla b_i)\right], \end{aligned}$$

where $\nabla\boldsymbol{\theta}$ are the gradients used in equations 2.4 to 2.6 and we assume that the learning rates for weights are biases are same. This is shown in appendix B.

Thus, there are $2^{n_v+n_h}$ different update rules defined by different combinations of binary $f_k$, $k = 1, \ldots, n_v + n_h$. All the update rules are well-founded maximum likelihood updates to the original model.

The new gradient is, then, proposed to be a weighted sum of the $2^{n_v+n_h}$ gradients with the following weights:

$$\prod_{k=1}^{n_v+n_h} \langle x_k\rangle_{\mathrm{dm}}^{f_k}\left(1 - \langle x_k\rangle_{\mathrm{dm}}\right)^{1-f_k}. \tag{3.5}$$

By using these weights, the new gradient prefers sparse data representations for which $\langle x_k \rangle_{\mathrm{dm}} \approx 0$ because the corresponding models get larger weights.

The proposed weighted sum yields the enhanced gradient (see appendix C),

$$\nabla_e w_{ij} = \mathrm{Cov}_{\mathrm{d}}(v_i, h_j) - \mathrm{Cov}_{\mathrm{m}}(v_i, h_j), \tag{3.6}$$

$$\nabla_e b_i = \langle v_i \rangle_{\mathrm{d}} - \langle v_i \rangle_{\mathrm{m}} - \sum_j \langle h_j \rangle_{\mathrm{dm}} \nabla_e w_{ij}, \tag{3.7}$$

$$\nabla_e c_j = \langle h_j \rangle_{\mathrm{d}} - \langle h_j \rangle_{\mathrm{m}} - \sum_i \langle v_i \rangle_{\mathrm{dm}} \nabla_e w_{ij}, \tag{3.8}$$

in which, by the choice of the weights 3.5, the effect of the bias gradient terms in the representation 3.1 is canceled out. Thus, the new update rules are

$$w_{ij} \leftarrow w_{ij} + \eta \nabla_e w_{ij}, \tag{3.9}$$

$$b_i \leftarrow b_i + \eta \nabla_e b_i, \tag{3.10}$$

$$c_j \leftarrow c_j + \eta \nabla_e c_j. \tag{3.11}$$

**3.2 Analysis of the Enhanced Gradient.** In appendix D, we show that the new update rules are invariant to the bit-flipping transformations. It is also easy to see that the enhanced gradient shares all zeros with the traditional gradient.

We compare the enhanced gradient to the traditional one on an RBM with 361 hidden neurons trained on MNIST. Figure 3 represents the angles between the update directions for the hidden neurons. Each element of the visualized matrices is the absolute value of the cosine of the angle between the update directions for two neurons. The gradients obtained with the traditional rule are highly correlated with each other. On the contrary, the new gradient yields update directions that are close to orthogonal, which allows the neurons to learn distinct features.

More details on how to obtain the bit-flipping transformation, the transformation-dependent update rules, and the enhanced gradient update rules for general Boltzmann machines are presented in our technical report (Raiko et al., 2011).

## 4 Experiments

In this section, we experimentally compare the proposed enhanced gradient with the traditional one.

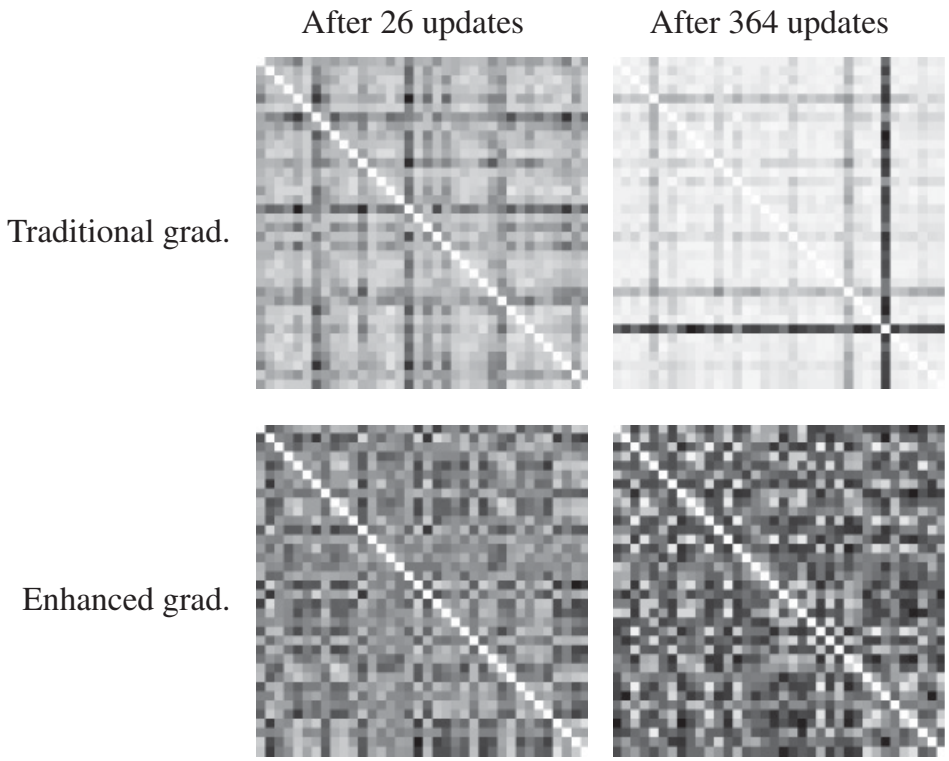



Figure 3: The angles between the update directions for the weights of the RBM with 361 hidden neurons. White pixels correspond to small angles, and black pixels correspond to orthogonal directions.

### 4.1 Experiments with MNIST and 1-MNIST

*4.1.1 Experiments with a Relatively Small Model.* We start by running the same experiment as in section 2.3.1 to test the sensitivity of the training procedure based on the enhanced gradient to the scale initialization and the learning rate. As can be seen from Figure 4, the results obtained with the enhanced gradient look better than the ones obtained with the traditional gradient (compare to Figure 1). After a relatively short training period, there are generally more filters that represent some meaningful features, which suggests that using the enhanced gradient can speed up learning.

In the next experiment, we compare the quality of resulting RBMs in longer training sessions. In order to be able to test different training settings in a reasonable amount of time, we train a relatively small RBM with only 361 hidden neurons. We trained RBMs on binarized MNIST and 1-MNIST data sets, in which the original grayscale pixels were rounded. RBMs were initialized according to the practical guide by Hinton (2010): The weights were randomly sampled from the normal distribution with zero mean and standard deviation $\frac{1}{n_v+n_h}$. Each visible bias $b_i$ was initialized to $\log \frac{m_i}{1-m_i}$, where $m_i$ is the sample mean of the $i$th pixel in the training data. All hidden biases $c_j$ were initially set to $-4$.

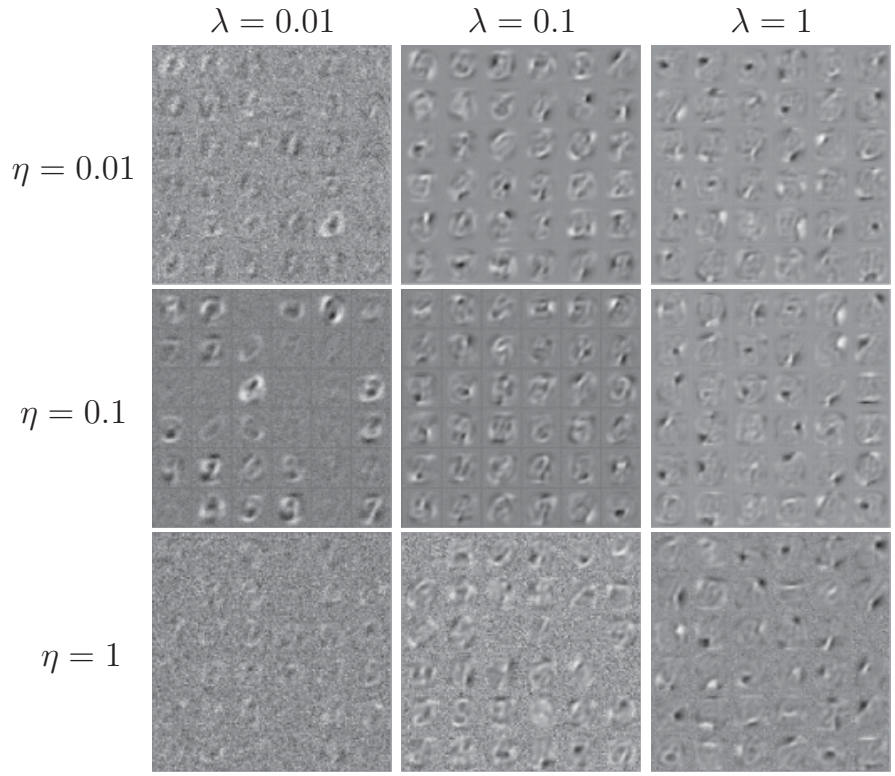
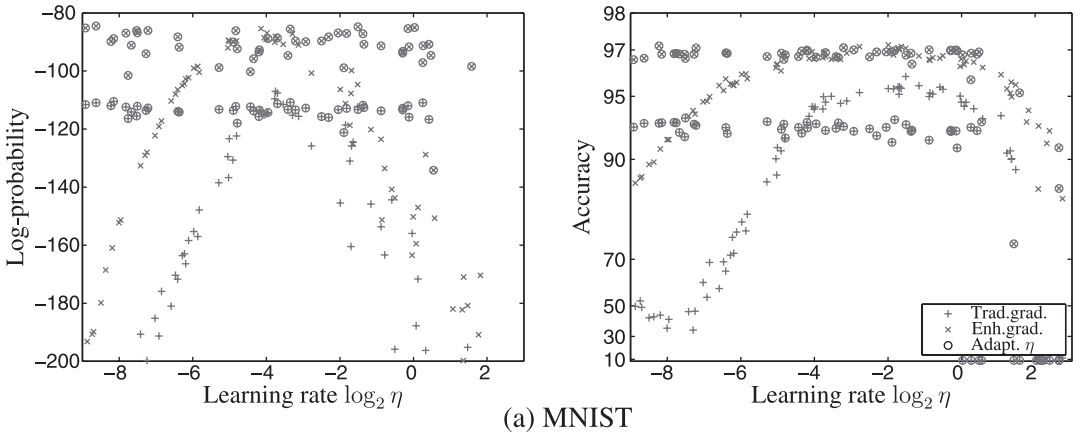



Figure 4: Visualization of the filters learned by RBMs with 36 hidden neurons on MNIST with various initial learning rates $\eta$ and initial weight scaling $\lambda$. Learning was performed for five epochs each using the enhanced gradient with a fixed learning rate.

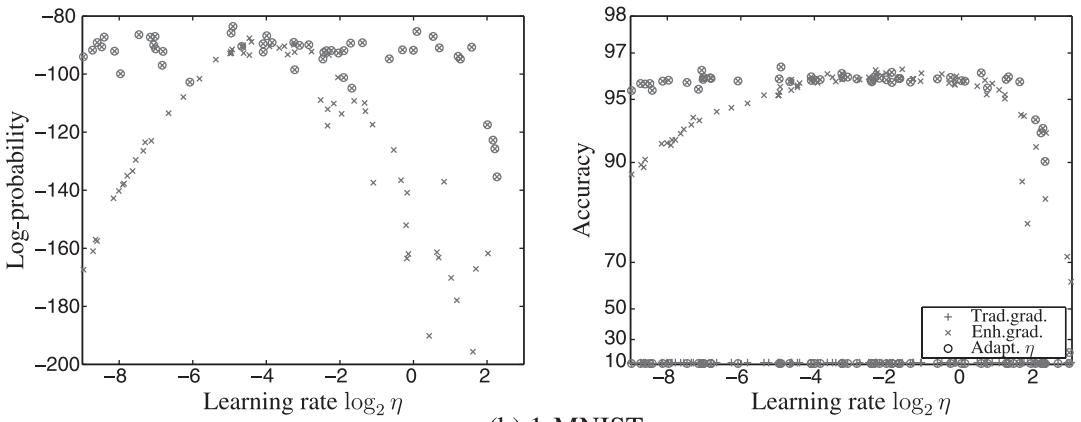
We ran a set of training sessions for 20 epochs each, varying the learning rate and the strategy for collecting samples from the model distribution. We used PT with 11 equally spaced temperatures and PCD and CD with a single Gibbs update ($n = 1$). The learning rate was initialized with values $2^l$, where $l$ was randomly sampled from a uniform distribution $[-9, 3]$. We tried both fixing the learning rate and using the adaptation scheme based on maximizing a local approximation of the likelihood (Cho, Ilin et al., 2011).

The quality of the trained models is assessed using the following quantities:

1. Log probability of test data under the trained model. The computation of this quantity requires the knowledge of the normalizing constant, which was estimated using annealed importance sampling (AIS), similarly to previous work (Salakhutdinov, 2009). The estimates of the normalizing constant were obtained by averaging 100 independent runs of AIS with different initializations. Each AIS run used 10,001 equally spaced temperatures.
2. Classification accuracy obtained with a logistic regression classifier trained on the activation probabilities of the hidden neurons




(a) MNIST




(b) 1-MNIST

Figure 5: Log probabilities and classification accuracies of test data for different initializations of the learning rate. The models were trained using the stochastic approximation with PT sampling.

conditioned on the original grayscale pixels. Additionally, we fine-tuned the network using stochastic backpropagation. Note that these are indirect measures of the RBM quality because they are not optimized during learning.

Figure 5 shows the comparison of the training results obtained with PT. Each marker represents a value of a performance index against the initial value of the learning rate. A cross corresponds to the traditional gradient, while an $\times$ corresponds to the enhanced gradient. Markers surrounded by circles represent the results obtained with the adaptive learning rate.

The main observation from Figure 5 is that in the case of PT, training with the enhanced gradient generally results in higher log probabilities and better classification accuracies. The variation of the results is much higher for the traditional approach, while most of the runs with the enhanced gradient provided relatively good RBMs. The improvement of the RBM quality is significant for MNIST and radical for 1-MNIST, where we were not able to train any reasonable model with the traditional gradient. Note

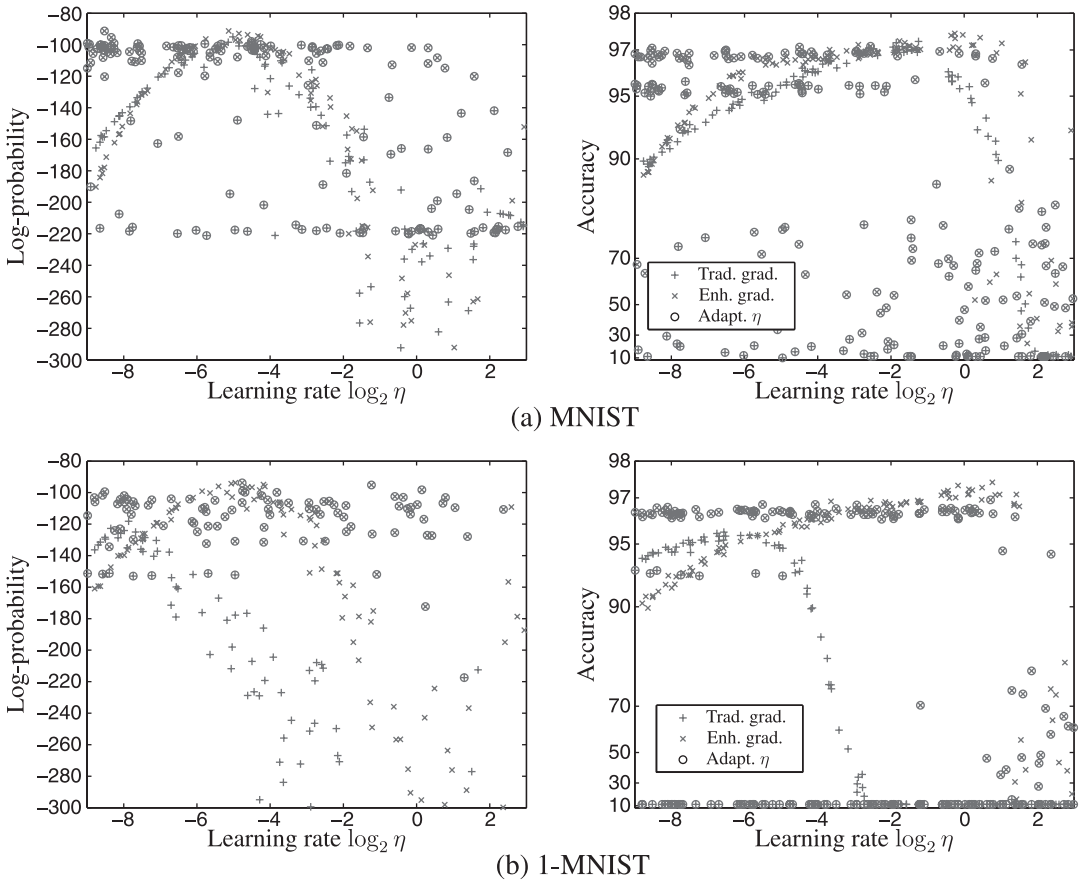



Figure 6: Log probabilities and classification accuracies of test data for different initializations of the learning rate. The models were trained using the stochastic approximation with Gibbs sampling. Note the difference in the $y$-axis scaling from Figures 5 and 7.

also that using the adaptive learning rate led in most cases to a relatively good model regardless of the learning rate initialization when it was used with the enhanced gradient. However, the adaptation mechanism did not find the optimal learning rate in the case of the traditional gradient, and the best results were obtained with a learning rate fixed to a proper value.

In spite of the fact that the enhanced gradient is invariant to data representation, note that the classification accuracies obtained on 1-MNIST are slightly worse than those on MNIST. This is due to different initial sparsity of the hidden units $h_j$. Approximately equivalent results for 1-MNIST and MNIST could be obtained when the hidden biases were initialized to $c_j \leftarrow -4 - \sum_i w_{ij} \langle v_i \rangle_d$. However, for consistency, we present only the results obtained with the initialization procedure proposed by Hinton (2010).

The results obtained using PCD with Gibbs sampling are shown in Figure 6. They indicate that the enhanced gradient is again superior to the traditional gradient. It is especially apparent in the case of 1-MNIST, where the enhanced gradient shows more robustness to changes in the

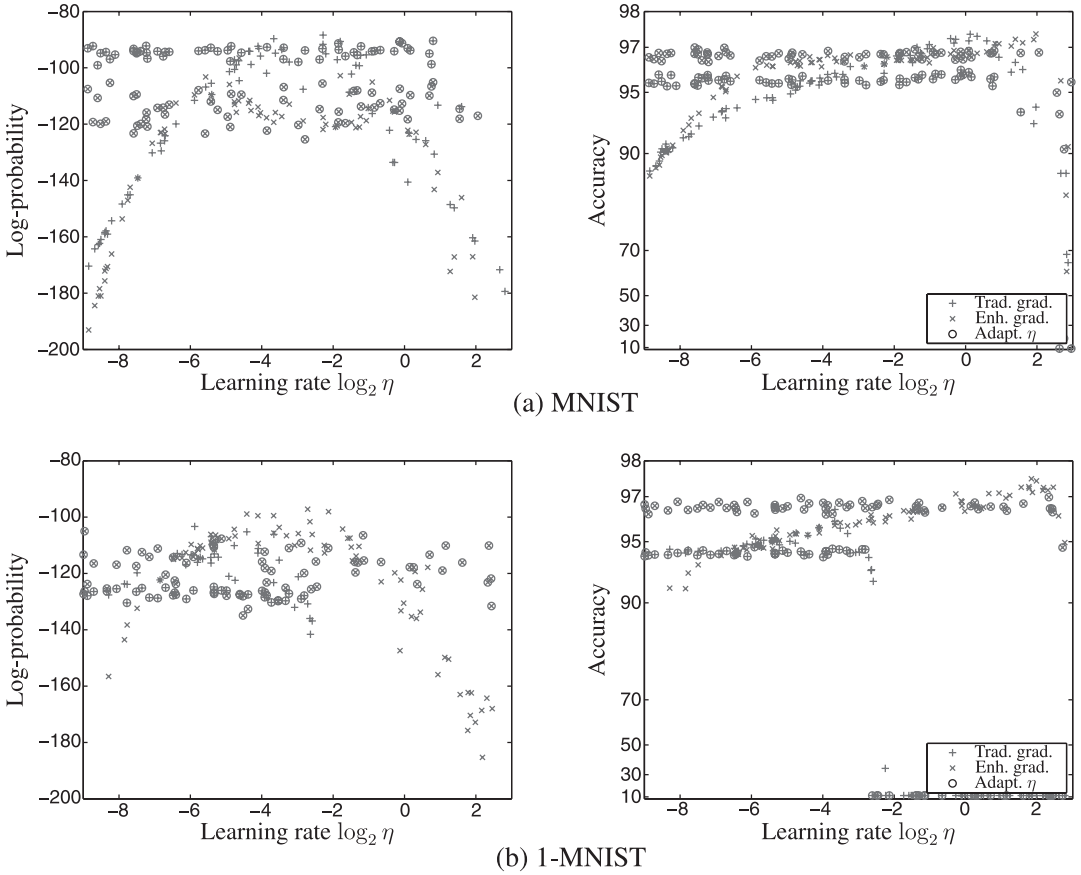
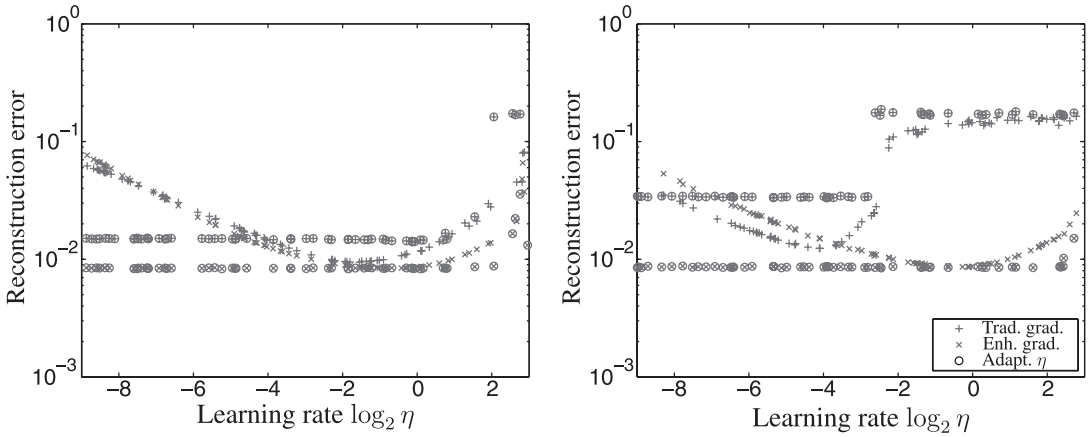



Figure 7: Log probabilities and classification accuracies of test data for different initializations of the learning rate. The models were trained by minimizing CD.

learning rate. In a few cases, we can observe that the adaptive learning rate was not able to adapt the learning rate appropriately. However, the results suggest that the enhanced gradient, together with the adaptive learning rate, performs better compared to the traditional gradient with or without the adaptive learning rate.

Figure 7 shows the results of the same experiment with CD. Again, learning from 1-MNIST is much easier with the enhanced gradient; note that learning with the traditional gradient often failed completely. The best classification accuracies obtained with the two gradients were quite similar. However, using the traditional gradient resulted in better generative models for MNIST. A possible explanation is that learning by minimizing CD does not maximize the log likelihood directly, and therefore one should not regard log likelihood as the ultimate performance indicator. Especially using a small number of Gibbs steps introduces a large bias in favor of minimizing the one-step reconstruction error (Bengio & Delalleau, 2009).

In Figure 8, we show the average mean-squared reconstruction error of the test samples. It is clear that in most cases, the models trained with the enhanced gradient achieved lower reconstruction errors. Together with the




(a) MNIST (b) 1-MNIST

Figure 8: Mean-squared one-step reconstruction errors of test samples for the models trained by minimizing CD. Note that smaller reconstruction error values represent better models, unlike log probabilities.

adaptive learning, those models were consistently better than the models trained with the traditional gradient for a broad range of initial learning rates.

*4.1.2 Experiments with Larger Models.* Additionally, in order to get higher classification accuracies on MNIST, we trained larger RBMs having 500, 1000, 2000, and 4000 hidden neurons. In this experiment, we used PT with 11 temperatures and CD with a single Gibbs step. All the models were trained for 1000 epochs, with the adaptive learning rate initialized to 0.0001 and upper-bounded by 0.1. In the second half of training, the learning rate was annealed inverse proportionally to the number of updates. In the case of PT, we also varied the number of model samples collected to compute the expectation over the model distribution. We used 128 and 1024 samples, which is marked by PT-128 and PT-1024 in the experimental results.

The achieved classification accuracies are shown in Table 1. The best result of 98.49% was obtained with an RBM having 4000 hidden neurons, which was trained with the enhanced gradient by minimizing CD after fine-tuning the model. In the case of using PT, we observed that the models trained with the enhanced gradient outperformed those trained with the traditional gradient. However, the accuracies were lower than those obtained by the models trained by minimizing CD.

The RBM having 4000 hidden neurons, trained by computing the expectation over the model distribution with 1024 samples collected by PT sampling, achieved the highest log probability, $-64.06$. However, the models trained with 128 model samples were not able to perform as well as those with 1024 model samples. Furthermore, the RBM trained by minimizing CD with the enhanced gradient systematically had lower one-step

Table 1: Classification Accuracies of the Test Data of MNIST.

| | Before Fine-Tuning | | | | After Fine-Tuning | | | |
|---|---|---|---|---|---|---|---|---|
| | PT (128) | | CD | | PT (128) | | CD | |
| Hidden Neurons | Enhanced | Traditional | Enhanced | Traditional | Enhanced | Traditional | Enhanced | Traditional |
| 500 | 97.58 | 97.25 | 97.15 | 97.04 | 97.58 | 97.18 | 97.15 | 97.04 |
| 1000 | 97.93 | 97.78 | 98.02 | 97.82 | 97.94 | 97.78 | 98.09 | 97.83 |
| 2000 | 98.33 | 97.89 | 98.17 | 97.81 | 98.31 | 97.94 | 98.16 | 97.80 |
| 4000 | 98.41 | 97.82 | 98.48 | 98.11 | 98.36 | 97.83 | **98.49** | 98.10 |

Note: The highest accuracy is shown in boldface.

reconstruction errors of the test samples compared to those trained with the traditional gradient.

One noticeable phenomenon observed when the RBMs having a large number of hidden neurons, such as 2000 or 4000, were trained, was that the enhanced gradient used more hidden neurons than the traditional gradient did. It can be investigated by computing $\mathrm{Var}[p(h_j \mid \mathbf{v})]$, where $\mathbf{v}$ comes from test data. Those unused hidden neurons will have a variance close to 0, indicating that they are not dependent on whatever samples are clamped to the visible neurons. In Figure 9, it is clear that the models trained with the enhanced gradient have more hidden neurons with nonzero variances of the conditional probabilities.

**4.2 Experiments with Caltech 101 Silhouettes.** In this section, we test the proposed enhanced gradient on the Caltech 101 Silhouettes data (Marlin, Swersky, Chen, & de Freitas, 2010; random samples from this data set are shown in Figure 10). We ran four experiments with RBMs with 500, 1000, 2000, and 4000 hidden neurons. In order to avoid tuning the learning rate, we used the adaptive learning rate. Training was performed for 1000 epochs with PT and the minibatch size 128. The learning rate was initialized to 0.0001, and the upper bound was set to 0.1. After 500 epochs, the learning rate was decreased inverse proportionally to the number of updates.

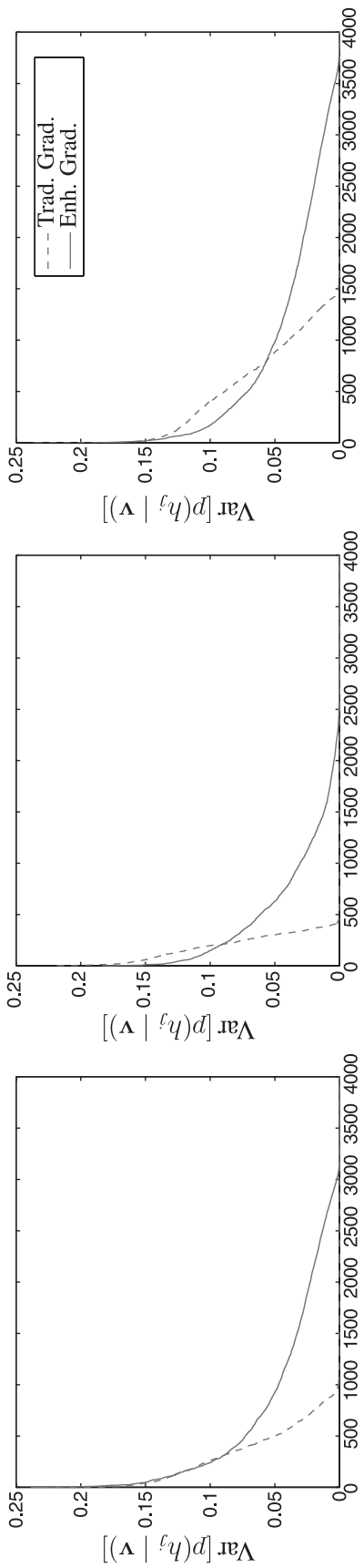
The obtained results are presented in Tables 2 to 5, where we used the same performance indexes as in section 4.1 to assess the quality of the results. Figure 10b shows samples drawn from the trained model. Remarkably, the classification accuracy improved by more than 5% over the best result reported by Marlin et al. (2010), with higher log probabilities when the enhanced gradient was used. The trend is more evident when PT was used to sample from the model distributions. However, we were not able to improve the accuracies with the discriminative fine-tuning.

We emphasize that we used the enhanced gradient with the adaptive learning rate without laborious tuning. In contrast, Marlin et al. (2010) used an extensive validation to choose the right learning rate and other metaparameters and also applied an advanced optimization method as a fine-tuning method to the initial stochastic gradient descent optimization.

## 5 Conclusion

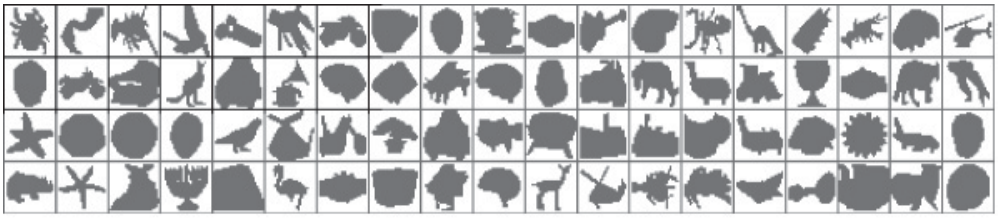
In this letter, we discussed several potential problems with training RBMs and proposed new update rules, based on the enhanced gradient. Unlike the traditional learning rules, which are dependent on data representation, the enhanced gradient was derived to be invariant to it. This allowed learning the flipped version of the MNIST data set without any difficulty.

The proposed gradient was compared against the traditional one using contrastive divergence, parallel tempering, and persistent contrastive divergence with plain Gibbs sampling. For MNIST, the classification accuracies obtained with the enhanced gradient were similar to the traditional one




(a) PT (128) (b) PT (1024) (c) CD

Figure 9: Variances of hidden activation probabilities of an RBM having 4000 hidden neurons given test samples of MNIST. The $x$-axis corresponds to each hidden neuron, and the $y$-axis represents the variance of the activation probabilities of the neuron given test samples. Hidden units are sorted by the variance of their conditional probabilities.



(a) Caltech 101 Silhouettes

(b) Generated samples

Figure 10: (a) One hundred randomly chosen training samples of Caltech 101 silhouettes. (b) Samples generated from the RBM with 2000 hidden neurons trained on Caltech 101 silhouettes.

Table 2: Log Probabilities of the Test Data of Caltech 101 Silhouettes by the RBMs Trained Using PT.

| | PT | | |
|---|---|---|---|
| Hidden Neurons | Enhanced | Traditional | (M) |
| 500 | −114.75 | −179.09 | ≈−120 |
| 1000 | −111.72 | −184.42 | |
| 2000 | −108.98 | −171.03 | |
| 4000 | −107.78 | −129.04 | |

Note: (M): Results reported by Marlin et al. (2010) using PCD.

Table 3: Log Probabilities and Reconstruction Errors of the Test Data of Caltech 101 Silhouettes by the RBMs Trained by Minimizing CD.

| | Log Probability | | | Reconstruction Error | |
|---|---|---|---|---|---|
| Hidden Neurons | Enhanced | Traditional | (M) | Enhanced | Traditional |
| 500 | −241.46 | −270.63 | $<-450$ | 0.0166 | 0.2447 |
| 1000 | −228.86 | −252.54 | | 0.0117 | 0.2362 |
| 2000 | −220.30 | −132.15 | | 0.0544 | 0.2481 |
| 4000 | −196.36 | −176.28 | | 0.0588 | 0.2489 |

Note: (M): Result reported by Marlin et al. (2010) using CD.

Table 4: Classification Accuracy of the Test Data of Caltech 101 Silhouettes.

| | PT | | | CD | | |
|---|---|---|---|---|---|---|
| Hidden Neurons | Enhanced | Traditional | (M) | Enhanced | Traditional | (M) |
| 500 | 70.91 | 67.92 | 65.4 | 69.14 | 70.22 | 64.9 |
| 1000 | 72.22 | 67.10 | | 70.61 | 70.83 | |
| 2000 | 72.17 | 69.57 | | 72.52 | 71.04 | |
| 4000 | 73.00 | 71.26 | | 72.35 | 69.09 | |

Note: (M): Results reported by Marlin et al. (2010) using PCD (left) and CD (right).

Table 5: Classification Accuracy of the Test Data of Caltech 101 Silhouettes After Discriminative Fine-Tuning.

| | PT | | | CD | | |
|---|---|---|---|---|---|---|
| Hidden Neurons | Enhanced | Traditional | (M) | Enhanced | Traditional | (M) |
| 500 | 70.83 | 67.75 | 65.4 | 68.96 | 70.05 | 64.9 |
| 1000 | 72.17 | 67.45 | | 70.57 | 70.74 | |
| 2000 | 72.17 | 69.14 | | 72.61 | 71.11 | |
| 4000 | 72.82 | 71.26 | | 72.39 | 69.09 | |

Note: (M): Results reported by Marlin et al. (2010) using PCD (left) and CD (right).

in the case of CD and significantly better in the case of PT and PCD. For 1-MNIST, the enhanced gradient clearly outperformed the traditional one. Our experiments also showed that the enhanced gradient is less sensitive to the choice of metaparameters than the traditional one, which makes training RBMs easier.

Finally, we note that the idea of the enhanced gradient can be applied to other types of Boltzmann machines. For example, we showed some preliminary experimental results on applying this idea to Boltzmann machines with gaussian visible neurons (Cho, Ilin et al., 2011; Cho, Raiko, & Ilin, 2011). Using the enhanced gradient for training other types of Boltzmann machines is a subject for future research.

## Appendix A: Bit-Flip Transformations

In this section, we present and prove two properties of the transformations 3.2 to 3.4.

**Theorem 1.** *The transformed machine is equivalent to the original one, that is,* $P(\tilde{\boldsymbol{x}} \mid \tilde{\boldsymbol{\theta}}) = P(\boldsymbol{x} \mid \boldsymbol{\theta})$ *for all states* $\boldsymbol{x}$.

**Proof.**

$$
\begin{aligned}
E(\tilde{\mathbf{x}} \mid \tilde{\boldsymbol{\theta}}) &= -\frac{1}{2}\tilde{\mathbf{x}}^{\top}\tilde{\mathbf{W}}\tilde{\mathbf{x}} - \tilde{\mathbf{b}}^{\top}\tilde{\mathbf{x}} \\
&= \sum_{ij} -\frac{1}{2} x_i^{1-f_i}(1-x_i)^{f_i}(-1)^{f_i+f_j} w_{ij} x_j^{1-f_j}(1-x_j)^{f_j} \\
&\quad - \sum_i (-1)^{f_i}\left(b_i + \sum_j f_j w_{ij}\right) x_i^{1-f_i}(1-x_i)^{f_i} \\
&= \sum_{ij} \frac{1}{2}(x_i - f_i) w_{ij} (x_j - f_j) - \sum_i \left(b_i + \sum_j f_j w_{ij}\right)(x_i - f_i) \\
&= \sum_{ij} -\frac{1}{2} x_i w_{ij} x_j - \sum_i b_i x_i + \sum_{ij} \frac{1}{2} x_i w_{ij} f_j + \sum_{ij} \frac{1}{2} x_i W_{ij} x_j \\
&\quad - \sum_{ij} \frac{1}{2} f_i w_{ij} f_j - \sum_{ij} f_j w_{ij} x_i + \sum_i \left(b_i + \sum_j f_j w_{ij}\right) f_i \\
&= \sum_{ij} -\frac{1}{2} x_i w_{ij} x_j - \sum_i b_i x_i \\
&\quad - \sum_{ij} \frac{1}{2} f_i w_{ij} f_j + \sum_i \left(b_i + \sum_j f_j w_{ij}\right) f_i \\
&= E(\mathbf{x} \mid \boldsymbol{\theta}) + \text{const.}
\end{aligned}
$$

Since the energy functions are the same up to a constant, the probability distributions P are the same.

## Appendix B: Transformation-Dependent Update Rules

Here we derive and show that the traditional gradient update rules are dependent on the bit-flipping transformation.

We define

$$
\begin{aligned}
\mathrm{Cov}_P(x_i, x_j) &= \langle x_i x_j \rangle_P - \langle x_i \rangle_P \langle x_j \rangle_P \\
\langle \cdot \rangle_{\mathrm{dm}} &= \frac{\langle \cdot \rangle_{\mathrm{d}} + \langle \cdot \rangle_{\mathrm{m}}}{2}
\end{aligned}
$$

and note that

$$\langle x_i\rangle_{\mathrm{dm}}\nabla b_j = \frac{1}{2}\langle x_i\rangle_{\mathrm{d}}\langle x_j\rangle_{\mathrm{d}} + \frac{1}{2}\langle x_i\rangle_{\mathrm{m}}\langle x_j\rangle_{\mathrm{d}} - \frac{1}{2}\langle x_i\rangle_{\mathrm{d}}\langle x_j\rangle_{\mathrm{m}} - \frac{1}{2}\langle x_i\rangle_{\mathrm{m}}\langle x_j\rangle_{\mathrm{m}}$$

to help get the form in equation B.1. The three-step update for the weights obtained by transforming, updating, and transforming back is

$$\begin{aligned}
w_{ij} &\leftarrow (-1)^{f_i+f_j}\left[(-1)^{f_i+f_j}w_{ij} + \eta(\langle\tilde{x}_i\tilde{x}_j\rangle_{\mathrm{d}} - \langle\tilde{x}_i\tilde{x}_j\rangle_{\mathrm{m}})\right] \\
&= w_{ij} + \eta(-1)^{f_i+f_j}\left[\left\langle x_i^{1-f_i}(1-x_i)^{f_i}x_j^{1-f_j}(1-x_j)^{f_j}\right\rangle_{\mathrm{d}}\right. \\
&\quad \left.-\left\langle x_i^{1-f_i}(1-x_i)^{f_i}x_j^{1-f_j}(1-x_j)^{f_j}\right\rangle_{\mathrm{m}}\right] \\
&= w_{ij} + \eta[\langle(x_i-f_i)(x_j-f_j)\rangle_{\mathrm{d}} - \langle(x_i-f_i)(x_j-f_j)\rangle_{\mathrm{m}}] \\
&= w_{ij} + \eta[\langle x_ix_j\rangle_{\mathrm{d}} - \langle x_ix_j\rangle_{\mathrm{m}} - f_i\nabla b_j - f_j\nabla b_i] \\
&= w_{ij} + \eta[\mathrm{Cov}_{\mathrm{d}}(x_i,x_j) - \mathrm{Cov}_{\mathrm{m}}(x_i,x_j) + (\langle x_i\rangle_{\mathrm{dm}} - f_i)\nabla b_j \\
&\quad +(\langle x_j\rangle_{\mathrm{dm}} - f_j)\nabla b_i]. \qquad \text{(B.1)}
\end{aligned}$$

We use a shorthand $\nabla_{\mathbf{f}}w_{ij} = \langle x_ix_j\rangle_{\mathrm{d}} - \langle x_ix_j\rangle_{\mathrm{m}} - f_i\nabla b_j - f_j\nabla b_i$ for the resulting gradient when using the transformation $\mathbf{f}$.

The three-step update for the bias is

$$\begin{aligned}
b_i &\leftarrow (-1)^{f_i}\left\{\tilde{b}_i + \eta(\langle\tilde{x}_i\rangle_{\mathrm{d}} - \langle\tilde{x}_i\rangle_{\mathrm{m}}) + \sum_j f_j[\tilde{w}_{ij} + \eta(\langle\tilde{x}_i\tilde{x}_j\rangle_{\mathrm{d}} - \langle\tilde{x}_i\tilde{x}_j\rangle_{\mathrm{m}})]\right\} \\
&= (-1)^{f_i}\left\{(-1)^{f_i}\left(b_i + \sum_j f_jW_{ij}\right) + \eta\left[\left\langle x_i^{1-f_i}(1-x_i)^{f_i}\right\rangle_{\mathrm{d}}\right.\right. \\
&\quad \left.\left.-\left\langle x_i^{1-f_i}(1-x_i)^{f_i}\right\rangle_{\mathrm{m}}\right] + \sum_j f_j(-1)^{f_i+f_j}[w_{ij} + \eta\nabla_{\mathbf{f}}w_{ij}]\right\} \\
&= b_i + \sum_j f_jw_{ij} + \eta(\langle x_i\rangle_{\mathrm{d}} - \langle x_i\rangle_{\mathrm{m}}) + \sum_j f_j(-w_{ij} - \eta\nabla_{\mathbf{f}}w_{ij}) \\
&= b_i + \eta\left(\nabla b_i - \sum_j f_j\nabla_{\mathbf{f}}w_{ij}\right). \qquad \text{(B.2)}
\end{aligned}$$

From equations B.1 and B.2, we note that the transformation vectors $\mathbf{f}$ do not cancel out, and thus we end up with different parameters after the update depending on the transformation.

## Appendix C: Enhanced Gradient

This appendix presents the derivation of the enhanced gradient update rules 3.6 to 3.8, starting from the transformation-dependent gradient update rules derived in the previous section.

The enhanced gradient is a weighted average of all possible transformation-dependent updates with different transformation vectors **f**. We choose the weights for the different updates inspired by equation B.1 to be

$$\prod_i \langle x_i \rangle_{\mathrm{dm}}^{f_i} (1 - \langle x_i \rangle_{\mathrm{dm}})^{1-f_i},$$

which sum up to one when considering all the exponentially many alternative transformation vectors **f**.

The enhanced gradient for the weights is derived as follows:

$$\begin{aligned}
\nabla_e w_{ij} = & \sum_{\mathbf{f}\in\{0,1\}^n} \left[ \prod_k \langle x_k \rangle_{\mathrm{dm}}^{f_k} (1 - \langle x_k \rangle_{\mathrm{dm}})^{1-f_k} \right] [\mathrm{Cov}_{\mathrm{d}}(x_i, x_j) - \mathrm{Cov}_{\mathrm{m}}(x_i, x_j) \\
& + (\langle x_i \rangle_{\mathrm{dm}} - f_i) \nabla b_j + (\langle x_j \rangle_{\mathrm{dm}} - f_j) \nabla b_i] \\
= & \sum_{f_i, f_j \in \{0,1\}} \prod_{k, i \neq k \neq j} [\langle x_k \rangle_{\mathrm{dm}} + (1 - \langle x_k \rangle_{\mathrm{dm}})] \\
& \times \langle x_i \rangle_{\mathrm{dm}}^{f_i} (1 - \langle x_i \rangle_{\mathrm{dm}})^{1-f_i} \langle x_j \rangle_{\mathrm{dm}}^{f_j} (1 - \langle x_j \rangle_{\mathrm{dm}})^{1-f_j} [\mathrm{Cov}_{\mathrm{d}}(x_i, x_j) \\
& - \mathrm{Cov}_{\mathrm{m}}(x_i, x_j) + (\langle x_i \rangle_{\mathrm{dm}} - f_i) \nabla b_j + (\langle x_j \rangle_{\mathrm{dm}} - f_j) \nabla b_i] \\
= & \sum_{f_i, f_j \in \{0,1\}} \langle x_i \rangle_{\mathrm{dm}}^{f_i} (1 - \langle x_i \rangle_{\mathrm{dm}})^{1-f_i} \langle x_j \rangle_{\mathrm{dm}}^{f_j} (1 - \langle x_j \rangle_{\mathrm{dm}})^{1-f_j} \\
& \times [\mathrm{Cov}_{\mathrm{d}}(x_i, x_j) - \mathrm{Cov}_{\mathrm{m}}(x_i, x_j) + (\langle x_i \rangle_{\mathrm{dm}} - f_i) \nabla b_j \\
& + (\langle x_j \rangle_{\mathrm{dm}} - f_j) \nabla b_i] \\
= & \mathrm{Cov}_{\mathrm{d}}(x_i, x_j) - \mathrm{Cov}_{\mathrm{m}}(x_i, x_j) \\
& + (1 - \langle x_i \rangle_{\mathrm{dm}})(1 - \langle x_j \rangle_{\mathrm{dm}}) [\langle x_i \rangle_{\mathrm{dm}} \nabla b_j + \langle x_j \rangle_{\mathrm{dm}} \nabla b_i] \\
& + \langle x_i \rangle_{\mathrm{dm}} (1 - \langle x_j \rangle_{\mathrm{dm}}) [(\langle x_i \rangle_{\mathrm{dm}} - 1) \nabla b_j + \langle x_j \rangle_{\mathrm{dm}} \nabla b_i] \\
& + (1 - \langle x_i \rangle_{\mathrm{dm}}) \langle x_j \rangle_{\mathrm{dm}} [\langle x_i \rangle_{\mathrm{dm}} \nabla b_j + (\langle x_j \rangle_{\mathrm{dm}} - 1) \nabla b_i] \\
& + \langle x_i \rangle_{\mathrm{dm}} \langle x_j \rangle_{\mathrm{dm}} [(\langle x_i \rangle_{\mathrm{dm}} - 1) \nabla b_j + (\langle x_j \rangle_{\mathrm{dm}} - 1) \nabla b_i] \\
= & \mathrm{Cov}_{\mathrm{d}}(x_i, x_j) - \mathrm{Cov}_{\mathrm{m}}(x_i, x_j).
\end{aligned}$$

For the bias term, it would be possible to derive similarly. However, rather than using a different weight update $\nabla_{\mathbf{f}}\mathbf{W}$ for each bias update, we use the enhanced gradient for the weights when doing each bias update, that is, replacing equation B.2 by $b_i \leftarrow b_i + \eta(\nabla b_i - \sum_j f_j \nabla_e w_{ij})$. Now we get the final formulation of the enhanced gradient, which was presented in equations 3.7 and 3.8, for the bias:

$$\begin{aligned}
\nabla_e b_i &= \sum_{\mathbf{f}\in\{0,1\}^n} \left[\prod_k \langle x_k\rangle_{\mathrm{dm}}^{f_k}(1-\langle x_k\rangle_{\mathrm{dm}})^{1-f_k}\right]\left(\nabla b_i - \sum_j f_j \nabla_e w_{ij}\right) \\
&= \nabla b_i - \sum_j \sum_{f_i, f_j\in\{0,1\}} \langle x_i\rangle_{\mathrm{dm}}^{f_i}(1-\langle x_i\rangle_{\mathrm{dm}})^{1-f_i}\langle x_j\rangle_{\mathrm{dm}}^{f_j}(1-\langle x_j\rangle_{\mathrm{dm}})^{1-f_j} f_j \nabla_e w_{ij} \\
&= \nabla b_i - \sum_j \langle x_j\rangle_{\mathrm{dm}}[(1-\langle x_i\rangle_{\mathrm{dm}})\nabla_e w_{ij} + \langle x_i\rangle_{\mathrm{dm}}\nabla_e w_{ij}] \\
&= \nabla b_i - \sum_j \langle x_j\rangle_{\mathrm{dm}}\nabla_e w_{ij}.
\end{aligned}$$

## Appendix D: Invariance of the Enhanced Gradient

**Theorem D.1.** *The enhanced gradient*

$$\begin{aligned}
\nabla_e w_{ij} &= \mathrm{Cov}_d(x_i, x_j) - \mathrm{Cov}_m(x_i, x_j) \\
\nabla_e b_i &= \nabla b_i - \sum_j \langle x_j\rangle_{dm}\nabla_e w_{ij}
\end{aligned}$$

*is invariant to the bit-flipping transformations as described in appendix A.*

**Proof.** We again compose a three-step update consisting of a transformation, update by enhanced gradient, and transformation back. If the resulting model is the same regardless of the transformation vector $\mathbf{f}$, we have proven the claim. The combined update for the weights is

$$\begin{aligned}
w_{ij} &\leftarrow (-1)^{f_i+f_j}[\tilde{w}_{ij} + \eta(\mathrm{Cov}_{\mathrm{d}}(\tilde{x}_i, \tilde{x}_j) - \mathrm{Cov}_{\mathrm{m}}(\tilde{x}_i, \tilde{x}_j))] \\
&= w_{ij} + (-1)^{f_i+f_j}\eta\big[\mathrm{Cov}_{\mathrm{d}}\big(x_i^{1-f_i}(1-x_i)^{f_i}, x_j^{1-f_j}(1-x_j)^{f_j}\big) \\
&\quad - \mathrm{Cov}_{\mathrm{m}}\big(x_i^{1-f_i}(1-x_i)^{f_i}, x_j^{1-f_j}(1-x_j)^{f_j}\big)\big] \\
&= w_{ij} + \eta[\mathrm{Cov}_{\mathrm{d}}(x_i, x_j) - \mathrm{Cov}_{\mathrm{m}}(x_i, x_j)].
\end{aligned}$$

The combined update for the bias is

$$
\begin{aligned}
b_i &\leftarrow (-1)^{f_i}\left[\tilde{b}_i + \eta\nabla_e\tilde{b}_i + \sum_j f_j(\tilde{w}_{ij} + \eta\nabla_e\tilde{w}_{ij})\right] \\
&= (-1)^{f_i}\Bigg\{(-1)^{f_i}\left(b_i + \sum_j f_j w_{ij}\right) + \eta\Bigg[\langle\tilde{x}_i\rangle_{\mathrm{d}} - \langle\tilde{x}_i\rangle_{\mathrm{m}} \\
&\quad - \sum_j \langle\tilde{x}_j\rangle_{\mathrm{dm}}(\langle\tilde{x}_i\tilde{x}_j\rangle_{\mathrm{d}} - \langle\tilde{x}_i\tilde{x}_j\rangle_{\mathrm{m}} - \langle\tilde{x}_j\rangle_{\mathrm{dm}}\langle\tilde{x}_i\rangle_{\mathrm{d}} + \langle\tilde{x}_j\rangle_{\mathrm{dm}}\langle\tilde{x}_i\rangle_{\mathrm{m}} \\
&\quad - \langle\tilde{x}_i\rangle_{\mathrm{dm}}\langle\tilde{x}_j\rangle_{\mathrm{d}} + \langle\tilde{x}_i\rangle_{\mathrm{dm}}\langle\tilde{x}_j\rangle_{\mathrm{m}})\Bigg] + \sum_j f_j[(-1)^{f_i+f_j}w_{ij} \\
&\quad + \eta(\langle\tilde{x}_i\tilde{x}_j\rangle_{\mathrm{d}} - \langle\tilde{x}_i\tilde{x}_j\rangle_{\mathrm{m}} - \langle\tilde{x}_i\rangle_{\mathrm{d}}\langle\tilde{x}_j\rangle_{\mathrm{d}} + \langle\tilde{x}_i\rangle_{\mathrm{m}}\langle\tilde{x}_j\rangle_{\mathrm{m}})]\Bigg\} \\
&= b_i + \eta\Bigg\{\langle x_i - f_i\rangle_{\mathrm{d}} - \langle x_i - f_i\rangle_{\mathrm{m}} - \sum_j[\langle x_j - f_j\rangle_{\mathrm{dm}}(\langle(x_i - f_i)(x_j - f_j)\rangle_{\mathrm{d}} \\
&\quad - \langle(x_i - f_i)(x_j - f_j)\rangle_{\mathrm{m}} - \langle x_j - f_j\rangle_{\mathrm{dm}}\langle x_i - f_i\rangle_{\mathrm{d}} + \langle x_j - f_j\rangle_{\mathrm{dm}}\langle x_i - f_i\rangle_{\mathrm{m}} \\
&\quad - \langle x_i - f_i\rangle_{\mathrm{dm}}\langle x_j - f_j\rangle_{\mathrm{d}} + \langle x_i - f_i\rangle_{\mathrm{dm}}\langle x_j - f_j\rangle_{\mathrm{m}}) \\
&\quad + f_j(\langle x_i x_j\rangle_{\mathrm{d}} - \langle x_i x_j\rangle_{\mathrm{m}} - \langle x_i\rangle_{\mathrm{d}}\langle x_j\rangle_{\mathrm{d}} + \langle x_i\rangle_{\mathrm{m}}\langle x_j\rangle_{\mathrm{m}})]\Bigg\} \\
&= b_i + \eta\Bigg\{\nabla b_i - \sum_j[(\langle x_j\rangle_{\mathrm{dm}} - f_j)(\nabla w_{ij} - f_j\nabla b_i - f_i\nabla b_j - \langle x_i\rangle_{\mathrm{dm}}\nabla b_j \\
&\quad + f_i\nabla b_j - \langle x_j\rangle_{\mathrm{dm}}\nabla b_i + f_j\nabla b_i) + f_j(\nabla w_{ij} - \langle x_i\rangle_{\mathrm{dm}}\nabla b_j - \langle x_j\rangle_{\mathrm{dm}}\nabla b_i)]\Bigg\} \\
&= b_i + \eta\Bigg[\nabla b_i - \sum_j(\langle x_j\rangle_{\mathrm{dm}}\nabla w_{ij} - \langle x_j\rangle_{\mathrm{dm}}\langle x_i\rangle_{\mathrm{dm}}\nabla b_j - \langle x_j\rangle_{\mathrm{dm}}^2\nabla b_i \\
&\quad - f_j\nabla w_{ij} + f_j\langle x_i\rangle_{\mathrm{dm}}\nabla b_j + f_j\langle x_j\rangle_{\mathrm{dm}}\nabla b_i \\
&\quad + f_j\nabla w_{ij} - f_j\langle x_i\rangle_{\mathrm{dm}}\nabla b_j - f_j\langle x_j\rangle_{\mathrm{dm}}\nabla b_i)\Bigg] \\
&= b_i + \eta\Bigg[\nabla b_i - \sum_j\langle x_j\rangle_{\mathrm{dm}}(\nabla W_{ij} - \langle x_i\rangle_{\mathrm{dm}}\nabla b_j - \langle x_j\rangle_{\mathrm{dm}}\nabla b_i)\Bigg] \\
&= b_i + \eta\left(\nabla b_i - \sum_j\langle x_j\rangle_{\mathrm{dm}}\nabla_e W_{ij}\right).
\end{aligned}
$$

## References


Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *2*, 1–127.

Bengio, Y., & Delalleau, O. (2009). Justifying and generalizing contrastive divergence. *Neural Comput.*, *21*, 1601–1621.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*, 281–305.

Carreira-Perpiñán, M. A., & Hinton, G. E. (2005). On contrastive divergence learning. In R. G. Cowell & Z. Ghahramani (Eds.), *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics* (pp. 33–40). Society for Artificial Intelligence and Statistics.

Cho, K., Ilin, A., & Raiko, T. (2011). Improved learning of Gaussian-Bernoulli restricted Boltzmann machines. In *Proceedings of the Twentieth International Conference on Artificial Neural Networks*. Berlin: Springer-Verlag.

Cho, K., Raiko, T., & Ilin, A. (2010). Parallel tempering is efficient for learning restricted Boltzmann machines. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 3246–3253). Piscataway, NJ: IEEE.

Cho, K., Raiko, T., & Ilin, A. (2011). Gaussian-Bernoulli deep Boltzmann machine. In *NIPS 2011 Workshop on Deep Learning and Unsupervised Feature Learning*, Sierra Nevada, Spain.

Desjardins, G., Courville, A., & Bengio, Y. (2010). Adaptive parallel tempering for stochastic maximum likelihood learning of RBMs. In *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning*.

Desjardins, G., Courville, A., Bengio, Y., Vincent, P., & Delalleau, O. (2009). *Tempered Markov chain Monte Carlo for training of restricted Boltzmann machines* (Tech. Rep. 1345). Montreal: Universitée de Montréal.

Desjardins, G., Courville, A., Bengio, Y., Vincent, P., & Delalleau, O. (2010). Parallel tempering for training of restricted Boltzmann machines. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (pp. 145–152). Cambridge, MA: MIT Press.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, *11*, 625–660.

Fischer, A., & Igel, C. (2010). Empirical analysis of the divergence of Gibbs sampling based learning algorithms for restricted Boltzmann machines. In *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III* (pp. 208–217). Berlin: Springer-Verlag.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Comput.*, *14*, 1771–1800.

Hinton, G. E. (2010). *A practical guide to training restricted Boltzmann machines* (Tech. Rep. 2010-003). Toronto: Department of Computer Science, University of Toronto.

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

Krizhevsky, A. (2010). *Convolutional deep belief networks on CIFAR-10* (Tech. Rep.). Toronto: Computer Science Department, University of Toronto.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document Recognition. In *Proceedings of the IEEE* (Vol. 86, pp. 2278–2324). Piscataway, NJ: IEEE.

Lee, H., Ekanadham, C., & Ng, A. (2008). Sparse deep belief net model for visual area V2. In J. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.), *Advances in neural information processing systems, 20* (pp. 873–880). Cambridge, MA: MIT Press.

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 609–616). New York: ACM.

Marlin, B. M., Swersky, K., Chen, B., & de Freitas, N. (2010). Inductive principles for restricted Boltzmann machine learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (pp. 509–516). Cambridge, MA: MIT Press.

Raiko, T., Cho, K., & Ilin, A. (2011). *Derivations on the enhanced gradient for the Boltzmann machine* (Tech. Rep.). Espoo: Aalto University School of Science, Department of Information and Computer Science.

Ranzato, M. A., Huang, F. J., Boureau, Y. L., & LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1–8). Los Alamitos, CA: IEEE.

Salakhutdinov, R. (2009). Learning in Markov random fields using tempered transitions. In Y. Bengio, D. Schuurmans, J. Lafferty, C.K.I. Williams, & A. Culotta (Eds.), *Advances in neural information processing systems, 22* (pp. 1598–1606). Red Hook, NY: Curran.

Salakhutdinov, R., & Hinton, G. E. (2009). Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics* (Vol. 5, pp. 448–455). Cambridge, MA: MIT Press.

Schulz, H., Müller, A., & Behnke, S. (2010). Investigating convergence of restricted Boltzmann machine learning. In *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning*.

Smolensky, P. (1986). Information processing in dynamical systems: foundations of harmony theory. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 1: Foundations* (pp. 194–281). Cambridge, MA: MIT Press.

Tieleman, T. (2008). *Training restricted Boltzmann machines using approximations to the likelihood gradient*. New York: ACM.

Younes, L. (1989). Parametric inference for imperfectly observed Gibbsian fields. *Probability Theory and Related Fields*, *82*, 625–645. doi: 10.1007/BF00341287

Received September 21, 2011; accepted August 13, 2012.

# Publication 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.**

# Enhanced Gradient and Adaptive Learning Rate for Training Restricted Boltzmann Machines

**KyungHyun Cho** KYUNGHYUN.CHO@AALTO.FI
**Tapani Raiko** TAPANI.RAIKO@AALTO.FI
**Alexander Ilin** ALEXANDER.ILIN@AALTO.FI
Department of Information and Computer Science, Aalto University School of Science, Finland

## Abstract

Boltzmann machines are often used as building blocks in greedy learning of deep networks. However, training even a simplified model, known as restricted Boltzmann machine (RBM), can be extremely laborious: Traditional learning algorithms often converge only with the right choice of the learning rate scheduling and the scale of the initial weights. They are also sensitive to specific data representation: An equivalent RBM can be obtained by flipping some bits and changing the weights and biases accordingly, but traditional learning rules are not invariant to such transformations. Without careful tuning of these training settings, traditional algorithms can easily get stuck at plateaus or even diverge. In this work, we present an enhanced gradient which is derived such that it is invariant to bit-flipping transformations. We also propose a way to automatically adjust the learning rate by maximizing a local likelihood estimate. Our experiments confirm that the proposed improvements yield more stable training of RBMs.

## 1. Introduction

Deep learning has gained its popularity recently as a way for learning complicated and large probabilistic models (see, e.g., Bengio, 2009). Especially, deep neural networks such as a deep belief network and a deep Boltzmann machine have been applied to various machine learning tasks with impressive improvements over conventional approaches (Hinton & Salakhutdinov, 2006; Salakhutdinov & Hinton, 2009; Salakhutdinov, 2009b).

Deep neural networks are characterized by the large number of layers of neurons and by using layer-wise unsupervised pretraining to learn a probabilistic model for the data. A deep neural network is typically constructed by stacking multiple restricted Boltzmann machines (RBM) so that the hidden layer of one RBM becomes the visible layer of another RBM. Layer-wise pretraining of RBMs then facilitates finding a more accurate model for the data. Various papers (Salakhutdinov & Hinton, 2009; Hinton & Salakhutdinov, 2006; Erhan et al., 2010) empirically confirmed that such multi-stage learning works better than conventional learning methods, such as the back-propagation with random initialization. It is thus important to have an efficient method for training RBM.

Unfortunately, training RBM is known to be difficult. Recent research suggests that without careful choice of learning parameters that are well suited to specific data sets and RBM structures, traditional learning algorithms may fail to model the data distribution correctly (Schulz et al., 2010; Fischer & Igel, 2010; Desjardins et al., 2010). This problem is often manifested in the fact that likelihood decreases during learning.

In this paper, we discuss the difficulties of training RBMs using the traditional gradient and propose a new training algorithm. The proposed improvements include an adaptive learning rate and a new enhanced gradient estimate. The adaptation rule for the learning rate is derived from maximizing a local approximation of the likelihood. The enhanced gradient is designed such that it does not contain the terms which often distract learning using the traditional gradient. The new gradient is also invariant to the data representation.

We conduct extensive experiments comparing the conventional learning algorithms with the proposed one. We use the MNIST handwritten digits data set (LeCun et al., 1998) and the Caltech 101 Silhouettes data set (Marlin et al., 2010) as benchmark problems. Some experiments were performed on the transformed MNIST data set in which each bit was flipped. We refer to this data set as 1-MNIST.

Appearing in *Proceedings of the $28^{th}$ International Conference on Machine Learning*, Bellevue, WA, USA, 2011. 

The data set 1-MNIST is known to be more difficult to learn, and we give an explanation fot this effect. The empirical results suggest that the new learning rules can avoid many difficulties in training RBMs.

## 2. Training Restricted Boltzmann Machines

RBM is a stochastic recurrent neural network consisting of binary neurons arranged in two layers (Smolensky, 1986). Each neuron $v_i$ in the visible layer is connected to all the hidden neurons, and each neuron $h_j$ in the hidden layer is connected to all the visible neurons. We denote by $\mathbf{v}$ a binary column vector containing the states $v_i$ of the visible neurons and similarly by $\mathbf{h}$ a vector of hidden states $h_j$.

The probability of a particular state $(\mathbf{v}, \mathbf{h})$ of the network is defined by the energy which is postulated to be

$$E(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta}) = -\mathbf{v}^T \mathbf{W} \mathbf{h} - \mathbf{b}^T \mathbf{v} - \mathbf{c}^T \mathbf{h} \tag{1}$$

where parameters $\boldsymbol{\theta}$ include weights $\mathbf{W} = [w_{ij}]$ and biases $\mathbf{b} = [b_i]$ and $\mathbf{c} = [c_j]$. Parameter $w_{ij}$ is the weight of the synaptic connections between neurons $v_i$ and $h_j$. The probability of a state $(\mathbf{v}, \mathbf{h})$ is

$$P(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left[-E(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})\right],$$

where $Z(\boldsymbol{\theta})$ is the normalizing constant.

### 2.1. Training

Maximum likelihood estimation of the parameters of RBM can be done using gradient-ascent update with learning rate $\eta$ and the following gradients:

$$\nabla w_{ij} = \langle v_i h_j \rangle_{\mathrm{d}} - \langle v_i h_j \rangle_{\mathrm{m}} \tag{2}$$

$$\nabla b_i = \langle v_i \rangle_{\mathrm{d}} - \langle v_i \rangle_{\mathrm{m}} \tag{3}$$

$$\nabla c_j = \langle h_j \rangle_{\mathrm{d}} - \langle h_j \rangle_{\mathrm{m}}. \tag{4}$$

We denote by $\langle \cdot \rangle_{\mathrm{d}}$ the expectation over the data, or in other words the distribution $P(\mathbf{h} \mid \{\mathbf{v}^{(t)}\}, \boldsymbol{\theta})$. Similarly, $\langle \cdot \rangle_{\mathrm{m}}$ denotes the expectation over the model distribution $P(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})$. We also use a shorthand notation $\langle \cdot \rangle_P$ which is the expectation over the probability distribution $P$.

A practical way to avoid computing the gradients exactly, which is not computationally feasible, is to use Markov-Chain Monte-Carlo (MCMC) sampling methods to compute the expectations $\langle \cdot \rangle_{\mathrm{m}}$ approximately. The restricted structure of RBM makes Gibbs sampling efficient in drawing samples from $P(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})$: Given one layer, either visible or hidden, the neurons in the other layer are mutually independent. This makes it possible to sample from the whole layer at once.

Training is typically done using only a subset of data examples for computing the expectations $\langle \cdot \rangle_{\mathrm{d}}$ on each iteration. This subset of training data is usually called mini-batch.

#### 2.1.1. Contrastive Divergence

Contrastive divergence (CD) learning (Hinton, 2002) approximates the true gradient in (2)–(4) by computing $\langle \cdot \rangle_{\mathrm{m}}$ using samples obtained after running $n$ steps of Gibbs sampling starting from each data sample of the corresponding mini-batch. The CD gradient for the weights is approximated as

$$\nabla w_{ij} \approx \langle v_i h_j \rangle_{\mathrm{d}} - \langle v_i h_j \rangle_{P_n} \tag{5}$$

where $P_n$ denotes the distribution after $n$ steps of Gibbs sampling. Even though CD learning is known to be biased (Carreira-Perpiñán & Hinton, 2005), it has been proven to work well in practice.

#### 2.1.2. Parallel Tempering

Parallel tempering (PT) sampling was recently proposed to replace Gibbs sampling for estimating $\langle \cdot \rangle_{\mathrm{m}}$ (Desjardins et al., 2010; Cho et al., 2010). The basic idea of PT is that multiple chains of Gibbs sampling are run for models with different "temperatures". Every now and then, the samples are swapped between the chains. Chains with higher temperatures correspond to more diffuse distributions and therefore they can produce a greater variety of samples. This facilitates better exploration of the state space.

In this paper, we use PT with a set of $N$ inverse temperatures $0 < \beta_2 < \cdots < \beta_{N-1} < 1$. Temperature $\beta_N = 1$ corresponds to the current RBM model with parameters $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b}, \mathbf{c})$. Smaller values $\beta_i$ correspond to less restricted models with parameters $\boldsymbol{\theta}_i = (\beta_i \mathbf{W}, \beta_i \mathbf{b}, \beta_i \mathbf{c})$. Thus $\beta_1 = 0$ corresponds to the most diffuse distribution.

We run separate $N$ chains for each sample in the mini-batch. After every step in the chains, swaps are proposed and accepted according to the Metropolis rule. The expectations $\langle \cdot \rangle_{\mathrm{m}}$ are computed from the samples of the chain with $\beta = 1$.

### 2.2. Annealed Importance Sampling

It is desirable to know the actual value of the likelihood which is optimized during learning. If the normalizing constant is known, computing the likelihood is straightforward.

*Annealed importance sampling* (AIS) provides a way to estimate the normalizing constant of RBM (Salakhutdinov, 2009a). AIS is based on *simple importance sampling* (SIS), which uses the fact that the ratio of two normalizing constants for two probability densities $P_A(\mathbf{v}) = P_A^*(\mathbf{v})/Z_A$ and $P_B(\mathbf{v}) = P_B^*(\mathbf{v})/Z_B$ can be computed as:

$$\frac{Z_B}{Z_A} = \left\langle \frac{P_B^*(\mathbf{v})}{P_A^*(\mathbf{v})} \right\rangle_{P_A}. \tag{6}$$

In SIS, (6) is estimated using samples from $P_A(\mathbf{v})$.

Using this idea, AIS estimates the normalizing constant of the model distribution by computing the ratio of the normalizing constants of consecutive intermediate distributions ranging from so-called base distribution and the target distribution.

### 2.3. Difficulties in Training RBMs

The fact that the objective function is very costly to estimate makes training RBM difficult. It is difficult to determine how well the learning is progressing. Furthermore, it is not possible to use advanced optimization methods such as conjugate gradient or even line-search.

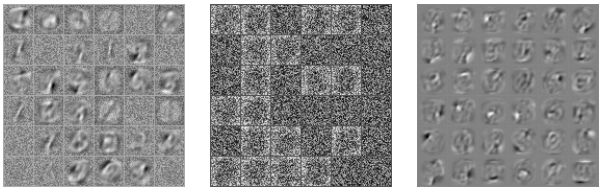
Learning is performed using stochastic gradient, and it converges to a local solution. It is generally not feasible to compare different local optima analytically. Schulz et al. (2010) and Fischer & Igel (2010) recently showed that depending on initialization and learning parameters the resulting RBMs can highly vary even for a small data set.

Furthermore, most learning algorithms discussed in the previous section can diverge if the learning parameters are not chosen appropriately (Desjardins et al., 2010; Schulz et al., 2010; Fischer & Igel, 2010). The use of advanced MCMC sampling methods such as PT has been shown to avoid divergence but the likelihood can highly fluctuate in the long run without using the appropriate learning rate scheduling (Desjardins et al., 2010; 2009).

One way to analyze the quality of a trained model is to look at the features (the weights $w_{ij}$) and the bias terms $c_j$ corresponding to different hidden neurons $h_j$. Neurons that have a large bias $c_j$ are most of the time active. They are not very useful because the weights associated to them can be incorporated into the bias term $\mathbf{b}$. Other neurons (e.g. with large negative biases $c_j$) can be always inactive or there can be neurons (with weights $w_{ij}$ close to zero) whose activations are independent of data. Such hidden neurons are also useless because they do not contribute to the modeling capacity of RBM.

Ideally, each hidden neuron should represent a distinct "meaningful" feature, for example, a typical part of an image. We have noticed, however, that very often the hidden neurons tend to learn features that resemble the visible bias term $\mathbf{b}$. This effect is more prominent at the initial stage of learning and for data set in which visible bits are mostly active, such as 1-MNIST.

Fig. 1(a)-(b) show the weights $\mathbf{W}$ of RBM with 36 hidden neurons trained using the traditional gradient (2)–(4) on MNIST and 1-MNIST with the constant learning rate 0.1 and weights initialized randomly from $\begin{bmatrix}-1 & 1\end{bmatrix}$. The features learned from MNIST look quite good, even though there are some useless neurons. However, the features learned from 1-MNIST are clearly bad: 18 hidden neurons



(a) (b) (c)

*Figure 1.* Visualization of filters learned after five epochs by RBM with 36 hidden neurons. (a) Traditional gradient, MNIST. (b) Traditional gradient, 1-MNIST. (c) Proposed algorithm (Section 3), 1-MNIST

are mostly active and represent global features that somewhat resemble the visible bias, the other 18 neurons are mostly inactive and hence useless.

There is a number of well-known heuristics proposed to improve the training results. They include proper scheduling of the learning rate, weight decay prior for the weights, adding momentum terms to the gradients, and forcing sparsity of the hidden activations. These heuristics are known to help in many practical applications, however, with extra parameters which should be selected very carefully. Good values of these parameters are typically found by trial and error and it seems that one requires a lot of experience to set the learning settings right (Hinton, 2010).

## 3. Improved Training Algorithm

This section describes the two novel contributions.

### 3.1. Adaptive Learning Rate

Here we propose an algorithm for automatically adapting the learning rate while training RBM using stochastic gradient. The automatic adaptation of the learning rate is based on maximizing the local estimate of the likelihood.

Let $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b}, \mathbf{c})$ be the current model, $\boldsymbol{\theta}' = (\mathbf{W}', \mathbf{b}', \mathbf{c}')$ is the updated model with some learning rate $\eta$ and $P_{\boldsymbol{\theta}}(\mathbf{v}) = P^*_{\boldsymbol{\theta}}(\mathbf{v})/Z_{\boldsymbol{\theta}}$ is the pdf with normalizing constant $Z_{\boldsymbol{\theta}}$ for the model with parameters $\boldsymbol{\theta}$. Now if we assume that the learning rate is small enough and therefore the two models are close to each other, the likelihood of $\boldsymbol{\theta}'$ can be computed as in SIS using (6):

$$P_{\boldsymbol{\theta}'}(\mathbf{v}_{\mathrm{d}}) = \frac{P^*_{\boldsymbol{\theta}'}(\mathbf{v}_{\mathrm{d}})}{Z_{\boldsymbol{\theta}}} \frac{Z_{\boldsymbol{\theta}}}{Z_{\boldsymbol{\theta}'}} = \frac{P^*_{\boldsymbol{\theta}'}(\mathbf{v}_{\mathrm{d}})}{Z_{\boldsymbol{\theta}}} \left\langle \frac{P^*_{\boldsymbol{\theta}'}(\mathbf{v})}{P^*_{\boldsymbol{\theta}}(\mathbf{v})} \right\rangle^{-1}_{P_{\boldsymbol{\theta}}}, \quad (7)$$

where $\mathbf{v}_{\mathrm{d}}$ denotes the training data. In practice, we use samples from the next mini-batch for $\mathbf{v}_{\mathrm{d}}$. [1].

[1] Our experiments showed that if the same samples were used both for obtaining the gradients and the adaptive learning rate, learning rate fluctuated too much in case of PT learning and diverged in case of CD learning .

Now we would like to select a learning rate so as to maximize the likelihood of the new parameters $\boldsymbol{\theta}'$. Equation (7) can be used to approximate the required likelihood. The unnormalized pdf $P^*_{\boldsymbol{\theta}'}$ is computed using the training samples and (1), and the expectation $\langle\cdot\rangle_{P_{\boldsymbol{\theta}}}$ can be estimated using the samples from $P_{\boldsymbol{\theta}}$, like in SIS. These samples are collected in order to estimate the negative term in (5) and therefore computing this expectation can be done practically for free.

In principle, one could find the optimal learning rate that maximizes the local estimate of the likelihood on each iteration. However, this would likely lead to large fluctuations of the learning rate because of the small sample size (mini-batch). In our experiments, we selected the new learning rate from the set $\{(1-\epsilon)^2\eta_0, (1-\epsilon)\eta_0, \eta_0, (1+\epsilon)\eta_0, (1+\epsilon)^2\eta_0\}$, where $\eta_0$ is the previous learning rate and $\epsilon$ is a small constant.

### 3.2. Enhanced Gradient

In this section, we propose a new gradient to be used instead of (2)–(4). Let us first define the covariance between two variables under distribution $P$

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

We can rewrite the standard gradient (2) as

$$\begin{aligned}\nabla w_{ij} = {} & \mathrm{cov}_{\mathrm{d}}\left(v_i, h_j\right) - \mathrm{cov}_{\mathrm{m}}\left(v_i, h_j\right) \\ & + \langle v_i\rangle_{\mathrm{dm}} \nabla c_j + \langle h_j\rangle_{\mathrm{dm}} \nabla b_i , \end{aligned} \tag{8}$$

where $\langle\cdot\rangle_{\mathrm{dm}} = \frac{1}{2}\langle\cdot\rangle_{\mathrm{d}} + \frac{1}{2}\langle\cdot\rangle_{\mathrm{m}}$ is the average activity of neuron under the data and model distributions.

The standard gradient (8) has several potential problems. The gradients w.r.t. the weights contain the terms that point to the same direction as the gradient w.r.t. the bias terms (and vice versa). This effect is prominent when there are many neurons which are mainly active, that is for which $\langle\cdot\rangle_{\mathrm{dm}} \approx 1$. These terms can distract learning of meaningful weights, which often leads to the case when many neurons try to learn features resembling the bias terms, as shown in Fig. 1(b).

When $\langle\cdot\rangle_{\mathrm{dm}} \approx 0$ for most of the neurons, this effect can be negligible, which might explain why learning 1-MNIST is more difficult than MNIST and partially explain why sparse Boltzmann machines (Lee et al., 2008), which ensure that the average activation of a hidden neuron is kept at low level, have been successful.

A related problem is that the update using (8) is different depending on the data representation. This can be shown by using transformations where some of the binary units of RBM are flipped such that zeros become ones and vice versa:

$$\begin{aligned}\tilde{v}_i &= v_i^{1-f_i^{(v)}}(1-v_i)^{f_i^{(v)}}, \qquad f_i^{(v)} \in \{0,1\}, \\ \tilde{h}_j &= h_j^{1-f_j^{(h)}}(1-h_j)^{f_j^{(h)}}, \qquad f_j^{(h)} \in \{0,1\}.\end{aligned}$$

The parameters can then be transformed accordingly to $\tilde{\boldsymbol{\theta}}$

$$\begin{aligned}\tilde{w}_{ij} &= (-1)^{f_i^{(v)}+f_j^{(h)}} w_{ij} \\ \tilde{b}_i &= (-1)^{f_i^{(v)}}\left(b_i + \sum_j f_j^{(h)} w_{ij}\right) \\ \tilde{c}_j &= (-1)^{f_j^{(h)}}\left(c_j + \sum_i f_i^{(v)} w_{ij}\right),\end{aligned}$$

such that the resulting RBM has an equivalent energy function, that is $E(\tilde{\mathbf{x}} \mid \tilde{\boldsymbol{\theta}}) = E(\mathbf{x} \mid \boldsymbol{\theta}) + \text{const}$ for all $\mathbf{x}$. When a model is transformed, updated, and transformed back, the resulting model depends on the transformations:

$$\begin{aligned}w_{ij} \leftarrow {} & w_{ij} + \eta\Big[\mathrm{cov}_{\mathrm{d}}\left(v_i, h_j\right) - \mathrm{cov}_{\mathrm{m}}\left(v_i, h_j\right) \\ & + \left(\langle v_i\rangle_{\mathrm{dm}} - f_i^{(v)}\right)\nabla c_j + \left(\langle h_j\rangle_{\mathrm{dm}} - f_j^{(h)}\right)\nabla b_i\Big] \end{aligned} \tag{9}$$

$$b_i \leftarrow b_i + \eta\Big[\nabla b_i - \sum_j f_j^{(h)}\left(\nabla w_{ij} - f_i^{(v)}\nabla c_j - f_j^{(h)}\nabla b_i\right)\Big]$$

$$c_j \leftarrow c_j + \eta\Big[\nabla c_j - \sum_i f_i^{(v)}\left(\nabla w_{ij} - f_i^{(v)}\nabla c_j - f_j^{(h)}\nabla b_i\right)\Big],$$

where $\nabla\boldsymbol{\theta}$ are the gradients defined in Eqs. (2)–(4).

We have thus $2^{n_v+n_h}$ different update rules defined by different combinations of binary $f_i^{(v)}$ and $f_j^{(h)}$, $i = 1, \ldots, n_v$ and $j = 1, \ldots, n_h$, where $n_v$, $n_h$ are the number of visible and hidden neurons, respectively. All the update rules are well-founded maximum likelihood updates to the original model. We propose to use as the new gradient a weighted sum of the $2^{n_v+n_h}$ gradients with the following weights:

$$\prod_i \langle v_i\rangle_{\mathrm{dm}}^{f_i^{(v)}}\left(1-\langle v_i\rangle_{\mathrm{dm}}\right)^{1-f_i^{(v)}} \prod_j \langle h_j\rangle_{\mathrm{dm}}^{f_j^{(h)}}\left(1-\langle h_j\rangle_{\mathrm{dm}}\right)^{1-f_j^{(h)}} \tag{10}$$

By using these weights we prefer sparse data representations for which $\langle\cdot\rangle_{\mathrm{dm}} \approx 0$ because the corresponding models get larger weights.

The proposed weighted sum yields the enhanced gradient

$$\tilde{\nabla} w_{ij} = \mathrm{cov}_{\mathrm{d}}\left(v_i, h_j\right) - \mathrm{cov}_{\mathrm{m}}\left(v_i, h_j\right) \tag{11}$$

$$\tilde{\nabla} b_i = \langle v_i\rangle_{\mathrm{d}} - \langle v_i\rangle_{\mathrm{m}} - \sum_j \langle h_j\rangle_{\mathrm{dm}} \tilde{\nabla} w_{ij} \tag{12}$$

$$\tilde{\nabla} c_j = \langle h_j\rangle_{\mathrm{d}} - \langle h_j\rangle_{\mathrm{m}} - \sum_i \langle v_i\rangle_{\mathrm{dm}} \tilde{\nabla} w_{ij}, \tag{13}$$

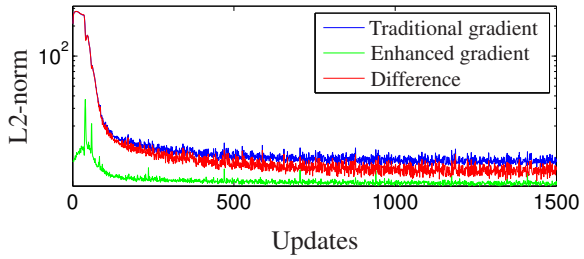



*Figure 2.* L2-norms of the gradients for weights during the learning of a RBM with 361 hidden neurons. The blue curve plots the norms of the traditional gradient, and the green curve plots the norms of the proposed robust gradient. The norms of the difference between two gradients are drawn with the red curve.

in which, by the choice of the weights (10), the effect of the bias gradients in $\nabla w_{ij}$ is canceled out. It can be shown that the new rules are invariant to the bit-flipping transformations. One can also note that the enhanced gradient shares all zeroes with the traditional gradient.

In Figs. 2–3, we present some experimental analysis of the proposed gradient. Fig. 2 shows the norms of the gradient for the weights of an RBM with 361 hidden neurons trained on MNIST data set. It is clear that the additional terms that distract learning dominate in the traditional gradient, especially at the early stage of training.

Fig. 3 shows the differences in the update directions for different neurons of an RBM trained on MNIST. Each element of a matrix is the absolute value of the cosine of the angle between the update directions for the two neurons. The gradients obtained by the traditional rule are highly correlated to each other, especially, at the early stage of learning. On the contrary, the new gradient yields update directions that are close to orthogonal, which allows the neurons to learn distinct features.

## 4. Experiments

In this section, we experimentally compare the proposed improvements to the traditional learning algorithms. In Sections 4.1–4.3, RBMs are trained on the MNIST data set, and in Section 4.4, we use the Caltech 101 Silhouettes data.

We run 20 epochs with a mini-batch size of 128 unless otherwise mentioned. Thus, each RBM was updated about 4,700 times. Both biases $\mathbf{b}$ and $\mathbf{c}$ of an RBM were initialized to all zeros. Weights were randomly initialized such that $w_{ij} = \lambda \cdot u$ where $\lambda$ is a weight scale and $u \sim \mathcal{U}(-1, 1)$ denotes a sample from the uniform random variable from $-1$ to 1. By default, we used $\lambda = 1/\sqrt{n_v + n_h}$.

For PT learning, we used 11 different inverse temperatures equally spaced from $\beta_1 = 0$ to $\beta_{11} = 1$. For CD learning, we used $n = 1$ steps of Gibbs sampling. For each setting, RBMs were independently trained with five different

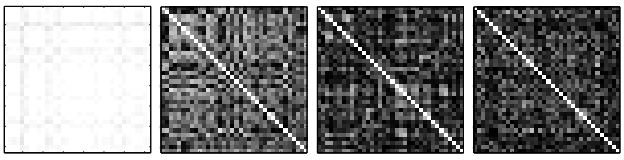


*Figure 3.* The angles between the update directions for the weights of RBM with 36 hidden neurons. White pixels correspond to small angles, while black pixels correspond to orthogonal directions. From left to right: traditional gradient after 26 updates, traditional gradient after 352 updates, enhanced gradient after 26 updates, and enhanced gradient after 352 updates.

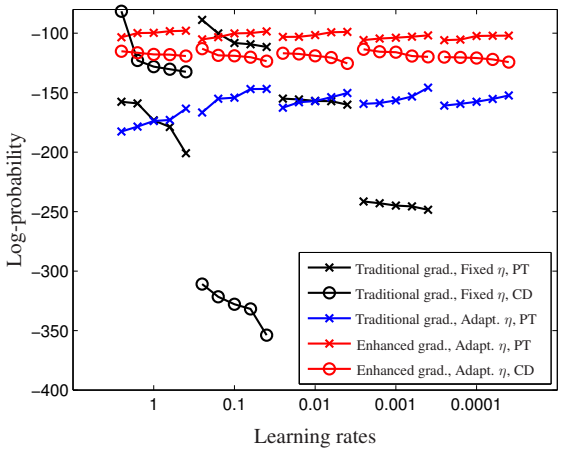



*Figure 4.* Log-probabilities of test data samples computed after 20 epochs for five runs with different initializations for different learning rates. Log-probabilities that do not appear on the plot are smaller than $-400$. The order of the points is arbitrary, and they were sorted in order to make the curves more discriminate.

initializations of parameters. After training, the normalizing constant of each model was estimated using AIS and the log-probability of the test data was computed. We used $\boldsymbol{\theta}_i = (\beta_i \mathbf{W}, \beta_i \mathbf{b}, \beta_i \mathbf{c})$ and 10,001 equally-spaced temperatures. Each estimate of $Z(\boldsymbol{\theta})$ was averaged over 100 independent AIS runs.

### 4.1. Sensitivity to Learning Rate

In order to demonstrate how the learning rate can greatly affect training results, we trained RBMs with 361 hidden neurons using the traditional gradient with five different learning rates $\{1, 0.1, 0.01, 0.001, 0.0001\}$. The black curves in Fig. 4 show the log-probability of the test data obtained with PT and CD sampling strategies. It is clear that the resulting RBMs have huge variance depending on the choice of the learning rate. Too small learning rate prevents RBM from learning barely anything, whereas too large learning rate often results in models which are worse than those RBMs trained with proper learning rates. In case of using a learning rate 10, the learning failed completely.

In order to test the proposed adaptive learning rate,

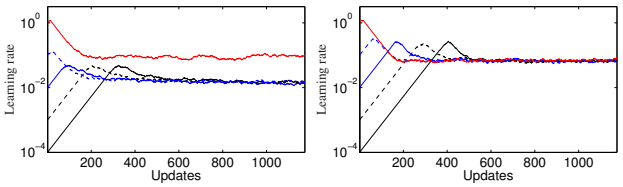



*Figure 5.* Evolution of the adaptive learning rate from five different initializations during learning. The learning rates are shown as a function of the number of updates. RBMs are trained with the traditional gradient (left) and the robust gradient (right).

we trained RBMs with 361 hidden neurons using the traditional gradient and the same five values $\{1, 0.1, 0.01, 0.001, 0.0001\}$ to initialize the learning rate. The blue curves in Fig. 4 show the obtained log-probabilities of the test data. The results are now more stable and the variance among the resulting RBMs is smaller compared to the results obtained with fixed learning rates (the black curves in the same figure). Regardless of the initial learning rate, all RBMs were trained quite well.

These results suggest that the adaptive learning rate works well. However, it was still slightly better to use manually tuned training parameters (using the constant learning rate of 0.1).

Fig. 5 shows the evolution of the learning rate during learning. Even with very small initial learning rate, the adaptive learning rate could find the appropriate learning after only a few hundred updates. Remarkably, the learning rates converge to the same value when the enhanced gradient is used.

The red curves in Fig. 4 show the log-probabilities of the test data obtained with the new gradient and the adaptive learning rate initialized with five different values. Both PT and CD sampling were tried. It is apparent that the enhanced gradient improves the overall learning performance compared to the traditional gradient. Similar performance was obtained on 1-MNIST (the results are not shown here) because the new gradient is invariant to data representation.

### 4.2. RBM as Feature Extractor

In addition to the log-probabilities of the test data, we trained simple logistic regression classifiers on top of the RBMs to check their feature extracting performance. The activation probabilities of the hidden neurons were used as the features. In order not to destroy the already learned structure of the RBM, no discriminative fine-tuning was performed. This explains why the accuracies reported in this paper are far from the state-of-the-art accuracy on MNIST using deep neural networks (Salakhutdinov, 2009b).

The black curves in Fig. 6 show high variance of the classification results for the traditional gradient depending on the chosen learning rate. The results obtained for MNIST (the upper plot) are pretty good although the choice of the learning does have an effect on performance. However, the classification accuracy obtained for 1-MNIST is very bad, which proves that 1-MNIST is more difficult for training using the traditional gradient.

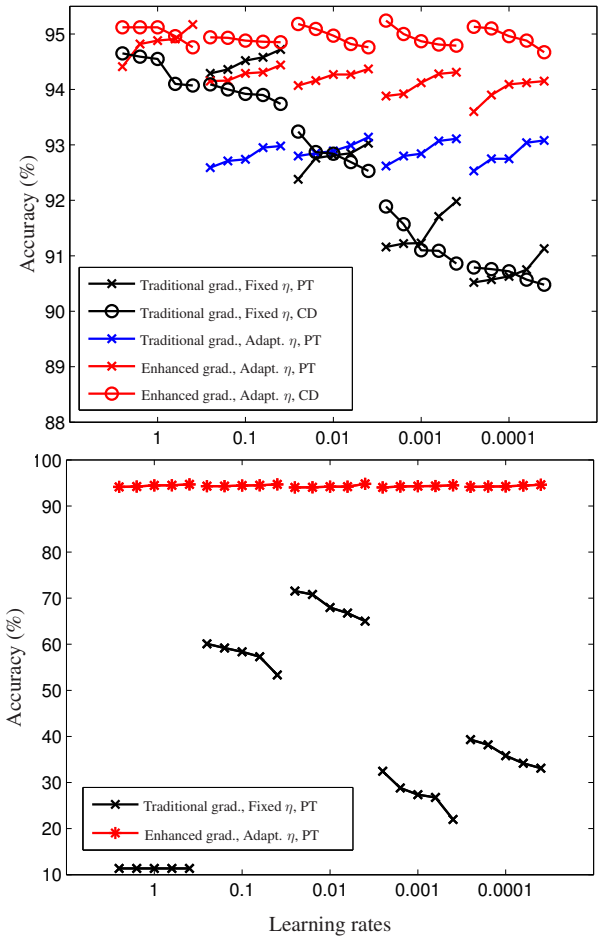



*Figure 6.* Classification accuracy of test data samples computed after 20 epochs for MNIST (above) and 1-MNIST (below). For each initial learning rate, the learning was conducted five times. The results that do not appear on the upper plot were below 88%. The order of the points is arbitrary, and they were sorted in order to make the curves more discriminate.

The blue curves in Fig. 6 show that the adaptive learning rate can reduce the variance of the results obtained with the traditional gradient. However, the results were quite significantly worse for the initial learning rate 1.

The red curves in Fig. 6 shows the superior performance of the enhanced gradient and the adaptive learning rate compared to the traditional gradient. Regardless of the initial learning rate, all RBMs leaned features which yielded high classification performance. Note that the results are excellent also for 1-MNIST.

### 4.3. Sensitivity to Weight Initialization

In the next experiment, we test the sensitivity of training results to the scale of the weight initialization. We trained small RBMs with 36 hidden neurons on MNIST using different scales of the initial weights and varying learning rates. PT sampling was used to draw model samples from RBM.

The plot on the left in Fig. 7 visualizes the filters learned by RBMs using the traditional gradient with fixed learning rate. It is clear that the results are highly dependent on the choice of the training parameters: The combination of the initial weight scale and the learning rate should be selected very carefully in order to learn reasonable features. The combination of learning rate $\eta = 0.1$ and weight scale $\lambda = 0.1$ seems to give the best results for the reported experiments. In practice, an optimal combination of the training parameters is usually found by trial and error, which makes training a laborious procedure.

The plot on the right in Fig. 7 shows the filters learned using the new gradient and the adaptive learning rate initialized with five different values. It is clear that the features are much better than the ones obtained with the traditional gradient. Remarkably, no hidden neuron is either dead or always active regardless of the scale of the initial weights and the choice of the initial learning rate.

### 4.4. Caltech 101 Silhouettes

Finally, we tested the proposed learning rules on Caltech 101 Silhouettes data set. RBMs with 500, 1000, and 2000 hidden neurons were trained using the proposed algorithm for 300 epochs with the mini-batch size set to 256. The learning rate was initialized to $0.0001$.

The obtained results are presented in Table 1. Remarkably, the classification accuracy improved by more than 5 % over the best result reported by Marlin et al. (2010).

| Hidden neurons | Log-probability | Accuracy (%) |
|---|---|---|
| 500 | -127.40, -280.91 | 71.56, 68.48 |
| 1000 | -129.69, -190.80 | 72.61, 70.39 |
| 2000 | -131.19, -166.72 | 71.82, 71.39 |

*Table 1.* Log-probabilities and classification accuracies of the test data of Caltech 101 Silhouettes after 300 epochs. First numbers were obtained by PT learning, and the following numbers were by CD learning.

## 5. Discussion

The paper discussed the main difficulties of training RBMs and their underlying reasons. The main reason is that RBMs are best trained using approximate stochastic gradient updates, which leads to high variance in resulting RBMs and possibly diverging behavior. The second reason is that many learning hyperparameters, e.g. learning rate scheduling, have to be carefully and manually chosen depending on the structure of the trained RBMs and the properties of the training data set. Additionally, a problem of having meaningless hidden neurons in RBMs during learning has been demonstrated and discussed.

We proposed a new algorithm for RBM training that addresses the above difficulties. It consists of an adaptive learning rate and an enhanced gradient, and is formulated with well-founded theoretical background. The enhanced gradient could overcome the problem of having hidden neurons learning the near-identical features and was able to speed up the overall learning significantly. Also, unlike the traditional gradient rules which were dependent on the sparsity of the data samples, the enhanced gradient was derived to be invariant to the sparsity could successfully learn very dense data set without any difficulty.

The paper mainly focused on parallel tempering learning, but we also showed that contrastive divergence learning is also improved by adopting the proposed improvements. Our future work will apply the proposed methods to other models in the Boltzmann family, such as deep BMs and Gaussian-Bernoulli BMs.

## Acknowledgments

This work was supported by the honours programme of the department, by the Academy of Finland and by the IST Program of the European Community, under the PASCAL2 Network of Excellence. The authors would like to thank Ilya Sutskever of University of Toronto, Andreas Müller and Hannes Schulz of University of Bonn.

## References

Bengio, Y. Learning Deep Architectures for AI. *Found. Trends Mach. Learn.*, 2:1–127, January 2009.

Carreira-Perpiñán, M. A. and Hinton, G. E. On Contrastive Divergence Learning. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, Jan 6-8, 2005, Savannah Hotel, Barbados*, pp. 33–40. Society for Artificial Intelligence and Statistics, 2005.

Cho, K., Raiko, T., and Ilin, A. Parallel Tempering is Efficient for Learning Restricted Boltzmann Machines. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2010)*, Barcelona, Spain, July 2010.

Desjardins, G., Courville, A., Bengio, Y., Vincent, P., and Delalleau, O. Tempered Markov Chain Monte Carlo for

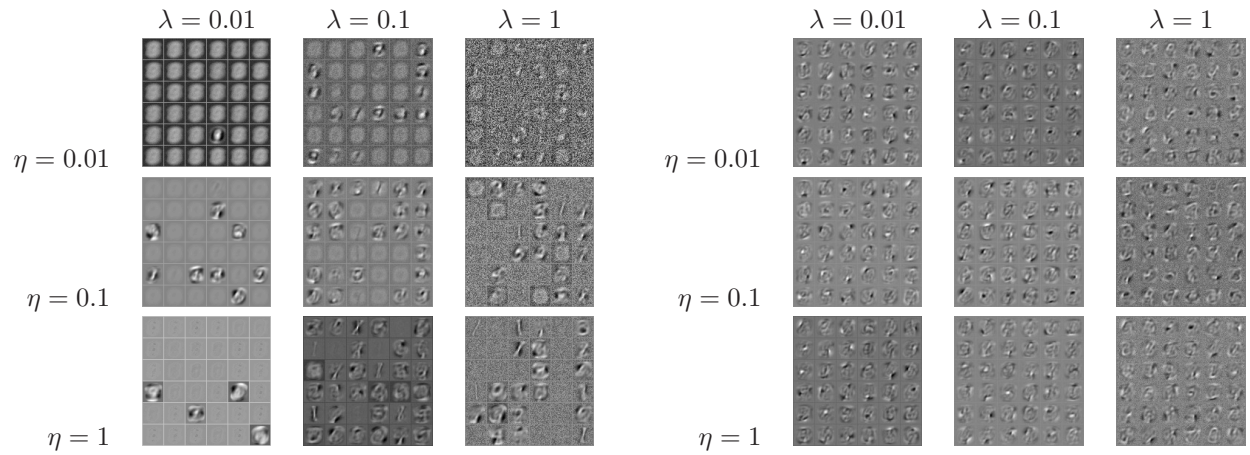



*Figure 7.* Visualization of filters learned by RBMs with 36 hidden neurons on MNIST with various initial learning rates and initial weights scaling. Left: using the traditional gradient with fixed learning rate, right: using the enhanced gradient with adaptive learning rate. Learning was performed for 5 epochs each.

training of Restricted Boltzmann Machines. Technical Report 1345, Universitée de Montréal, 2009.

Desjardins, G., Courville, A., Bengio, Y., Vincent, P., and Delalleau, O. Parallel Tempering for Training of Restricted Boltzmann Machines. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 145–152, 2010.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., and Bengio, S. Why does unsupervised pretraining help deep learning? *J. Mach. Learn. Res.*, 11: 625–660, 2010.

Fischer, A. and Igel, C. Empirical analysis of the divergence of Gibbs sampling based learning algorithms for restricted Boltzmann machines. In *Proceedings of the 20th international conference on Artificial neural networks: Part III*, ICANN'10, pp. 208–217, Berlin, Heidelberg, 2010. Springer-Verlag.

Hinton, G. E. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14:1771–1800, August 2002. ISSN 0899-7667.

Hinton, G. E. A Practical Guide to Training Restricted Boltzmann Machines. Technical Report 2010–003, Department of Computer Science, University of Toronto, 2010.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, July 2006.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-Based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, volume 86, pp. 2278–2324, 1998.

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

Marlin, B. M., Swersky, K., Chen, B., and de Freitas, N. Inductive Principles for Restricted Boltzmann Machine Learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 509–516, 2010.

Salakhutdinov, R. *Learning Deep Generative Models*. PhD thesis, University of Toronto, 2009a.

Salakhutdinov, R. Learning in Markov Random Fields using Tempered Transitions. In *In Advances in Neural Information Processing Systems*, pp. 1598–1606, 2009b.

Salakhutdinov, R. and Hinton, G. E. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pp. 448–455, 2009.

Schulz, H., Müller, A., and Behnke, S. Investigating Convergence of Restricted Boltzmann Machine Learning. In *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

Smolensky, P. Information processing in dynamical systems: foundations of harmony theory. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*, pp. 194–281. MIT Press, Cambridge, MA, USA, 1986.

# Publication 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.**

# Parallel Tempering is Efficient for Learning Restricted Boltzmann Machines

KyungHyun Cho, Tapani Raiko, Alexander Ilin

***Abstract*—A new interest towards restricted Boltzmann machines (RBMs) has risen due to their usefulness in greedy learning of deep neural networks. While contrastive divergence learning has been considered an efficient way to learn an RBM, it has a drawback due to a biased approximation in the learning gradient. We propose to use an advanced Monte Carlo method called parallel tempering instead, and show experimentally that it works efficiently.**

## I. Introduction

Recently, deep neural networks such as a deep belief network [1] and a deep Boltzmann machine [2] have become widely applied to various machine learning tasks. These deep neural networks are characterized and distinguished from the conventional multi-layer perceptron and other shallow neural networks by their large number of layers of neurons and its adaptation by a layer-wise unsupervised greedy learning method.

A deep neural network is typically constructed by stacking multiple restricted Boltzmann machines (RBM) so that the hidden layer of one RBM becomes the visible layer of another RBM which is situated one level up. This way of constructing deep neural networks allows for using layer-wise training of RBMs, which facilitates finding a more accurate model for the data. Such multi-stage learning has been empirically shown to work better than conventional learning methods such as the widely used back-propagation [2]. It is thus important to have an efficient and well-behaving learning method for RBM. In this paper, we explore some advanced sampling procedures for that.

The paper starts by briefly discussing theoretical background behind a general Boltzmann machine (BM). Then, we state the difference between the general BM and RBM, and introduce a learning algorithm specific to RBM. Contrastive divergence learning which is a successful learning algorithm for RBM is explained, and later a learning algorithm utilizing parallel tempering is introduced. In the experimental part, we demonstrate the capability of RBM to learn the data distribution by drawing samples from a trained RBM. We also compare the newly proposed learning algorithm with contrastive divergence learning.

The authors are with the Department of Information and Computer Science, School of Science and Technology, Aalto University, Finland (email: firstname.lastname@tkk.fi)

## II. Boltzmann Machine

Boltzmann machine (BM) is a stochastic recurrent neural network consisting of binary neurons [3]. The network is fully connected, and we use 0 or 1 as the state of each neuron $x_i$. The links between each pair of neurons are symmetric (meaning that the effect of one neuron on the state of the other one is symmetric for each pair) and it is assumed that there are no edges going from the neurons to themselves.

The probability of a particular state $\mathbf{x} = [x_1, x_2, \cdots, x_d]^T$ is defined by the energy of BM which is postulated as

$$E(\mathbf{x} \mid \mathbf{W}) = -\frac{1}{2}\sum_i \sum_{j>i} w_{ij} x_i x_j,$$

where $\mathbf{W}$ is a weight matrix consisting of weights $w_{ij}$ of the synaptic connections between neurons $i$ and $j$. We assume that $w_{ii} = 0$ and that $w_{ij} = w_{ji}$. The bias terms can be omitted by using an auxiliary component in the state vector that always has the value 1. The probability of a state $\mathbf{x}$ is

$$P(\mathbf{x} \mid \mathbf{W}) = \frac{1}{Z(\mathbf{W})} \exp\left[-E(\mathbf{x} \mid \mathbf{W})\right] \tag{1}$$

where

$$Z(\mathbf{W}) = \sum_{\mathbf{x}} \exp\left[-E(\mathbf{x} \mid \mathbf{W})\right]$$

is the normalizing constant.

It follows from (1) that the conditional probability of a single neuron being either 0 or 1 given the states of the other neurons can be written in the following way:

$$P(x_i = 1 \mid \mathbf{x}_{\setminus i}, \mathbf{W}) = \frac{1}{1 + \exp\left(-\sum_{j \neq i} w_{ij} x_j\right)}, \tag{2}$$

where $\mathbf{x}_{\setminus i}$ denotes a vector $[x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_d]^T$. It is obvious that this probability is expressed using the standard nonlinear sigmoid function used in multi-layer perceptron networks.

The neurons of BM are usually divided into visible and hidden ones $\mathbf{x} = [\mathbf{v}^T, \mathbf{h}^T]^T$, where the states $\mathbf{v}$ of the visible neurons are clamped to observed data, and the states $\mathbf{h}$ of the hidden neurons can change freely.

### A. Learning Boltzmann machine

The parameters of BM can be learnt from the data using standard maximum likelihood estimation. Given a data set

$\{\mathbf{v}^{(t)}\}_{t=1}^{N}$, the log-likelihood of the parameters of BM is

$$\mathcal{L}(\mathbf{W}) = \sum_{t=1}^{N} \log P(\mathbf{v}^{(t)}|\mathbf{W}) = \sum_{t=1}^{N} \log \sum_{\mathbf{h}} P(\mathbf{v}^{(t)}, \mathbf{h} \mid \mathbf{W}),$$

where the states $\mathbf{h}$ of the hidden neurons have to be marginalized out. This yields

$$\begin{aligned}\mathcal{L}(\mathbf{W}) &= \sum_{t=1}^{N} \log \frac{\sum_{\mathbf{h}} \exp(\frac{1}{2}\mathbf{x}^{(t)^T}\mathbf{W}\mathbf{x}^{(t)})}{\sum_{\mathbf{x}} \exp(\frac{1}{2}\mathbf{x}^T\mathbf{W}\mathbf{x})} \\ &= \sum_{t=1}^{N} \left[ \log \sum_{\mathbf{h}} \exp(\frac{1}{2}\mathbf{x}^{(t)^T}\mathbf{W}\mathbf{x}^{(t)}) \right. \\ &\qquad \left. - \log \sum_{\mathbf{x}} \exp(\frac{1}{2}\mathbf{x}^T\mathbf{W}\mathbf{x}) \right].\end{aligned}$$

The gradient of the log-likelihood is obtained by taking partial derivative with respect to parameters $w_{ij}$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{ij}} &= \frac{1}{2} \sum_{t=1}^{N} \left[ \frac{\sum_{\mathbf{h}} x_i^{(t)} x_j^{(t)} \exp(\frac{1}{2}\mathbf{x}^{(t)^T}\mathbf{W}\mathbf{x}^{(t)})}{\sum_{\mathbf{h}} \exp(\frac{1}{2}\mathbf{x}^{(t)^T}\mathbf{W}\mathbf{x}^{(t)})} \right. \\ &\qquad \left. - \frac{\sum_{\mathbf{x}} x_i x_j \exp(\frac{1}{2}\mathbf{x}^T\mathbf{W}\mathbf{x})}{\sum_{\mathbf{x}} \exp(\frac{1}{2}\mathbf{x}^T\mathbf{W}\mathbf{x})} \right] \\ &= \frac{N}{2} \left[ \frac{1}{N} \sum_{t=1}^{N} \sum_{\mathbf{h}} x_i^{(t)} x_j^{(t)} P(\mathbf{h} \mid \mathbf{v}^{(t)}, \mathbf{W}) \right. \\ &\qquad \left. - \sum_{\mathbf{x}} x_i x_j P(\mathbf{x} \mid \mathbf{W}) \right] \\ &= \frac{N}{2} \left[ \langle x_i x_j \rangle_{P(\mathbf{h}|\{\mathbf{v}^{(t)}\},\mathbf{W})} - \langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})} \right],\end{aligned}$$

where we used a shorthand notation $\langle \cdot \rangle_{P(\cdot)}$ which can be understood as the expectation computed over the probability distribution $P(\cdot)$.

The overall update formula for a parameter $w_{ij}$ is

$$w_{ij} \leftarrow w_{ij} + \eta \left[ \langle x_i x_j \rangle_{P(\mathbf{h}|\{\mathbf{v}^{(t)}\},\mathbf{W})} - \langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})} \right], \quad (3)$$

where $\eta$ denotes the learning rate, and it has absorbed the factor $\frac{N}{2}$. Thus the direction of the gradient is the difference between the correlations under two distinct probability distributions. The first one $\langle x_i x_j \rangle_{P(\mathbf{h}|\{\mathbf{v}^{(t)}\},\mathbf{W})}$ is the correlation obtained while the visible nodes are clamped to the training data, and it represents the target probability distribution to which the trained BM is intended towards. The second term $\langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})}$ is the correlation obtained from the current probability distribution represented by the BM. According to the sign of each term, the computations of the two terms can be called the positive phase and the negative phase, respectively.

The learning algorithm can be seen as driving BM so that the correlations between each pair of neurons in the two phases coincide with each other. In other words, we want the probability distribution represented by BM to be exactly identical to the probability distribution defined by the training data set. Although the analytical formulation of the exact probability distribution of the training data set is unknown, the positive phase mimics it by clamping the visible neurons to the training data, and lets the hidden neurons freely have their own states. It can then be compared to the probability distribution represented by BM where both the visible and hidden neurons can freely choose their states according to the distribution determined by the weights. When the difference between the distributions in the two phases becomes zero or small enough, then learning effectively stops.

### *B. Practical limitation and approximate approach*

Although the activation and learning rules of BM are both clearly formulated, there are practical limitations in using BM. Especially, the gradient-based update formula is computationally unfeasible, as the distributions required in both phases can only be obtained by evaluating all possible combinations of the states of the neurons in the machine.

There exist exponentially many possible combinations of the states. For example, BM which was designed to handle $28 \times 28$ black-and-white image with 500 hidden nodes has $2^{28 \cdot 28 + 500}$ possible combinations, and the number is unimaginably huge. Evaluating all those states at every gradient update step is simply unfeasible. In fact, even the evaluation of a probability of a single combination is almost impossible, as all combinations of the states must be evaluated regardlessly to compute the normalizing constant $Z(\mathbf{W})$.

The obvious approach to overcome this difficulty is to use Gibbs sampling (see e.g. [4]). Gibbs sampling can easily be implemented because the conditional distribution of the state of a single neuron in BM given the states of all the other neurons is given by (2).

This approach can greatly reduce the computational burden of the gradient update rule. If we assume that the number of samples required for explaining the probability distribution of the whole state space is sufficiently smaller than the size of the state space, that is the number of all possible combinations of the states of the neurons, the learning of BM is not anymore as computational unfeasible as the exact computation of the probability masses.

However, there also exist other kinds of limitations in using Gibbs sampling for training BM. The biggest problem is due to the full-connectivity of BM. Since each neuron is connected to and influenced by all the other neurons, it takes as many steps as the number of neurons to get one sample of the BM state. Even when the visible neurons are clamped to the training data, the number of required steps for a single fresh sample is still at least the number of hidden neurons. This makes the successive samples in the

chain highly correlated with each other and this poor mixing affects the performance of learning. Another limitation of this approach is that multi-modal distributions are problematic for Gibbs sampling [5]: Due to the nature of component-wise sampling, the samples might miss some modes of the distribution.

## III. Restricted Boltzmann Machine

To overcome these practical limitations imposed on the general Boltzmann machine, a structurally restricted version of Boltzmann machine called Restricted Boltzmann Machine (RBM) has been proposed [6]. RBM is constructed by removing the lateral connections in-between the visible neurons and the hidden neurons. Therefore, a visible neuron would only have edges connected to the hidden neurons, and a hidden neuron would only have edges connected to the visible neurons. Now, the structure of RBM can be divided into two layers with inter-connecting edges, and it resembles the structure of the bi-partite graph.

The most important advantage over the general BM is on the improved effectiveness in doing sampling. It follows from the fact that all neurons in one layer are independent of each other given the states of the neurons in the other layer. Now Gibbs sampling can be done layer-wise rather than component-wise. It can then greatly reduce the number of sampling runs required to get enough samples to represent the probability distribution.

Furthermore, in the view of computational efficiency, the layer-wise sampling can fully utilize the modern parallelized computing environment, as sampling of each neuron (or component) in the same layer can be done independently of each other and simultaneously, whereas Gibbs sampling on the general BM requires that sampling of each neuron must be done sequentially.

As the restriction has been imposed on the structure, the energy, the state probability must be modified accordingly:

$$E(\mathbf{v}, \mathbf{h} \mid \mathbf{\Theta}) = -\frac{1}{2}\left(\mathbf{v}^T\mathbf{W}\mathbf{h} + \mathbf{b}^T\mathbf{v} + \mathbf{c}^T\mathbf{h}\right)$$
$$P(\mathbf{v}, \mathbf{h} \mid \mathbf{\Theta}) = \frac{1}{Z(\mathbf{\Theta})}\exp\left\{-E(\mathbf{v}, \mathbf{h} \mid \mathbf{\Theta})\right\},$$

where now parameters $\mathbf{\Theta} = (\mathbf{W}, \mathbf{b}, \mathbf{c})$ include biases $\mathbf{b}$ and $\mathbf{c}$. The learning rules then become

$$w_{ij} \leftarrow w_{ij} + \eta_w\left[\langle v_i h_j\rangle_{P(\mathbf{h}|\{\mathbf{v}^{(t)}\},\mathbf{\Theta})} - \langle v_i h_j\rangle_{P(\mathbf{v},\mathbf{h}|\mathbf{\Theta})}\right]$$
$$b_i \leftarrow b_i + \eta_b\left[\langle v_i\rangle_{P(\mathbf{h}|\{\mathbf{v}^{(t)}\},\mathbf{\Theta})} - \langle v_i\rangle_{P(\mathbf{v},\mathbf{h}|\mathbf{\Theta})}\right]$$
$$c_j \leftarrow c_j + \eta_c\left[\langle h_j\rangle_{P(\mathbf{h}|\{\mathbf{v}^{(t)}\},\mathbf{\Theta})} - \langle h_j\rangle_{P(\mathbf{v},\mathbf{h}|\mathbf{\Theta})}\right],$$

where we used the same shorthand notation $\langle\cdot\rangle_{P(\cdot)}$ as before.

For computing the correlations during the positive phase, the exact computation of the correlation between the clamped neurons $\mathbf{v}$ and the (free) hidden neurons $\mathbf{h}$ is possible, since all the hidden neurons are independent of each other conditioned on the training data. For instance, the expectation for updating the weights is

$$\langle v_i h_j\rangle_{P(\mathbf{h}|\{\mathbf{v}^{(t)}\},\mathbf{\Theta})} = \frac{1}{N}\sum_{t=1}^{N} v_i^{(t)} P(h_j^{(t)} \mid \mathbf{v}^{(t)}, \mathbf{\Theta}), \quad (4)$$

where $P(h_j^{(t)} \mid \{\mathbf{v}^{(t)}\}, \mathbf{\Theta})$ is computed using (2) with the bias term included.

Although the difficulties in learning have partly been solved, the practical limitations of the general Boltzmann machine still remain. As the number of neurons in RBM increases, a greater number of samples must be gathered by Gibbs sampling in order to properly explain the probability distribution represented by RBM. The computational load has been greatly reduced but it still remains large. Moreover, the problem of the multi-modal probability distribution has not at all been addressed.

### A. Contrastive divergence learning

Contrastive divergence (CD) learning [7] does not follow the gradient obtained by the maximum likelihood criterion. Rather, CD learning approximates the true gradient by replacing the expectation over $P(\mathbf{v}, \mathbf{h} \mid \mathbf{\Theta})$ with an expectation over a distribution $P_n$ that is obtained by running $n$ steps of Gibbs sampling from the empirical distribution. In practice, parallel chains of Gibbs sampling are run starting separately from each observation in the data set. The samples at step $n$ are used to compute the expectation.

The learning formula, then, becomes

$$w_{ij} \leftarrow w_{ij} + \eta\left[\langle x_i h_j\rangle_{P_0} - \langle x_i h_j\rangle_{P_n}\right].$$

It should be noted that the case $n = 0$ produces the empirical distribution $P(\mathbf{h} \mid \{\mathbf{v}^{(t)}\}, \mathbf{\Theta})$ used in the positive phase, whereas the case $n = \infty$ produces the true distribution of the negative phase $P(\mathbf{x} \mid \mathbf{\Theta})$.

As it can be anticipated from the fact that the direction of the gradient is not identical to the exact gradient, CD learning is known to be biased [8]. Nevertheless, CD learning has been proven to work well in practice. A good property of CD is that in case the data distribution is multimodal, running the chains starting from each data sample guarantees, that the samples approximating the negative phase have representatives from different modes.

## IV. Restricted Boltzmann Machine and Parallel Tempering

A problem that has not been addressed neither by Gibbs sampling nor by CD learning is that the samples generated during the negative phase do not tend to explain the whole

1) Create a sequence of RBMs $(R_0, R_1, \cdots, R_K)$ such that parameters of $R_k$ are $\mathbf{\Theta}_k = (T_k\mathbf{W}, \mathbf{b}, \mathbf{c})$, where $0 \leq T_0 < T_1 < \cdots < T_K = 1$.
2) Create an empty set of samples $\boldsymbol{X} = \{\}$.
3) Set $\mathbf{x}_0 = (\mathbf{x}_{0,0}, \cdots, \mathbf{x}_{K,0})$ such that every $\mathbf{x}_{k,0}$ is a uniformly distributed random vector (or use old ones from the previous epoch).
4) For $m$ = 1 to $M$, repeat
   a) Sample $\mathbf{x}_m = (\mathbf{x}_{0,m}, \cdots, \mathbf{x}_{K,m})$ from the sequence of RBMs such that $\mathbf{x}_{k,m}$ is sampled by one-step Gibbs sampling starting from $\mathbf{x}_{k,m-1}$.
   b) For $j$ = $K$ to 1, repeat
      - Swap $\mathbf{x}_{j,m}$ and $\mathbf{x}_{j-1,m}$ according to $P_{\text{swap}}(\mathbf{x}_{j,m}, \mathbf{x}_{j-1,m})$ computed using (5).
   c) Add $\mathbf{x}_{K,m}$ to $\boldsymbol{X}$.
5) $\boldsymbol{X}$ is the set of samples collected by parallel tempering sampling.

**TABLE I:** Sequence of steps for sampling from RBM using parallel tempering.

state space. This paper, therefore, proposes to use yet another, improved variant of Markov-Chain Monte Carlo sampling method called *parallel tempering* (PT) [9]. PT sampling used in this paper utilizes multiple Gibbs sampling chains with varying levels of temperatures, where a term *temperature* denotes the level of the energy of the overall system, in this case, RBM. The higher the temperature of the chain, the more likely the samples collected by Gibbs sampling to move freely.

The use of PT sampling in training RBM is simply to use it instead of Gibbs sampling in the negative phase. Due to the previously mentioned characteristics, it is expected that the samples collected during the negative phase would explain the model distribution better, and that the learning process would be done well even with a smaller number of samples than those required if Gibbs sampling is used.

The basic idea of PT sampling is that samples are collected from multiple chains of Gibbs sampling with different temperatures from the highest temperature $T = 0$ to the current temperature $T = 1$[1]. For every pair of collected samples from two distinct chains, the swap probability is computed, and the samples are swapped according to the probability. The swap probability of a pair of samples is formulated according to the Metropolis rule (see e.g. [4]) as

$$P_{\text{swap}}(\mathbf{x}_{T_1}, \mathbf{x}_{T_2}) = \min\left(1, \frac{P_{T_1}(\mathbf{x}_{T_2})P_{T_2}(\mathbf{x}_{T_1})}{P_{T_1}(\mathbf{x}_{T_1})P_{T_2}(\mathbf{x}_{T_2})}\right), \quad (5)$$

where $T_1$ and $T_2$ denote the temperatures of the two chains, and $\mathbf{x}_{T_1}$ and $\mathbf{x}_{T_2}$ denote samples collected from the two chains. $P_T(\cdot)$ is the probability function of the RBM with parameters $\mathbf{\Theta} = (T\mathbf{W}, \mathbf{b}, \mathbf{c})$, where the effect of different temperatures is emulated by multiplying the current weights $\mathbf{W}$ by the corresponding temperature $T$.

After each round of sampling and the swaps, the sample at the true temperature $T = 1$ is gathered as the sample for the iteration. The samples come from the true distribution $P(\mathbf{v}, \mathbf{h} \mid \mathbf{\Theta})$ assuming that enough iterations are run to diminish the effect of the initialization.

It must be noted that the Gibbs sampling chain with the highest temperature ($T = 0$) is never multimodal. So, the samples from the chain are less prone to missing some modes that exist in RBM. From the chain with the highest temperature to the lowest temperature, samples from each chain become more and more likely to follow the target model distribution.

This nature of swapping samples between the different temperatures enables better mixing of samples from different modes with much less number of samples than that would have been required if Gibbs sampling was used. A brief description of how PT sampling can be done for training RBM is shown in Table I. This is the procedure that is run between each parameter update during learning.

## V. Estimating Log-Likelihood of Restricted Boltzmann Machines

For estimating the normalizing constant, this paper adapts the method utilizing *annealed importance sampling* (AIS) [10] which has been successfully adapted for computing the normalizing constant of RBM [11].

AIS is based on *simple importance sampling* (SIS) method that could estimate the ratio of two normalizing constants. For two probability densities $P_A(\mathbf{x}) = \frac{P_A^*(\mathbf{x})}{Z_A}$ and $P_B(\mathbf{x}) = \frac{P_B^*(\mathbf{x})}{Z_B}$, the ratio of two normalizing constants $Z_A$ and $Z_B$ can be estimated by a Monte Carlo sampling method without any bias if it is possible to sample from $P_A(\cdot)$:

$$\frac{Z_A}{Z_B} = E_{P_A}\left[\frac{P_B^*(\mathbf{x})}{P_A^*(\mathbf{x})}\right] \approx \frac{1}{M}\sum_{i=1}^{M}\frac{P_B^*(\mathbf{x})}{P_A^*(\mathbf{x})}.$$

[1]Since the lower value denotes the higher temperature, a term *inverse temperatures* is frequently used, but in this paper, *temperature* will be used.

1) Create a sequence of intermediate temperatures $T_k$ such that $0 \leq T_0 < T_1 < \cdots < T_K = 1$.
2) Create a base RBM $R_0$ with parameters $\mathbf{\Theta}_0 = (\mathbf{W}_0, \mathbf{b}, \mathbf{c})$, where $\mathbf{W}_0 = 0$.
3) Create a sequence of intermediate RBMs $R_k$ such that
   - It has twice as many hidden nodes as the target RBM has.
   - Parameters are $\mathbf{\Theta}_k = ([(1 - T_k)\mathbf{W}_0 \;\; T_k\mathbf{W}], \mathbf{b}, \left[\mathbf{c}^T \mathbf{c}^T\right]^T)$.
4) For $m$ = 1 to $M$, repeat
   a) Sample $\mathbf{x}_1$ from $R_0$.
   b) For $k$ = 1 to $K - 1$, repeat
      - Sample $\mathbf{x}_{k+1}$ from $R_k$ by one-step Gibbs sampling starting from $\mathbf{x}_k$.
   c) Set $u_m = \prod_{k=1}^{K} \frac{P_k^*(\mathbf{x}_k)}{P_{k-1}^*(\mathbf{x}_k)}$, where $P_k^*(\cdot)$ is an unnormalized marginal distribution function of $R_k$.
5) The estimate of $\frac{Z_K}{Z_0}$ is $\frac{1}{M}\sum_{m=1}^{M} u_m$.

**TABLE II:** Sequence of estimating the normalizing constant by annealed importance sampling.

Based on SIS, AIS estimates the normalizing constant of the model distribution by computing the ratio of the normalizing constants of consecutive intermediate distributions ranging from so-called base distribution and the target distribution. The base distribution is chosen such that its normalizing constant $Z_0$ can be computed exactly and it is possible to collect independent samples from it. A natural choice of the base distribution for RBM is an RBM with zero weights $\mathbf{W}$. This yields the normalizing constant

$$Z_0 = \prod_i (1 + \exp\{b_i\}) \prod_j (1 + \exp\{c_j\}),$$

where indices $i$ and $j$ go through all the visible and hidden neurons, respectively.

By computing the product of the estimated ratios of the intermediate normalizing constants and $Z_0$, the normalizing constant of the target RBM can be estimated. The algorithm implementing AIS is outlined in Table II.

To achieve an accurate estimate of the normalizing constant, a large number of intermediate RBMs should be used, and the normalizing constant must be estimated by as many annealing runs as possible [11]. This means that $K$ and $M$ in the algorithm from Table II should be large. The runs of AIS are independent from each other, so they can be fully parallelized.

## VI. Experiments

Two different sets of experiments were done using MATLAB. The goal of the first experiment was to test the capability of RBMs to capture the data distribution. We generated samples from RBM trained on the OptDigits data set. The data set was acquired from the UCI Machine Learning Repository [12] and it consisted of handwritten digits of the size $8 \times 8$ pixels. The samples were collected by parallel tempering sampling starting from a randomly drawn state. Most of the samples were observed to resemble the digits regardless of the initial state.

The second experiment was conducted in order to compare the performance of RBM depending on two different learning methods: CD learning and learning using sampling with PT. The performance was evaluated by the approximated likelihood of the training data set and the approximated probability of the test data set. We observed that the performance is better when the gradient was estimated using PT sampling.

Furthermore, in the second experiment we computed the probability of uniformly randomly generated data in the current RBM model. The goal was to observe a potential problem of CD learning that the samples generated during the negative phase do not represent the state space as well as the samples generated by PT sampling, but only represent the region centered around the training samples [13]. The probability of random data was computed for different learning methods and compared.[2]

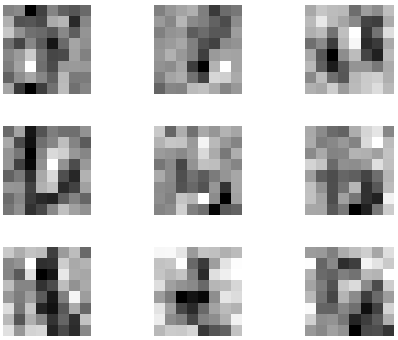
### *A. Generating samples from a trained restricted Boltzmann machine*

RBM was constructed such that there are 64 visible neurons and 100 hidden neurons. Each neuron had a bias parameter. RBM was trained with 3822 training samples of $8 \times 8$ handwritten digits. The original OptDigits data set provides 17-level greyscale digits but for simplicity we rounded the intensity of each pixel so that the intensity less than 8 became 0 (and 1 otherwise).

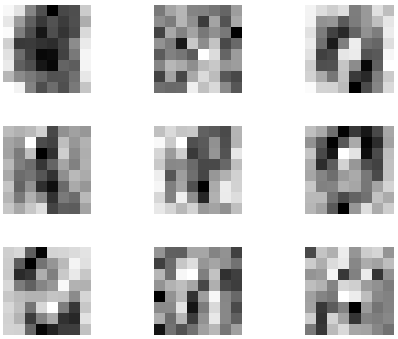
RBM was trained separately by CD learning with $n = 1$ and learning with PT sampling. PT sampling was done with $K = 20$ and temperatures $T_0 = 0$, $T_1 = 0.05, \ldots, T_{20} = 1$. The models represented by the RBMs are named CD1 and

[2]We assume that uniformly drawn samples do not lie close to the training data because the size of the training data set is much smaller than the size of the state space which is $2^{64}$.

(a) Training data set



(b) Visualization of hidden nodes (CD1)



(c) Visualization of hidden nodes (PT)

**Fig. 1:** Training data set and visualization of hidden nodes. (a) shows 10 training samples where for each digit one sample was randomly chosen. (b) and (c) shows the weights connected to nine randomly chosen hidden neurons.

(a) Model learned with CD1

(b) Model learned with PT

**Fig. 2:** Samples generated by parallel tempering sampling from the RBM trained with (a) CD1 and (b) PT started from the random sample. The first digits of both figures are the random initial samples.

PT, respectively. Each gradient update was done in the batch style so that all the training samples were used. CD1 and PT were trained for 2000 epochs, and the learning rate $\eta$ started from $0.05$ and gradually decreased following the search-then-converge strategy.[3]

Figure 1 shows the training data samples and the visualization of the hidden nodes after training. The visualization of the hidden node was done by displaying the weights associated with the node as a grey-scale digit. It can be observed that each hidden node represents a distinct feature.

To see the generative behavior of RBM, the samples were gathered using PT sampling starting from a random initial sample. $K = 20$ was used for PT sampling. Figure 2 shows the activation probabilities for the visible neurons of the generated samples from the models learned with CD1 and PT. The digits in the figure are 19 samples chosen out of 2000 samples collected by PT sampling starting from the random sample. Each consecutive samples are separated by 100 sampling steps, and the first digit in both figures of Figure 2 represents the random initial sample. It is clear that the trained RBM is able to generate digits which look similar to the training data. The proposed method of learning with PT sampling works as well as the conventional CD learning.

### *B. Comparison between contrastive divergence learning and parallel tempering*

For the second experiment, we trained RBMs with the same 100 hidden neurons using four learning algorithms: CD1, CD5, CD25 and PT (where CD$n$ is CD learning with $n$ sampling steps).

The parameters $K$ and $M$ of parallel tempering were chosen so that the number of total Gibbs sampling steps during one gradient update matches that of CD1 which uses as many samples as the training data samples. PT was, therefore, trained with $K = 20$ temperatures and $M = 192$ samples per gradient update. This choice is reasonable in a sense that the difference in CD learning and learning with PT sampling only depends on the number of Gibbs sampling steps, whereas the computational cost of additional operations may vary largely depending on the implementation.

Each RBM was trained for 635 epochs and the probabilities of both training and test data were estimated. The parameters used in AIS were $M = 50$ and $K = 5000$. All the models were trained 30 times and the averaged performance indices were calculated.

Figure 3 shows that the probability of the test data increases, while the probability of the random data decreases over the gradient updates. This is consistent with the fact that the

[3] $\eta(n) = \frac{\eta_0}{1+\frac{n}{n_0}}$, where $\eta_0 = 0.05$ for both the weight and the bias, and $n_0 = 300$ for both CD1 and PT.

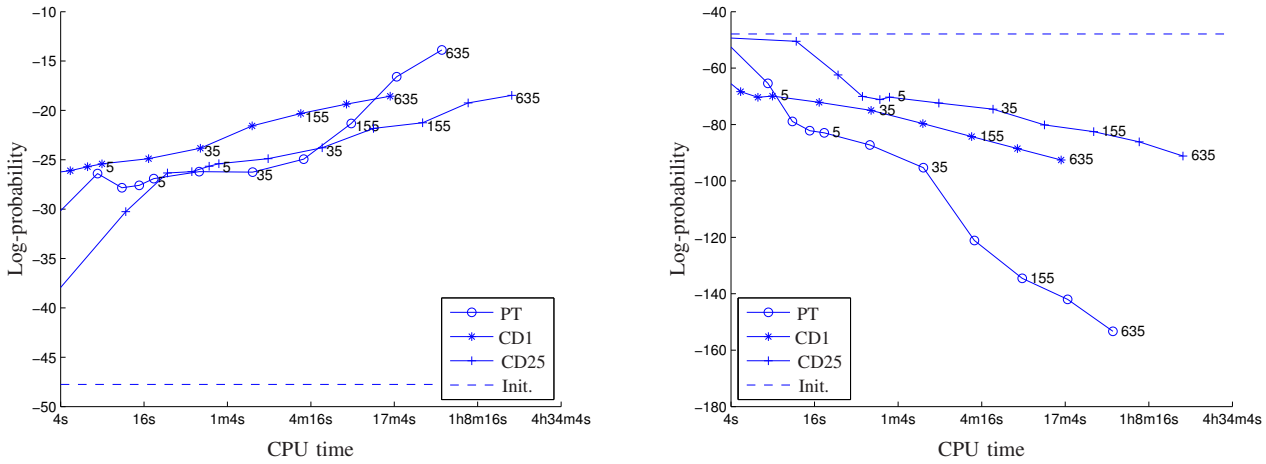



**Fig. 3:** Left: Average probabilities of test data against the processor time. Right: Average probabilities of random data against the processor time. The time scale is logarithmic. The dashed lines indicate the log-probability of the initial weights for both test and random data. The numbers denote the number of epochs after which the value was measured.

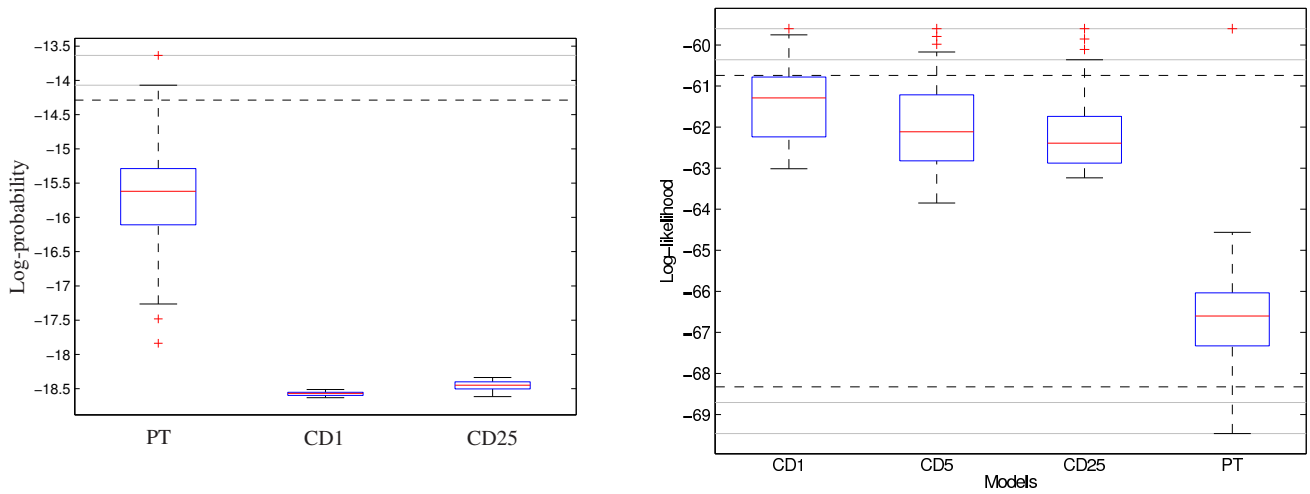



**Fig. 4:** Left: Box plots of probabilities of test data after 635 epochs over 30 repeated runs. Right: Box plots of probabilities of random data over 30 repeated runs. For probability values, the red line inside the box denotes the median, the edges of the box are 25-th and 75-th percentiles, and the whiskers are extended to the extreme value not considering possible outliers.

gradient maximizes the likelihood according to the distribution of the training data. It also confirms that the probability of the unseen samples that are not close to any training sample is decreased. However, the rate of the changes in the likelihood and the probability of the test data over updates differs from one model to another. PT achieves the highest average likelihood and the highest average probability of the test data, and at the same time achieves the lowest probability of the random data at the fastest rate. It can be further observed that PT learning is computationally more favorable than CD25 and comparable to CD1.

Figure 4 which shows the average probability of the test data set and the random data set by 30 independent trials, further, confirms that PT indeed achieves the highest probability of the test data set and the lowest probability of the random data set. It should be, however, noted that the variance of PT is greater than those of both CD1 and CD25.

The increase of $n$ in CD learning certainly boosts up the rate of the increase in the likelihood as a function of learning epochs, but even with $n$ as large as 25 CD learning cannot achieve as large likelihood as PT does. CD learning with $n = 25$ is much more computationally demanding than PT. This result indicates that the use of the advanced sampling technique can yield faster and better training of RBM.

## VII. Discussion

As an alternative to Gibbs sampling, contrastive divergence learning has been proposed and made learning of RBMs faster. Despite its computationally favorability and the wide acceptance, contrastive divergence learning is biased in the sense that the computed gradient computed does not lead

exactly to the maximum of the likelihood. This paper, therefore, proposed an alternative approach which utilizes parallel tempering for training RBMs. This approach does not sacrifice the optimality of the direction of the gradient but reduces the computational cost by improving the quality of the samples.

Two separate experiments were done: (1) to confirm the capability of RBM to capture the data distribution and (2) to show that RBM trained by the proposed PT approach is superior to that trained by the conventional CD learning. The former experiment confirmed that RBM trained by either CD learning or learning with PT sampling is able to generate samples resembling the training data. The second experiment confirmed that the use of the proposed PT approach can result in a more accurate RBM. As a performance measure, the log-likelihood estimated using annealed importance sampling was used.

Learning with PT sampling was superior in all aspects of the experimental results. We observed higher likelihood computed on the training data and higher probability of the test data. The increase of the likelihood over the gradient updates was also faster. The probability of random samples by PT sampling was less than any other model trained with CD learning. This confirmed the existence of the potential problem of CD learning that the samples generated by CD learning during the negative phase do not represent the state space well and fail to decrease the probabilities over the regions which are far from the training data. At the same time, the computational complexity of the gradient update by PT sampling was comparable to that of CD learning.

Recently, the use of PT learning for RBMs has been proposed independently also in [14]. The work in [14] illustrates the possible explanations why PT learning performs better than CD learning, and presents the experimental results showing the superiority of PT learning. It, however, lacks showing the efficiency of PT learning compared to CD learning in terms of the computational complexity, whereas we showed that even in the terms of the computational load PT learning is as efficient as CD learning.

A similar attempt at improving the learning by adapting an advanced sampling method has been proposed in [5]. The work also does sampling by considering multiple distributions of different temperatures, but the details of the proposed algorithm (called *tempered transition*) differ greatly from that presented in this paper.

Findings of this paper raise further research issues related to improving PT sampling for training RBM. The in-depth study of how the parameters of PT sampling influence the performance must be done, as the experiments in this report were done only with one specific setting of PT sampling. The adjustable parameters such as the learning rate, the number of temperatures and the number of samples, significantly affects the performance of learning, and the relationship between the choice of parameters and the performance must be further studied in order for PT sampling to be widely used.

*Acknowledgements*

This work was supported by the honours programme of the department, by the Academy of Finland and by the IST Program of the European Community, under the PASCAL2 Network of Excellence. This publication only reflects the authors' views.

## REFERENCES

[1] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, July 2006.

[2] R. Salakhutdinov and G. Hinton, "Deep Boltzmann machines," in *Artificial Intelligence and Statistics*, 2009.

[3] S. Haykin, *Neural Networks: A Comprehensive Foundation (2nd Edition)*, 2nd ed. Prentice Hall, July 1998.

[4] D. J. C. Mackay, *Information Theory, Inference & Learning Algorithms*, 1st ed. Cambridge University Press, June 2002.

[5] R. Salakhutdinov, "Learning in Markov random fields using tempered transitions," *NIPS 2009*, 2009.

[6] P. Smolensky, "Information processing in dynamical systems: foundations of harmony theory," in *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 194–281.

[7] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comp.*, vol. 14, no. 8, pp. 1771–1800, August 2002.

[8] M. A. Carreira-Perpiñ and G. Hinton, "On contrastive divergence learning," in *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, Jan 6-8, 2005, Savannah Hotel, Barbados*, R. G. Cowell and Z. Ghahramani, Eds. Society for Artificial Intelligence and Statistics, 2005, pp. 33–40.

[9] D. J. Earl and M. W. Deem, "Parallel tempering: Theory, applications, and new perspectives," *Phys. Chem. Chem. Phys.*, vol. 7, no. 23, pp. 3910–3916, 2005.

[10] R. M. Neal, "Annealed importance sampling," *Statistics and Computing*, vol. 11, pp. 125–139, 1998.

[11] R. Salakhutdinov, "Learning deep generative models," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.

[12] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[13] Y. Bengio, "Learning deep architectures for ai," Dept. IRO, Universite de Montreal, Tech. Rep., 2007.

[14] G. Desjardins, A. Courville, Y. Bengio, P. Vincent, and O. Delalleau, "Tempered Markov chain Monte Carlo for training of restricted Boltzmann machines," Dept. IRO, Universite de Montreal, Tech. Rep. 1345, 2009.

# Publication 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.**

# Tikhonov-Type Regularization for Restricted Boltzmann Machines

KyungHyun Cho, Alexander Ilin, and Tapani Raiko

Department of Information and Computer Science
Aalto University School of Science, Finland
{kyunghyun.cho,alexander.ilin,tapani.raiko}@aalto.fi

**Abstract.** In this paper, we study a Tikhonov-type regularization for restricted Boltzmann machines (RBM). We present two alternative formulations of the Tikhonov-type regularization which encourage an RBM to learn a smoother probability distribution. Both formulations turn out to be combinations of the widely used weight-decay and sparsity regularization. We empirically evaluate the effect of the proposed regularization schemes and show that the use of them could help extracting better discriminative features with sparser hidden activation probabilities.

**Keywords:** Restricted Boltzmann Machine, Tikhonov Regularization.

## 1 Introduction

Restricted Boltzmann machines (RBM) play an important role in deep learning. In many deep neural networks each layer of the network is pre-trained as if it were an RBM, and it has been empirically shown to facilitate training the whole network (see, e.g., [10,8]).

It is common to use a stochastic gradient method for training RBMs. Both contrastive divergence learning [12] and approximate maximum-likelihood learning (see, e.g., [21]), two of the most popular learning methods, are based on the stochastic gradient method.

One important research direction in using the stochastic gradient method for RBMs is to design a regularization term. For instance, one of the most naive, but widely-used, regularization methods called weight-decay regularizes the growth of parameters in order to avoid overfitting and to stabilize learning. Another completely different regularization technique introduced in [14] forces learning to result in an RBM that gives sparser hidden activations given visible data.

Along this line of research, we investigate a Tikhonov-type regularization (see, e.g., [1,9]) by which we refer to regularizing the derivative of either an approximating function or a function related to it. In this paper, two different formulations of the Tikhonov-type regularization for RBMs are derived. We found that both formulations appear as a combination of weight-decay and sparsity regularization, and present an empirical evaluation on their effect in training RBMs.

Recently, a form of Tikhonov-type regularization was successfully applied to auto-encoders, which are closely related to RBMs [22], in [18,17] to explicitly encourage

A.E.P. Villa et al. (Eds.): ICANN 2012, Part I, LNCS 7552, pp. 81–88, 2012.

hidden variables to be invariant to (small) deformation of input representations. It was done by regularizing the squared derivative of a latent variable with respect to an input variable, which makes it a modified form of Tikhonov-type regularization.

## 2 Restricted Boltzmann Machines

The restricted Boltzmann machine is a stochastic neural network with a bipartite structure such that each visible neuron is connected to all the hidden neurons and each hidden neuron is connected to all the visible ones [20].

We define a log-probability assigned to a given visible vector $\mathbf{v}$ by an RBM as:

$$\log p(\mathbf{v} \mid \boldsymbol{\theta}) = f(\mathbf{v} \mid \boldsymbol{\theta}) + \sum_{j=1}^{N_h} \log \left( 1 + \exp \left( c_j + \sum_{i=1}^{N_v} w_{ij} \frac{v_i}{\sigma^2} \right) \right) - \log Z(\boldsymbol{\theta}), \quad (1)$$

where $\mathbf{v}$ and $\mathbf{h}$ are a column vector and a binary column vector representing the state of the visible and hidden neurons, and parameters $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b}, \mathbf{c}, \boldsymbol{\sigma})$ include weights $\mathbf{W} = [w_{ij}]_{N_v \times N_h}$, biases $\mathbf{b} = [b_i]_{N_v \times 1}$, $\mathbf{c} = [c_j]_{N_h \times 1}$ and standard-deviations $\boldsymbol{\sigma} = [\sigma_i]_{N_v \times 1}$. $N_v$ and $N_h$ are the numbers of visible and hidden neurons, respectively. $Z(\boldsymbol{\theta})$ denotes the normalizing constant which is intractable and it can be calculated by summing exponentially many terms.

Function $f(\mathbf{v} \mid \boldsymbol{\theta})$ in (1) indicates a contribution of visible neurons' biases to an energy of an RBM. $f(\mathbf{v} \mid \boldsymbol{\theta})$ together with $\sigma_i$ decides whether a visible neuron may have a binary value or a continuous real value.

When $f(\mathbf{v} \mid \boldsymbol{\theta}) = \mathbf{b}^\top \mathbf{v}$, it requires a visible neuron to have either 0 or 1, making a standard binary RBM [20]. In this case each $\sigma_i$ is set to 1. On the other hand, each visible neuron can have a continuous real value when $f(\mathbf{v} \mid \boldsymbol{\theta}) = -\sum_{i=1}^{N_v} \frac{(v_i - b_i)^2}{2\sigma_i^2}$, and each $\sigma_i$ can either be set to a pre-defined value or learned [2,10]. We call this model a Gaussian-Bernoulli RBM (GRBM).

Given a training set $\{\mathbf{v}^{(n)}\}_{n=1}^N$ an RBM can be trained by maximizing log-likelihood. The maximization is usually done by the stochastic gradient method, and in this paper, we use the recently introduced method of the enhanced gradient [4] together with parallel tempering [3,7].

### 2.1 Regularization for Restricted Boltzmann Machines

There are a number of regularization techniques that are widely used.

One most widely used technique is *weight-decay* regularization. Training an RBM with the weight-decay regularization maximizes the following objective function:

$$\mathcal{L}(\boldsymbol{\theta}) - \frac{\beta_w}{2} \sum_{ij} w_{ij}^2, \quad (2)$$

where $\mathcal{L}(\boldsymbol{\theta})$ and $\beta_w$ are the log-likelihood function and the regularization constant, respectively [1].

[1] The weight-decay may be applied to visible and hidden biases, as we have done in this paper.

Another widely used technique, called *sparsity regularization*, regularizes the average activation probability of each hidden neuron. An RBM trained using the sparsity regularization is commonly referred to as sparse RBM (sRBM) [14]. Sparse RBMs have been popular due to the fact that an RBM with low average hidden activation probabilities can extract better discriminative features than non-regularized RBMs (see, e.g., [16,5]). In [14], the sRBM was introduced by modifying the objective function to

$$\mathcal{L}(\boldsymbol{\theta}) - \frac{\beta_s}{2}\sum_{j=1}^{N_h}\left(\rho - \frac{1}{N}\sum_{n=1}^{N} p(h_j \mid \mathbf{v}^{(n)}, \boldsymbol{\theta})\right)^2, \tag{3}$$

where $\rho$ and $\beta_s$ are a target average activation of each hidden neuron and the regularization constant, respectively.

## 3 Tikhonov Regularization for Restricted Boltzmann Machines

In this section, we present two possible formulations of the Tikhonov-type regularization for RBMs. We refer to them as **TYPE-1** and **TYPE-2** formulations, respectively.

### 3.1 TYPE-1 and TYPE-2 Regularizations

One basic approach of the Tikhonov-type regularization is to minimize

$$\frac{\beta}{2}\mathbb{E}_{p(\mathbf{v})}\left[\|\nabla_{\mathbf{v}} y(\mathbf{v})\|^2\right], \tag{4}$$

when the task is to approximate some function $y(\mathbf{v})$ (see, e.g., [1,9]) of inputs $\mathbf{v}$. Here, $\beta$ is a regularization parameter. $p(\mathbf{v})$ can be defined by a set of training samples or be approximated by a probabilistic model.

Intuitively, by minimizing the derivative of the approximating function, Eq. (4) keeps the function smooth around training samples or around regions of high probability. In other words, it makes function $y(\mathbf{v})$ more invariant to (small) deformations of $\mathbf{v}$.

Under this intuition, it is natural to use as the approximating function $y(\mathbf{v})$ the probability density function $p(\mathbf{v})$ learned by an RBMs. Thus, the RBM model distribution is regularized to be smoother.

After replacing $y(\mathbf{v})$ with Eq. (1), we get the following **TYPE-1** regularization term:

$$J_1 = \frac{\beta}{2}\mathbb{E}_{p(\mathbf{v})}\left[\sum_{i=1}^{N_v}\left(\frac{\partial}{\partial v_i}\log p(\mathbf{v} \mid \boldsymbol{\theta})\right)^2\right] \approx \frac{\beta}{2N}\sum_{n=1}^{N}\sum_{i=1}^{N_v}\left(\frac{\partial f(\mathbf{v}^{(n)} \mid \boldsymbol{\theta})}{\partial v_i} + \sum_{j=1}^{N_h}\frac{w_{ij}}{\sigma_i^2}h_j^{(n)}\right)^2,$$

where $\mathbf{v}^{(n)}$ is either an empirical sample or a sample drawn from the model distribution[2] and $h_j^{(n)}$ is a short-hand notation for $p(h_j = 1 \mid \mathbf{v}^{(n)}, \boldsymbol{\theta})$.

[2] In the experiments, we used the samples from the model distribution which are readily available when computing the gradients.

Instead, we may formulate the Tikhonov-type regularization by regularizing the derivative of another function. From Eq. (1) it is apparent that an RBM is a special-case of product-of-experts models [12], which implies that a probability given to $\mathbf{v}$ by an RBM consists of contributions from experts which, in the case of RBMs, are hidden neurons. Hence, it is reasonable to regularize each contribution of a hidden neuron by minimizing the derivative of the logarithmic conditional probability distribution of each hidden neuron $\log p(h_j \mid \mathbf{v}, \boldsymbol{\theta})$[3].

The **TYPE-2** Tikhonov regularization is then formulated to minimize the following term:

$$J_2 = \frac{\beta}{2}\mathbb{E}_{p(\mathbf{v})}\left[\sum_{i=1}^{N_v}\sum_{j=1}^{N_h}\left(\frac{\partial}{\partial v_i}\log h_j^{(n)}\right)^2\right] \approx \frac{\beta}{2N}\sum_{n=1}^{N}\sum_{i=1}^{N_v}\sum_{j=1}^{N_h}\left(\frac{w_{ij}}{\sigma_i^2}h_j^{(n)}\right)^2. \quad (5)$$

In the case of a standard binary RBM, it is easy to see that the derivatives in both formulations are not well-defined as $\mathbf{v}$ is a binary vector. However, we can simply assume that $p(\mathbf{v} \mid \boldsymbol{\theta})$ has a domain of $\mathbb{R}^{N_v}$ instead, which is obviously followed by $p(h_j \mid \mathbf{v}, \boldsymbol{\theta})$ having the same domain[4].

The idea behind this choice is that a probability distribution defined by a binary RBM is constructed by taking values of all $\mathbf{v}$ such that each component of $\mathbf{v}$ is restricted to be either 0 or 1. Hence, we make the distribution defined by the RBM smoother by smoothing another continuous distribution with the same probability density function.

It is easy to see that both **TYPE-1** and **TYPE-2** can be seen as a combination of the weight-decay and sparsity regularizations. Both terms decrease when the absolute $l^2$-norm of each weight and the average activation probability of each hidden neuron decrease.

### 3.2 Optimization

A straightforward way to train an RBM with one of the two types of the Tikhonov-type regularization is to optimize the regularization term together with the log-likelihood. However, this approach makes it difficult to utilize the enhanced gradient which has been shown to perform better than the traditional gradient [4]. Hence, we follow the approach introduced in [14]. At each iteration, following the normal stochastic update of the parameters using the enhanced gradient we update the parameters $w_{ij}$, $b_i$ and $c_j$ again according to one of the regularization terms computed with the current minibatch.

## 4 Experiments

In this section we try to see the effect of the proposed regularization. From here on we refer to an RBM trained using the proposed Tikhonov regularization as a regularized RBM (rRBM). Note that in this paper we only focus on a standard RBM which constraints each visible neuron to be binary. *rRBMt1* and *rRBMt2* indicate the **TYPE-1**

[3] As noted earlier, a similar idea has been applied to auto-encoders in [18,17].

[4] This assumption does not need to be made in case of RBMs with continuous visible neurons.

and **TYPE-2** regularization, respectively. Additionally, we tested the weight-decay and sparsity regularization techniques in order to see how they perform differently compared to the proposed Tikhonov-type regularization. They are denoted by *wRBM* and *sRBM*, respectively.

We take a look at three metrics that can explain the effect of the proposed regularization term. We trained RBMs with 500 hidden neurons on two different data sets which are the handwritten digits (MNIST) [13] and the Caltech-101 Silhouettes [15]. As they have been quite well studied previously, we can easily compare to results obtained by other researchers.

Firstly, log-probabilities of test samples are checked. It may happen that the log-probabilities become larger for the rRBMs, as smoothing could potentially decrease the peaks around training samples systematically resulting in higher probability being assigned to nearby test samples.

Secondly, we consider classification accuracies using the learned features from an RBM, which indirectly suggests how discriminative extracted features are. In order to see how discriminative features were, we did not fine-tune the already trained weights of RBMs.

Additionally, in order to see how the proposed scheme biases a resulting model we check the average hidden activation probabilities given test samples. It can be expected that rRBMs will achieve higher sparsity.

For each data set we chose a regularization constant $\beta$ through validation. We grid-searched from $2^{-8}$ to $2^{-20}$ and estimated log-probabilities of validation samples. For each grid point five RBMs were trained for a small number of epochs with different random initializations, we considered their medians. Starting from a large $\beta$ we logarithmically decreased it until the log-probabilities of validation samples stopped increasing or decreasing significantly. Then, we chose the largest $\beta$ with the converged performance and sparsity.

We followed the same validation strategy to choose $\beta_w$ for RBMs trained using the weight-decay regularization. For sparse RBMs, we chose the target sparsity $\rho$ through validation. $\rho$ was grid-searched from $2^{-1}$ to $2^{-8}$, and $\rho$ with the best log-probabilities was chosen. The regularization constant $\beta_s$ was fixed to the inverse of the target sparsity, as recommended by [14].

Finally, we chose $2^{-16}$ and $2^{-17}$ for MNIST with the **TYPE-1** and **TYPE-2**, respectively. For Caltech-101 Silhouettes, $2^{-17}$ was chosen for both formulations of the Tikhonov-type regularization. $\beta_w$ for the weight-decay was chosen to be $2^{-14}$ and $2^{-11}$ for MNIST and Caltech-101 Silhouettes. $2^{-2}$ and $2^{-5}$ were chosen to be the target sparsity $\rho$ for MNIST and Caltech-101 Silhouettes, respectively.

For initializing parameters, we followed the strategy recommended in [11]. Each weight $w_{ij}$ was drawn from a zero-mean normal distribution with its variance $\frac{1}{\sqrt{N_v+N_h}}$. Visible biases $\mathbf{b}$ were set according to the training samples, and hidden biases $\mathbf{c}$ were initialized to negative values $(-4)$ in order to encourage sparse hidden activation probabilities.

We independently trained RBMs five times with different parameters initializations. Each RBM was trained for 200 epochs and 3000 epochs for MNIST and Caltech-101 Silhouettes, which amount to about 93,800 and 99,000 updates, respectively.

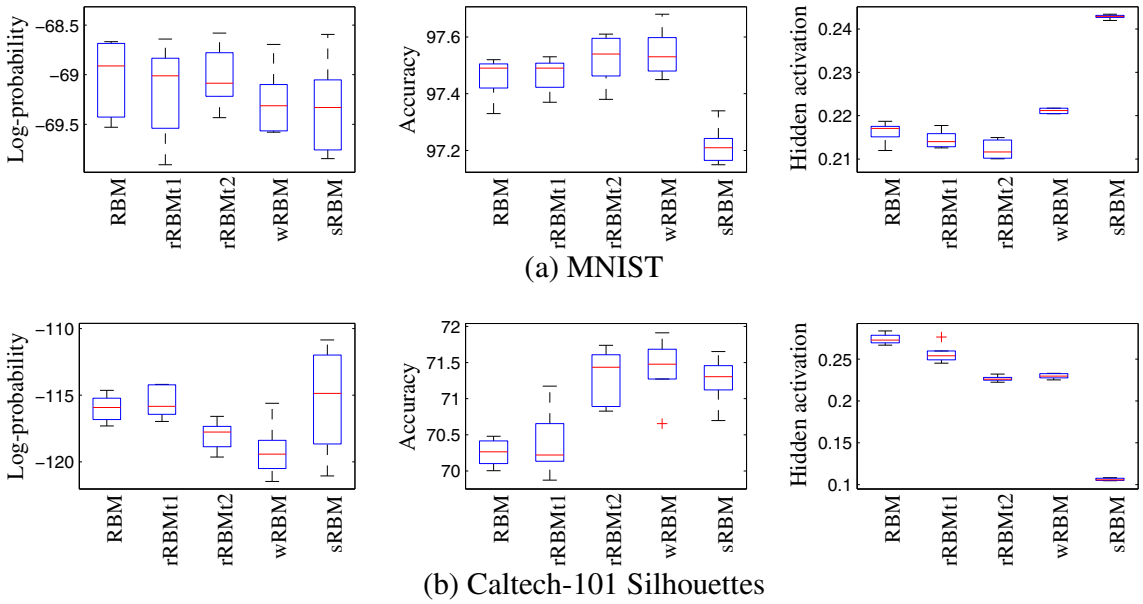



**Fig. 1.** Log-probabilities, classification accuracies and average hidden activation probabilities of test samples computed from the RBMs trained on MNIST and Caltech-101 Silhouettes with the proposed Tikhonov regularization schemes (*rRBMt1* and *rRBMt2*) and without it (*RBM*)

Log-probabilities were computed using a normalizing constant estimated using the AIS [19]. A simple logistic classifier was trained on hidden activation probabilities to compute classification accuracies. We used parallel tempering to sample from the model distribution [7,3], and used the enhanced gradient and the adaptive learning rate, with both an initial learning rate and an upper-bound set to $0.1$, proposed in [2]. For each experiment we decreased the learning rate proportionally to the inverse of the number of updates for the last half of training.

### 4.1 Result

In Fig. 1, we see the log-probabilities and the classification accuracies of the test samples and the average activation probabilities of the hidden neurons given the test samples.

The most obvious difference between the non-regularized RBM and the rRBMs is the lower average hidden activation probabilities given test samples [5]. As discussed previously the proposed regularization schemes resulted in a model with sparser hidden activation probabilities. It is also noticeable that **TYPE-2** tends to bias a resulting model to have sparser hidden activation probabilities even compared to the RBMs trained using the **TYPE-1** regularization or the RBMs trained with the weight-decay.

A general trend of extracting better discriminative features could be observed when the RBMs were regularized with either the **TYPE-1** or **TYPE-2** schemes. It was especially obvious with the **TYPE-2** regularization while the use of the **TYPE-1** formulation gave only marginal improvement over the non-regularized RBMs.

[5] Inconsistently high or low average hidden activation probabilities achieved by the sparse RBMs are due to the fact that the target sparsity $\rho$ was chosen by the validation to be as high as $2^{-2} = 0.25$ for MNIST and as low as $2^{-5} = 0.0312$ for Caltech-101 Silhouettes.

On the other hand, it could be observed that the proposed regularization schemes were not able to improve the resulting models' generative performance. In the case of MNIST, it could be seen that the better discriminative performance was achieved with slight degradation in the log-probabilities of test samples. However, in the case of Caltech-101 Silhouettes, we could observe that better generative models were learned using the **TYPE-1** scheme. It indirectly suggests that smoothing the overall probability distribution (1) could potentially improve the generalization of the model by removing highly peaked probability mass on training samples, while smoothing a contribution from each hidden neuron does not necessarily help.

## 5 Discussion

We have presented two possible types of the Tikhonov-type regularization for training RBMs in this paper. Both the **TYPE-1** and **TYPE-2** schemes prefer an RBM to learn a smoother probability distribution by minimizing the derivatives of either log-probability distribution or log-conditional distribution of hidden neurons. It was shown that both types were formulated as a combination of the weight-decay and sparsity regularizations which are widely used when training RBMs.

The experiments showed that both types were able to extract better discriminative features with sparser hidden activation probabilities while marginally sacrificing the generative capability of the resulting RBMs. The trend was more visible with the **TYPE-2** scheme, while it was not so apparent with the **TYPE-1** regularization. However, we were not able to see any significant performance improvement over other conventional regularization techniques with binary RBMs.

We noticed through the validation step of the experiments that the regularization constant $\beta$ needs to be carefully chosen. Too large $\beta$ overly simplified the distribution learned by an RBM and failed to give a good fit of a training distribution. More thorough investigation in choosing an appropriate regularization constant will need to be done. Regardlessly, the proposed formulations of the Tikhonov regularization reduce the number of hyper-parameters from at least three ($\rho$, $\beta_s$, and $\beta_w$) to one ($\beta$) against using both the weight-decay and the sparse regularization together.

We tested the proposed regularization schemes with a standard, binary RBM only. Both schemes, however, are not restricted to an RBM, but applicable to any other variant that can explicitly sum out hidden variables. One such variant is a GRBM which replaces a binary visible neuron with a continuous, real-valued visible neuron. Another possibility is a recently introduced spike-and-slab RBM [6]. It is natural to test the proposed method with this model as a next step in future work.

## References

1. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics), 1st edn. Springer (2006); 2nd printing edn. (October 2007)
2. Cho, K., Ilin, A., Raiko, T.: Improved learning of gaussian-bernoulli restricted boltzmann machines. In: Honkela, T. (ed.) ICANN 2011, Part I. LNCS, vol. 6791, pp. 10–17. Springer, Heidelberg (2011)

3. Cho, K., Raiko, T., Ilin, A.: Parallel tempering is efficient for learning restricted boltzmann machines. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN 2010), Barcelona, Spain (July 2010)
4. Cho, K., Raiko, T., Ilin, A.: Enhanced Gradient and Adaptive Learning Rate for Training Restricted Boltzmann Machines. In: Proceedings of the Twenty-Seventh International Conference on Machine Learning, ICML 2011 (2011)
5. Coates, A., Lee, H., Ng, A.Y.: An analysis of single-layer networks in unsupervised feature learning. In: AISTATS (2011)
6. Courville, A., Bergstra, J., Bengio, Y.: Unsupervised models of images by spike-and-slab rbms. In: Getoor, L., Scheffer, T. (eds.) Proceedings of the 28th International Conference on Machine Learning (ICML 2011), pp. 1145–1152. ACM, New York (2011)
7. Desjardins, G., Courville, A., Bengio, Y., Vincent, P., Delalleau, O.: Parallel Tempering for Training of Restricted Boltzmann Machines. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 145–152 (2010)
8. Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., Bengio, S.: Why does unsupervised pre-training help deep learning? Journal of Machine Learning Research (2010)
9. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice-Hall (July 1998)
10. Hinton, G.E., Salakhutdinov, R.R.: Reducing the Dimensionality of Data with Neural Networks. Science 313(5786), 504–507 (2006)
11. Hinton, G.: A Practical Guide to Training Restricted Boltzmann Machines. Tech. rep., Department of Computer Science, University of Toronto (2010)
12. Hinton, G.E.: Training products of experts by minimizing contrastive divergence. Neural Comput. 14, 1771–1800 (2002)
13. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE 86, 2278–2324 (1998)
14. Lee, H., Ekanadham, C., Ng, A.: Sparse deep belief net model for visual area V2, pp. 873–880 (2008)
15. Marlin, B.M., Swersky, K., Chen, B., de Freitas, N.: Inductive Principles for Restricted Boltzmann Machine Learning. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 509–516 (2010)
16. Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., Ng, A.Y.: Multimodal deep learning. In: International Conference on Machine Learning (ICML), Bellevue, USA (June 2011)
17. Rifai, S., Dauphin, Y.N., Vincent, P., Bengio, Y., Muller, X.: The manifold tangent classifier. In: Shawe-Taylor, J., Zemel, R.S., Bartlett, P., Pereira, F.C.N., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 24, pp. 2294–2302 (2011)
18. Rifai, S., Vincent, P., Muller, X., Glorot, X., Bengio, Y.: Contractive Auto-Encoders: Explicit Invariance During Feature Extraction. In: Getoor, L., Scheffer, T. (eds.) Proceedings of the 28th International Conference on Machine Learning (ICML 2011), pp. 833–840. ACM, New York (2011)
19. Salakhutdinov, R.: Learning Deep Generative Models. Ph.D. thesis, University of Toronto (2009)
20. Smolensky, P.: Information processing in dynamical systems: foundations of harmony theory. In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations, pp. 194–281. MIT Press, Cambridge (1986)
21. Tieleman, T.: Training restricted Boltzmann machines using approximations to the likelihood gradient. In: Proceedings of the 25th International Conference on Machine Learning, ICML 2008, pp. 1064–1071. ACM, New York (2008)
22. Vincent, P.: A connection between score matching and denoising autoencoders. Neural Computation 23(7), 1661–1674 (2011)

# Publication 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.**

# Improved Learning of Gaussian-Bernoulli Restricted Boltzmann Machines

KyungHyun Cho, Alexander Ilin and Tapani Raiko

Department of Information and Computer Science
Aalto University School of Science, Finland
{firstname.lastname@aalto.fi}

**Abstract.** We propose a few remedies to improve training of Gaussian-Bernoulli restricted Boltzmann machines (GBRBM), which is known to be difficult. Firstly, we use a different parameterization of the energy function, which allows for more intuitive interpretation of the parameters and facilitates learning. Secondly, we propose parallel tempering learning for GBRBM. Lastly, we use an adaptive learning rate which is selected automatically in order to stabilize training. Our extensive experiments show that the proposed improvements indeed remove most of the difficulties encountered when training GBRBMs using conventional methods.

**Keywords:** Restricted Boltzmann Machine, Gaussian-Bernoulli Restricted Boltzmann Machine, Adaptive Learning Rate, Parallel Tempering

## 1 Introduction

Conventional restricted Boltzmann machines (RBM) [1, 17] define the state of each neuron to be binary, which seriously limits their application area. One popular approach to address this problem is to replace the binary visible neurons with the Gaussian ones. The corresponding model is called Gaussian-Bernoulli RBM (GBRBM) [8]. Unfortunately, training GBRBM is known to be a difficult task (see, e.g. [9, 11, 12]).

In this paper, we propose a few improvements to the conventional training methods for GBRBMs to overcome the existing difficulties. The improvements include another parameterization of the energy function, parallel tempering learning, which has previously been used for ordinary RBMs [6, 5, 3], and the use of an adaptive learning rate, similarly to [2].

## 2 Gaussian-Bernoulli RBM

The energy of GBRBM [8] with real-valued visible neurons $\mathbf{v}$ and binary hidden neurons $\mathbf{h}$ is traditionally defined as

$$E(\mathbf{v}, \mathbf{h}|\theta) = \sum_{i=1}^{n_v} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{n_v}\sum_{j=1}^{n_h} W_{ij} h_j \frac{v_i}{\sigma_i} - \sum_{j=1}^{n_h} c_j h_j, \tag{1}$$

where $b_i$ and $c_j$ are biases corresponding to hidden and visible neurons, respectively, $W_{ij}$ are weights connecting visible and hidden neurons, and $\sigma_i$ is the standard deviation associated with a Gaussian visible neuron $v_i$ (see e.g. [11]).

The traditional gradient-based update rules are obtained by taking the partial derivative of the log-likelihood function $\log \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}|\theta))$, in which the hidden neurons are marginalized out, with respect to each model parameter. However, training GBRBMs even using well-defined gradients is generally difficult and takes long time (see, e.g., [11, 12]). One of the main difficulties is learning the variance parameters $\sigma_i$, which are, unlike other parameters, are constrained to be positive. Therefore, in many existing works, those parameters are often fixed to unity [9, 11, 15].

# 3 Improved Learning of Gaussian-Bernoulli RBM

## 3.1 New Parameterization of the Energy Function

The traditional energy function in (1) yields somewhat unintuitive conditional distribution in which the noise level defined by $\sigma_i$ affects the conditional mean of the visible neuron. In order to change this, we use a different energy function:

$$E(\mathbf{v}, \mathbf{h}|\theta) = \sum_{i=1}^{n_v} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{n_v}\sum_{j=1}^{n_h} W_{ij} h_j \frac{v_i}{\sigma_i^2} - \sum_{j=1}^{n_h} c_j h_j. \tag{2}$$

Under the modified energy function, the conditional probabilities for each visible and hidden neurons given the others are

$$p(v_i = v|\mathbf{h}) = \mathcal{N}\left(v \mid b_i + \sum_j h_j W_{ij}, \sigma_i^2\right),$$

$$p(h_j = 1|\mathbf{v}) = \text{sigmoid}\left(c_j + \sum_i W_{ij} \frac{v_i}{\sigma_i^2}\right),$$

where $\mathcal{N}(\cdot \mid \mu, \sigma^2)$ denotes the Gaussian probability density function with mean $\mu$ and variance $\sigma^2$. The update rules for the parameters are, then,

$$\nabla W_{ij} = \left\langle \frac{1}{\sigma_i^2} v_i h_j \right\rangle_{\text{d}} - \left\langle \frac{1}{\sigma_i^2} v_i h_j \right\rangle_{\text{m}}, \tag{3}$$

$$\nabla b_i = \left\langle \frac{1}{\sigma_i^2} v_i \right\rangle_{\text{d}} - \left\langle \frac{1}{\sigma_i^2} v_i \right\rangle_{\text{m}}, \tag{4}$$

$$\nabla c_j = \langle h_j \rangle_{\text{d}} - \langle h_j \rangle_{\text{m}}, \tag{5}$$

where a shorthand notations $\langle \cdot \rangle_d$ and $\langle \cdot \rangle_m$ denote the expectation computed over the data and model distributions accordingly [1].

Additionally, we use a different parameterization of the variance parameters: $\sigma_i^2 = e^{z_i}$. Since we learn log-variances $z_i = \log \sigma_i^2$, $\sigma_i$ is naturally constrained to stay positive. Thus, the learning rate can be chosen with less difficulty. Under the modified

energy function, the gradient with respect to $z_i$ is

$$\nabla z_i = e^{-z_i} \left( \left\langle \frac{1}{2}(v_i - b_i)^2 - \sum_j v_i h_j w_{ij} \right\rangle_{\mathrm{d}} - \left\langle \frac{1}{2}(v_i - b_i)^2 - \sum_j v_i h_j w_{ij} \right\rangle_{\mathrm{m}} \right).$$

### 3.2 Parallel Tempering

Parallel tempering (PT) learning. However, applying the same methodology to GBRBM is not straightforward: For example, a naive approach of multiplying $\sigma_i$ with the temperature results in the base model with zero variances for the visible neurons, or scaling the energy function by temperature would yield infinite variances. Here, we follow the methodology of [3].

In order to overcome this problem, we propose a new scheme for constructing the intermediate models with inverse temperatures $\beta$ such that

$$W_{ij}^{(t)} = \beta W_{ij}, \qquad b_i^{(t)} = \beta b_i + (1-\beta) m_i,$$
$$c_j^{(t)} = \beta c_j, \qquad \sigma_i^{(\beta)} = \sqrt{\beta \sigma_i^2 + (1-\beta) s_i^2},$$

where $W_{ij}$, $b_i$ and $c_j$ are the parameters of the current model, and $m_i$ and $s_i^2$ are the overall mean and variance of the $i$-th visible component in the training data.

The intermediate model is thus an interpolation between the base model and the current model, where the base model consists of independent Gaussian variables fitted to the training data.

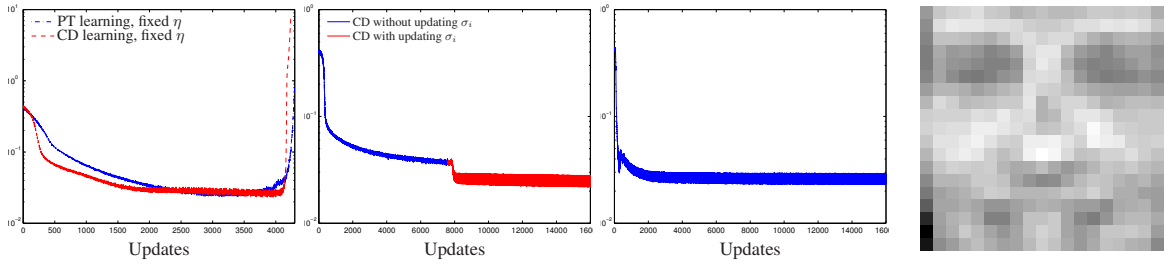
### 3.3 Adaptive Learning Rate

Many recent papers [2, 16, 7] point out that training RBM is sensitive to the choice of learning rate $\eta$ and its scheduling. According to our experience, GBRBM tends to be even more sensitive to this choice compared to RBM. It will be shown later that, if the learning rate is not annealed towards zero, GBRBM can easily diverge in the late stage of learning.

The adaptive learning rate proposed in [2] addresses the problem of automatic choice of the learning rate. The adaptation scheme proposed there is based on an approximation of the likelihood that is valid only for small enough learning rates. In this work, we use the same adaptive learning rate strategy but we introduce an upper-bound for the learning rate so that the approximation does not become too crude.

## 4 Experiments

In all the experiments, we used the following settings. The weights were initialized to uniform random values between $\pm \frac{1}{n_v + n_h}$. Biases $b_i$ and $c_j$ were initialized to zero and variances $\sigma_i$ to ones. Adaptive learning rate candidates (see [2]) were $\{0.9\eta, \eta, 1.1\eta\}$, where $\eta$ is the previous learning rate. In PT learning, we used 21 equally spaced $\beta \in \{0, 0.05, \ldots, 1\}$, and in CD learning, we used a single Gibbs step.




(a) (b) (c) (d)

**Fig. 1.** (a)-(c): The reconstruction errors obtained by training GBRBM using a learning rate fixed to 0.001 (a), with the adaptive learning rate while updating variances from the 650-th epoch using CD learning (b) and using PT learning (c). (d): Visualization of the learned variances.

### 4.1 Learning Faces

The CBCL data [13] used in the experiment contains 2,429 faces and 4,548 non-faces as training set and 472 faces and 23,573 non-faces as test set. Only the faces from the training set of the CBCL data were used.

In the first experiment, we trained two GBRBMs with 256 hidden neurons using both CD and PT learning with the learning rate fixed to 0.001 while updating all parameters including $\sigma_i^2$. As can be observed from Fig. 1(a), learning diverged in both cases (CD and PT learning), which is manifested in the increasing reconstruction error. This result confirms that GBRBMs are sensitive to the learning rate scheduling. The divergence became significant when the variances decreased significantly (not shown in Fig. 1(a)), indirectly indicating that the sensitivity is related to learning the variances.

**Learning Variances is Important** We again trained GBRBMs with 256 hidden neurons by CD and PT learning. The upper-bound and the initial learning rate were set to 0.01 and 0.0001, respectively.

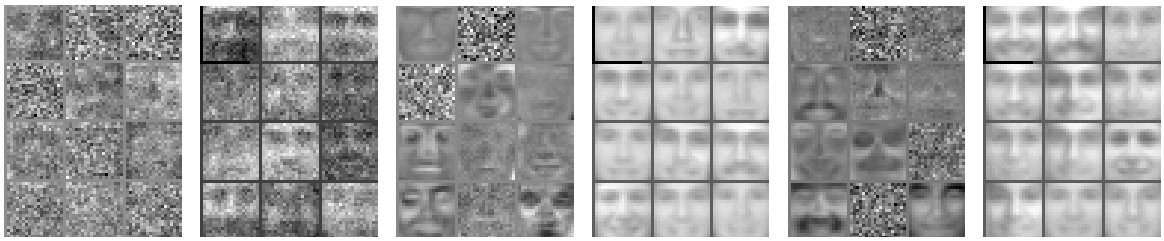
Initially, the variances of the visible neurons were not updated, but fixed to 1. The training was performed for 650 epochs. Afterwards, the training was continued for 1000 epochs, however, with updating variances.

Fig. 2(a) shows the learned filters and the samples generated from the GBRBM after the first round of training. The reconstruction error nearly converged (see the blue curve of Fig. 1(b)), but it is clear to see that both the filters and the samples are very noisy. However, the continued training significantly reduced the noise from the filters and the samples, as shown in Fig. 2(b).

From Fig. 1(b), it is clear that learning variances decreased the reconstruction error significantly. The explanation could be that the GBRBM has learned the importance, or noisiness, of pixels so that it focuses on the important ones.

The visualization of the learned variances in Fig. 1(d) reveals that important parts for modeling the face , for example, eyes and mouth, have lower variances while those of other parts are higher. Clearly, since the important parts are rather well modeled, the noise levels of corresponding visible neurons are lower.

**Parallel Tempering** In order to see if the proposed scheme of PT learning works well with GBRBM, an additional experiment using PT learning was conducted under the same setting, however, now updating the variances from the beginning.



(a) (b) (c)

**Fig. 2.** Example filters (left) and samples (right) generated by GBRBM trained using CD learning without updating variances (a), continued with updating variances (b), and trained using PT learning with updating variances from the beginning (c). 12 randomly chosen filters are shown, and between each consecutive samples 1000 Gibbs sampling steps were performed.

The observation of Fig. 1(c) suggests that learning variances from the beginning helps. It is notable that the learning did not diverge as the adaptive learning rate could anneal the learning rate appropriately.

The samples were generated from the trained GBRBM. Comparing the samples in the right figures of Fig. 2(a)–(c) suggests that the GBRBM trained using PT learning provides more variety of distinct samples, which indirectly suggests that the better generative model was learned by PT learning.

### 4.2 Learning Natural Images

CIFAR-10 data set [11] consists of three-channel (R, G, B) color images of size $32 \times 32$ with ten different labels.

**Learning Image Patches** In this experiment, the procedure proposed in [14] is roughly followed which was successfully used for classification tasks [11, 12, 4]. The procedure, first, trains a GBRBM on small image patches.

Two GBRBMs, each with 300 hidden neurons, following the modified energy function were trained on $8 \times 8$ images patches using CD and PT learning for 300 and 200 epochs, respectively.

Fig. 3 visualizes the filters learned by the GBRBMs. Apparently, the filters with the large norms mostly learn the global structure of the patches, whereas those with smaller norms tend to model more fine details. It is notable that this behavior is more obvious in the case of PT learning, whereas in the case of CD learning, the filters with the small norms mostly learned not-so-useful global structures.

The learned variances $\sigma_i^2$ of different pixels $i$ were distributed in $[0.0308\ 0.0373]$ and $[0.0283\ 0.0430]$ in case of CD and PT learning. In both cases, they were smaller than those of the training samples $s_i^2$, lying between $0.0547$ and $0.0697$. This was expected and is desirable [11].

**Classifying Natural Images** The image patches were preprocessed with independent component analysis (ICA) [10] and were transformed to vectors of 64 independent components each. Then, they were used as training data for GBRBMs. GBRBMs had 200 or 300 binary hidden neurons, and were trained by persistent CD learning [18] with

(a) CD learning (b) PT learning

**Fig. 3.** (a) two figures visualize 128 filters with the largest norms and 128 filters with the smallest norms of the GBRBM trained using CD learning, and (b) same figures obtained from PT learning.

a fixed learning rate $\eta = 0.005$ and variances fixed to one. The minibatch of size 20 was used, and we denote this model ICA+GBRBM.

Afterwards, 49 patches were extracted from each image in a convolutional way, and the hidden activations were obtained for each patch. Those activations were concatenated to form a feature vector which was used for training a logistic regression classifier.

The best classification accuracy of 63.75% was achieved with ICA+GBRBM having 64 independent components and 300 hidden neurons after training the GBRBM for only about 35 epochs. The obtained accuracy is comparable to the accuracies from the previous research. Some of them using the variants of RBM include 63.78% by GBRBM with whitening [11], and 68.2% obtained by the mean and covariance RBM with principal component analysis [14].

Also, slightly worse accuracies were achieved when the raw pixels of the image patches were used. Using the filters obtained in the previous experiment, 55.20% (CD) and 57.42% (PT) were obtained. This suggests that it is important to preprocess samples appropriately.

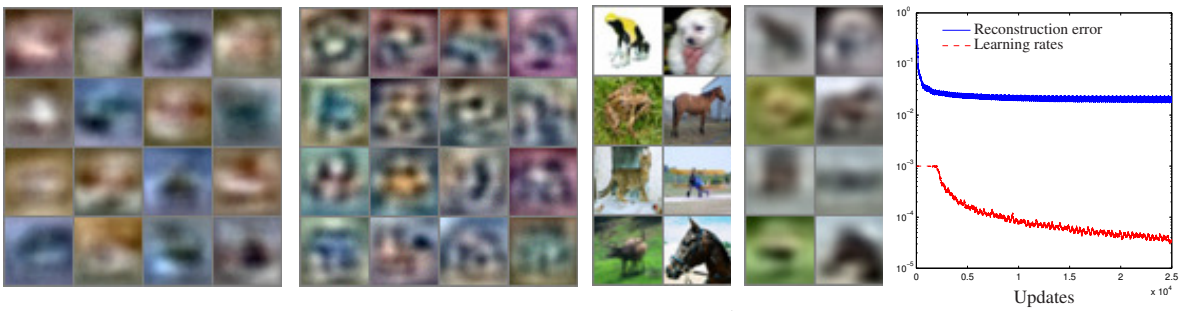
**Learning Whole Images** Due to the difficulty in training GBRBM, only data sets with comparably small dimensions have been mainly used in various recent papers. In case of CIFAR-10 the GBRBM was unable to learn any meaningful filters from whole images in [11].

In this experiment, a GRBM with 4000 hidden neurons was trained on whole images of CIFAR-10. It was expected that learning the variances, which became easier due to the proposed improvements, would encourage GBRBM to learn interesting interior features. CD learning with the adaptive learning rate was used. The initial learning rate and the upper-bound were set to 0.001. The training lasted for 70 epochs, and the minibatch of size 128 was used.

As shown in Fig. 4(a) the filters with the large norms tend to model the global features such as the position of the object, whereas the filters with the smaller norms model fine details, which coincides with the filters of the image patches. It is notable that the visualized filters do not possess those global, noisy filters (see Fig. 2.1 of [11]).

This visualization shows that the proposed improvements in training GBRBMs prevents the problem raised in [12] that a GBRBM easily fails to model the whole images by focusing mostly on the boundary pixels only.

Also, according to the evolution of the reconstruction error in Fig. 4(c), the learning proceeded stably. The red curve in the same plot suggests that the adaptive learning rate was able to anneal the learning rate automatically.




(a) (b) (c)

**Fig. 4.** (a): Two figures visualize 16 filters each with the largest norms (left) and the least norms (right) of the GBRBM trained on the whole images of CIFAR-10. (b): Two figures visualize original images (left) and their reconstructions (right). (c): The evolution of the reconstruction error and the learning rate.

Looking at Fig. 4(b), it is clear that the GBRBM was able to capture the essence of the training samples. The reconstructed images look like the blurred versions of the original ones while maintaining the overall structures. Apparently, both the boundary and the interior structure are rather well maintained.

## 5 Discussion

Based on the widely used GBRBM, we proposed a modified GBRBM which uses a different parameterization of the energy function. The modification led to the perhaps more elegant forms for visible and hidden conditional distributions given each other and gradient update rules.

We, then, applied two recent advances in training an RBM, PT learning and the adaptive learning rate, to a GBRBM. The new scheme of defining the tempered distributions for applying PT learning to GBRBM was proposed. The difficulty of preventing the divergence of learning was shown to be addressed by the adaptive learning rate with some practical considerations, for example, setting the upper bound of the learning rate.

Finally, the use of GBRBM and the proposed improvements were tested through the series of experiments on realistic data sets. Those experiments showed that a GBRBM and the proposed improvements were able to address the practical difficulties such as the sensitivity to the learning parameters and the inability of learning meaningful features from high dimensional data.

Despite these successful applications of GBRBM presented in this paper, training GBRBM is still more challenging than training a RBM. Further research in improving and easing the training is required.

**Acknowledgements** This work was supported by the summer internship and the honours programme of the department, by the Academy of Finland and by the IST Program of the European Community, under the PASCAL2 Network of Excellence. This publication only reflects the authors' views.

## References


1. Ackley, D.H., Hinton, G.E., Sejnowski, T.J.: A learning algorithm for Boltzmann machines. Cognitive Science 9, 147–169 (1985)
2. Cho, K.: Improved Learning Algorithms for Restricted Boltzmann Machines. Master's thesis, Aalto University School of Science (2011)
3. Cho, K., Raiko, T., Ilin, A.: Parallel tempering is efficient for learning restricted boltzmann machines. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN 2010). Barcelona, Spain (July 2010)
4. Coates, A., Lee, H., Ng, A.Y.: An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In: NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning (2010)
5. Desjardins, G., Courville, A., Bengio, Y.: Adaptive Parallel Tempering for Stochastic Maximum Likelihood Learning of RBMs. In: NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning (2010)
6. Desjardins, G., Courville, A., Bengio, Y., Vincent, P., Delalleau, O.: Parallel Tempering for Training of Restricted Boltzmann Machines. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. pp. 145–152 (2010)
7. Fischer, A., Igel, C.: Empirical analysis of the divergence of Gibbs sampling based learning algorithms for restricted Boltzmann machines. In: Proceedings of the 20th international conference on Artificial neural networks: Part III. pp. 208–217. ICANN'10, Springer-Verlag, Berlin, Heidelberg (2010)
8. Hinton, G.E., Salakhutdinov, R.R.: Reducing the Dimensionality of Data with Neural Networks. Science 313(5786), 504–507 (July 2006)
9. Hinton, G.: A Practical Guide to Training Restricted Boltzmann Machines. Tech. rep., Department of Computer Science, University of Toronto (2010)
10. Hyvärinen, A., Karhunen, J., Oja, E.: Independent Component Analysis. Wiley-Interscience, 1 edn. (May 2001)
11. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., Computer Science Department, University of Toronto (2009)
12. Krizhevsky, A.: Convolutional Deep Belief Networks on CIFAR-10. Tech. rep., Computer Science Department, University of Toronto (2010)
13. MIT Center For Biological and Computation Learning: CBCL Face Database #1, `http://www.ai.mit.edu/projects/cbcl`
14. Ranzato, M.A., Hinton, G.E.: Modeling pixel means and covariances using factorized third-order Boltzmann machines. In: CVPR. pp. 2551–2558 (2010)
15. Salakhutdinov, R.: LEARNING DEEP GENERATIVE MODELS. Ph.D. thesis, University of Toronto (2009)
16. Schulz, H., Müller, A., Behnke, S.: Investigating Convergence of Restricted Boltzmann Machine Learning. In: NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning (2010)
17. Smolensky, P.: Information processing in dynamical systems: foundations of harmony theory. In: Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations, pp. 194–281. MIT Press, Cambridge, MA, USA (1986)
18. Tieleman, T.: Training restricted Boltzmann machines using approximations to the likelihood gradient. In: Proceedings of the 25th international conference on Machine learning. pp. 1064–1071. ICML '08, ACM, New York, NY, USA (2008)

# Publication 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.**

# Gaussian-Bernoulli Deep Boltzmann Machine

KyungHyun Cho, Tapani Raiko and Alexander Ilin
Department of Information and Computer Science,
Aalto University School of Science
Email: firstname.lastname@aalto.fi

***Abstract*—In this paper, we study a model that we call Gaussian-Bernoulli deep Boltzmann machine (GDBM) and discuss potential improvements in training the model. GDBM is designed to be applicable to continuous data and it is constructed from Gaussian-Bernoulli restricted Boltzmann machine (GRBM) by adding multiple layers of binary hidden neurons. The studied improvements of the learning algorithm for GDBM include parallel tempering, enhanced gradient, adaptive learning rate and layer-wise pretraining. We empirically show that they help avoid some of the common difficulties found in training deep Boltzmann machines such as divergence of learning, the difficulty in choosing right learning rate scheduling, and the existence of meaningless higher layers.**

## I. Introduction

Deep Boltzmann machine (DBM) [1] is a recent extension of the simple restricted Boltzmann machine (RBM) in which several RBMs are stacked on top of each other. Unlike in other models, such as deep belief network or deep autoencoders (see, e.g., [2], [3]), in DBM, each neuron in the intermediate layers receives both top-down and bottom-up signals, which facilitates propagation of uncertainty during the inference procedure [1]. The original DBM is constructed such that each visible neuron represents a binary variable, that is DBM learns distributions over binary vectors.

A popular approach to modeling real-valued data is normalizing each input variable to $[0, 1]$ and treating it as a probability (e.g., using a gray-scale value of a pixel as a probability [2], [4]). This approach is however restrictive and it fits best to bounded variables. In the original DBM [1], real-valued data are first transformed into binary codes by training a model called Gaussian-Bernoulli RBM (GRBM) [2], and DBM is learned for the binary codes extracted from the data. This approach showed promising results [1], [5], [6] but it may be beneficial to combine GRBM and DBM in a single model and allow their joint optimization.

In this paper, we study a Gaussian-Bernoulli deep Boltzmann machine (GDBM) which uses Gaussian units in the visible layer of DBM. Even though deriving stochastic gradient is rather easy for GDBM, the training procedure can easily run into problems without careful selection of the learning parameters. This is largely caused by the fact that GRBM is known to be difficult to tune, especially the variance parameters of the visible neurons (see, e.g., [7]). We propose an algorithm for training GDBM based on the improvements introduced recently for training GRBM [8]. Also, we discuss the universal approximator property of GDBM.

The rest of the paper is organized as follows. The GDBM model is introduced in Section II. In Section III, we present the training algorithm and explain how to compute the terms of the stochastic gradient using mean-field approximations and parallel tempering sampling. In Section IV-A, we describe the update rules that are invariant to the data representation and more robust to the initialization of the parameters. In Section IV-B, we describe how we adapt the learning rate using the ideas proposed in [9].

## II. Gaussian-Bernoulli Deep Boltzmann Machine

GDBM with a single visible layer and $L$ hidden layers is parameterized with weights $\mathbf{W}$ of synaptic connections between the visible layer and the first hidden layer, $\mathbf{W}^{(l)}$ between layers $l$ and $l+1$, biases $\mathbf{b}$ of the visible layer, $\mathbf{b}^{(l)}$ of each hidden layer $l$, and standard-deviations $\sigma_i$ of the visible neurons. For a certain state $[\mathbf{v}^T\ {\mathbf{h}^{(1)}}^T\ \cdots\ {\mathbf{h}^{(L)}}^T]^T$, the energy is defined as:

$$E(\mathbf{v}, \mathbf{h}^{(1)}, \cdots, \mathbf{h}^{(L)} \mid \boldsymbol{\theta}) = \sum_{i=1}^{N_v} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{N_v}\sum_{j=1}^{N_1} \frac{v_i}{\sigma_i^2} h_j^{(1)} w_{ij} - \sum_{l=1}^{L}\sum_{j=1}^{N_l} b_j^{(l)} h_j^{(l)} - \sum_{l=1}^{L-1}\sum_{j=1}^{N_l}\sum_{k=1}^{N_{l+1}} h_j^{(l)} h_k^{(l+1)} w_{jk}^{(l)}$$

and the corresponding probability is

$$p\left(\mathbf{v}, \left\{\mathbf{h}^{(l)}\right\}_{l=1}^{L} \middle| \boldsymbol{\theta}\right) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(-E\left(\mathbf{v}, \left\{\mathbf{h}^{(l)}\right\}_{l=1}^{L} \middle| \boldsymbol{\theta}\right)\right), \tag{1}$$

where $N_v$ and $N_l$ are the number of neurons in the visible layer and the $l$-th hidden layer, respectively, and $Z(\boldsymbol{\theta})$ is the normalizing constant. Note that we use the GRBM parameterization, including learning $z_i = \log \sigma_i^2$ instead of $\sigma_i$ directly, from [8]. Note also that GRBM is a special case of GDBM with $L = 1$.

The states of the neurons in the same layer are independent of each other given the adjacent upper and lower layers. The conditional probability of a visible neuron is

$$p(v_i|\mathbf{h}^{(1)}, \boldsymbol{\theta}) = \mathcal{N}\left(v_i \middle| \sum_{j=1}^{N_1} h_j^{(1)} w_{ij} + b_i, \sigma_i^2\right),$$

where $\mathcal{N}(\cdot \mid \mu, \sigma^2)$ is a probability density of Normal distribution with a mean $\mu$ and a standard deviation $\sigma$, and

the conditional probabilities of the hidden neurons are

$$P(h_j^{(1)}|\mathbf{v},\mathbf{h}^{(2)},\boldsymbol{\theta}) = f\left(\sum_{i=1}^{N}\frac{v_i}{\sigma_i^2}w_{ij} + \sum_{k=1}^{N_2} h_k^{(2)} w_{jk}^{(1)} + b_j^{(1)}\right),$$

$$P(h_j^{(l)}|\mathbf{h}^{(l-1)},\mathbf{h}^{(l+1)},\boldsymbol{\theta}) = f\left(\sum_{i=1}^{N_{l-1}} h_i^{(l-1)} w_{ij}^{(l-1)} + \sum_{k=1}^{N_{l+1}} h_k^{(l+1)} w_{jk}^{(l)} + b_j^{(l)}\right),$$

where $f(\cdot)$ is a sigmoid function and $N_{L+1} = 0$.

### A. GDBM is a universal approximator

GRBM belongs to the family of mixture of Gaussians (MoG) since its joint distribution can be factorized into $p(\mathbf{v},\mathbf{h}^{(1)}) = P(\mathbf{h}^{(1)})p(\mathbf{v} \mid \mathbf{h}^{(1)})$ where $P(\mathbf{h}^{(1)})$ is the mixture coefficients and $p(\mathbf{v} \mid \mathbf{h}^{(1)})$ is Gaussian. MoGs are known to be universal approximators [10]. However, the center points of the exponentially many Gaussians in the data space are defined by only a linear number of parameters w.r.t. the number of hidden units, so not all MoGs can be written as a GRBM.

Given a MoG, we could transform it into a GRBM if we further constrain that exactly one of the hidden units is active at a time, that is, $\sum_j h_j^{(1)} = 1$. We set the columns of $\mathbf{W}$ to match the center points of the MoG and $\mathbf{b}$ to $\mathbf{0}$, and $\mathbf{b}^{(1)}$ is set such that the marginals $P(\mathbf{h}^{(1)})$ would match the mixing coefficients of the MoG.

As the final step, we implement the restriction $\sum_j h_j^{(1)} = 1$ using another hidden layer. We set $N_2 = N_1$ and $b_j^{(2)} = -\omega$ for each $j$, $w_{ij}^{(1)} = 3\omega$ for all $i = j$ and $w_{ij}^{(1)} = -\omega$ for all $i \neq j$, and further subtract $\omega$ from each $b_i^{(1)}$. As $\omega$ goes to infinity, it is clear that the probability of all states where $\mathbf{h}^{(2)} \neq \mathbf{h}^{(1)}$ or $\sum_j h_j^{(1)} \neq 1$ goes to zero and other states implement the previous GRBM with the wanted restriction. We have thus shown that any MoG can be modeled with a GDBM, and GDBM is a universal approximator.

## III. Training GDBM

GDBM can be trained with the stochastic maximization of the likelihood, where the likelihood function is computed by marginalizing out all the hidden neurons. For each parameter $\theta$, the partial-derivative of the log-likelihood function is

$$\frac{\partial \mathcal{L}}{\partial \theta} \propto \left\langle \frac{\partial\left(-E(\mathbf{v}^{(t)},\mathbf{h} \mid \boldsymbol{\theta})\right)}{\partial\theta} \right\rangle_{\mathrm{d}} - \left\langle \frac{\partial\left(-E(\mathbf{v},\mathbf{h} \mid \boldsymbol{\theta})\right)}{\partial\theta} \right\rangle_{\mathrm{m}}, \tag{2}$$

where $\langle\cdot\rangle_{\mathrm{d}}$ and $\langle\cdot\rangle_{\mathrm{m}}$ denote the expectation over the data distribution $P(\mathbf{h} \mid \{\mathbf{v}^{(t)}\},\boldsymbol{\theta})$ and the model distribution $P(\mathbf{v},\mathbf{h} \mid \boldsymbol{\theta})$, respectively.$\{\mathbf{v}^{(t)}\}$ is a set of all the training samples.

### A. Computing expectation over the data distribution

Computing the first term of (2) is straightforward for restricted Boltzmann machines because in that model the hidden neurons are independent of each other given the visible neurons. However, this does not apply to GDBM and therefore one needs to use some sort of approximation. We employ the mean-field approximation that was used for training binary DBM in [1].

In the mean-field approximation, the visible neurons are fixed to a training data sample, and the state of each hidden neuron $h_j^{(l)}$ is described with its probability $\mu_j^{(l)}$ of being active, which is updated with the following fixed-point iterations:

$$\mu_j^{(l)} \leftarrow f\left(\sum_{i=1}^{N_{l-1}} \mu_i^{(l-1)} w_{ij}^{(l-1)} + \sum_{k=1}^{N_{l+1}} \mu_k^{(l+1)} w_{jk}^{(l)} + b_j^{(l)}\right),$$

Note that $N_0 = N_v$, $\mu_i^{(0)} = v_i/\sigma_i^2$, and the update rule for the top-most layer not contain the summation term $\sum_{k=1}^{N_{l+1}}$. Using the mean-field approximation is known to introduce a bias. However, this is a computationally efficient scheme to capture only one mode of the posterior distribution, which can be desirable in many practical applications [1].

### B. Computing expectation over the model distribution: Parallel Tempering approach

The second term of the gradient (2) can be computed using Markov-chain Monte-Carlo (MCMC) sampling. The original approach proposed in [11] uses persistent Gibbs sampling with only a few steps of sampling at each update. This is equivalent to persistent contrastive divergence (PCD) introduced for training RBM [12]. Unfortunately the persistent Gibbs sampling suffers from poor mixing of samples, which results in the fact that trained models may have probability mass in the areas which are not represented in the training data (false modes) [13], [11], [5], [14]. In our experiments, we were able to observe that learning can easily diverge when persistent Gibbs sampling is used (see Section V).

We therefore use parallel tempering recently proposed in the context of RBM and GRBM [13], [15], [8] as a sampling procedure for GDBM. Parallel tempering overcomes the poor-mixing problem by maintaining multiple chains of sampling with different temperatures. In the chain with a high temperature, particles are more likely to explore the state space easily, whereas particles in the chain with low temperatures more closely follow the target model distribution.

We define the tempered distributions by varying parameters $\boldsymbol{\theta}$ of the original GDBM (1). We denote by $\boldsymbol{\theta}_\beta$ the parameters of the intermediate models defined by *inverse* temperatures $\beta$, where $\beta = 0$ corresponds to the base (most diffuse) distribution and $\beta = 1$ corresponds to the target distribution defined by the original GDBM. Defining appropriate intermediate distributions is quite straightforward for binary RBM [13], [15] but it is not as trivial for models with real-valued visible units [8]. In this work, we use the tempering scheme

defined by the following choice of $\boldsymbol{\theta}_\beta$:

$$\begin{aligned}
\mathbf{b}_\beta &= \beta\mathbf{b} + (1-\beta)\mathbf{m},\\
\mathbf{b}_\beta^{(l)} &= \beta\mathbf{b}^{(l)} \quad (l \geq 1),\\
\sigma_{\beta,i} &= \sqrt{\beta\sigma_i^2 + (1-\beta)s_i^2},\\
\mathbf{W}_\beta^{(l)} &= \beta\mathbf{W}^{(l)},
\end{aligned}$$

where $\mathbf{m} = [m_i]_{i=1}^{N_v}$ and $s_i$ are the means and variances estimated from the samples obtained from *all the tempered distributions*. Thus, the base distribution is the Gaussian distribution fitted to the samples from all the intermediate chains. The proposed scheme assures that the swapping happens even if the target distribution diverges from the data distribution. According to our experiments, this results in more stable learning compared to the scheme proposed in [8].

Adapting the temperature during learning can improve mixing and therefore facilitate learning [16]. In our experiments, we adapt the temperatures so as to maintain that the numbers of particle swaps between the consecutive chains as equal as possible. This is done by adjusting the inverse temperatures $\{\beta_i, i = 1, \ldots, N_{\text{chains}}\}$ after every swapping round using the following heuristic:

$$\beta_i^{(t)} \leftarrow \eta\beta_i^{(t-1)} + (1-\eta)\frac{\sum_{j=1}^{i-1} n_j}{\sum_{k=1}^{N_{\text{chains}}-1} n_k},$$

where $\eta$ is the damping factor, $n_j$ denote the number of swaps between the chains defined by $\beta_j$ and $\beta_{j+1}$, and the hottest chain is kept at the initial temperature, that is $\beta_1 = 0$. This simple approach does not have much computational overhead and seems to improve learning, as we show in Section V.

The proposed scheme of adapting the base distribution and the temperatures seem to work well in practice, although it may introduce a bias. The analysis of the possible biases of this approach is a question for further research.

## IV. Improving the Training Procedure

In this section, we show how to improve training of GDBM by adapting several ideas introduced for training RBM and deep networks.

### A. Enhanced gradient for GDBM

Enhanced gradient was introduced recently to make the update rules of binary Boltzmann machines invariant to data representation [17], [9]. The gradient was derived by introducing bit-flipping transformations and making the update rule, which follows from (2), invariant to such transformations. It has been shown to improve learning of RBM by making the results less sensitive to the learning parameters and initialization.

The same ideas can be used for enhancing the gradient in the models with Gaussian visible neurons such as GRBM and GDBM. Instead of the bit-flipping transformations, one can transform the original model by shifting each visible unit as $\tilde{v}_i = v_i - \Delta_i$ and correcting the bias terms accordingly: $\tilde{b}_i = b_i + \Delta_i$ and $\tilde{b}_i^{(1)} = b_i^{(1)} - \sum_i^N \frac{\Delta_i}{\sigma_i^2} w_{ij}$, which would result in an equivalent model. Following the methodology of [9], one can select the shifting parameters $\Delta_i$ such that the resulting gradient w.r.t. weights $w_{ij}^{(l)}$ and biases $b_i^{(l)}$ do not contain the same terms. This yields the following update rules:

$$\begin{aligned}
\nabla_e\, w_{ij} &= \mathrm{Cov}_{\mathrm{d}}\left(\frac{v_i}{\sigma_i^2}, h_j^{(1)}\right) - \mathrm{Cov}_{\mathrm{m}}\left(\frac{v_i}{\sigma_i^2}, h_j^{(1)}\right),\\
\nabla_e\, w_{ij}^{(l)} &= \mathrm{Cov}_{\mathrm{d}}\left(h_i^{(l)}, h_j^{(l+1)}\right) - \mathrm{Cov}_{\mathrm{m}}\left(h_i^{(l)}, h_j^{(l+1)}\right)\\
&\qquad\qquad (1 \leq l < L),\\
\nabla_e\, b_i &= \nabla b_i - \sum_j \left\langle h_j^{(1)}\right\rangle_{\mathrm{dm}} \nabla_e\, w_{ij},\\
\nabla_e\, b_i^{(1)} &= \nabla b_i^{(1)} - \sum_j \left\langle h_j^{(2)}\right\rangle_{\mathrm{dm}} \nabla_e\, w_{ij}^{(1)} - \sum_k \left\langle v_k\right\rangle_{\mathrm{dm}} \nabla_e\, w_{ki},\\
\nabla_e\, b_i^{(l)} &= \nabla b_i^{(l)} - \sum_j \left\langle h_j^{(l+1)}\right\rangle_{\mathrm{dm}} \nabla_e\, w_{ij}^{(l)}\\
&\qquad - \sum_k \left\langle h_k^{(l-1)}\right\rangle_{\mathrm{dm}} \nabla_e\, w_{ki}^{(l-1)} (l > 1),
\end{aligned}$$

where $\left\langle h_j^{(l)}\right\rangle_{\mathrm{dm}} = \frac{1}{2}\left\langle h_j^{(l)}\right\rangle_{\mathrm{d}} + \frac{1}{2}\left\langle h_j^{(l)}\right\rangle_{\mathrm{m}}$ is the average activity of a neuron under the data and model distributions and $\langle v_i\rangle_{\mathrm{dm}} = \frac{1}{2}\left\langle v_i/\sigma_i^2\right\rangle_{\mathrm{d}} + \frac{1}{2}\left\langle v_i/\sigma_i^2\right\rangle_{\mathrm{m}}$. $\mathrm{Cov}_P(\cdot,\cdot)$ is a covariance between two variables under the distribution $P$ defined as

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

### B. Adaptive learning rate

The choice of the learning rate to be used with the stochastic gradient (2) greatly affects the training procedure [18], [19], [9]. In order to diminish this problem, we adopt the strategy of automatic adaptation of the learning rate, as proposed in [9]. The adaptation is done based on the estimate of the likelihood computed using the identity

$$p(\mathbf{v}_{\mathrm{d}}|\boldsymbol{\theta}_\eta) = \frac{p^*(\mathbf{v}_{\mathrm{d}}|\boldsymbol{\theta}_\eta)}{Z(\boldsymbol{\theta})}\left\langle\frac{p^*(\mathbf{v}|\boldsymbol{\theta}_\eta)}{p^*(\mathbf{v}|\boldsymbol{\theta})}\right\rangle_{p(\mathbf{v}|\boldsymbol{\theta})}^{-1}, \tag{3}$$

where $\boldsymbol{\theta}_\eta$ are the model parameters obtained by updating $\boldsymbol{\theta}$ with learning rate $\eta$, $p^*$ denotes an unnormalized pdf such that $p(\mathbf{v}|\boldsymbol{\theta}) = p^*(\mathbf{v}|\boldsymbol{\theta})/Z(\boldsymbol{\theta})$ and the required expectation is approximated using samples from $p(\mathbf{v}|\boldsymbol{\theta})$.

The unnormalized probabilities $p^*(\mathbf{v}|\boldsymbol{\theta})$ are obtained by integrating out the hidden neurons from the joint model:

$$p^*(\mathbf{v}|\boldsymbol{\theta}) = \sum_{\mathbf{h}} p^*(\mathbf{v}, \mathbf{h}),$$

which yields a simple analytical form in the case of RBM or GRBM. However, explicit integration of the hidden neurons is not tractable for GDBM and therefore one has to employ some approximations. We use the approximation

$$\sum_{\mathbf{h}} E(\mathbf{v}, \mathbf{h}) \approx E(\mathbf{v}, \boldsymbol{\mu}),$$

where $\boldsymbol{\mu}$ is the mean-field approximation of the hidden activations which is computed as discussed in Section III-A.

Note that we use different portions of data (mini-batches) for computing the stochastic gradient and adaptation of the

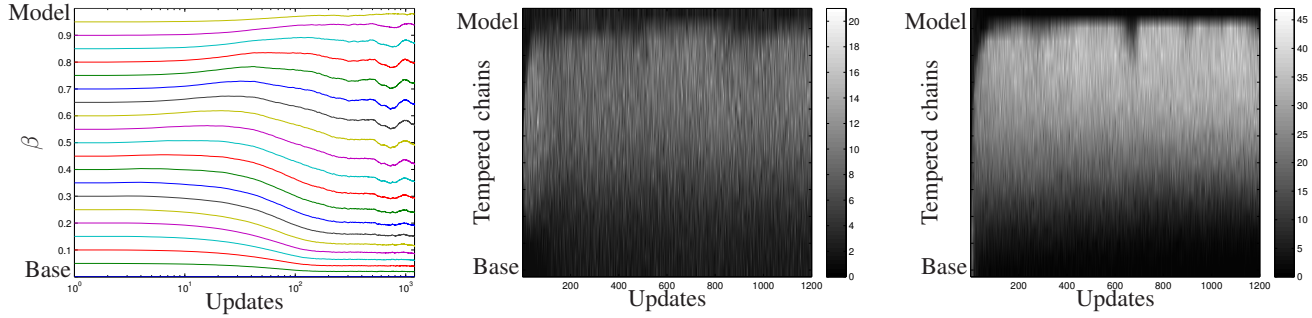



Fig. 1. The left figure shows the evolution of the inverse temperatures during the learning while $\beta_1$ is fixed to 0. The middle and right figures plot the number of swaps between a pair of consecutive samples at each update (swap) while the temperatures were adapted (middle) and the temperatures were fixed at the equally spaced levels (right).

learning rate, as was suggested in [9]. Therefore, the fixed-point iterations required for computing the mean-field approximation have to be run twice. However, initializing the mean-field values with samples from the model distribution seems to yield fast convergence. Also, making only a few fixed-point iterations (without convergence) seems to be enough to get stable behavior of the learning rate adaptation.

### C. Layer-wise pretraining

Layer-wise pretraining is commonly used in deep networks to help obtain better models by initializing weights sensibly [20]. DBM requires special care during the pretraining phase because the neurons in the intermediate layers receive signal both from the upper and the lower layers, unlike in deep belief networks [2]. Salakhutdinov proposed cope with this problem by halving the pretrained weights in the intermediate layers and duplicating the visible and topmost layers during the pretraining [11]. The pretrained GRBM containing the visible layer has the following energy:

$$E(\mathbf{v}, \mathbf{h}^{(1)} \mid \boldsymbol{\theta}) = \sum_i \frac{(v_i - b_i)^2}{2\sigma_i^2/N_v} - \sum_j c_j h_j^{(1)} - \sum_i \sum_j \frac{v_i}{\sigma_i^2 N_v} h_j^{(1)} w_{ij},$$

where $N_v = 2$ corresponds to duplicating the visible layer. Similarly, the topmost RBM during pretraining has the energy

$$E(\mathbf{h}^{(L-1)}, \mathbf{h}^{(L)} \mid \boldsymbol{\theta}) = -\sum_i b_i h_i^{(L-1)} - \sum_j (N_h c_j) h_j^{(L)} - \sum_i \sum_j h_i^{(L-1)} h_j^{(L)} (N_h w_{ij}^{(L-1)}),$$

where we also use $N_h = 2$.

## V. Experiments

We train the GDBM model on Olivetti face dataset [21]. Out of 400 faces, we used the first 350 faces of 35 people for training and the remaining ones of 5 people as test samples. We test the model in the problem of reconstructing the left half of the face from the right half. We compared the reconstruction results using the methodology presented in [22]. Note that the test set does not contain faces of people from the training set for better assessment of the generalization skills. The dataset was initially normalized such that each pixel has zero-mean and unit variance. We trained GDBM models with three hidden layers and 500 neurons in each hidden layer.

Unless specified otherwise, GDBM is trained using pretraining, enhanced gradient and adaptive learning rate. The size of a minibatch and the number of samples from the model distribution were both set to 64. The initial learning rate and its upper-bound were both set to $0.001$ for pretraining and to $0.0005$ for joint training of all the layers. Weight-decay of $0.005$ was used both during training RBM and pretraining DBM. The reconstruction was performed by fixing the known half of the image, computing the mean-field approximation of the posterior distribution over all the unknown neurons including the missing half of the image.

**Adaptive Learning Rate.** Our experiments confirmed that the learning rate was able to adapt automatically using the proposed strategy. The left plot in Figure 3 shows that the algorithm very quickly finds the appropriate region of learning rates. After that, the learning rate slowly decreases, which is a desired property in stochastic optimization (see, e.g., [23]).

**Parallel Tempering.** Parallel tempering yields pretty good results (see Fig. 4) while we were not able to achieve convergence of GDBM with PCD. Fig. 1 indicates that proposed scheme of temperature adaptation results in the increased and consistent number of swaps among all consecutive pairs of tempered chains, and hence, in better mixing.

**Enhanced Gradient and Pretraining.** One obvious way to check whether the higher layers of GDBM have learned any useful structure is to inspect the mean-field approximation values of the neurons in those layers given the training or test data samples. When no useful structure was learned, most hidden neurons in the top layer converge near $0.5$ which means that nearly no signal was received from the neighboring layers. On the other hands, when those neurons actually affect the modeling of the distribution, they converge to the values close to either $0$ or $1$.

One GDBM was trained without the layer-wise pretraining, starting from the random initialization. In this case it was clear that except for the first hidden layer no upper layer was able to learn any useful structure, as shown on the top row of Figure 2.

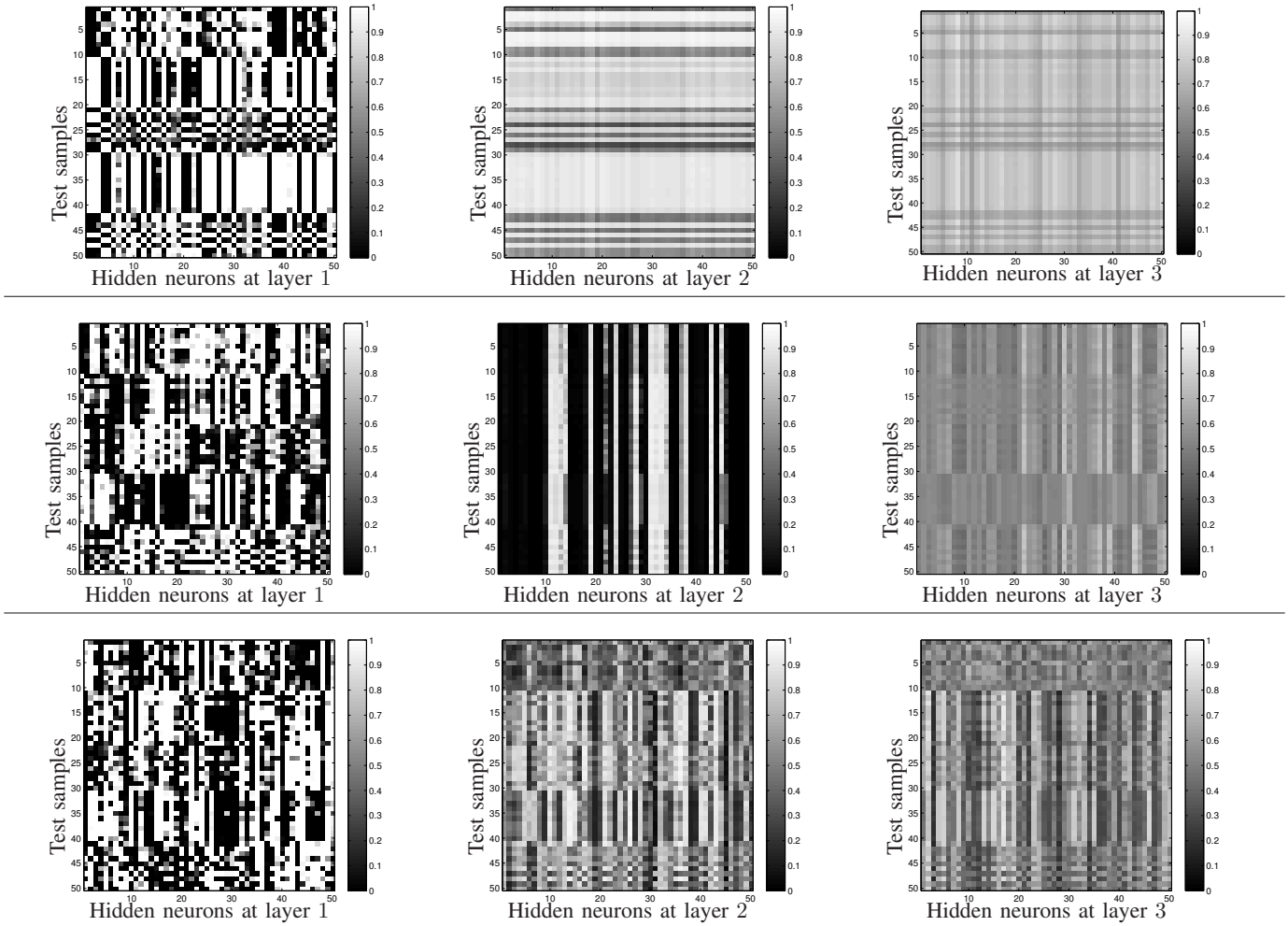



Fig. 2. The figures visualize the mean-field values of the hidden neurons at different hidden layers when the visible neurons are fixed to the test data samples. The top figures were obtained using the GDBM trained without any pretraining. The middle and bottom figures were obtained from the GDBM trained with both the pretraining and the joint-training using either using the traditional gradient or the enhanced gradient, respectively. All three GDBMs were trained for 215 epochs.

Most of approximated values of the hidden neurons in the second and third layers are near $0.5$.

We trained the second GDBM by first initializing it with layer-wise pretraining. However, this time, we did not use the enhanced gradient for either the pretraining or the joint training. Comparing with the top row of Figure 2, it is apparent from the figures of the middle row that the hidden neurons in the upper layers are approximated closer to $0$ or $1$ when the layers were pretrained. It suggests that the pretraining is important in a sense that it enables the upper layers to learn the structure of the data distribution.

However, we were able to observe that many hidden neurons in the higher layers (see the hidden layer 2, for instance) do not contribute much to the modeling of the distribution, as they are either always inactive ($= 0$) or always active ($= 1$). This behavior was already discovered in case of RBM, and it was shown that the enhanced gradient can resolve the problem [9].

Thus, we trained yet another GDBM now by using the enhanced gradient. The bottom row of Figure 2 clearly indicates that the enhanced gradient was able to address the problem. Now the hidden neurons in the layer 3 respond differently to the distinct test samples, and by doing so, encourages the flow of the uncertainty between the hidden layer 1 and the hidden layer 3, enabling the hidden neurons in the layer 3 to capture the structure also.

**Comparison with other models.** We trained principal component analysis (PCA) and GRBM on the same dataset. PCA used 100 principal components [22], and GRBM had 500 hidden neurons. We limited the number of the principal components to 100 because including more components resulted in stronger overfitting and larger reconstruction errors.

The right plot in Fig. 3 shows the evolution of the difference between the original test faces and the reconstructed faces for each model. It indicates that GRBM start overfitting quickly, which results in the increase of the reconstruction error of the unseen faces. It is also evident from the reconstructed faces in Fig. 4. Ultimately, GRBM performs worse than PCA


Fig. 4. Reconstructed test samples using PCA, GRBM and GDBM.

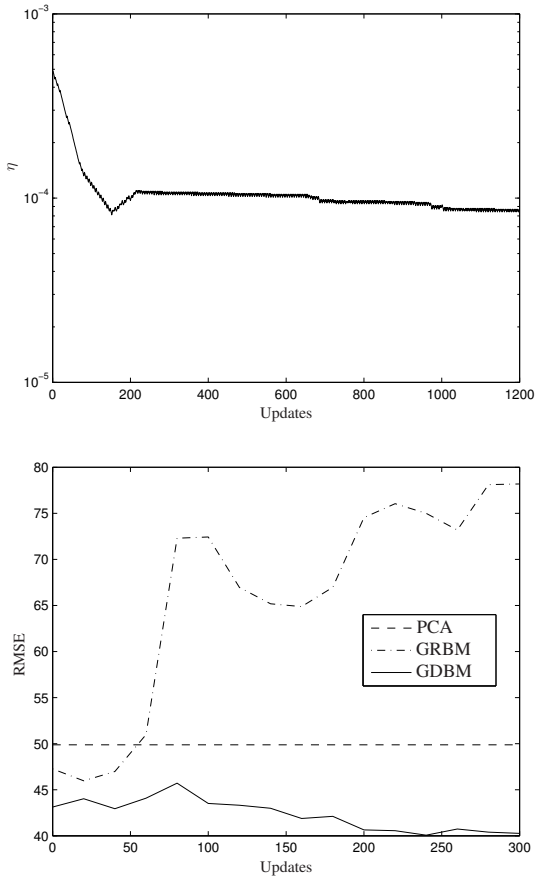



Fig. 3. The evolution of the learning rate when the adaptive learning rate was used (left) and the reconstruction errors using three different methods: PCA, GRBM, and GDBM (right).

evidenced by both RMSE and the visual inspection of the reconstructed faces.

GDBM trained only with pretraining using the identical GRBM training procedure, however, does not become overfitted and performs better than GRBM, which indicates that the additional hidden layers help perform and generalize better.

Furthermore, the joint training of GDBM by the proposed learning algorithm improves the performance significantly. It suggests that it is indeed important to jointly train all layers of GDBM in order to obtain a better generative model.

## VI. Discussion

In this paper, we described Gaussian-Bernoulli deep Boltzmann machine and discussed its universal approximator property. Based on the learning algorithm for the binary DBM [1], we adapted three improvements which are parallel tempering, enhanced gradient, and adaptive learning rate for training a GDBM. Through the experiments we were able to empirically show that GDBM trained using these improvements can achieve good generative performance.

Although they are not presented in this paper, using the same hyper-parameters for learning faces, we were able to train GDBM on other data sets such as NORB [24] and CIFAR-10 [7]. It clearly indicates that the proposed improvements make learning insensitive to the choice of the learning hyper-parameters and thus easier. However, the trained GDBMs were not able to produce any state-of-the-art classification accuracy. The discrepancy between the generative capability and the classification performance of GDBM is left for the future research.

Recently, several approaches have been proposed for efficiently training DBM, and it will be interesting to see how they perform when they are used for training GDBM compared to the learning algorithm proposed in this paper. Some of them are; adaptive MCMC sampling [5], tempered-transition [11], using a separate set of recognition weights [6], centering trick [25] and two-stage pretraining algorithm [26].

## REFERENCES


[1] R. Salakhutdinov and G. E. Hinton, "Deep Boltzmann machines," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, vol. 5, 2009, pp. 448–455.

[2] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504–507, July 2006.

[3] Y. Bengio, "Learning Deep Architectures for AI," *Found. Trends Mach. Learn.*, vol. 2, pp. 1–127, January 2009.

[4] B. Lake, R. Salakhudinov, J. Gross, and J. Tenenbaum, "One shot learning of simple visual concepts," in *Proceedings of the 33rd Annual Meeting of the Cognitive Science Society*, 2011.

[5] R. Salakhutdinov, "Learning Deep Boltzmann Machines using Adaptive MCMC," in *ICML*, J. Frnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 943–950.

[6] R. Salakhutdinov and H. Larochelle, "Efficient learning of deep boltzmann machines." *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 693–700, 2010.

[7] A. Krizhevsky, "Learning multiple layers of features from tiny images," Computer Science Department, University of Toronto, Tech. Rep., 2009.

[8] K. Cho, A. Ilin, and T. Raiko, "Improved Learning of Gaussian-Bernoulli Restricted Boltzmann Machines," in *Proceedings of the Twentith International Conference on Artificial Neural Networks*, ser. ICANN 2011, 2011.

[9] K. Cho, T. Raiko, and A. Ilin, "Enhanced Gradient and Adaptive Learning Rate for Training Restricted Boltzmann Machines," in *Proceedings of the Twenty-seventh International Conference on Machine Learning*, ser. ICML 2011, 2011.

[10] A. S. D.M. Titterington and U. Makov, *Statistical Analysis of Finite Mixture Distributions*. New York, London, Sydney: John Wiley & Sons, 1985.

[11] R. Salakhutdinov, "Learning in Markov Random Fields using Tempered Transitions," in *Advances in Neural Information Processing Systems 22*, Bengio, Y. and Schuurmans, D. and Lafferty, J. and Williams, C. K. I. and Culotta, A., Ed., 2009, pp. 1598–1606.

[12] T. Tieleman, *Training restricted Boltzmann machines using approximations to the likelihood gradient*, ser. ICML '08. New York, NY, USA: ACM, 2008.

[13] G. Desjardins, A. Courville, Y. Bengio, P. Vincent, and O. Delalleau, "Parallel Tempering for Training of Restricted Boltzmann Machines," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 145–152.

[14] K. Cho, "Improved Learning Algorithms for Restricted Boltzmann Machines," Master's thesis, Aalto University School of Science, 2011.

[15] K. Cho, T. Raiko, and A. Ilin, "Parallel tempering is efficient for learning restricted boltzmann machines," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2010)*, Barcelona, Spain, July 2010.

[16] 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*, 2010.

[17] T. Raiko, K. Cho, and A. Ilin, "Enhanced gradient for learning boltzmann machines (abstract)," in *The Learning Workshop*, Fort Lauderdale, Florida, April 2011.

[18] A. Fischer and C. Igel, "Empirical analysis of the divergence of Gibbs sampling based learning algorithms for restricted Boltzmann machines," in *Proceedings of the 20th international conference on Artificial neural networks: Part III*, ser. ICANN'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 208–217.

[19] H. Schulz, A. Müller, and S. Behnke, "Investigating Convergence of Restricted Boltzmann Machine Learning," in *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[20] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, 2010.

[21] F. Samaria and A. Harter, "Parameterisation of a stochastic model for human face identification," in *Proceedings of the Second IEEE Workshop on Applications of Computer Vision, 1994.*, dec 1994, pp. 138 –142.

[22] H. Poon and P. Domingos, "Sum-Product networks: A new deep architecture," in *Proceedings of the Twenty Seventh Conference on Uncertainty in Artificial Intelligence*, 2011.

[23] H. Kushner and G. Yin, *Stochastic Approximation and Recursive Algorithms and Applications*. Springer, 2003.

[24] Y. Lecun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings of CVPR'04*, vol. 2, 2004.

[25] G. Montavon and K.-R. Müller, "Deep Boltzmann machines and the centering trick," in *Neural Networks: Tricks of the trade, Reloaded*, 2nd ed., ser. LNCS, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Springer, 2012, vol. 7700.

[26] K. Cho, T. Raiko, A. Ilin, and J. Karhunen, "A Two-Stage Pretraining Algorithm for Deep Boltzmann Machines," in *NIPS 2012 Workshop on Deep Learning and Unsupervised Feature Learning*, Lake Tahoe, December 2012.

# Publication 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)*, Pages 106–113, September 2013.**

# A Two-Stage Pretraining Algorithm for Deep Boltzmann Machines

KyungHyun Cho, Tapani Raiko, Alexander Ilin and Juha Karhunen

Department of Information and Computer Science
Aalto University School of Science, Finland
{firstname.lastname@aalto.fi}

**Abstract.** A deep Boltzmann machine (DBM) is a recently introduced Markov random field model that has multiple layers of hidden units. It has been shown empirically that it is difficult to train a DBM with approximate maximum-likelihood learning using the stochastic gradient unlike its simpler special case, restricted Boltzmann machine (RBM). In this paper, we propose a novel pretraining algorithm that consists of two stages; obtaining approximate posterior distributions over hidden units from a simpler model and maximizing the variational lower-bound given the fixed hidden posterior distributions. We show empirically that the proposed method overcomes the difficulty in training DBMs from randomly initialized parameters and results in a better, or comparable, generative model when compared to the conventional pretraining algorithm.

**Keywords:** Deep Boltzmann Machine, Deep Learning, Pretraining

## 1 Introduction

Deep Boltzmann machine (DBM), proposed in [14], is a recently introduced variant of Boltzmann machines which extends widely used restricted Boltzmann machines (RBM) to a model that has multiple hidden layers. It differs from the popular deep belief network (DBN) [5] in that every edge in the DBM model is undirected. In this way, DBMs facilitate propagating uncertainties across multiple layers of hidden variables.

Although it is straightforward to derive a learning algorithm for DBMs using a variational approximation and stochastic maximum likelihood method, recent research (see, for example, [14, 4]) has shown that learning the parameters of DBMs is not trivial. Especially the generative performance of the trained model, commonly measured by the variational lower-bound of log-probabilities of test samples, tends to degrade as more hidden layers are added.

In [14] a greedy layer-wise pretraining algorithm was proposed to be used to initialize parameters of DBMs, and it was shown that it largely overcomes the difficulty of learning a good generative model.

Along this line of research, we propose another way to approach pretraining DBMs in this paper. The proposed scheme is based on an observation that training DBMs consists of two separate stages; approximating a posterior distribution over hidden units and updating parameters to maximize the lower-bound of log-likelihood given those states.

Based on this observation, our proposed method pretrains a DBM in two stages. During the first stage we train a simpler, directed deep model such as DBNs or stacked denoising autoencoders (sDAE) to obtain an approximate posterior distribution over hidden units. With this fixed approximate posterior distribution, we train an RBM that learns a distribution over a combination of data samples and their corresponding posterior distributions of hidden units. Finetuning the model is then trivial as one only needs to free hidden variables from the approximate posterior distribution computed during the first stage.

We show that the proposed algorithm helps learning a good generative model which is empirically comparable to the pretraining method proposed in [14]. Furthermore, we discuss the potential degrees of freedom in extending the proposed approach.

## 2 Deep Boltzmann Machines

We start by describing deep Boltzmann machines (DBM) [14]. A DBM with $L$ layers of hidden neurons is defined by the following energy function:

$$-E(\mathbf{v},\mathbf{h}\mid\boldsymbol{\theta}) = \sum_{i}^{N_v} v_i b_i + \sum_{i}^{N_v}\sum_{j}^{N_1} v_i h_j^{(1)} w_{i,j} + \sum_{j}^{N_1} h_j^{(1)} c_j^{(1)} + \sum_{l=2}^{L}\left(\sum_{j}^{N_l} h_j^{(l)} c_j^{(l)} + \sum_{j}^{N_l}\sum_{k}^{N_{l+1}} h_j^{(l)} h_k^{(l+1)} u_{j,k}^{(l)}\right), \tag{1}$$

where $\mathbf{v} = [v_i]_{i=1\ldots N_v}$ and $\mathbf{h}^{(l)} = \left[h_j^{(l)}\right]_{j=1\ldots N_l}$ are $N_v$ binary visible units and $N_l$ binary hidden units in the $l$-th hidden layer. $\mathbf{W} = [w_{i,j}]$ is the set of weights between the visible neurons and the first layer hidden neurons, while $\mathbf{U}^{(l)} = \left[u_{j,k}^{(l)}\right]$ is the set of weights between the $l$-th and $l+1$-th hidden neurons. $b_i$ and $c_j^{(l)}$ are a bias to the $i$-th visible neuron and the $j$-th hidden neuron in the $l$-th hidden layer, respectively. We use $\boldsymbol{\theta}$ to denote a set of all these parameters.

With the energy function, a DBM can assign a probability to each state vector $\mathbf{x} = [\mathbf{v};\mathbf{h}^{(1)};\cdots;\mathbf{h}^{(L)}]$ using a Boltzmann distribution $p(\mathbf{x}\mid\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})}\exp\{-E(\mathbf{x}\mid\boldsymbol{\theta})\}$. Based on this property the parameters can be learned by maximizing the log-likelihood $\mathcal{L} = \sum_{n=1}^{N}\log\sum_{\mathbf{h}} p(\mathbf{v}^{(n)},\mathbf{h}\mid\boldsymbol{\theta})$ given $N$ training samples $\{\mathbf{v}^{(n)}\}_{n=1,\ldots,N}$, where $\mathbf{h} = \left[\mathbf{h}^{(1)};\cdots;\mathbf{h}^{(L)}\right]$.

The gradient computed by taking the partial derivative of the log-likelihood function with respect to each parameter is used in most cases with a mini-batch per update. It is then used to update the parameters, effectively forming a stochastic gradient ascent method. A standard way of computing gradient results in the following update rule for each parameter $\theta$:

$$\nabla\theta = \left\langle -\frac{\partial E(\mathbf{v}^{(n)},\mathbf{h}\mid\boldsymbol{\theta})}{\partial\theta}\right\rangle_{\mathrm{d}} - \left\langle -\frac{\partial E(\mathbf{v},\mathbf{h}\mid\boldsymbol{\theta})}{\partial\theta}\right\rangle_{\mathrm{m}}, \tag{2}$$

where $\langle\cdot\rangle_{\mathrm{d}}$ and $\langle\cdot\rangle_{\mathrm{m}}$ denote the expectation over the data distribution $P(\mathbf{h}\mid\{\mathbf{v}^{(n)}\},\boldsymbol{\theta})$ and the model distribution $P(\mathbf{v},\mathbf{h}\mid\boldsymbol{\theta})$, respectively [3].

## 3 Training Deep Boltzmann Machines

Although the update rules in Eq. (2) are well defined, it is intractable to compute them exactly. Hence, an approach that uses variational approximation together with Markov chain Monte Carlo (MCMC) sampling was proposed in [14].

First, the variational approximation is used to compute the expectation over the data distribution. It starts by approximating $p(\mathbf{h} \mid \mathbf{v}, \boldsymbol{\theta})$, which is intractable unless $L = 1$, by a factorial distribution $Q(\mathbf{h}) = \prod_{l=1}^{L} \prod_{j=1}^{N_l} \mu_j^{(l)}$. The variational parameters $\mu_j^{(l)}$ can then be estimated by the following fixed-point equation:

$$\mu_j^{(l)} \leftarrow f\left(\sum_{i=1}^{N_{l-1}} \mu_i^{(l-1)} w_{ij}^{(l-1)} + \sum_{k=1}^{N_{l+1}} \mu_k^{(l+1)} w_{kj}^{(l)} + c_j^{(l)}\right), \tag{3}$$

where $f(x) = \frac{1}{1+\exp\{-x\}}$. Note that $\mu_i^{(0)} = v_i$ and the update rule for the top layer does not contain the second summation term, that is $N_{L+1} = 0$.

This variational approximation provides the values of variational parameters that maximize the following lower-bound with respect to the current parameters:

$$p(\mathbf{v} \mid \boldsymbol{\theta}) \geq \mathbb{E}_{Q(\mathbf{h})}\left[-E(\mathbf{v}, \mathbf{h})\right] + \mathcal{H}(Q) - \log Z(\boldsymbol{\theta}), \tag{4}$$

where

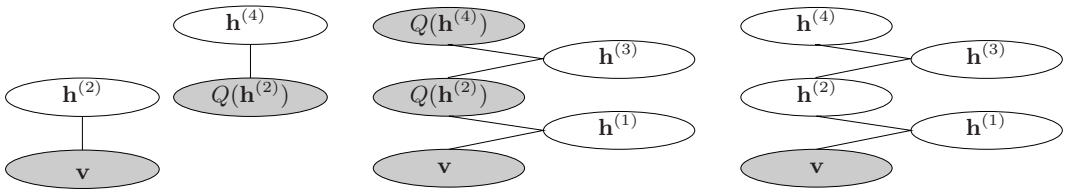
$$\mathcal{H}(Q) = -\sum_{l=1}^{L} \sum_{j=1}^{N_l} \left(\mu_j^{(l)} \log \mu_j^{(l)} + (1 - \mu_j^{(l)}) \log(1 - \mu_j^{(l)})\right)$$

is an entropy functional. Hence, each gradient update step does not increase the exact log-likelihood but its variational lower-bound.

Second, the expectation over the model distribution is computed by persistent sampling. The simplest approach is to use Gibbs sampling.

This approach closely resembles variational expectation-maximization (EM) algorithm (see, for example, [2]). Learning proceeds by alternating between finding the variational parameters $\boldsymbol{\mu}$ and updating the DBM parameters to maximize the given variational lower-bound using the stochastic gradient method. However, it has been known and will be shown in the experiments in this paper that training a DBM using this approach starting from randomly initialized parameters is not trivial [14, 4].

Hence, in [14] a pretraining algorithm to initialize the parameters of DBMs was proposed. The pretraining algorithm greedily trains each layer of a DBM by considering each layer as an RBM, following a pretraining approach used for training deep belief networks (DBN) [5]. However, due to the undirectedness of edges in DBMs it has been proposed to use the first layer RBM with two duplicate copies of visible units with tied weights and the last layer RBM with two duplicate copies of hidden units with tied weights. Once one layer has been trained, another layer can be trained on the aggregate posterior distribution of the hidden units of the lower layer to extend the depth. After the pretraining, learned weights are used as initializations of weights of DBMs.




(a) Stage 1 (b) Stage 2 (c) Finetuning

**Fig. 1.** Illustration of the two-stage pretraining algorithm followed by finetuning of all parameters. Shaded nodes indicate clamped variables whereas white nodes are free variables.

# 4 A Two-stage Pretraining Algorithm

In this paper, we propose an alternative way of initializing parameters of a DBM compared with the one described at the end of Section 3. We employ an approach that separately obtains posterior distributions over hidden units and initializes parameters.

Before proceeding to the description of the proposed algorithm, we first divide the hidden layers of a DBM into two sets. Let us denote a vector of hidden units in the odd-numbered layers as $\mathbf{h}_+$ and the respective vector in the even-numbered layers as $\mathbf{h}_-$. In this sense we may define $\boldsymbol{\mu}_+$ and $\boldsymbol{\mu}_-$ as variational parameters of the hidden units in the odd-numbered layers and the even-number layers, respectively.

**Stage 1**: We focus on finding a good set of variational parameters $\boldsymbol{\mu}_-$ of $Q(\mathbf{h}_-)$ that has a potential to give a reasonably high variational lower-bound in Eq. (4). In other words, we propose to first find a good posterior distribution over hidden units given a visible vector regardless of parameter values of a DBM. Although it might sound unreasonable to find a good set of variational parameters without any fixed parameter values, we can do this by *borrowing* posterior distributions over latent variables from other models.

DBNs and sDAE's, described in [5] and [16], are natural choices to find a good approximate posterior distribution over units in the even-numbered hidden layers. One justification for using either of them is that they can be trained efficiently and well (see, e.g., [1] and references therein). It becomes a trivial task as one can iteratively train each even-numbered layer as either an RBM or a DAE on top of each other, as is a common practice when a DBN or a sDAE is trained.

**Stage 2**: Once a set of initial variational parameters $\boldsymbol{\mu}_-$ is found from a DBN or an sDAE, we train a model that has predictive power of the variational parameters given a visible vector. It can be simply done by letting an RBM learn a joint distribution of $\mathbf{v}$ and $\boldsymbol{\mu}_-$.

The structure of the RBM can be directly derived from the DBM such that its visible layer corresponds to the visible layer and the even-numbered hidden layers of the DBM and its hidden layer to the odd-numbered hidden layers of the DBM. The connections between them can also follow those of the DBM. This corresponds to finding a set of DBM parameters that *fit* the variational parameters obtained in the first stage.

Once an RBM has been trained, we can use the learned parameters as initializations for training the DBM, which corresponds to freeing $\mathbf{h}_-$ from its variational posterior distribution obtained in the first stage.

A simple illustration of the two-stage pretraining algorithm is given in Fig. 1.

### 4.1 Discussion

It is quite easy to see that the proposed algorithm has high degree of freedom to plug in alternative algorithms and models in both stages.

The most noticeable flexibility can be found in Stage 1. Any other machine learning model that gives reasonable posterior distributions over multiple layers of binary hidden units can be used instead of RBMs or DAEs. Also, instead of stacking each layer at a time, one could opt to train deep autoencoders at once using advanced backpropagation algorithms (see, for instance, [10]).

In Stage 2, one may opt to use a DAE instead of an RBM. It will make learning faster and therefore leave more time for finetuning the model afterward. Also, the use of different algorithms for training an RBM can be considered. For quicker pretraining, one may use contrastive divergence [6] , or for better initial models, advanced MCMC sampling methods could be used.

Another obvious possibility is to utilize the conventional pretraining algorithm proposed in [14] during the first stage. This approach gives approximate posterior distributions over all hidden units as well as initial values of the parameters. In this way, one may use either an RBM or a fully visible BM (FVBM) during the second stage starting from the initialized parameters. When an RBM is used in the second stage, one could simply discard $\boldsymbol{\mu}_+$.

One important point of the proposed algorithm is that it provides another research perspective in training DBMs. The existing pretraining scheme developed in [14, 11] was based on the observation that under certain assumptions the variational lower-bound could be increased by learning weight parameters layer wise. However, the success of the proposed scheme suggests that it may not be the set of parameters that need to be directly pretrained, but the set of variational parameters that determine how tight the variational lower-bound is and their corresponding parameters.

## 5 Experiments

In the experiments, we train DBMs on two datasets which are a handwritten digit dataset (MNIST) [7] and Caltech-101 Silhouettes dataset [8]. We used the MNIST and Caltech-101 Silhouettes datasets because experimental results of using DBMs for both datasets are readily available for direct comparison [13, 12, 9].

We train DBMs with varying numbers of units in the hidden layers; 500-1000, 500-500-1000, 500-500-500-1000. The first two architectures were used in [13, 12], which enables us to directly compare our proposed algorithm with the conventional pretraining algorithm.

For learning algorithms, we extensively tried various combinations. They are presented in Table 1. In summary, a $DBM^{stage1}_{stage2}$ denotes a deep Boltzmann machine in which its superscript and subscript denote the algorithms used during the first and second stages, respectively.

We used contrastive divergence (CD) to train RBMs in the first stage, and the persistent CD [15] with coupled adaptive simulated annealing (CAST) was used in the second stage. DAEs were trained using stochastic backpropagation algorithm.

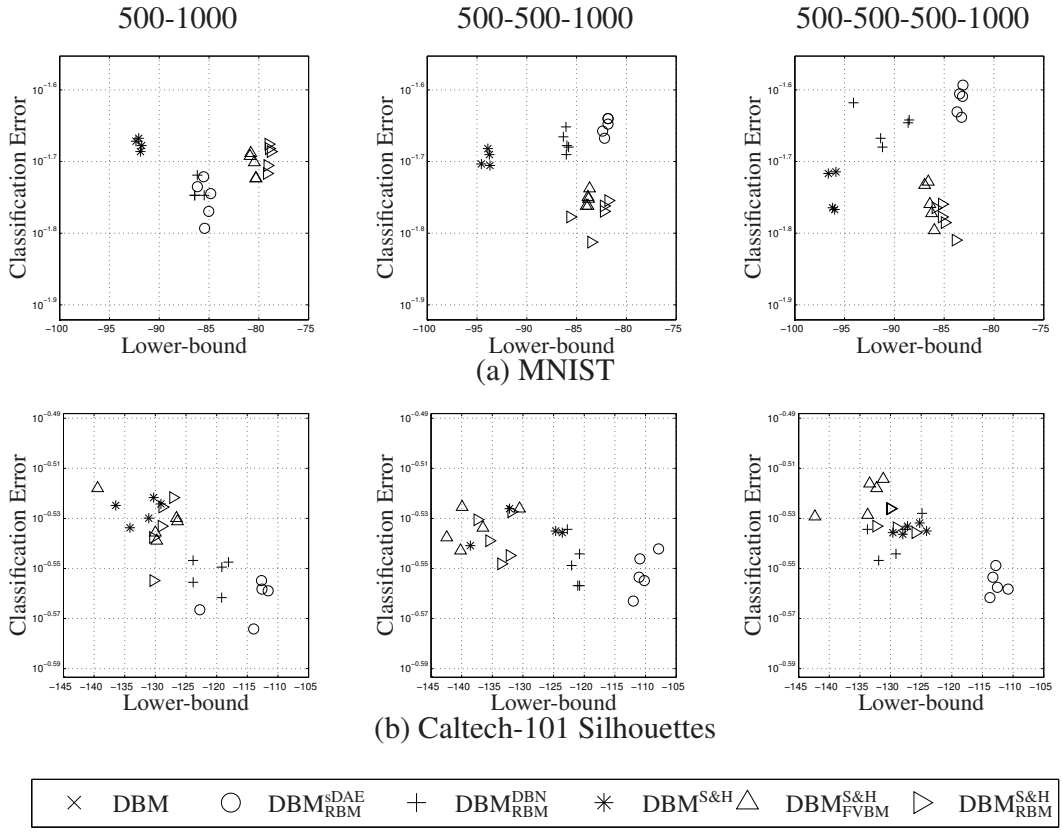



**Fig. 2.** Performance of the trained DBMs. Best performing models are in bottom right corners of each plot.

When a DBM was finetuned, we estimated the variational parameters by running at most 30 mean-field fixed-point updates. The model statistics, the negative part of the gradient, was computed by CAST.

| | Stage 1 | Stage 2 | Finetuning |
|---|---|---|---|
| DBM | $\times$ | $\times$ | DBM |
| DBM$^{\text{sDAE}}_{\text{RBM}}$ | sDAE | RBM | DBM |
| DBM$^{\text{DBN}}_{\text{RBM}}$ | DBN | RBM | DBM |
| DBM$^{\text{S\&H}}$ | (S) | $\times$ | DBM |
| DBM$^{\text{S\&H}}_{\text{RBM}}$ | (S) | RBM | DBM |
| DBM$^{\text{S\&H}}_{\text{FVBM}}$ | (S) | FVBM | DBM |

**Table 1.** Algorithms used in the experiment. (S) – the pretraining algorithm from [14].

We evaluated the resulting models with the variational lower-bound of log-probabilities and the classification error of test samples. The variational lower-bounds reflect the generative performance of the model. The classification accuracy computed from a linear support vector machine (SVM) tells us the discriminative property of the hidden units. We trained a linear SVM for each hidden layer $l$ using $\boldsymbol{\mu}_l$ as its features. This is expected to show how much information about input samples is captured by each hidden layer of the model.

All models were trained five times starting from different random initializations. We report medians over these random trials.

### 5.1 Result and Analysis

Fig. 2 presents the result using both the lower-bound of log-probabilities and the classification error of the test samples. As has already been expected, none of the models

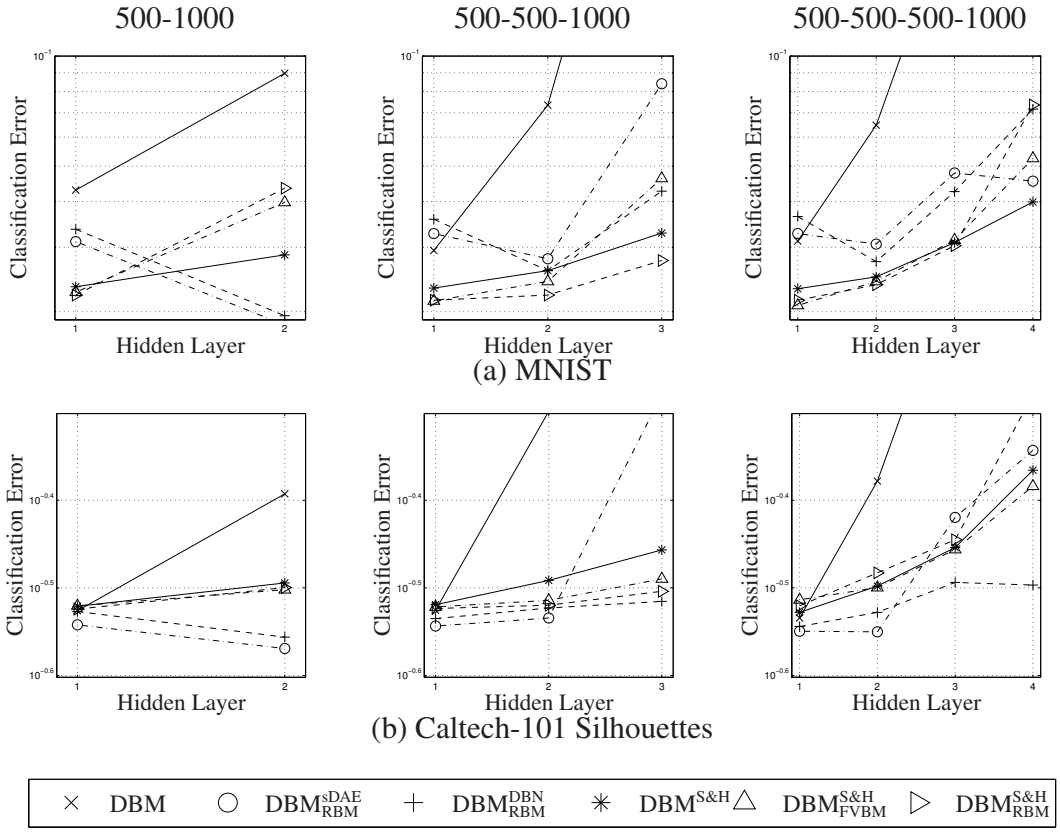



**Fig. 3.** Layer-wise Discriminative Performance. Lower is better.

trained *without* pretraining have been able to perform well enough to be presented inside the boundaries of the boxes in Fig. 2.

It is clear from the figures that the proposed two-stage pretraining algorithm outperforms, in all cases, the conventional pretraining algorithm (DBM$^{S\&H}$). On MNIST, the DBMs pretrained with the proposed algorithm using the conventional pretraining algorithm in the first stage achieved the best performance. In the case of Caltech-101 Silhouettes, DBM$^{sDAE}_{RBM}$was able to achieve superior performance in both generative and discriminative modeling. It is notable that without any pretraining (DBM) we were not able to achieve any reasonable performance.

Fig. 3 presents layer-wise classification errors. It is clear from the significantly lower accuracies in the higher hidden layers of the DBMs trained without pretraining that pretraining is essential to allow upper layers to capture structures of data. DBM$^{DBN}_{RBM}$and DBM$^{S\&H}_{RBM}$were most effective in ensuring the upper hidden layers to have better discriminative property.

## 6 Conclusions

The experimental success of the proposed two-stage pretraining algorithm in training DBMs suggests that the difficulty of DBM learning might be due to the fact that the estimated variational lower-bound at the initial stage of learning is too crude, or too loose. Once one initializes the variational parameters well enough by utilizing another deep

hierarchical model, the parameters of a DBM can be fitted to give a tighter variational lower-bound which facilitates jointly estimating all parameters.

The proposed two-stage pretraining algorithm provides a general framework in which many hierarchical deep learning models can be used. It even makes possible to include the conventional pretraining algorithm as a part of the proposed algorithm and improve upon it. This is a significant step in developing and improving a training algorithm for DBMs, as it allows us to fully utilize other learning algorithms that have been extensively studied previously.

## References

1. Bengio, Y., Courville, A., Vincent, P.: Representation Learning: A Review and New Perspectives. arXiv:1206.5538 [cs.LG] (Jun 2012)
2. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, corrected 2nd printing edn. (2007)
3. Cho, K.: Improved Learning Algorithms for Restricted Boltzmann Machines. Master's thesis, Aalto University School of Science (2011)
4. Desjardins, G., Courville, A., Bengio, Y.: On training deep Boltzmann machines. arXiv:1203.4416 [cs.NE] (Mar 2012)
5. Hinton, G., Salakhutdinov, R.: Reducing the dimensionality of data with neural networks. Science 313(5786), 504–507 (July 2006)
6. Hinton, G.: Training products of experts by minimizing contrastive divergence. Neural Computation 14, 1771–1800 (August 2002)
7. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE. pp. 2278–2324. No. 11
8. Marlin, B.M., Swersky, K., Chen, B., de Freitas, N.: Inductive principles for restricted Boltzmann machine learning. In: Proc. of the 13th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2010). pp. 509–516 (2010)
9. Montavon, G., Müller, K.R.: Deep Boltzmann machines and the centering trick. In: Montavon, G., Orr, G.B., Müller, K.R. (eds.) Neural Networks: Tricks of the trade, Reloaded, LNCS, vol. 7700. Springer, 2nd edn. (2012)
10. Raiko, T., Valpola, H., LeCun, Y.: Deep learning made easier by linear transformations in perceptrons. In: Proc. of the 15th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2012). La Palma, Canary Islands, Spain (April 2012)
11. Salakhutdinov, R., Hinton, G.E.: A Better Way to Pre-Train Deep Boltzmann Machines. In: Advances in Neural Information Processing Systems (2012)
12. Salakhutdinov, R.: Learning deep Boltzmann machines using adaptive MCMC. In: Fürnkranz, J., Joachims, T. (eds.) Proc. of the 27th Int. Conf. on Machine Learning (ICML 2010). pp. 943–950. Omnipress, Haifa, Israel (June 2010)
13. Salakhutdinov, R., Hinton, G.: An efficient learning procedure for deep Boltzmann machines. Tech. Rep. MIT-CSAIL-TR-2010-037, MIT (August 2010)
14. Salakhutdinov, R., Hinton, G.E.: Deep Boltzmann machines. In: Proc. of the Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2009). pp. 448–455 (2009)
15. Tieleman, T., Hinton, G.E.: Using fast weights to improve persistent contrastive divergence. In: Proceedings of the 26th Annual International Conference on Machine Learning. pp. 1033–1040. ICML '09, ACM, New York, NY, USA (2009)
16. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: 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)

# Publication 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)*, Pages 432–440, June 2013.**

# Simple Sparsification Improves Sparse Denoising Autoencoders in Denoising Highly Noisy Images

**KyungHyun Cho** KyungHyun.Cho@aalto.fi
Department of Information and Computer Science, Aalto University School of Science, Finland

## Abstract

Recently Burger et al. (2012) and Xie et al. (2012) proposed to use a denoising autoencoder (DAE) for denoising noisy images. They showed that a plain, deep DAE can denoise noisy images as well as the conventional methods such as BM3D and KSVD. Both of them approached image denoising by denoising small, image patches of a larger image and combining them to form a clean image. In this setting, it is usual to use the encoder of the DAE to obtain the latent representation and subsequently apply the decoder to get the clean patch. We propose that a simple sparsification of the latent representation found by the encoder improves denoising performance, both when the DAE was trained with and without sparsity regularization. The experiments confirm that the proposed sparsification indeed helps both denoising a small image patch and denoising a larger image consisting of those patches. Furthermore, it is found out that the proposed method improves even classification performance when test samples are corrupted with noise.

## 1. Introduction

Many latent variable models can be cast as a model that learns an encoder and a decoder, either explicitly or implicitly (Ranzato et al., 2007). The encoder maps a given data sample to a latent space, and the decoder decodes the latent representation into the data space. For instance, principal component analysis (PCA) learns a linear encoder and decoder so that the $L_2$ error between a given sample and the sample reconstructed by applying the encoder and decoder sequentially is minimal (see, e.g., Bishop, 2006).

Often these models are trained to minimize the $L_2$ reconstruction error together with a sparsity regularization. The sparsity regularization ensures that the number of non-zero components in the latent representation given a sample is small. Sparse coding (see, e.g., Olshausen & Field, 1996) is one example that aims to minimize the reconstruction error while regularizing the sparsity of (overcomplete) latent representations.

One popular application of an encoder-decoder model with sparsity regularization has been image denoising. Elad & Aharon (2006) showed that a clean image can be constructed by denoising and combining small image patches of the noisy, original image, where sparse coding is used for denoising. Hyvärinen et al. (1999) denoised a large image by explicitly sparsifying the latent representation of each small image patch extracted from the larger image with a shrinkage nonlinearity.

More recently, Xie et al. (2012) proposed to use, yet, another encoder-decoder model called sparse denoising autoencoders (spDAE) for image denoising. An spDAE is a variant of a denoising autoencoder (DAE) proposed by Vincent et al. (2010) that uses a sparsity regularization during training. It was shown that the spDAE is also effective in denoising noisy images, especially when the number of hidden layers is larger than one.

These two approaches, sparse coding and denoising autoencoder, have two important differences. Firstly, DAEs, including an spDAE, have a parameterized encoder while sparse coding relies on optimization to obtain a latent representation. Secondly, the latent representation encoded by a DAE is not necessarily sparse in a strict sense due to the usual use of smooth, saturating nonlinearity functions. Sparse coding, on the other hand, finds a truly sparse latent representation.

Based on these observations, we claim that faster and

*Proceedings of the 30$^{th}$ International Conference on Machine Learning*, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. 

improved denoising can be done by explicitly sparsifying the latent representation found by spDAEs. We explain an intuition behind the claim by considering an spDAE as a previously mentioned encoder-decoder model with, potentially multi-layered, nonlinear mappings between data space and (sparse) latent space. We, then, propose a *simple sparsification* that explicitly sparsifies the latent representation of an spDAE.

We empirically confirm that the proposed simple sparsification leads to better performance by denoising various types of images corrupted with noise using spDAEs having a number of different structures. Furthermore, we show that the proposed sparsification also improves the discriminative properties of the latent representations obtained by spDAEs.

## 2. Sparse Denoising Autoencoders and Simple Sparsification

### 2.1. Sparse Denoising Autoencoder

We begin with a single-layer spDAE which is a special form of multi-layer perceptron network with a single hidden layer (Vincent et al., 2010). An spDAE tries to learn a network that reconstructs an input vector optimally by minimizing the following cost function:

$$\sum_{n=1}^{N} \left\| g \circ f \left( \eta(\mathbf{x}^{(n)}) \right) - \mathbf{x}^{(n)} \right\|^2 + \lambda \Omega(\mathbf{W}, \{\mathbf{x}^{(n)}\}), \quad (1)$$

where $\Omega(\mathbf{W}, \{\mathbf{x}^{(n)}\})$ is a sparsity regularizer, and

$$f(\mathbf{x}) = \phi(\mathbf{W}^\top \mathbf{x}) \text{ and } g(\mathbf{h}) = \mathbf{W}\mathbf{h}$$

are, respectively, an encoder and decoder with a component-wise nonlinearity function $\phi$. $\eta$ explicitly adds noise to an input sample $\mathbf{x}^{(n)}$. $\mathbf{W}$ is a matrix of the weights between the input layer and the hidden layer and is shared by the encoder and decoder. For notational simplicity, we omit biases to all units.

### 2.2. Non-linear Mapping with Constrained Range

If we assume $[0, 1]$ hidden units with a sigmoid activation function $\phi(x) = \frac{1}{1+\exp(-x)}$, the encoder $f$ is a non-linear function that maps a sample $\mathbf{x}$ in a data space[1] $\mathbb{P} \subseteq \mathbb{R}^p$ to a potentially higher-dimensional, latent space $\mathbb{Q} \subseteq [0, 1]^q$, where $p$ and $q$ are the number of visible and hidden units, respectively. The decoder does exactly the opposite with $\mathbb{Q}$ and $\mathbb{P}$ as its domain and range, respectively[2]. In this setting, the sparsity regularizer $\Omega$ defines, or restricts, the range $\mathbb{Q}$ of the learned encoder $f$.

We should note that, for any sample $\mathbf{x} \in \mathbb{R}^p$, the sample does not belong to $\mathbb{P}$, if $f(\mathbf{x}) \notin \mathbb{Q}$. It is further expected that any sample that was corrupted by error smaller than that injected by $\eta$ is still encoded to a hidden code in $\mathbb{Q}$, but any highly noisy or corrupted sample $\mathbf{x}$ maps through the encoder $f$ to a region outside $\mathbb{Q}$, albeit not necessarily.

Let us consider a specific case of regularizing the average hidden activation to a predefined sparsity hyperparameter $\rho$ (Lee et al., 2008). In this case,

$$\Omega(\mathbf{W}, \{\mathbf{x}^{(n)}\}) = \frac{1}{2} \left\| \frac{1}{N} \sum_{n=1}^{N} f(\mathbf{x}^{(n)}) - \rho \mathbf{1} \right\|_2^2. \quad (2)$$

Assuming that the model consists of stochastic binary hidden units with their probabilities given by the encoder $f$, we can see that $f(\mathbf{x})$ will have, on average, $\rho \times q$ components active while all others are inactive. If $\rho$ is set close to 0, a sparse latent representation will be produced by the encoder $f$.

This leads to an encoder $f$ with a (approximate) range, conditioned on the data set $\mathbb{P}$,

$$\mathbb{Q} \approx \left\{ \mathbf{h} = f(\mathbf{x}) \,\middle|\, \mathbf{x} \in \mathbb{P}, \|\mathbb{E}_{\mathbf{x} \in \mathbb{P}}[h_j] - \rho\|_2^2 = 0 \right\} \quad (3)$$

with the amount of error controlled by the regularization constant $\lambda$, where $h_j$ is the $j$-th component of $\mathbf{h}$.

In this case, $f(\mathbf{x})$ of any sample $\mathbf{x}$ that is close to one of training samples will fall in $\mathbb{Q}$. In other words, the average activation of $f(\mathbf{x})$ will be around the predefined $\rho$. If the average activation of $f(\mathbf{x})$ is either too smaller or too larger than $\rho$, it can be suspected that the sample $\mathbf{x}$ is either not of the same type as training samples or corrupted with high level of noise.

### 2.3. Simple Sparsification

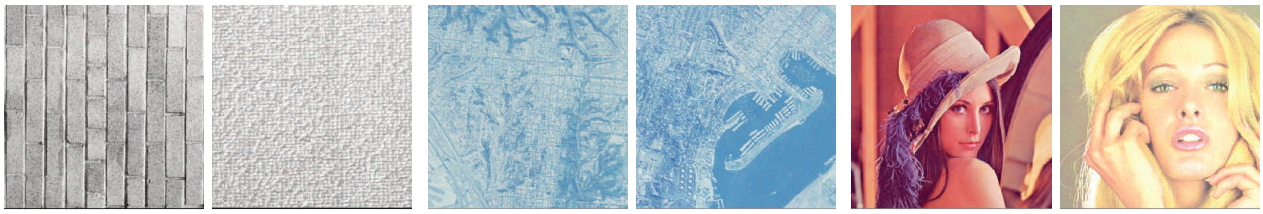
Obviously, when the latent representation $f(\tilde{\mathbf{x}})$ of a corrupted sample $\tilde{\mathbf{x}}$ is found to be outside $\mathbb{Q}$, we cannot

[1] For an spDAE that explicitly adds white Gaussian noise via $\eta$ when learning the parameters, the data space $\mathbb{P}$ is defined as

$$\mathbb{P} = \left\{ \mathbf{x} \in \mathbb{R}^p \,\middle|\, \exists \mathbf{x}^{(n)} \in D, \|\mathbf{x} - \mathbf{x}^{(n)}\|_2^2 \leq \epsilon \right\},$$

where $D = \left\{ \mathbf{x}^{(n)} \right\}_{n=1}^{N}$ is a training set and $\epsilon$ is decided by the variance of white Gaussian noise explicitly added by $\eta(\cdot)$.

[2] Note that the decoder $g$ is a *linear* mapping in the case of an spDAE with a single hidden layer. However, when an spDAE has more than one hidden layer, $g$ becomes nonlinear as described in Section 2.4.



(a) Textures (b) Aerials (c) Miscellaneous

*Figure 1.* Sample images from the test image sets

expect the decoder $g$ to correctly reconstruct the clean sample, since $g$ was trained to map from only $\mathbb{Q}$ to $\mathbb{P}$. It is, hence, not desirable to simply apply the encoder and decoder sequentially to denoise an input sample.

Instead, it must be checked whether $\mathbf{h} = f(\tilde{\mathbf{x}})$ belongs to $\mathbb{Q}$ before the decoder $g$ is applied. If $\mathbf{h} \notin \mathbb{Q}$, one must project it onto $\mathbb{Q}$ such that the decoder will correctly map $\mathbf{h}$ to a clean, denoised sample. As $\mathbb{Q}$ is defined by the sparsity of its data points, another way to put it is that $\mathbf{h}$ needs to be *sparsified*.

We define a sparsification $R$ by

$$R(\mathbf{h}) = \arg\min_{\mathbf{q} \in \mathbb{Q}} d(\mathbf{h} - \mathbf{q}), \tag{4}$$

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

Simply put, $R$ projects $\mathbf{h} \in [0, 1]^q$ onto $\mathbb{Q}$. Depending on a type of a sparsity regularizer and a target application, one must choose a suitable $d$, and the choice may have impact on the denoising performance.

In the case of the previously described sparsity regularizer (2), we can, for instance, use a variant of orthogonal matching pursuit which stops when the number of zero hidden units reaches the target sparsity $\bar{\rho}$ or the average activation reaches $1 - \bar{\rho}$. However, this type of approaches using optimization will be prohibitively expensive, especially for image denoising task which requires evaluating the encoder tens and hundreds of thousands times per image.

Alternatively, we can define $R(\mathbf{h})$ to decrease each component of $\mathbf{h}$ so that the average activation is closer to $1 - \bar{\rho}$. We call this approach a *simple sparsification*, and this effectively sets small components to zero by

$$\mathbf{h} \leftarrow \max\left(\mathbf{h} - \max\left(\frac{1}{q}\|\mathbf{h}\|_1 - (1 - \bar{\rho}), 0\right), 0\right), \tag{5}$$

where max applies to each component.

Note that it does not attempt to increase the components of $\mathbf{h}$ even if the average activation of $\mathbf{h}$ is smaller than $1 - \bar{\rho}$. This is justified by the fact that noise is likely to encourage more hidden units to respond meaninglessly, assuming white Gaussian additive noise. Fig. 2(c) shows that the average activation of hidden units increases as noise does.

It might seem obvious to choose the target sparsity $\bar{\rho}$ to be $1 - \rho$ of which $\rho$ was used when training the spDAE. Another possibility is to estimate $\bar{\rho}$ to minimize the reconstruction error of noisy training samples corrupted with a predefined level and type of noise. The latter approach can be useful when the spDAE was *not* trained with the sparsity regularization.

### 2.4. Deep Denoising Autoencoders

Unfortunately, applying the sparsity regularizer, for instance, the one in (2), is not computationally efficient in DAEs with multiple hidden layers. Hence, it has been common to use the sparsity regularizer only during pretraining (see, e.g., Xie et al., 2012).

Once the weights of a deep DAE are initialized by pretraining, this works as regularization that controls the sparsity of the hidden activations. After pretraining, the whole deep DAE can be further finetuned by stochastic backpropagation algorithm (Rumelhart et al., 1986).

Considering a deep DAE with $2L-1$ hidden layers, we have an encoder

$$f(\mathbf{x}) = \phi\left(\mathbf{W}^{(L-1)}\phi\left(\mathbf{W}^{(L-2)}\cdots\phi\left(\mathbf{W}^{(1)}\mathbf{x}\right)\cdots\right)\right) \tag{6}$$

that maps from $\mathbb{P}$ to $\mathbb{Q}$, and a decoder

$$g(\mathbf{h}) = \mathbf{W}^{(1)\top}\phi\left(\mathbf{W}^{(2)\top}\cdots\phi\left(\mathbf{W}^{(L)\top}f(\mathbf{x})\right)\cdots\right) \tag{7}$$

that reversely maps from $\mathbb{Q}$ to $\mathbb{P}$.

In this case, it is not obvious at which stage the proposed sparsification should apply. For instance, the simple sparsification can be used at each layer of the encoder, the decoder or both of them. On the other hands, if all hidden layers in the encoder are considered as a single non-linear mapping function in whole, the simple sparsification should be applied at the bottleneck layer.

In this paper, we obtain the denoised sample by only applying the simple sparsification at the bottleneck

layer:

$$\hat{\mathbf{x}} = g(R(f(\mathbf{x}))),$$

where $f$ and $g$ are defined by Eq. (6) and Eq. (7), respectively. It is, however, left for future to investigate other possibilities.

### 2.5. Related Approaches

#### 2.5.1. Sparse Coding

Assuming a linear generative model, sparse coding (see, e.g., Olshausen & Field, 1996) tries to find a (overcomplete) dictionary of features $\mathbf{W}$ and a set of *sparse* codes $\{\mathbf{h}^{(n)}\}_{n=1}^N$ given a set of training samples $\{\mathbf{x}^{(n)}\}_{n=1}^N$. This is achieved by minimizing the following cost:

$$\sum_{n=1}^{N} \left\| \mathbf{x}^{(n)} - \mathbf{W}\mathbf{h}^{(n)} \right\|_2^2 + \lambda\Omega(\mathbf{W}, \{\mathbf{h}^{(n)}\}),$$

where $\Omega$ is, again, a sparsity regularizer. Although this is closely related to the cost function of spDAEs in Eq. (1), there is no explicit encoder in sparse coding, and encoding has to be done via optimization.

If one makes the model more precise by requiring the sparsity of the sparse code $\mathbf{h}$ in a form $\|\mathbf{h}\|_0 \leq L$ for some integer $L \ll q$, then, given a sample $\mathbf{x}$ and the weights $\mathbf{W}$, encoding by optimization finds the latent representation $\mathbf{h}$ approximately inside $\mathbb{Q} = \{\mathbf{h} \mid \|\mathbf{h}\|_0 \leq L\}$ (Elad & Aharon, 2006). This is contrast to the spDAEs which do not guarantee that the encoded representation, without any explicit sparsification, resides in $\mathbb{Q}$.

This optimization-based approach of sparse coding and the proposed sparsification combined with an spDAE are two opposite approaches in finding a sparse latent representation. The former, for instance, based on the pursuit algorithms (see, e.g., (Chen et al., 2001)), starts from the center of $\mathbb{Q}$ (all zero hidden units) and sequentially finds the non-zero hidden units that decrease the reconstruction error, until a stopping criterion is met. The latter, proposed method, first encodes a given sample to a superset $[0, 1]^q$ of $\mathbb{Q}$, and then, projects the found latent representation onto $\mathbb{Q}$.

#### 2.5.2. Shrinkage Nonlinearity

In a similar context of sparse coding, Hyvärinen (1999) proposed to find a dictionary by independent component analysis (ICA) and use a shrinkage nonlinearity function to find a sparse code. This approach makes obtaining sparse code computationally less demanding compared to the optimization-based encoding in the conventional sparse coding.

This approach is closely related to the proposed simple sparsifiation. Hyvärinen (1999), for instance, suggested the following shrinkage nonlinear function in the case of each latent component following a super-Gaussian distribution:

$$s(h) = \frac{1}{1 + \sigma^2 a} \text{sign}(h) \max(0, |h| - b\sigma^2),$$

where $a$ and $b$ are parameters to be estimated, $\sigma^2$ is a noise variance, and $h$ is a latent component. If we assume a unit noise variance ($\sigma^2 = 1$) and $a = 0$, it becomes

$$s(h) = \text{sign}(h) \max(0, |h| - b).$$

If we set $b$ to $\|\mathbf{h}\|_1 - (1 - \bar{\rho})$ from Eq. (5), it is easy to see that applying the shrinkage nonlinearity $s$ to each latent component is equivalent to the proposed simple sparsification. Both of them reduce the absolute value of each latent component.

Compared to these approaches based on sparse coding, an spDAE, using the simple sparsification, has an advantage that it is natural to extend the model into a deeper model. This allows us to build a *deep* sparse generative model with a fast inference procedure, unlike the conventional sparse coding models.

## 3. Image Denoising

A noisy large image can be denoised by denoising small patches of the image and combining them together (see, e.g., Hyvärinen, 1999; Elad & Aharon, 2006). Let us define a set of $N$ binary matrices $\mathbf{D}_n \in \mathbb{R}^{p \times d}$ that extract a set of small image patches given a large, whole image $\mathbf{x} \in \mathbb{R}^d$, where $d = wh$ is the product of the width $w$ and the height $h$ $p$ is the size of image patches (e.g., $p = 64$ if the size of an image patch is $8 \times 8$). Then, the denoised image is constructed by

$$\tilde{\mathbf{x}} = \left( \sum_{n=1}^{N} \mathbf{D}_n^\top r_{\boldsymbol{\theta}}(\mathbf{D}_n \mathbf{x}) \right) \oslash \left( \sum_{n=1}^{N} \mathbf{D}_n^\top \mathbf{D}_n \mathbf{1} \right), \quad (8)$$

where $\oslash$ is a element-wise division and $\mathbf{1}$ is a vector of ones. $r_{\boldsymbol{\theta}}(\cdot)$ is an image denoising function, parameterized by $\boldsymbol{\theta}$, that denoises $N$ image patches extracted from the input image $\mathbf{x}$.

Eq. (8) essentially extracts and denoises image patches from the input image. Then, it combines them by taking an average of those overlapping pixels. For a computational reason, it is usual to use overlapping image patches separated by a few pixels rather than all possible patches.

Recently, Burger et al. (2012), Xie et al. (2012) and Cho (2013) proposed to utilize a denoising autoen-

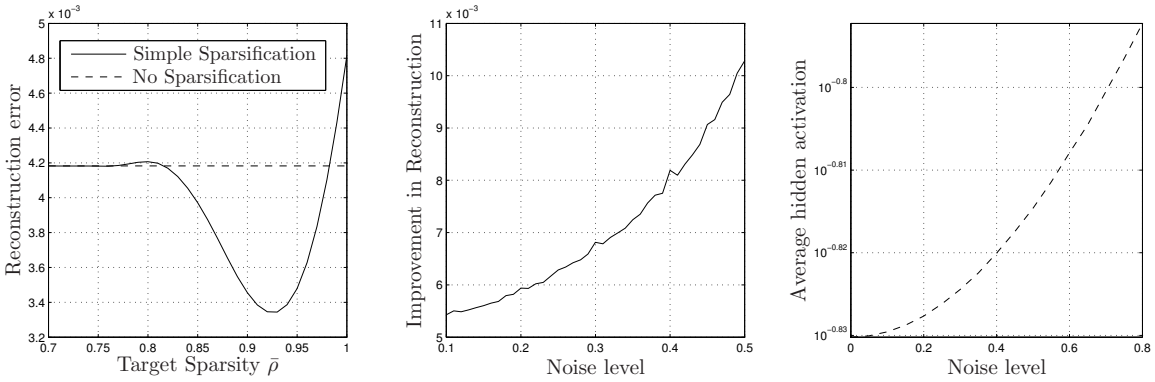



*Figure 2.* (a) Reconstruction error of image patches with varying target sparsity $\bar{\rho}$. (b) Improvement of reconstruction error of image patches achieved by using the simple sparsification with fixed $\bar{\rho} = 0.9$. (c) Average hidden activations given a set of noisy image patches with varying noise levels. A solid line and dashed line denote reconstruction errors obtained by the spDAE with and without the explicit sparsification, respectively. Image patches were randomly collected from the test sets and corrupted with white Gaussian additive noise of standard deviation 0.1. All errors were obtained using a single-layer DAE trained on $8 \times 8$ image patches with and without the simple sparsification.

coder (DAE) in place of $r_{\boldsymbol{\theta}}(\cdot)$ to perform image denoising. It is straightforward to denoise an image patch with a DAE, or in our case, spDAE. Given a noisy image patch, we first obtain a latent representation by applying the encoder $f$. Then, the decoder $g$ will reconstruct a clean patch from the hidden representation.

The proposed sparsification $R$ can be plugged in before the decoder is applied, that is, the denoised patch $\hat{\mathbf{x}} = g\left(R\left(f\left(\mathbf{x}\right)\right)\right)$. This applies to spDAEs with any number of hidden layers.

## 4. Experiments

Following the approach used by Cho (2013), we used the images from three separate sets, *textures*, *aerials* and *miscellaneous*, from the USC-SIPI Image Database[3] as test images. Fig. 1 presents six sample images from the three image sets.

All images were converted to grayscale by averaging three color channels into a single grayscale pixel. Also, each pixel of the images was normalized into $[0, 1]$ instead of the original $[0, 255]$.

Although we mainly focused on a single-layer spDAE in this paper, we trained spDAEs with one, two and four hidden layers on images patches randomly collected from CIFAR-10 dataset (Krizhevsky, 2009) to see the effect of the simple sparsification on deeper models. The size of each hidden layer was fixed to a constant multiple of the size of a visible layer[4]. We use the shorthand notations DAE, DAE(2) and DAE(4) for denoting the trained DAEs with one, two and four hidden layers, respectively.

A single-layer spDAE was trained with the sparsity regularizer given in Eq. (2) with $\rho = 0.1$. Each layer of all deep spDAEs with more than one hidden layers were pretrained as a single-layer spDAE with, again, $\rho$ set to 0.1. We used $1 - \rho$ for the target sparsity $\bar{\rho}$ of the simple sparsification.

Unlike in (Burger et al., 2012) and (Xie et al., 2012), all the models were trained in a completely *blind* way. In other words, no prior knowledge about the level or type of noise in the test images was used. Regardless of the types or levels of noise injected in the test images, each model was trained by adding white Gaussian noise with 0.1 standard deviation and dropping 10% of input pixels at each stochastic gradient update.

### 4.1. Image Patch Denoising

We have extracted from each image of the test set 50 random image patches. White Gaussian additive noise of standard deviation 0.1 was added to each pixel, and it was denoised by the trained single-layer spDAE with the simple sparsification using the heuristic introduced in Section 2.3. To see the effect of the simple sparsification, we varied $\bar{\rho}$ from 0.7 to 1.

Fig. 2 (a) shows the reconstruction errors obtained by

[3] http://sipi.usc.edu/database/

[4] We used 5 as suggested by Xie et al. (2012). For instance, the models trained on $8 \times 8$ pathces have $8 \times 8 \times 5 = 320$ units per hidden layer.

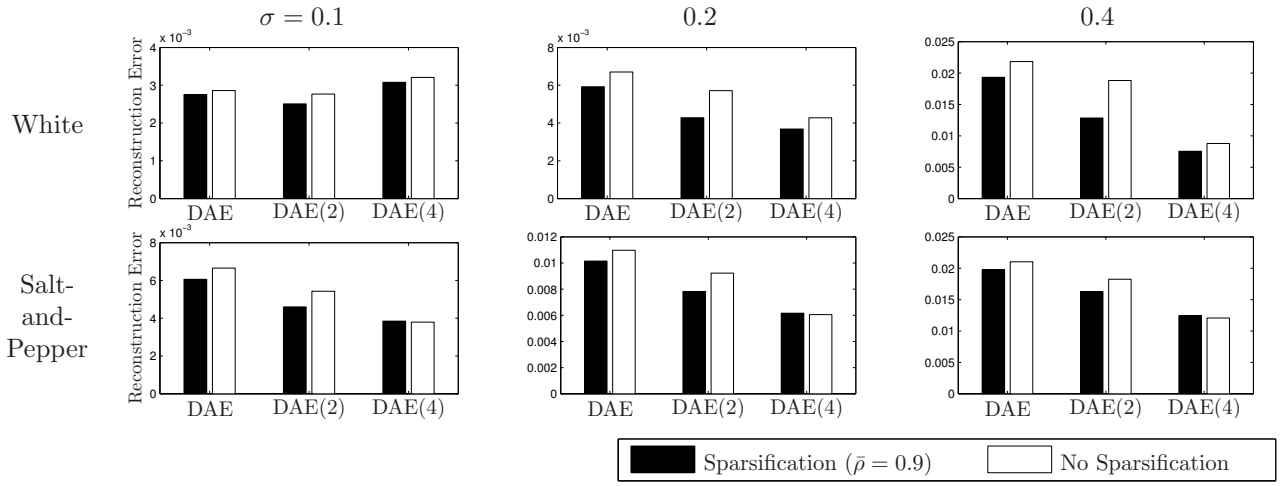



*Figure 3.* MSEs obtained with and without the explicit sparsification. The top and bottom rows show the denoising performance on the test images corrupted with white Gaussian additive noise and salt-and-pepper noise, respectively, of different noise levels $\sigma$. All images were denoised with the models trained on $8 \times 8$ patches. Lower is better.

the trained single-layer spDAE using the simple sparsification with varying target sparsities $\bar{\rho}$. The dashed-line shows the reconstruction error obtained without the simple sparsification. It is clear that lower reconstruction error could be achieved with the simple sparsification around $1 - \rho = 0.9$. We could also observe similar trend with the deeper spDAEs.

Furthermore, in Fig. 2 (b), we can see that the performance improvement by the simple sparsification grows as the level of noise increases. This suggests that the simple sparsification may help denoising an image especially when the image is highly corrupted.

### 4.2. Large Image Denoising

We tested two types of noise which were white Gaussian additive noise and salt-and-pepper noise. Three levels of noise were tested; 0.1, 0.2 and 0.4. In the case of white Gaussian noise, those levels correspond to standard deviations, while they correspond to noisy pixel probabilities with salt-and-pepper noise. The denoising performance was measured by the reconstruction error between the denoised image from the original, clean image. $L_2$-norm of the difference between them was used as the reconstruction error.

For the large, noisy test images, we first applied Wiener filtering with $3 \times 3$ neighborhood, as was done by Cho (2013). Subsequently, we denoised the images with the trained spDAEs following Eq. (8). We used overlapping image patches extracted every second pixel.

In Fig. 3, we can see the effect of the simple sparsification. It is clear that the proposed sparsification improves the performance of the spDAEs regardless of the level of noise. However, the improvement is more visible when the amount of noise is high (0.2 or 0.4).

This result was expected, as each image patch from those images corrupted with low level of noise are likely to be encoded into a latent representation that (approximately) resides in $\mathbb{Q}$. Hence, the simple sparsification will not alter it much, giving a performance similar to the one obtained without the simple sparsification. On the other hand, when the level of noise is high, the latent representation is likely to be outside $\mathbb{Q}$, and the simple sparsification correctly projects it onto $\mathbb{Q}$ to make the decoder behave better, resulting in superior performance.

The performance improvement was especially apparent with the spDAEs with the smaller number of hidden layers (one or two). The improvement decreased as the number of hidden layers increased. This might be due to the fact that the deep spDAEs were not fine-tuned with the sparsity regularizer, which made them encode a given sample into a *less* sparse latent representation.

### 4.3. Non-Regularized Denoising Autoencoders

If we consider the case where $\bar{\rho}$ is chosen to minimize the reconstruction error of noisy training samples, it is possible to consider the case where the DAEs were not trained with the sparsity regularization. The non-regularized DAEs are further interesting, as Burger et al. (2012) showed that the non-regularized ones perform comparably to, or better than, the conventional state-of-the-art denoising methods.

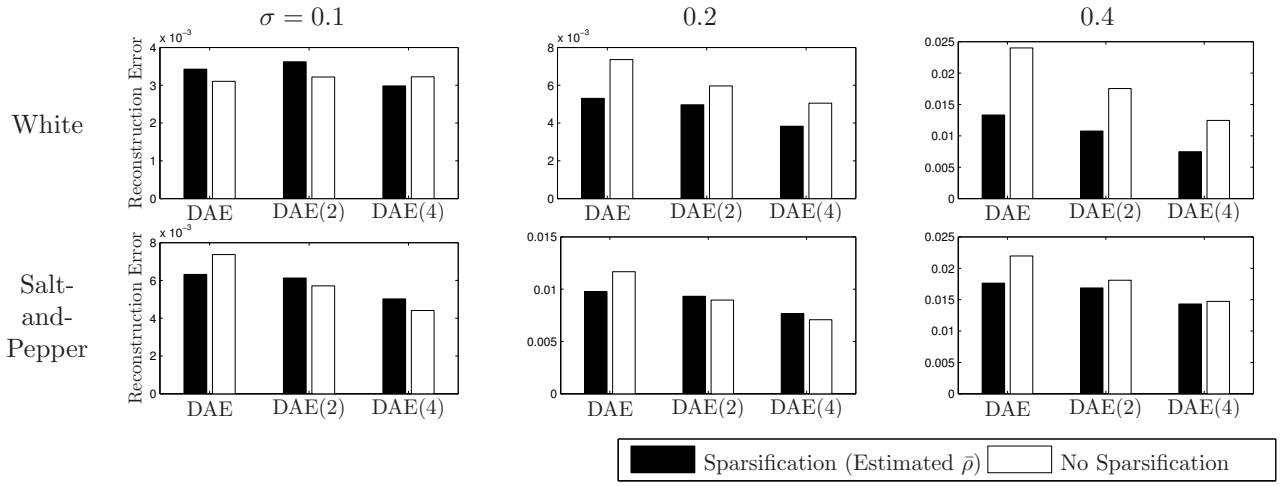



*Figure 4.* MSEs obtained with and without the explicit sparsification using the non-regularized DAEs. The top and bottom rows show the denoising performance on the test images corrupted with white Gaussian additive noise and salt-and-pepper noise, respectively, of different noise levels $\sigma$. All images were denoised with the models trained on $8 \times 8$ patches. Lower is better.

| | 4 | 8 | 16 |
|---|---|---|---|
| DAE | 0.83 | 0.84 | 0.83 |
| DAE(2) | 0.75 | 0.73 | 0.74 |
| DAE(4) | 0.81 | 0.75 | 0.78 |

*Table 1.* The estimated $\bar{\rho}$ for each non-regularized denoising autoencoders trained on square image patches of width 4, 8 and 16.

Hence, we have run the same set of experiments using the non-regularized DAEs. The test images were denoised with the simple sparsification using the estimated $\bar{\rho}$'s in Tab. 1.

In Fig. 4, we can see that the non-regularized DAEs also benefit from using the proposed simple sparsification when the level of noise is high. However, when only small amount of noise was injected, in some cases, the performance degraded with the sparsification.

Interestingly, we observed that the sparse DAEs outperformed the non-regularized DAEs when no sparsification was used. When the proposed sparsification was applied, however, the performance gap between them decreased. This suggests that the existing denoising approach based on non-regularized DAEs, for instance, used by Burger et al. (2012), may also benefit by simply plugging in the simple sparsification.

### 4.4. Discriminative Performance

Although we focus mainly on image denoising in this paper, we have also briefly investigated the potential improvement in the discriminative performance.

We used MNIST handwritten digits dataset (LeCun et al., 1998) and trained an spDAE with two hidden layers. Each layer had 4000 units and was pretrained with the sparsity regularization.

We trained a support vector machine (SVM) using libsvm (Chang & Lin, 2011) with a radial-basis function (RBF) kernel on the raw pixels of MNIST. As our interest is in a case where there are only noisy test samples available, we tried classifying, with the traind SVM, the noisy test set of MNIST after denoising them with the trained spDAE. The test set was corrupted with salt-and-pepper noise.

Additionally, we checked the effect of the simple sparsification on the discriminative properties of the latent representation. Two SVMs, again with the RBF kernel, were separately trained on the original latent representations and explicitly sparsified ones, respectively.

The robustness of the classification performance to the level of noise was measured by

$$m_p = \frac{\mathcal{E}_0}{\mathcal{E}_p},$$

where $\mathcal{E}_p$ is the classification error with the noise level $p$. The classifier with $m_p$ dropping more slowly can be considered more robust to noise in the test samples.

In Table 2, we can clearly see that the proposed sparsification makes the classifier more robust to noise in the test samples. This is especially apparent as the level of noise increased. This experiment, albeit brief and simple, suggests that the proposed simple sparsification improves even the discriminative property of the latent representations.

| Noise Level $p$ | | 0.000 | 0.100 | 0.200 | 0.300 | 0.400 |
|---|---|---|---|---|---|---|
| No Sparsification | $m_p$ | 1 | 0.861 | 0.565 | 0.217 | 0.106 |
| | $\mathcal{E}_p$ | 0.032 | 0.037 | 0.057 | 0.148 | 0.304 |
| Sparsification | $m_p$ | 1 | 0.873 | 0.615 | 0.252 | 0.117 |
| | $\mathcal{E}_p$ | 0.035 | 0.039 | 0.056 | 0.137 | 0.295 |

(a)

| $p$ | 0.000 | 0.100 | 0.200 | 0.300 | 0.400 |
|---|---|---|---|---|---|
| $m_p$ | 1 | 0.393 | 0.071 | 0.035 | 0.025 |
| $\mathcal{E}_p$ | 0.015 | 0.039 | 0.216 | 0.438 | 0.623 |
| $m_p$ | 1 | 0.443 | 0.083 | 0.039 | 0.026 |
| $\mathcal{E}_p$ | 0.016 | 0.035 | 0.189 | 0.399 | 0.591 |

(b)

*Table 2.* The robustness of the classification performance, measured by $m_p$, and the classification error $\mathcal{E}_p$, with varying noise levels. (a) The SVM was applied to the raw pixels denoised by the spDAE with and without the simple sparsification. (b) The SVMs were trained on the latent representations obtained by the spDAE with and without the simple sparsification.

This is interesting to notice that the classifier benefited from the feature extraction by the spDAE only when low level of noise (0 or 0.1) was injected to the test samples. When, the amount of noise was larger, we were able to see that the classification using the denoised raw pixels, especially with the proposed sparsification, outperformed that using the latent representations.

## 5. Discussion and Conclusion

A sparse denoising autoencoder (spDAE) learns an encoder that maps from a data space, defined by training samples corrupted with explicit noise, to a sparse latent space, defined by the sparsity regularization. A decoder of the spDAE, then, maps back from the sparse latent space to the data space. As its name suggests, an spDAE encodes a given noisy sample into a *sparse* latent representation and decodes the found representation into a *denoised* sample.

However, we noticed that the learned non-linear mappings, both encoder and decoder, are not well defined when a given sample is highly corrupted. In other words, if a given sample does not belong to the same data space, then the spDAE is not expected to denoise it well. Under this observation, we have proposed that explicit sparsification is necessary after the encoder is applied. For an spDAE trained with a sparsity regularization given in Eq. (2), a *simple sparsification* which makes the average latent activation closer to the target sparsity $\bar{\rho}$ was proposed.

Only very recently have *deep* neural networks, including denoising autoencoders, been applied to image denoising by Burger et al. (2012) and Xie et al. (2012). They all showed that these deep neural networks perform comparably to, or better than, conventional denoising methods. In this context, we have empirically shown that the proposed simple sparsification further improves denoising performance of deep neural networks. In addition to image denoising, we were able to see that the proposed sparsification could be used to improve even the discriminative performance when test samples were noisy.

In future, different sparsification methods should be sought for other available sparsity regularizers. Also, in the context of denoising, another type of datasets, such as speech, should be investigated in future.

## References


Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

Burger, H., Schuler, C., and Harmeling, S. Image denoising: Can plain neural networks compete with bm3d? In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 2392 –2399, june 2012.

Chang, C.-C. and Lin, C.-J. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

Chen, S. S., Donoho, D. L., and Saunders, M. A. Atomic decomposition by basis pursuit. *SIAM Rev.*, 43(1):129–159, January 2001.

Cho, K. Boltzmann machines and denoising autoencoders for image denoising. *ArXiv e-prints*, January 2013.

Elad, M. and Aharon, M. Image denoising via sparse and redundant representations over learned dictionaries. *Image Processing, IEEE Transactions on*, 15 (12):3736–3745, December 2006.

Hyvärinen, A. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–34, 1999.

Hyvärinen, A., Hoyer, P., and Oja, E. Image denoising by sparse code shrinkage. In *Intelligent Signal Processing*. IEEE Press, 1999.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, Computer Science Department, University of Toronto, 2009.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pp. 2278–2324, 1998.

Lee, H., Ekanadham, C., and Ng, A. Sparse deep belief net model for visual area v2. In Platt, J., Koller, D., Singer, Y., and Roweis, S. (eds.), *Advances in Neural Information Processing Systems 20*, pp. 873–880. MIT Press, Cambridge, MA, 2008.

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

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. Efficient learning of sparse representations with an energy-based model. In Schölkopf, B., Platt, J., and Hoffman, T. (eds.), *Advances in Neural Information Processing Systems 19*, pp. 1137–1144. MIT Press, Cambridge, MA, 2007.

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

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, December 2010.

Xie, J., Xu, L., and Chen, E. Image denoising and inpainting with deep neural networks. In Bartlett, P., Pereira, F., Burges, C., Bottou, L., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 350–358. 2012.

# Publication IX

**Kyunghyun Cho. Boltzmann Machines for Image Denoising. In *Proceedings of the 23rd International Conference on Artificial Neural Networks (ICANN 2013)*, Pages 611–618, September 2013.**

# Boltzmann Machines for Image Denoising

KyungHyun Cho

Department of Information and Computer Science
Aalto University School of Science, Finland
{firstname.lastname@aalto.fi}

**Abstract.** Image denoising based on a probabilistic model of local image patches has been employed by various researchers, and recently a deep denoising autoencoder has been proposed in [2] and [17] as a good model for this. In this paper, we propose that another popular family of models in the field of *deep learning*, called Boltzmann machines, can perform image denoising as well as, or in certain cases of high level of noise, better than denoising autoencoders. We empirically evaluate these two models on three different sets of images with different types and levels of noise. The experiments confirmed our claim and revealed that the denoising performance can be improved by adding more hidden layers, especially when the level of noise is high.

**Keywords:** Image Denoising, Deep Learning, Restricted Boltzmann Machine, Deep Boltzmann Machine

## 1 Introduction

Numerous approaches based on machine learning have been proposed for image denoising tasks over time. A dominant approach has been to perform denoising based on local statistics of image patches. Under this approach, small image patches from a larger noisy image are denoised and combined afterward to form a clean image.

For instance, in [9] independent component analysis (ICA) was used to estimate a dictionary of sparse elements and compute the sparse code of image patches. Subsequently, a shrinkage nonlinear function is applied to the estimated sparse code elements to suppress those elements with small absolute magnitude. These shrunk sparse codes are then used to reconstruct a noise-free image patch. More recently, it was shown in [5] that sparse overcomplete representation may be more useful in denoising images.

In essence, these approaches build a probabilistic model of natural image patches using a single layer of sparse latent variables. The posterior distribution of each noisy patch is either exactly computed or estimated, and the noise-free patch is reconstructed as an expectation of a conditional distribution over the posterior distribution.

Some researchers have proposed very recently to utilize a model that has more than one layers of latent variables for image denoising. It was shown in [2] that a deep denoising autoencoder [16] that learns a mapping from a noisy image patch to its corresponding clean version, can perform as good as the state-of-the-art denoising methods. Similarly, a variant of a stacked sparse denoising autoencoder that is more effective in image denoising was recently proposed in [17].

Along this line of research, we propose yet another type of deep neural networks for image denoising, in this paper. A Gaussian-Bernoulli restricted Boltzmann machines (GRBM) [6] and deep Boltzmann machines (GDBM) [13] are empirically shown to perform well in image denoising, compared to stacked denoising autoencoders. Furthermore, we evaluate the effect of the number of hidden layers of both Boltzmann machines and denoising autoencoders. The empirical evaluation is conducted using different noise types and levels on three different sets of images.

## 2 Deep Neural Networks

We start by briefly describing Boltzmann machines and denoising autoencoders which have become increasingly popular in the field of machine learning.

### 2.1 Boltzmann Machines

Originally proposed in 1980s, a Boltzmann machine (BM) [1] and especially its structural constrained version, a restricted Boltzmann machine (RBM) [14] have become increasingly important in machine learning since it was shown that a deep multi-layer perceptron can be trained easily by stacking RBMs on top of each other [6]. More recently, another variant of a BM, called a deep Boltzmann machine (DBM), has been proposed and shown to outperform other conventional machine learning methods in many tasks (see, e.g., [12]).

We first describe a Gaussian-Bernoulli DBM (GDBM) that has $L$ layers of binary hidden units and a single layer of Gaussian visible units. A GDBM is defined by its energy function

$$-E(\mathbf{v},\mathbf{h}\mid\boldsymbol{\theta}) = \sum_i -\frac{(v_i-b_i)^2}{2\sigma^2} + \sum_{i,j}\frac{v_i}{\sigma^2}h_j^{(1)}w_{i,j} + \sum_j h_j^{(1)}c_j^{(1)} + \sum_{l=2}^{L}\left(\sum_j h_j^{(l)}c_j^{(l)} + \sum_{j,k} h_j^{(l)}h_k^{(l+1)}u_{j,k}^{(l)}\right), \quad (1)$$

where $\mathbf{v} = [v_i]_{i=1\ldots N_v}$ and $\mathbf{h}^{(l)} = \left[h_j^{(l)}\right]_{j=1\ldots N_l}$ are $N_v$ Gaussian visible units and $N_l$ binary hidden units in the $l$-th hidden layer. $\mathbf{W} = [w_{i,j}]$ is the set of weights between the visible neurons and the first layer hidden neurons, while $\mathbf{U}^{(l)} = \left[u_{j,k}^{(l)}\right]$ is the set of weights between the $l$-th and $l+1$-th hidden neurons. $\sigma^2$ is the shared variance of the conditional distribution of $v_i$ given the hidden units.

With the energy function, a GDBM can assign a probability to each state vector $\mathbf{x} = [\mathbf{v};\mathbf{h}^{(1)};\cdots;\mathbf{h}^{(L)}]$ using a Boltzmann distribution: $p(\mathbf{x}\mid\boldsymbol{\theta}) = \exp\{-E(\mathbf{x}\mid\boldsymbol{\theta})\}/Z(\boldsymbol{\theta})$. Then, the parameters can be learned by maximizing the log-likelihood

$$\mathcal{L} = \sum_{n=1}^{N}\log\sum_{\mathbf{h}} p(\mathbf{v}^{(n)},\mathbf{h}\mid\boldsymbol{\theta})$$

given $N$ training samples $\{\mathbf{v}^{(n)}\}_{n=1,\ldots,N}$, where $\mathbf{h} = \left[\mathbf{h}^{(1)}; \cdots; \mathbf{h}^{(L)}\right]$.

Although the update rules based on the gradients of the log-likelihood function are well defined, it is intractable to exactly compute them. Hence, an approach that uses variational approximation together with Markov chain Monte Carlo (MCMC) sampling was proposed in [13]. This approach is often used together with a pretraining algorithm [13, 4]. In this paper, we initialized GDBMs using the two-stage pretraining algorithm recently proposed in [4].

A Gaussian-Bernoulli RBM (GRBM) is a special case of a GDBM, where the number of hidden layers is restricted to one, $L = 1$. Unlike GDBMs, it is possible to compute the posterior distribution over the hidden units exactly and tractably by

$$p(h_j = 1 \mid \mathbf{v}, \boldsymbol{\theta}) = f\left(\sum_i w_{ij} \frac{v_i}{\sigma^2} + c_j\right), \tag{2}$$

where $f$ is a logistic sigmoid function, and we dropped the superscript $(1)$ from $h_j$ for simplicity.

This removes the need for the variational approximation in computing the gradient and allows an efficient block Gibbs sampling. Based on this property many fast approximate algorithms for training GRBMs, such as contrastive divergence [7], have been proposed.

### 2.2 Denoising Autoencoders

A denoising autoencoder (DAE) [16] is a special form of multi-layer perceptron network with $2L - 1$ hidden layers and $L - 1$ sets of (tied) weights. A DAE tries to learn a network that denoises an explicitly corrupted input vector by minimizing the following cost function:

$$\sum_{n=1}^{N} \left\| \mathbf{W} g^{(1)} \circ \cdots \circ g^{(L-1)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)} \left(\eta(\mathbf{v}^{(n)})\right) - \mathbf{v}^{(n)} \right\|^2, \tag{3}$$

where $f^{(l)} = \phi({\mathbf{W}^{(l)}}^\top \mathbf{h}^{(l-1)})$ and $g^{(l)} = \phi(\mathbf{W}^{(l)} \mathbf{h}^{(2L-l)})$ are, respectively, encoding and decoding functions for $l$-th layer with a component-wise nonlinearity function $\phi$. $\eta$ stochastically adds noise to an input vector $\mathbf{x}$ at each update step. $\mathbf{W}^{(l)}$ is the weights between the $l$-th and $(l+1)$-th layers and is shared by the encoder and decoder.

## 3 Image Denoising

Let us define a set of $N$ binary matrices $\mathbf{D}_n \in \mathbb{R}^{p \times d}$ that extract a set of small image patches given a large, whole image $\mathbf{x} \in \mathbb{R}^d$, where $d = wh$ is the product of the width $w$ and the height $h$ of the image and $p$ is the size of image patches. Then, an image can be denoised by

$$\mathbf{x} = \left(\sum_{n=1}^{N} \mathbf{D}_n^\top r_{\boldsymbol{\theta}}(\mathbf{D}_n \tilde{\mathbf{x}})\right) \oslash \left(\sum_{n=1}^{N} \mathbf{D}_n^\top \mathbf{D}_n \mathbf{1}\right), \tag{4}$$

where $\oslash$ is an element-wise division and $\mathbf{1}$ is a vector of ones. $r_{\boldsymbol{\theta}}(\cdot)$ is an image denoising function, parameterized by $\boldsymbol{\theta}$, that denoises each image patch $\mathbf{D}_n\mathbf{x}$.

In short, Eq. (4) extracts and denoises image patches from the input image. Then, it combines them by taking an average of those overlapping pixels.

One of the popular choices for $r_{\boldsymbol{\theta}}(\cdot)$ has been to construct a probabilistic model with a set of latent variables that describe natural image patches (see, e.g., [8, 5]). Under this approach denoising can be considered as a two-step reconstruction. First, the posterior distribution over the latent variables is computed given an image patch. Based on that, the conditional distribution, or its mean, over the visible units is computed and used as a denoised image patch.

### 3.1 Boltzmann machines

We consider a BM with a set of Gaussian visible units $\mathbf{v}$ that correspond to the pixels of an image patch and a set of binary hidden units $\mathbf{h}$. Then, the goal of denoising can be written as

$$p(\mathbf{v} \mid \tilde{\mathbf{v}}) = \sum_{\mathbf{h}} p(\mathbf{v} \mid \mathbf{h}) p(\mathbf{h} \mid \tilde{\mathbf{v}}) = \mathbb{E}_{\mathbf{h}|\tilde{\mathbf{v}}} \left[ p(\mathbf{v} \mid \mathbf{h}) \right], \tag{5}$$

where $\tilde{\mathbf{v}}$ is a noisy input patch. In other words, we find a mean of the conditional distribution of the visible units with respect to the posterior distribution over the hidden units given the visible units fixed to the corrupted input image patch.

However, since taking the expectation over the posterior distribution is usually not tractable, it is often easier and faster to approximate it. We approximate the marginal conditional distribution in (5) with $p(\mathbf{v} \mid \tilde{\mathbf{v}}) \approx p(\mathbf{v} \mid \mathbf{h}) Q(\mathbf{h})$, where $Q(\mathbf{h})$, parameterized by $\boldsymbol{\mu} = \mathbb{E}_{Q(\mathbf{h})}[\mathbf{h}]$, is a fully-factorial (approximate) posterior distribution $p(\mathbf{h} \mid \tilde{\mathbf{v}})$.

Given a noisy image patch $\tilde{\mathbf{v}}$, following this approach, a GRBM reconstructs a noise-free patch by

$$\hat{v}_i = \sum_{j=1}^{N_h} w_{ij} \mathbb{E}\left[ \mathbf{h} \mid \tilde{\mathbf{v}} \right] + b_i.$$

The conditional distribution over the hidden units can be computed exactly from Eq. (2).

Unlike a GRBM, the posterior distribution of the hidden units of a GDBM is neither tractably computable nor has an analytical form. Instead, we use a fully-factorial *approximate* posterior $Q(\mathbf{h}) = \prod_{l=1}^{L} \prod_j \mu_j^l$, where the parameters $\mu_j^{(l)}$'s can be estimated by maximizing the variational lower-bound [13].

Once the variational parameters are converged, a GDBM reconstructs a noise-free patch by

$$\hat{v}_i = \sum_{j=1}^{N_l} w_{ij} \mu_j^{(1)} + b_i.$$

The convergence of the variational parameters may take too much time in practice. Hence, in the experiments, we initialize the variational parameters by the feed-forward propagation using the doubled weights [13] and performing the fixed-point update for at most five iterations only.

| Set | # of all images | # of color images | Min. Size | Max. Size |
|---|---|---|---|---|
| Textures | 64 | 0 | $512 \times 512$ | $1024 \times 1024$ |
| Aerials | 38 | 37 | $512 \times 512$ | $2250 \times 2250$ |
| Miscellaneous | 44 | 16 | $256 \times 256$ | $1024 \times 1024$ |

**Table 1.** Descriptions of the test image sets.

### 3.2 Denoising autoencoders

An encoder part of a DAE can be considered as performing an approximate inference of a fully-factorial posterior distribution of top-layer hidden units, i.e. a bottleneck, given an input image patch [16]. Hence, a similar approach can be applied to DAEs.

Firstly, the variational parameters $\boldsymbol{\mu}^{(L)}$ of the fully-factorial posterior distribution $Q(\mathbf{h}^{(L)}) = \prod_j \mu_j^{(L)}$ are computed by

$$\boldsymbol{\mu}^{(L)} = f^{(L-1)} \circ \cdots \circ f^{(1)} \left( \tilde{\mathbf{v}} \right).$$

Then, the denoised image patch can be reconstructed simply by propagating the variational parameters through the decoding nonlinearity functions $g^{(l)}$ such that

$$\hat{\mathbf{v}} = g^{(1)} \circ \cdots \circ g^{(L-1)} \left( \boldsymbol{\mu}^{(L)} \right).$$

## 4 Experiments

In the experiment, we test both GRBMs and GDBMs together with DAEs having varying numbers of hidden layers. These models are compared to each other on a *blind* image denoising task, where no prior knowledge about target images nor the type or level of noise is known when the models are trained. In other words, no separate training was done for different types or levels of noise injected to the test images. Unlike this, for instance, in [17] each DAE was trained specifically for the target noise level and type by changing $\eta(\cdot)$ accordingly.

### 4.1 Datasets

We used three sets of images, *textures*, *aerials* and *miscellaneous*, from the USC-SIPI Image Database[1] as test images. Tab. 1 lists the details of the image sets.

These datasets are, in terms of contents and properties of images, very different from each other. For instance, most of the images in the texture set have highly repetitive patterns that are not present in the images in the other two sets. Most images in the aerials set have simultaneously both coarse and fine structures. Also, the sizes of the images vary quite a lot across the test sets and across the images in each set.

As we are aiming to evaluate the performance of denoising a very general image, we used a large separate data set of natural image patches to train the models. We used

[1] http://sipi.usc.edu/database/

a large number of image patches randomly of sizes $4 \times 4$, $8 \times 8$ and $16 \times 16$ extracted from CIFAR-10 dataset [10]. The same set of experiments was run with the models trained on the patches from the Berkeley Segmentation Dataset [11], and the similar results were observed.

### 4.2 Settings

We tried three different depth settings for both Boltzmann machines and denoising autoencoders; a single, two and four hidden layers. Each hidden layer had the same number of hidden units which was the constant factor $5$ multiplied by the number of pixels in an image patch, as suggested in [17].

We denote Boltzmann machines with one, two and four hidden layers by GRBM, GDBM(2) and GDBM(4), respectively. Denoising autoencoders are denoted by DAE, DAE(2) and DAE(4), respectively.

The GRBMs were trained using the enhanced gradient [3] and persistent contrastive divergence (PCD) [15]. The GDBMs were trained by PCD after initializing the parameters with a two-stage pretraining algorithm [4]. DAEs were trained by a stochastic backpropagation, and when there were more than one hidden layers, we pretrained each pair of consecutive layers as a single-layer DAE with sparsity target set to $0.1$.

Two types of noise have been tested; white Gaussian and salt-and-pepper. White Gaussian noise simply adds zero-mean normal random value with a predefined variance to each image pixel, while salt-and-pepper noise sets a randomly chosen subset of pixels to either black or white. Three different noise levels ($0.1$, $0.2$ and $0.4$) were tested. In the case of white Gaussian noise, they were used as standard deviations, and in the case of salt-and-pepper noise, as a noise probability.

After noise was injected, each image was preprocessed by pixel-wise adaptive Wiener filtering, following the approach of [9]. The width and height of the pixel neighborhood were chosen to be small enough ($3 \times 3$) to avoid removing too much detail.

### 4.3 Results and Analysis

In Fig. 1, the performances of all the tested models trained on $8 \times 8$ image patches, measured by the peak signal-to-noise ratio (PSNR), are presented. Those trained on the patches of different sizes showed similar trend, and they are omitted here.

Interestingly, the GRBMs consistently performed comparably to much deeper DAEs in most cases regardless of the level of noise. This indirectly suggests that learning a good generative model might be important in image denoising. Considering that the number of parameters of the GRBM is, for instance, only half compared to DAE(2), training a model in a fully generative manner helps learning more compact representation of natural image patches that are more suitable for image denoising.

However, the GDBMs which are essentially deeper version of the GRBM were only able to outperform the other models in the high noise regime. In the cases of the aerials and miscellaneous sets, in the low noise regime, the GDBMs lag behind all the other models. A possible explanation for this rather poor performance of the GDBMs in the low noise regime is that the posterior distribution had to be approximated, whereas it

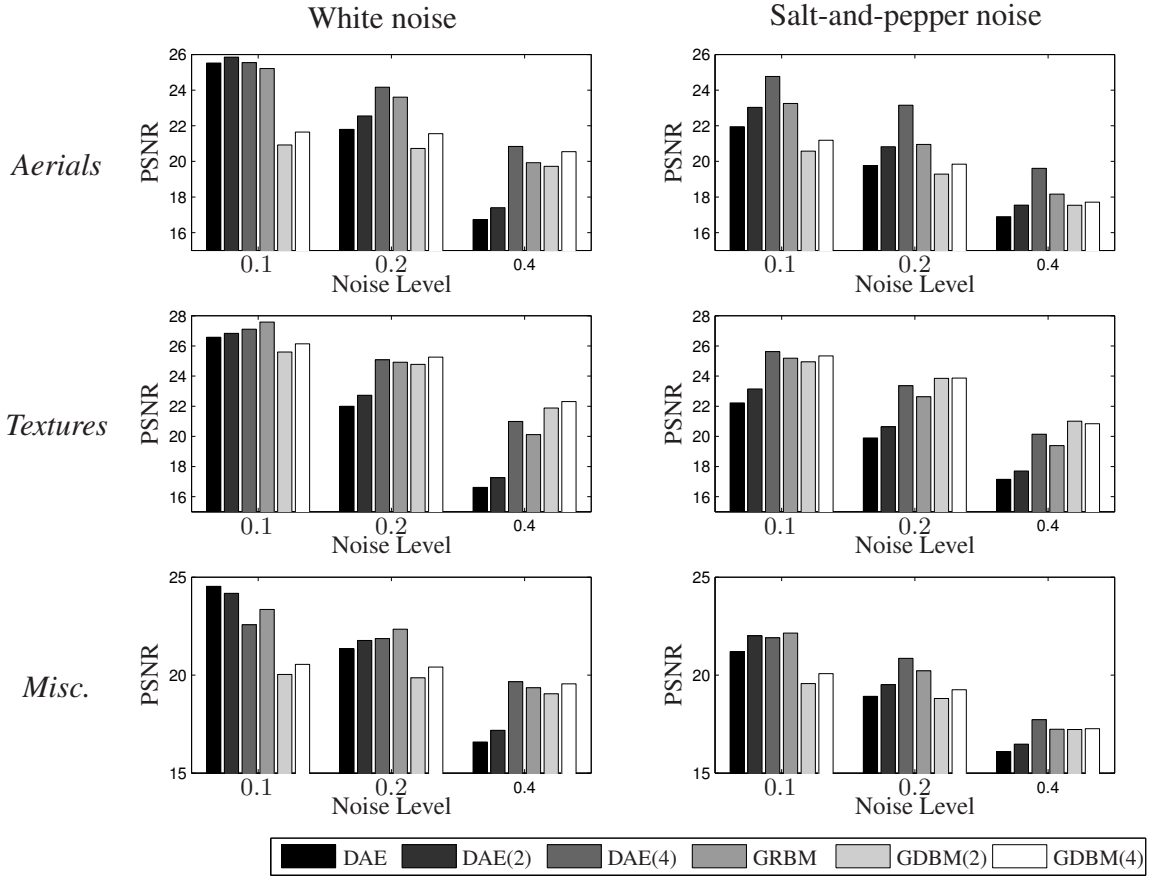



**Fig. 1.** The median PSNRs of grayscale images corrupted by different types and levels of noise.

was computed exactly in the case of GRBMs. A better approximation strategy might resolve this problem.

An important observation is that the deeper models significantly outperformed their corresponding shallower models as the level of injected noise grew. In other words, the power of the deep models became more evident as the difficulty of the task increased.

## 5 Discussion

In this paper, we proposed that, in addition to DAEs, Boltzmann machines (BM) can also be used efficiently for denoising images. Boltzmann machines and DAEs were empirically evaluated against each other in the *blind* image denoising task where no prior knowledge about target images and their level of corruption was assumed.

The experimental results showed that Boltzmann machines (BM) are good, potential alternatives to DAEs. BMs and DAEs performed comparably to each other in the low noise regime, while BMs were able to, in many cases, outperform DAEs when the level of injected noise was high. This suggests that BMs may be more robust to noise than DAEs are.

More careful look at the experimental results clearly showed that, in the case of DAEs, hidden layers do improve performance, especially when the level of noise is high. This did not always apply to BMs, where we found that the GRBMs outperformed, or performed as well as, the GDBMs in many cases. Regardlessly, in the high noise regime, it was always beneficial to have more hidden layers, even for BMs.

## References


1. Ackley, D.H., Hinton, G.E., Sejnowski, T.J.: A learning algorithm for Boltzmann machines. Cognitive Science 9, 147–169 (1985)
2. Burger, H., Schuler, C., Harmeling, S.: Image denoising: Can plain neural networks compete with bm3d? In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. pp. 2392 –2399 (june 2012)
3. Cho, K., Raiko, T., Ilin, A.: Enhanced gradient for training restricted Boltzmann machines. Neural Computation 25(3), 805–831 (2013)
4. Cho, K., Raiko, T., Ilin, A., Karhunen, J.: A Two-Stage Pretraining Algorithm for Deep Boltzmann Machines. In: NIPS 2012 Workshop on Deep Learning and Unsupervised Feature Learning. Lake Tahoe (December 2012)
5. Elad, M., Aharon, M.: Image denoising via sparse and redundant representations over learned dictionaries. Image Processing, IEEE Transactions on 15(12), 3736–3745 (Dec 2006)
6. Hinton, G., Salakhutdinov, R.: Reducing the dimensionality of data with neural networks. Science 313(5786), 504–507 (July 2006)
7. Hinton, G.: Training products of experts by minimizing contrastive divergence. Neural Computation 14, 1771–1800 (August 2002)
8. Hyvärinen, A.: Fast and robust fixed-point algorithms for independent component analysis. IEEE Transactions on Neural Networks 10(3), 626–34 (1999)
9. Hyvärinen, A., Hoyer, P., Oja, E.: Image denoising by sparse code shrinkage. In: Intelligent Signal Processing. IEEE Press (1999)
10. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., Computer Science Department, University of Toronto (2009)
11. Martin, D., Fowlkes, C., Tal, D., Malik, J.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: Proc. 8th Int'l Conf. Computer Vision. vol. 2, pp. 416–423 (July 2001)
12. Salakhutdinov, R., Hinton, G.: An efcient learning procedure for deep Boltzmann machines. Neural Computation 24, 1967–2006 (2012)
13. Salakhutdinov, R., Hinton, G.E.: Deep Boltzmann machines. In: Proc. of the Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2009). pp. 448–455 (2009)
14. Smolensky, P.: Information processing in dynamical systems: foundations of harmony theory. In: Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations, pp. 194–281. MIT Press, Cambridge, MA, USA (1986)
15. Tieleman, T.: Training restricted Boltzmann machines using approximations to the likelihood gradient. ICML '08, ACM, New York, NY, USA (2008)
16. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: 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)
17. Xie, J., Xu, L., Chen, E.: Image denoising and inpainting with deep neural networks. In: Bartlett, P., Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 25, pp. 350–358 (2012)

# Publication 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)*, Pages 6729–6733, May 2013.**

# GAUSSIAN-BERNOULLI RESTRICTED BOLTZMANN MACHINES AND AUTOMATIC FEATURE EXTRACTION FOR NOISE ROBUST MISSING DATA MASK ESTIMATION

*Sami Keronen* *KyungHyun Cho* *Tapani Raiko* *Alexander Ilin* *Kalle Palomäki*

Aalto University School of Science
Department of Information and Computer Science
PO Box 15400, FI-00076 Aalto, Finland

## ABSTRACT

A missing data mask estimation method based on Gaussian-Bernoulli restricted Boltzmann machine (GRBM) trained on cross-correlation representation of the audio signal is presented in the study. The automatically learned features by the GRBM are utilized in dividing the time-frequency units of the spectrographic mask into noise and speech dominant. The system is evaluated against two baseline mask estimation methods in a reverberant multisource environment speech recognition task. The proposed system is shown to provide a performance improvement in the speech recognition accuracy over the previous multifeature approaches.

***Index Terms***— Noise robust, speech recognition, mask estimation, GRBM, deep learning

## 1. INTRODUCTION

Missing data methods, one of the many approaches for reducing the gap between human listeners and automatic speech recognition (ASR) in noisy environments, are based on studies motivated by the human auditory system [1]. In missing data methods, the noise corrupted speech is divided into reliable, speech-dominated, and unreliable, noise-dominated, components. The unreliable components can be discarded, as in the marginalization approach, used as an upper bound to the missing clean speech values [2], or they can be reconstructed by the respective clean speech estimates [2, 3].

Estimating the reliable and unreliable spectro-temporal regions of the speech signal, i.e. mask estimation, can be challenging in varying noise environments. Some of the previous work on the field have considered mask estimation a binary classification problem by training machine learning based classifiers such as Gaussian mixture models (GMMs) [4, 5] or support vector machines (SVMs) [6] with several acoustic features in conjunction. These multifeature approaches counteract the adverse environmental factors with their comprehensive set of features – cues discriminating between speech and non-speech are effective in non-speech noisy environments [4], whereas directional cues provide information on competing speakers [5, 7].

As an alternative to founding the multifeature approach to a set of "design" features, a GRBM [8] can be trained to learn the acoustical patterns for an arguably better performing set of features. Due to the contrastive divergence algorithm [9] and the recent advances in general purpose graphics processing units, GRBMs and deep belief networks (DBNs) are displacing the traditional combination of hidden Markov models and GMMs as the basis of the state of the art ASR systems. DBNs are capable of learning the acoustical patterns efficiently, which has been shown in many speech related tasks such as phone recognition [10], monaural speech separation [11], and large vocabulary speech recognition [12, 13].

Ultimately the confrontation between design and automatically learned features reduces to quantity versus quality; the discrimination power of a single automatically learned feature may be small but the number of them can be made arbitrarily large, whereas a single design feature such as interaural time difference (ITD) or interaural level difference (ILD) [5, 7] may be effective alone but the overall number of them is usually much smaller. Additionally, the multilayer DBNs may provide higher level information on the audio signal [13], which may further improve the system performance.

In this paper, we use GRBMs to learn the cross-correlation representation of a dual channel multisource reverberant CHiME corpus and apply it in a missing data reconstruction-based automatic speech recognition task. The speech recognition performance of the proposed system is evaluated against systems based on 14 design features and on the unprocessed channel-wise cross-correlation values.

## 2. METHODS

In this section, we describe the proposed method starting with an introduction to missing data mask estimation, followed by descriptions of GRBM, feature extraction and GRBM training, classifier, and reconstruction of missing data.

### 2.1. Missing data mask estimation

Missing data techniques are based on the assumption that the spectro-temporal units of the noisy speech can be divided into

speech and noise dominated regions [2]. Typically a set of log-mel features $\boldsymbol{Y}$ is computed for each time-frequency (TF) unit of the speech signal and a so called spectrographic mask labels the feature observations as speech or noise dominated. A TF unit $Y_r(\tau, d)$ is considered reliable if $Y(\tau, d) \approx S(\tau, d)$, where $\tau$ denotes the time frame, $d$ the frequency channel, and $S(\tau, d)$ the clean speech signal without corrupting noise. Units $Y_u(\tau, d)$ are considered unreliable if $Y(\tau, d) \geq S(\tau, d)$.

In this work, binary valued masks are used for labeling and cluster-based imputation [3] (see Sec. 2.5), which has been shown to perform well on various speech recognition tasks, is used to reconstruct the missing values.

### 2.2. Gaussian-Bernoulli Restricted Boltzmann Machines

A GRBM is a neural network that models the probability density of continuous-valued data using binary latent variables. It consists of a layer of Gaussian visible units that correspond to components of data vectors, and a layer of binary hidden units. Each unit is connected to all units in the other layer (i.e. no lateral connections). It has been shown that the latent variables of a learnt GRBM can be used as meaningful unsupervised features (see, e.g., [10, 14]).

The energy given by a GRBM to each state of visible units $v_i$ and hidden units $h_j$ is defined as

$$E(\mathbf{v}, \mathbf{h} \mid \theta) = \sum_{i=1}^{n_v} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{n_v}\sum_{j=1}^{n_h} w_{ij} h_j \frac{v_i}{\sigma_i^2} - \sum_{j=1}^{n_h} c_j h_j,$$

where $n_v$ and $n_h$ are the numbers of hidden and visible units, and the parameters $\boldsymbol{\theta}$ include weights $w_{ij}$ connecting the visible and hidden units, the standard deviation $\sigma_i$ associated with a Gaussian visible unit $v_i$ and biases $b_i$ and $c_j$ for each unit [15]. Based on it, one can define a Boltzmann distribution by $p(\mathbf{v}, \mathbf{h} \mid \theta) = \frac{1}{Z(\theta)} \exp\left\{-E(\mathbf{v}, \mathbf{h} \mid \theta)\right\}$.

Due to the bipartite structure, the visible units given the the hidden units are conditionally independent, and the probability of each visible unit is

$$p(v_i = v|\mathbf{h}) = \mathcal{N}\left(v \mid b_i + \sum_j h_j w_{ij}, \sigma_i^2\right),$$

where $\mathcal{N}(\cdot \mid \mu, \sigma^2)$ denotes the Gaussian p.d.f. with mean $\mu$ and variance $\sigma^2$. Similarly, the hidden units are conditionally independent, and their probabilities are given by

$$p(h_j = 1|\mathbf{v}) = \text{sigmoid}\left(c_j + \sum_i w_{ij} \frac{v_i}{\sigma_i^2}\right),$$

which makes it simple to compute the activations of features learned by the GRBM for further use.

Learning parameters of a GRBM is commonly done by maximizing the log-likelihood function with a stochastic gradient. Recently in [16], the enhanced gradient was proposed and shown to outperform the conventional gradient update direction. In the enhanced gradient, each weight parameter $w_{ij}$ is updated by

$$w_{ij} \leftarrow w_{ij} + \eta \left[\text{Cov}_\text{d}\left(v_i, h_j\right) - \text{Cov}_\text{m}\left(v_i, h_j\right)\right], \quad (1)$$

where $\text{Cov}_p(v_i, h_j)$ is a covariance between $v_i$ and $h_j$ under distribution $p$, and d and m denote the data distribution $p(\mathbf{h} \mid \mathbf{v}, \boldsymbol{\theta})D(\mathbf{v})$ and the model distribution $p(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})$, respectively. Learning rate $\eta$ can be automatically adjusted by the adaptive learning rate proposed in [15, 16].

In [10], it was shown that a GRBM can learn more meaningful filters when the binary hidden units were replaced by the noisy rectified linear units (NReLU). When NReLU hidden units are used, the approximate mean activation of the hidden unit becomes

$$h_j = \max\left(0, a_j\right), \quad (2)$$

where $a_j = \sum_i w_{ij} \frac{v_i}{\sigma_i^2} + c_j$ is the input to the $j^{\text{th}}$ hidden unit.

### 2.3. Feature Extraction and GRBM training

In this work, cross-correlation vectors from bandpass filtered speech signal were used as input to the GRBM to generate a set of features, according to the following description.

First, the left-ear $x_l(n)$ and right-ear $x_r(n)$ speech signals, where $n$ denotes the sample number, were filtered into 21 bandpass signals $X_l(n, d)$ and $X_r(n, d)$, respectively, where $d$ is the frequency channel. The center frequencies between 171 Hz and 7097 Hz of the bandpass filters were designed to match the centers of the triangular filters used in conventional audio-MFCC conversion.

Second, the unscaled cross-correlation values between windowed (i.e. framed) bandpass filtered signals, starting from sample $n$, $\boldsymbol{w}_l(n, d) = [X_l(n, d), \ldots, X_l(n + N - 1, d)]$ and $\boldsymbol{w}_r(n, d) = [X_r(n, d), \ldots, X_r(n + N - 1, d)]$ with lags $l$ ranging from $-50$ to 49 are computed as follows

$$R(n, l, d) = \begin{cases} \sum_{t=0}^{N-l-1} w_l(t+l, n, d) w_r(t, n, d)^* & l \geq 0 \\ R^*(n, -l, d) & l < 0 \end{cases}, \quad (3)$$

where $N = 256$ denotes the length of the rectangular window and $[]^*$ the complex conjugate. The cross-correlation vector for a frame starting at sample $n$ on channel $d$ is obtained by

$$\boldsymbol{x}_{corr}(n, d) = [R(n, -50, d), \ldots, R(n, 49, d)]. \quad (4)$$

A single GRBM with 50 hidden units was trained with 20,000 sample vectors in 2000 epochs and a mini-batch size of 64. Initially, the number of hidden units $n_h$ was varied from 25 to 150 at 25 unit intervals and in small scale testing, 50 units was found optimal. The sample vectors, $\boldsymbol{x}_{corr}(n, d)$ with random $n$ and $d$ values, were arbitrarily selected from the

CHiME development set utterances, described in Section 3.1, in a way that the training corpus contained approximately equal amount of data from all the frequency channels and signal-to-noise ratios (SNRs). NReLU hidden units, CD with the enhanced gradient and adaptive learning rate were used. A single $\sigma$ was shared and learned for all visible units.

In evaluation, the input vectors $\boldsymbol{x}_{corr}(n, d)$ to the GRBM were computed by converting the bandpass filtered and cross-correlated speech signals into a series of 256 samples long frames with consecutive frames overlapping by 128 samples.

### 2.4. Classifier

For classifying the TF units into reliable and unreliable, separate SVMs with radial basis function (RBF) kernels were trained for each frequency channel $d$. Oracle masks were used as targets, while the mean hidden activations of the GRBM given in Eq. (2) were taken as input features. The oracle masks were constructed using the noisy and clean CHiME development data to compute the exact SNR of each TF unit; only the units with SNR over 0 dB were labeled reliable. For each SVM-based system, a single RBF kernel width $\gamma$ was used. $\gamma$ values of $2.0$ were found optimal for both the baseline mask estimation system (BME+SVM) and for the system taking the $\boldsymbol{x}_{corr}(n, d)$ vectors directly as SVM input (XCOR+SVM). For the proposed GRBM mask estimation system (GME+SVM), the optimal $\gamma$ value was $2.5$. The widths were tuned by using the features computed from a set of 200 randomly selected utterances from all the SNRs of the CHiME development set.

In evaluation, TF regions that contained less than 20 connected reliable elements were removed from the masks.

### 2.5. Reconstruction of missing data

In this work, cluster-based imputation (CBI) is used to reconstruct the missing data. In CBI, a GMM is created to represent the distribution of feature vectors of clean speech. The model is used to fill the missing values of the observed feature vector with the most probable values. CBI assumes that the reliable components of the observation vector are the real values of a clean speech feature vector and the unreliable components represent an upper bound to the clean speech estimate; this is derived from the additive noise assumption which states that the energy of a signal with additive noise is always higher than the energy of a clean signal. A more detailed description of CBI can be found in [3] and [17].

The missing feature components were reconstructed in 21-dimensional log-compressed mel-spectral domain and the features were processed in 5-frame windows with a window shift of one frame as described in [18]. 1,500 randomly selected utterances from the CHiME training set were used to train a 13-component clean speech GMM with 105-variate component densities and full covariance matrices.

## 3. EXPERIMENTS

### 3.1. Data

Here, we use CHiME challenge data [19], where spoken commands are recognized from recordings made in a noisy living room using a binaural dummy head. Target speaker, represented by the binaural impulse responses of the dummy head, was in a fixed 2 meter in front position relative to the head. The data set is divided into training, development and evaluation sets. The training set consists of 17,000 utterances of reverberated but noise-free speech. The development and evaluations sets consist of 600 shared speaker utterances mixed with 6 different SNRs (from $-6$ to 9 dB at 3 dB intervals) giving 3,600 utterances in total.

### 3.2. Speech recognition system

The baseline system (BL) used in this work is a hidden Markov model (HMM) based large vocabulary continuous speech recognizer (LVCSR). The acoustic models of the BL system are speaker independent state-tied triphones. The triphone segmentations of the CHiME training data were generated by an LVCSR trained on the WSJ British English corpus [20] in "forced alignment" mode. The HMM states are modeled with at most 100 Gaussians (with diagonal covariance matrices) and their durations are modeled with gamma distributions. The speech signal is represented as frames of 12 MFCC and a frame power feature together with their first- and second-order derivatives. Cepstral mean subtraction and maximum likelihood linear transformation are also applied. A language model based on no-backoff bigrams with uniform frequencies for all valid bigrams is used.

The performances of the CHiME challenge baseline system (CBL) and a baseline mask estimation system based on 14 design features [5] and an SVM classifier (BME+SVM) are also presented in the current study. CBL and BL systems differ in that CBL is trained speaker dependently and whole-word HMMs are used.

For comparison, results of the BME system coupled with a GMM classifier (BME+GMM), trained with nine times more data than the SVM classifiers applied here, are presented from our previous paper [5]. The classifier of the BME+SVM system was trained with the 14 design features computed from the same set of 200 randomly selected utterances used to train the other SVM classifiers. The acoustic features used for mask estimation in the BME systems includes modulation-filtered spectrogram, mean-to-peak-ratio and gradient of the temporal envelope, harmonic and inharmonic energies, noise estimates from long-term inharmonic energy and channel difference, noise gain, spectral flatness, subband energy to subband noise floor ratio, ITD, ILD, peak ITD and interaural coherence.

**Table 1**. Keyword accuracy rates of CHiME baseline (CBL) system, our baseline system (BL), baseline mask estimation methods based on a 14-component design feature set with GMM (BME+GMM) and SVM (BME+SVM) classifiers, mask estimation method based on direct use of cross-correlation representation (XCOR+SVM), and the proposed GRBM mask estimation system (GME+SVM) for the CHiME development and evaluation sets.

| | Development set | | | | | | |
|---|---|---|---|---|---|---|---|
| | 9 dB | 6 dB | 3 dB | 0 dB | -3 dB | -6 dB | Avg. |
| CBL | 83.1 | 73.8 | 64.0 | 49.1 | 36.8 | 31.1 | 56.3 |
| BL | 83.3 | 80.0 | 69.8 | 55.2 | 46.0 | 40.6 | 62.5 |
| BME + GMM | 88.6 | 85.3 | 78.1 | 68.6 | 60.6 | 55.1 | 72.7 |
| BME + SVM | 89.7 | 87.4 | 78.7 | 67.7 | 58.3 | 53.6 | 72.5 |
| XCOR + SVM | 89.1 | 87.7 | 80.6 | 71.8 | 63.5 | 58.2 | 75.2 |
| GME + SVM | 90.0 | 87.1 | 82.2 | 73.1 | 64.1 | 59.4 | 76.0 |
| | Evaluation set | | | | | | |
| CBL | 82.5 | 75.0 | 62.9 | 49.5 | 35.4 | 30.3 | 55.9 |
| BL | 86.3 | 78.3 | 68.5 | 53.9 | 44.3 | 41.9 | 62.2 |
| BME + GMM | 90.3 | 84.3 | 76.9 | 68.2 | 58.2 | 56.3 | 72.3 |
| BME + SVM | **91.0** | 85.3 | 79.4 | 68.8 | 56.2 | 53.7 | 72.4 |
| XCOR + SVM | 90.5 | **86.1** | 80.0 | 69.2 | 57.4 | 55.8 | 73.1 |
| GME + SVM | 90.7 | 85.8 | **81.0** | **69.8** | **61.4** | **58.9** | **74.6** |

**Table 2**. Statistical significances of pairwise system comparisons of the evaluation set presented in Table 1. "+" denotes a statistically significant and "-" a non-significant difference.

| Pair | 9 dB | 6 dB | 3 dB | 0 dB | -3 dB | -6 dB | Avg. |
|---|---|---|---|---|---|---|---|
| BME+GMM - BME+SVM | - | - | + | - | - | + | - |
| BME+GMM - GME+SVM | - | - | + | - | + | - | + |
| BME+GMM - XCOR+SVM | - | - | - | - | - | - | + |
| BME+SVM - XCOR+SVM | + | - | - | - | - | - | + |
| BME+SVM - GME+SVM | - | - | - | - | + | + | + |
| XCOR+SVM - GME+SVM | - | - | - | - | + | + | + |

### 3.3. Results

The keyword accuracies of the systems are gathered in Table 1. The highest scores on each evaluation set SNR is shown in bold type. On evaluation set, the lowest accuracy rates at every SNR is obtained by CBL (55.9% on average) followed by BL (62.2% on average). BME+GMM and BME+SVM offer similar average performance (72.3% and 72.4%, respectively) but BME+GMM achieves higher accuracy rates in below zero SNR cases, whereas BME+SVM outperforms the BME+GMM at the higher SNR cases. The highest score at 9 dB is obtained by BME+SVM (91.0%). XCOR+SVM exceeds the accuracy rates of both BME systems on average (73.1%) and achieves the highest score on 6 dB case (86.1%). The proposed GME+SVM is the best performing system on average (74.6%) and at SNRs from 3 dB to $-6$ dB (81.0%, 69.8%, 58.9%, 74.6%, respectively).

Statistical significance of the keyword accuracy difference between each system pair on the evaluation set was computed by the Wilcoxon signed-rank test with a 95% confidence level and the results of the analysis are presented in Table 2.

## 4. DISCUSSION

We have presented a mask estimation method based on automatic feature extraction from cross-correlation representation of binaural speech signal using a GRBM and an SVM classifier (GME+SVM). The proposed method is able to learn features exceeding the performance of advanced design features.

In some of the previous studies, the common approach for time-frequency unit classification has been to develop descriptive heuristic measures, or design features, some of which are processed through a rather complex model [4, 5]. However, relevant information may be lost when data is described with just a few features. With the help of modern machine learning methods such as GRBM feature extraction coupled with an SVM classifier, we can overcome the problem by using input signals in a less refined format. Even without GRBM feature extraction, we achieved better results with SVM classifier utilizing raw cross-correlation data as input than with the design features. Similarly, a recent study by Wang et al. [21] suggested combining a number of standard ASR features that were less processed than design features for missing data mask estimation.

Initially, two alternative approaches were also considered for the feature extraction. First, learning the discriminating patterns directly from the noisy signal provided only a small improvement over the BL system in the current task. Second, making the net "deeper" showed no improvement in performance over the proposed method. These findings suggest that there may be room for an improvement in GRBM training algorithms. The next step could also be to investigate whether GRBMs were useful in reconstructing the missing data.

The recent work by Wang and Wang [11] boosted speech separation by modeling temporal dynamics with DBNs on monaural audio signal, and the work by Dahl et al. [12] and Hinton et al. [13] successfully adopted DBNs on various LVCSR tasks. Inspired by the recent advances in combining neural networks and ASR, we have continued our previous study applying multiple design features to missing data mask estimation [5] by using automatically generated features taking advantage of the modeling capabilities of GRBMs.

## 5. ACKNOWLEDGMENTS

The work was financially supported by Langnet (Keronen) and FICS (Cho) graduate schools, and by the Academy of Finland under the grants no 133145 (Raiko), 134935 (Ilin), 136209 (Palomäki), and 251170 Finnish Centre of Excellence Program (2012-2017).

This work was supported in part by the IST Programme of the European Community, under the PASCAL2 Network of Excellence, IST-2007-216886. This publication only reflects the authors' views.

## 6. REFERENCES


[1] M. Cooke, P. Green, and M. Crawford, "Handling missing data in speech recognition," in *Proc. ICSLP*, Yokohama, Japan, September 1994, pp. 1555–1558.

[2] M. Cooke, P. Green, L. Josifovski, and A. Vizinho, "Robust automatic speech recognition with missing and unreliable acoustic data," *Speech Communication*, vol. 34, no. 3, pp. 267–285, 2001.

[3] B. Raj, M. Seltzer, and R. M. Stern, "Reconstruction of missing features for robust speech recognition," *Speech Communication*, vol. 43, no. 4, pp. 275–296, 2004.

[4] M. Seltzer, B. Raj, and R. M. Stern, "A Bayesian classifier for spectrographic mask estimation for missing feature speech recognition," *Speech Communication*, vol. 43, no. 4, pp. 379–393, 2004.

[5] S. Keronen, H. Kallasjoki, U. Remes, G. J. Brown, J. F. Gemmeke, and K. J. Palomäki, "Mask estimation and imputation methods for missing data speech recognition in a multisource reverberant environment," *Computer Speech & Language*, in press.

[6] J. F. Gemmeke, Y. Wang, M. Van Segbroeck, B. Cranen, and H. Van hamme, "Application of noise robust MDT speech recognition on the SPEECON and SpeechDat-Car databases," in *Proc. INTERSPEECH*, Brighton, UK, September 2009, pp. 1227–1230.

[7] S. Harding, J. Barker, and G. J. Brown, "Mask estimation for missing data speech recognition based on statistics of binaural interaction," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 14, no. 1, pp. 58–67, 2006.

[8] G. Hinton and R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504–507, July 2006.

[9] G. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, pp. 1771–1800, August 2002.

[10] N. Jaitly and G. Hinton, "Learning a better representation of speech soundwaves using restricted Boltzmann machines," in *Proc. ICASSP 2011*, Prague, Czech Republic, May 2011, pp. 5884–5887.

[11] Y. Wang and D. Wang, "Boosting classification based speech separation using temporal dynamics," in *Proc. INTERSPEECH*, Portland, OR, USA, September 2012.

[12] 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*, vol. 20, no. 1, pp. 30–42, 2012.

[13] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, vol. 28, no. 6, 2012.

[14] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., Computer Science Department, University of Toronto, 2009.

[15] K. Cho, A. Ilin, and T. Raiko, "Improved Learning of Gaussian-Bernoulli Restricted Boltzmann Machines," in *Proc. ICANN*, 2011.

[16] K. Cho, T. Raiko, and A. Ilin, "Enhanced Gradient and Adaptive Learning Rate for Training Restricted Boltzmann Machines," in *Proc. ICML*, 2011.

[17] B. Raj and R. M. Stern, "Missing-feature approaches in speech recognition," *IEEE Signal Processing Magazine*, vol. 22, no. 5, pp. 101–116, 2005.

[18] U. Remes, Y. Nankaku, and K. Tokuda, "GMM-based missing feature reconstruction on multi-frame windows," in *Proc. INTERSPEECH*, Florence, Italy, 2011, pp. 2407–2410.

[19] J. Barker, E. Vincent, N. Ma, H. Christensen, and P. Green, "The PASCAL CHiME Speech Separation and Recognition Challenge," *Computer Speech & Language*, in press.

[20] T. Robinson, J. Fransen, D. Pye, J. Foote, and S. Renals, "WSJCAM0: a British English speech corpus for large vocabulary continuous speech recognition," in *Proc. ICASSP*, Detroit, MI, USA, 1995, pp. 81–84.

[21] Y. Wang, K. Han, and D. Wang, "Acoustic features for classification based speech separation," in *Proc. INTERSPEECH*, Portland, OR, USA, September 2012.

DISSERTATIONS IN INFORMATION AND COMPUTER SCIENCE

Aalto-DD158/2012 Virpioja, Sami
Learning Constructions of Natural Language: Statistical Models and Evaluations. 2012.

Aalto-DD20/2013 Pajarinen, Joni
Planning under uncertainty for large-scale problems with applications to wireless networking. 2013.

Aalto-DD29/2013 Hakala, Risto
Results on Linear Models in Cryptography. 2013.

Aalto-DD44/2013 Pylkkönen, Janne
Towards Efficient and Robust Automatic Speech Recognition: Decoding Techniques and Discriminative Training. 2013.

Aalto-DD47/2013 Reyhani, Nima
Studies on Kernel Learning and Independent Component Analysis. 2013.

Aalto-DD70/2013 Ylipaavalniemi, Jarkko
Data-driven Analysis for Natural Studies in Functional Brain Imaging. 2013.

Aalto-DD61/2013 Kandemir, Melih
Learning Mental States from Biosignals. 2013.

Aalto-DD90/2013 Yu, Qi
Machine Learning for Corporate Bankruptcy Prediction. 2013.

Aalto-DD128/2013 Ajanki, Antti
Inference of relevance for proactive information retrieval. 2013.

Aalto-DD205/2013 Lijffijt, Jefrey
Computational methods for comparison and exploration of event sequences 2013.