Gaussian Discrete Restricted Boltzmann Machine: Theory and Its Applications

Subha Manoharan

A thesis presented in partial fulfilment of the requirements of the degree of

Master of Engineering

in

Electronics and Computer Engineering

at Massey University, Albany,

New Zealand

2015

Abstract

Restricted Boltzmann Machine (RBM) is a two-layer neural network, popular for its efficient training methodology in many applications involving data recall, classification, and recognition. Traditionally RBM is designed with binary neurons in both layers. RBMs with Gaussian (continuous-valued) neurons in visible layer have been introduced for ease of integration with real data. However, the hidden layer still consists of binary neurons. Recently, theoretical studies in discrete RBM with discrete visible and hidden nodes have shown that increasing the number of hidden states improves reconstruction error. Motivated by this finding, the research in this thesis aims to develop an RBM with a Gaussian visible layer and a discrete multi-state hidden layer, called the Gaussian Discrete RBM (GDRBM). The equations governing this new model have been worked out and a contrastive divergence training algorithm has been developed based on these equations. Performance results using the MNIST and CBCL benchmark datasets show that the performance of a GDRBM with 4-state hidden neurons is approximately the same as that of other Gaussian RBMs with binary hidden neurons when the size of the hidden layer is doubled. This GDRBM has also been used to form one layer of a deep autoencoder. This is the first time an autoencoder has been designed with a multi-state discrete layer. Initial experimental results show that a GDRBM-based deep autoencoder is able to reconstruct the inputs reasonably well. However the pretraining is not very effective and the amount of initial reconstruction error need to be reduced to make it perform at the same level of a traditional deep autoencoder. Further research will be needed to understand how GDRBM could be used in a deep autoencoder.

Acknowledgements

I would like to express my heartfelt thanks to my supervisor Assoc. Prof. Edmund Lai for his guidance and great efforts throughout the duration of this degree. I am grateful to him for his unbelievable support to complete this thesis. His guidance not only teaches me research skills but also many life skills which help in my entire life.

I would like to extend my gratitude to Graduate Research School, Professors, PhD students and International office in Massey University for their support and assistance for this course.

I wish to express my special thanks to Chathurani Silvia and her family, Vaitheki Sanjeeharan, Ibtisam Abbas, Sadia Alam and Annaniah Sundarajan for their encouragement and moral support during the initial stage of my studies. I would also like to thank Mr. Raj Narayanan for his financial support during my stay in New Zealand. Thank you all for your personal support to complete this degree.

Lastly, I thank my entire family for their co-operation to complete this study.

This thesis would not be completed without the support from all the above mentioned people. Once again, I appreciate all friends, family and Massey University for their precious resources and encouragement.

I dedicate this thesis to my supervisor Dr. Edmund Lai to express my thankful towards his support.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
BM	Boltzmann Machine
CD	Contrastive Divergence
CBCL	Center for Biological and Computational Learning
DBN	Deep Belief Network
DBM	Deep Boltzmann Machine
FPCD	Fast Persistent Contrastive Divergence
GRBM	Gaussian RBM
GBRBM	Gaussian Bernoulli RBM
GDRBM	Gaussian Discrete RBM
IGRBM	Improved Gaussian Bernoulli RBM
IBGRBM	Improved Bernoulli Gaussian RBM
MSE	Mean Square Error
PCD	Persistent Contrastive Divergence
PT	Parallel Tempering
RNN	Recurrent Neural Network
RBM	Restricted Boltzmann Machine

Chapter 1
Introduction

1.1 Background

Artificial Neural Network (ANN) is a branch of Artificial Intelligence (AI) that is inspired by the architecture of the human brain. An ANN typically consists of a large number of relatively simple processing nodes (or artificial neurons) that are organized in layers and are interconnected with each other in specific ways. Some nodes are visible to the outside world while others are "hidden" and are not accessible directly from the input or the output. Associated with each connection between two nodes is a synaptic weight. Training an ANN involves computing the weights to adequately model a set of training data.

Since the discovery of the learning capabilities of the multi-layer perceptron, many different ANNs have been proposed. They include Hopfield network, Recurrent Neural Network (RNN), Radial Basis Network (RBN), self-organizing networks, neural-fuzzy networks, etc. In 1985, a type of recurrent ANN known as the Boltzmann Machine (BM) was introduced. It consists of a visible and a hidden layer, with symmetric and bidirectional connection between nodes in the two layers and between the hidden nodes. In training, the values taken from the training dataset is set as input to the visible layer. When training is completed, the weights represent an internal model of the training data [1]. It turns out that the connections among nodes in the same layer make the network very difficult to train [2], [3]. Thus a variant of BM, known as the Restricted Boltzmann Machine (RBM), is introduced by removing the connection between neurons in the same layer. This simplifies the equations involved and consequently promotes faster training. Recently, RBM has been used as the building block for constructing multilevel architectures generally known as deep learning architectures [4],[5], with applications to feature extraction, classification, image and speech recognition [6],[7],[8], among others.

The nodes in the original BM and RBM are binary. That is, the inputs and outputs of each node are binary values. This presents two main limitations. First, these networks are only applicable to binary datasets. However many applications involve data that

cannot be represented accurately in binary values [9]. Second, binary hidden nodes limit the amount of information that can be represented. Although these problems can be partially solved by exponentially increasing the number of nodes, this solution is hampered by training algorithms that are inefficient and sometimes ineffective. While training methodologies have recently improved [1]-[10], the accuracy of continuous-valued inputs is still compromised.

This problem led researchers to develop a variant of RBM that accepts continuous-valued inputs. One such RBM is the Gaussian RBM (GRBM). It has been shown that GRBM provides better representation than binary RBM [6, 10-13]. GRBM has allowed the RBM to be applied to more areas. While some GRBM characteristics have been studied [10, 14], much remains to be discovered. It should be noted that the hidden nodes in a GRBM are still binary even though the visible nodes accept continuous inputs. The reason for this is Gaussian neuron provides linear transformation of inputs [6] which is highly undesirable for learning interesting features from input [15].

Another type of neuron known as softmax that gives discrete multistate space is studied by many neural networks [9], [11, 16, 17]. But it is less popularized, due to its computational complexity [18]. However, there is evidence that multistate neurons reproduce the input with minimal error [19]. This hypothesis improves representation power in RBM.

Normally, RBM models and its variants are trained with gradient methods. The states and weights are updated by following the direction that minimizes global energy. However, these types of algorithms tend to get stuck in local minima [20]. Furthermore, difficulties in determining the normalizing constant and other difficulties have been reported, [21], [3]. A better training algorithm, known as Contrastive Divergence (CD), that makes use of a single step of Gibbs sampling have been proposed [1]. Following this, other related efficient algorithms such as Persistent Contrastive Divergence (PCD) [22], Fast Contrastive Divergence [23], and Parallel Tempering (PT) [10] appeared in the literature. In RBM, training efficiency is determined by parameters such as learning rate, weight cost and weight decay [9]. GRBM has an additional Standard Deviation parameter. This parameter controls the amount of noise in the GRBM which affects the training efficiency. Most training

algorithms keep the standard deviation constant [8]. However, there is evidence that scaling the energy expression by the variance improves results [13].

1.2 Research Aims and Objectives

The main aim of this research is to further improve the representation power of GRBM. Recently, theoretical study in discrete RBM with discrete visible and hidden nodes has shown that increasing the number of hidden states improves reconstruction error [19]. The hypothesis is that this is also true for RBM with a Gaussian visible layer. The main objective of this work is therefore to test the validity of this hypothesis. In order to do this, the binary hidden nodes in GRBM are replaced by discrete multi-state ones. This modified network will be called Gaussian Discrete RBM (GDRBM).

Since GDRBM is different from GRBM, the set of equations governing the model and a corresponding training algorithm will need to be developed. The second objective is therefore to formulate a training algorithm that is effective in training GDRBMs. The algorithm will be based on the CD algorithm due to its simplicity and effectiveness.

The final objective is to evaluate the performance of GDRBM by applying it to a number of benchmark problems and compare with those produced by RBM and GRBM. The applications include handwritten image and face recognition using the MNIST [24] and CBCL [25] datasets respectively. Its application to deep learning architectures will also be explored.

1.3 Contributions

First, a new RBM known as the GDRBM has been developed. It has a Gaussian visible layer and a non-binary discrete output layer. The conditional probabilities of the hidden layer as well as the CD-1 training algorithm are modified for this particular network. The performance of GDRBM has been compared through simulation experiments using the MNIST and CBCL datasets.

Secondly, a deep autoencoder is constructed with GDRBM as one of the layers and the results are analyzed. This is the first attempt in using a non-binary RBM layer for a deep autoencoder. Some initial experimental results are obtained and analyzed.

1.4 Thesis Organization

This thesis is divided into five chapters.

Chapter 2 reviews details of the RBM and its training methodology. Two types of discrete neurons – Binary and Softmax units and three variants of continuous RBMs-Hinton RBM, GBRBM, and IGBRBM are discussed. In particular, this chapter discusses modification of the energy function of GRBM from the one used for binary RBM.

The new GDRBM is described in detail in Chapter 3. The training algorithm is developed by CD1 algorithm. Its characteristics are explored through various experiments in MNIST and CBCL face datasets.

Chapter 4 applies the GDRBM to a deep learning network. In particular, a deep autoencoder is constructed with a GDRBM layer. The performance of this new deep autoencoder is compared with the traditional deep autoencoder.

Finally chapter 5 concludes this thesis with suggestions for future works.

Chapter 2
Restricted Boltzmann Machines

This chapter reviews the RBM and some of its variants which are suitable for discrete-valued data and continuous valued data. The efficient training methodologies also described.

2.1 Boltzmann Machine

Boltzmann Machine (BM) is an ANN constructed by two state neurons connected through symmetric weights [2]. Neurons are either visible or hidden from the outside world. Those that have connections to external inputs/outputs form the visible layer while the rest form the hidden layer. The state of each neuron is determined stochastically by calculating the energy of that particular neuron compared with the energy of whole network. Every neuron in this network is connected to all other neurons [26]. This network architecture makes it very difficult to perform statistical sampling during the learning phase and it takes a long time to reach stationary distribution [1], [5], [27]. In general, training a BM is time consuming [28], [29].

2.2 Restricted Boltzmann Machine

In order to overcome the problems in training the BM, connections between neurons in the same layer is removed, creating what is known as a Restricted Boltzmann Machine [1],[4]. The two-layer structure of an RBM is shown in Figure 2.1. The visible layer has neurons labelled as v_i and hidden layers neurons are labelled h_j. Connections between the two layers are symmetric with weight w_{ij} between v_i and h_j. In addition, each layer has its own external bias, denoted by a_i and b_j for the visible and the hidden layers respectively.

The energy of the network is defined by

$$E(v,h) = -a_i v_i - b_j h_j - v_i h_j w_{ij} \tag{2.1}$$

Since there are no connections between neurons in the same layer, neurons in each layer is conditionally independent given the other [12]. Hence, the state of each neuron can be obtained by the following conditional probabilities:

$$p(h_j \mid v_i) = \frac{e^{h_j(b_j + w_{ij} v_i)}}{\sum_{h_j} e^{h_j(b_j + w_{ij} v_i)}} \tag{2.2}$$

$$p(v_i \mid h_j) = \frac{e^{v_i(a_i + w_{ij} h_j)}}{\sum_{v_i} e^{v_i(a_i + w_{ij} h_j)}} \tag{2.3}$$

where h_j represents the state of the *j*-th hidden neuron and v_i denotes the state of the *i*-th visible neuron. Detailed derivations of conditional distributions can be found in Appendix 1.

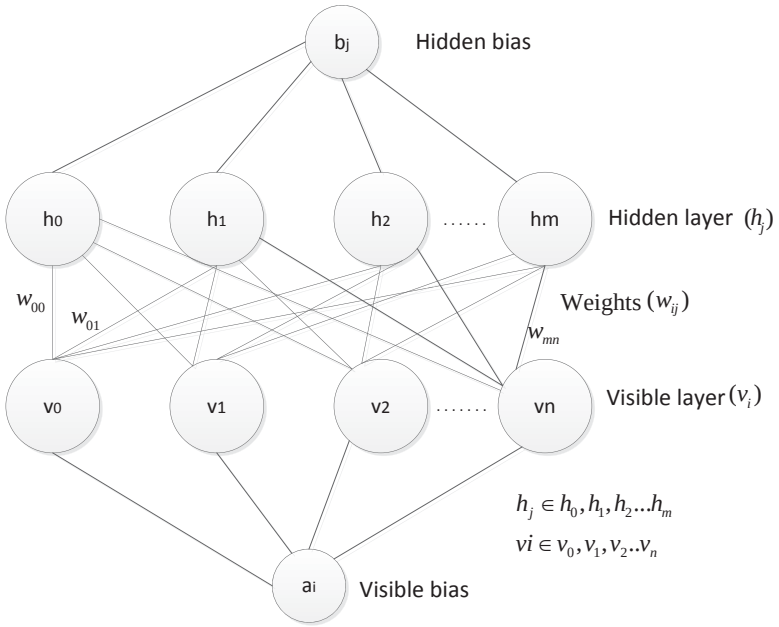


Figure 2.1 Architecture of Restricted Boltzmann Machine

2.2.1 Training Algorithms

Training an RBM involves estimating the parameters of the network $\theta = (w_{ij}, a_i, b_j)$ given a set of training data. This can be done using the maximum likelihood method

[20]. The idea is to find the parameter that maximizes the probability of input presented to the visible layer [1]. Since RBM is an energy based model, maximizing this probability is equivalent to minimizing the energy for the given input. Hence, the goal of the learning algorithm is to find the parameters that will lower the energy state of each neuron to maximize the joint probability distribution of whole network [30] which is given by,

$$p(v,h) = \frac{1}{z} e^{-E(v,h)} \tag{2.4}$$

The likelihood of the parameters θ of the RBM given the data in visible layer is defined as [31]:

$$L(\theta \mid v) = p(v \mid \theta) \tag{2.5}$$

For computational convenience, the log likelihood function is used instead [20]:

$$\ln L(\theta \mid v) = \ln\left(1/z\left(\sum_{h} e^{-E(v,h)}\right)\right) \tag{2.6}$$

Where z is normalizing constant and is given by $z = \sum_{v,h} e^{-E(v,h)}$. Substituting into(2.6), we have

$$\ln L(\theta \mid v) = \ln\left(\sum_{h} e^{-E(v,h)}\right) - \ln\left(\sum_{v,h} e^{-E(v,h)}\right) \tag{2.7}$$

With a large number of visible and hidden neurons, numerical approximation methods such as gradient ascent [32, 33] is usually used to find the maximum of this log likelihood function. Maximizing the likelihood function is equivalent to minimizing the weights. As weights are linearly related to energy by(2.1), minimizing the weights leads to minimizing energy. In each iteration of the gradient ascent, parameters are updated based on the direction of measured likelihood [34], which is given by the derivative:

$$\frac{\partial \ln L(\theta \mid v)}{\partial \theta} = -\frac{\sum_h e^{-E(v,h)}}{\sum_h e^{-E(v,h)}} \frac{\partial E(v,h)}{\partial \theta} + \frac{\sum_{v,h} e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}} \frac{\partial E(v,h)}{\partial \theta} \qquad (2.8)$$
$$= -\sum_h p(h \mid v) \frac{\partial E(v,h)}{\partial \theta} + \sum_{v,h} p(v,h) \frac{\partial E(v,h)}{\partial \theta}$$

Since θ consists of weights (w_{ij}), visible layer bias (a_i) and hidden layer bias (b_j), Equation (2.8) [27] can be split up into three parts:

$$\begin{aligned} \frac{\partial \ln L(\theta \mid v)}{\partial w_{ij}} &= < v_i h_j >_{data} - < v_i h_j >_{model} \\ \frac{\partial \ln L(\theta \mid v)}{\partial b_j} &= < v_i >_{data} - < v_i >_{model} \\ \frac{\partial \ln L(\theta \mid v)}{\partial a_i} &= < h_j >_{data} - < h_j >_{model} \end{aligned} \qquad (2.9)$$

Here $< v_i h_j >_{data}$ and $< v_i h_j >_{model}$ represent the expected values with respect to data and model distribution respectively. The idea behind these expectations is to find the number of times both layer units are on together during input layer takes inputs from training data and while recalling inputs from hidden layer [35].

These gradient values are used to update the parameters of the model:

$$\begin{aligned} \nabla w_{ij} &= \nabla w_{ij}(t-1) + \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \right) \\ \nabla b_j &= \nabla b_j(t-1) + \left(\langle h_j \rangle_{data} - \langle h_j \rangle_{model} \right) \\ \nabla a_i &= \nabla a_i(t-1) + \left(\langle v_i \rangle_{data} - \langle v_i \rangle_{model} \right) \end{aligned} \qquad (2.10)$$

A momentum term α and a learning rate ε can be introduced to control the convergence of the parameters [21]. With these two terms, the learning rules become:

$$\begin{aligned} \nabla w_{ij} &= \alpha \nabla w_{ij}(t-1) + \varepsilon \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \right) \\ \nabla b_j &= \alpha \nabla b_j(t-1) + \varepsilon \left(\langle h_j \rangle_{data} - \langle h_j \rangle_{model} \right) \\ \nabla a_i &= \alpha \nabla a_i(t-1) + \varepsilon \left(\langle v_i \rangle_{data} - \langle v_i \rangle_{model} \right) \end{aligned} \qquad (2.11)$$

In order to reduce unnecessary large weights, another useful learning parameter known as weight decay λ is used in the learning rule of weights [9]. Hence the learning rule for weights becomes,

$$\nabla w_{ij} = \alpha \nabla w_{ij}(t-1) + \varepsilon\left(\left\langle v_i h_j \right\rangle_{data} - \left\langle v_i h_j \right\rangle_{model}\right) - \lambda \nabla w_{ij}(t-1) \tag{2.12}$$

In order to speed up gradient updates for large datasets, training data are divided into several mini-batches. The learning rule for the gradient of each mini-batch is obtained by adding previous batch gradient ($w_{ij}(t-1)$) with current batch gradient (w_{ij}).

$$\begin{aligned} w_{ij} &= \nabla w_{ij} + w_{ij}(t-1) \\ b_j &= \nabla b_j + b_j(t-1) \\ a_i &= \nabla a_i + a_i(t-1) \end{aligned} \tag{2.13}$$

However, computing the gradient of the log likelihood is also practically difficult due to the expectations of the hidden and model distribution. Hence sampling methods are used to estimate the gradient of log likelihood [29]. Among many sampling methods, Gibbs sampling is one of the simplest if we know the conditional distribution of the variable (see Figure 2.2).

Since there is no connection between neurons of the same layer in a RBM, the conditional distribution of each layer is dependent only on the other layer. Thus Gibbs sampling for the two layers can be done in parallel and independently [1]. In other words, sampling does not depend on the previous states of the same layer. Hence, approximation of expectations in (2.11) and (2.12) are faster with the Gibbs sampling process.

All the efficient learning algorithms including contrastive divergence (CD) [1], Persistent Contrastive Divergence (PCD) [22], Fast Persistent Contrastive Divergence (FPCD) [23] and Parallel Tempering (PT) [36] all make use of Gibbs sampling to approximate log likelihood. Other approximation method such as Pseudo Likelihood [37], Score matching and Ratio matching [38] also exists to train RBM. These methods follow a straightforward mathematical calculation of log likelihood and are therefore computationally more expensive than Gibbs sampling methods.

Contrastive Divergence

CD is a basic RBM training methodology from which a number of other methods are derived. It consists of recurrent steps of Gibbs sampling to approximate the expectations of the data and model distributions by a large number of consecutive updates. Implementing Gibbs sampling for (2.8), the log likelihood approximation for one training pattern is given by [39]:

$$CD_k(\theta, v_0) = -\sum_h p(h \mid v_0) \frac{\partial E(v_0, h)}{\partial \theta} + \sum_h p(h \mid v_k) \frac{\partial E(v_k, h)}{\partial \theta} \qquad (2.14)$$

where $k = 1, 2, 3 \ldots \infty$ is the number of Gibbs steps. With sufficient Gibbs steps, the distribution of the model converges to the target distribution.

A single step of Gibbs sampling is processed in two phases, known as the positive and the negative phase. First, Gibbs sampling is initialized with the input data in the visible layer and probability distribution of hidden layer given visible layer (the first term in(2.14)) is obtained from the conditional distribution given in(2.2). This is the positive phase. Then the visible layer is updated given the hidden layer and reconstructed data is obtained from equation(2.3). Following this, the hidden layer is updated with the reconstructed data in the visible layer (the second term in (2.14)). This is the negative phase. This Gibbs step is illustrated in Figure 2.2.

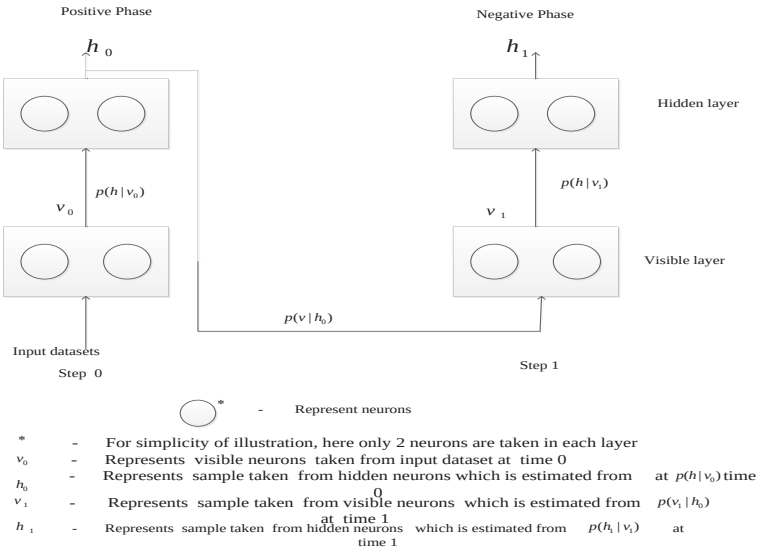


Figure 2.2 One step Gibbs Sampling

Accordingly, the first term of the learning rules in (2.11) can be easily calculated from the samples obtained in the positive phase of Gibbs sampling instead of calculating expected values over all the hidden nodes. For the second term we need to iterate the Gibbs step *k* times [39]. In this way, maximum likelihood estimation is performed through the CD-*k* algorithm without having to compute the actual expectations of the model. The full algorithm is shown in Algorithm 2.1.

The main advantage of this algorithm is that the speed of algorithm is independent of the number of neurons in the network. However, the training time increases linearly with *k*. Thus many neural network models use $k=1$, compromising accuracy with a speed up in training. Though some controversies exists when a small number of Gibbs iterations are used (see [40], [41]), RBMs trained using CD-1 could reconstruct the input reasonably well [9].

CDk Algorithm

For all minibatches,

1. In positive phase, Clamp input datasets in visible layer (v_0)

2. Find conditional Probability of Hidden layer given visible,

$$p(h_j \mid v_i) = \frac{e^{h_j(b_j + w_{ij}v_i)}}{\sum_{h_j} e^{h_j(b_j + w_{ij}v_i)}}$$

3. Find Deterministic hidden states by sampling (h_0)
4. Calculate positive phase samples,

$$posprods = v_0 * h_0$$
$$posvisact = sum(data)$$
$$poshidact = sum(h_0)$$

For CDk =1:K,

5. In negative phase, find conditional Probability of Visible layer given hidden layer (v_k)

$$p(v_i \mid h_j) = \frac{e^{v_i(a_i + w_{ij}h_j)}}{\sum_{v_i} e^{v_i(a_i + w_{ij}h_j)}}$$

6. Find Conditional probability of Hidden layer given visible layer(h_k)

$$p(h_j \mid v_i) = \frac{e^{h_j(b_j + w_{ij}v_i)}}{\sum_{h_j} e^{h_j(b_j + w_{ij}v_i)}}$$

7. Calculate negative phase samples,

$$negprods = v_k * h_k$$
$$negvisact = sum(v_k)$$
$$neghidact = sum(h_k)$$

end

6. Find gradient and update learning parameters (θ)

$$\nabla w_{ij} = \alpha \nabla w_{ij}(t-1) + \varepsilon(posprods - negprods)$$
$$\nabla b_j = \alpha \nabla b_j(t-1) + \varepsilon(poshidact - neghidact)$$
$$\nabla a_i = \alpha \nabla a_i(t-1) + \varepsilon(posvisact - negvisact)$$

$$w_{ij} = \nabla w_{ij} + w_{ij}(t-1)$$
$$b_j = \nabla b_j + b_j(t-1)$$
$$a_i = \nabla a_i + a_i(t-1)$$

end

Repeat all the above steps for large number of epochs

Algorithm 2.1 CD-k algorithm

Persistent Contrastive Divergence

Persistent Contrastive Divergence (PCD) [22] is a variant of the CD-*k* algorithm. With CD-*k*, in every Gibbs step, the change in the model distribution between consecutive parameter updates is small. In addition, it takes a long time to converge to a stationary distribution. PCD is designed to improve the speed of CD-*k* while estimating the model distribution. This algorithm is described in Algorithm 2.2.

PCD Algorithm

1. Initialize the visible layer with input datasets(v0)
   for epoch= 1:maxepoch
      for batch=1: minibatch

2. conditional distribution of hidden given visible(h0)

$$poshidprobs = p(h_j \mid v_i) = \frac{e^{h_j(b_j + w_{ij}v_i)}}{\sum_{h_j} e^{h_j(b_j + w_{ij}v_i)}}$$

3. Calculate positive phase

   poshidact = sum(poshidprobs)
   posvisact = sum(data)
   Posprods= poshidprobs *data

   if epoch =1 and batch=1
      neghidprobs= poshidprobs
   end

4. conditional distribution of visible given hidden(v1)

$$negdata = p(v_i \mid neghidprobs) = \frac{e^{v_i(a_i + w_{ij}h_j)}}{\sum_{v_i} e^{v_i(a_i + w_{ij}h_j)}}$$

5. conditional distribution of hidden given visible(h1)

$$neghidprobs = p(h_j \mid v_i) = \frac{e^{h_j(b_j + w_{ij}v_i)}}{\sum_{h_j} e^{h_j(b_j + w_{ij}v_i)}}$$

6. calculate negative phase

   neghidact = sum(poshidprobs)
   negvisact = sum(negdata)

   negprods= neghidprobs *negdata

7. calculate gradient and update parameters

$$\nabla w_{ij} = \alpha \nabla w_{ij}(t-1) + \varepsilon\left(posprods - negprods\right) \qquad w_{ij} = \nabla w_{ij} + w_{ij}(t-1)$$
$$\nabla b_j = \alpha \nabla b_j(t-1) + \varepsilon\left(poshidact - neghidact\right) \qquad b_j = \nabla b_j + b_j(t-1)$$
$$\nabla a_i = \alpha \nabla a_i(t-1) + \varepsilon\left(posvisact - negvisact\right) \qquad a_i = \nabla a_i + a_i(t-1)$$

end
end

Algorithm 2.2 PCD Algorithm

Instead of initializing the Gibbs chain from the input datasets for every mini-batch, the negative phase of the current mini-batch is initialized with negative samples of the previous mini-batch. Then a single Gibbs chain runs for the rest of the mini-batches to obtain the negative samples. Consequently, the negative samples do not depend on the input dataset after the first mini-batch (see step 3 in Algorithm 2.2). This feature shortens the time to estimate the negative samples, increasing the speed of approximation in maximum likelihood estimation. In particular, this algorithm improves the mixing rate of the Markov chain. The mixing rate can be further improved by adding fast weights with the actual weight, as with the Fast PCD algorithm [23].

Parallel Tempering

In general, Gibbs sampling does not mix well. That is, it takes a long time to reach a new region of the state space [42]. As a result, the negative phase samples do not follow the data distribution efficiently. Moreover, it requires large number of samples to cover all the regions of the data distribution. To overcome this issue, a sampling method known as parallel tempering is introduced [10].

The idea behind PT is to collect samples from multiple Gibbs chain with different energy levels. So it mixes well and the model distribution samples are close to that of the data distribution. At first, samples from two different chains each with a different temperature (i.e. energy) are collected and swapped according to

$$P_{swap}(X_{T1}, X_{T2}) = \min\left(1, \frac{P_{T1}(X_{T2})P_{T2}(X_{T1})}{P_{T1}(X_{T1})P_{T2}(X_{T2})}\right) \tag{2.15}$$

Then, from every pair, the one with $T = 1$ is used for learning the model. By iterating these steps a large number of times, the model distribution samples become very close to the data distribution. Experimental results showed that PT learning achieve better log likelihood estimation than CD with few Gibbs steps, e.g. CD-1. However, CD-*k* with $k > 25$ has better log likelihood estimates than PT.

With the proliferation of training algorithms for RBM, CD-1 remains the most popular for its simplicity and efficiency.

2.3 Discrete Restricted Boltzmann Machine

An RBM with discrete neurons is known as a Discrete RBM. The most popular discrete neurons are binary, softmax and multinomial [9]. However all these model follows the same Maximum Likelihood learning procedure and discussed in the previous section.

2.3.1 Binary Neurons

The original RBM has neurons with binary states (0 or 1) [1]. The conditional probability distributions of the binary states are expressed as sigmoid functions [39]:

$$p\left(h_j = 1 \middle| v_i\right) = sigmoid\left(a_i + w_{ij} v_i\right) \tag{2.16}$$

$$p\left(v_i = 1 \middle| h_j\right) = sigmoid\left(b_j + w_{ij} h_j\right) \tag{2.17}$$

Where, $sigmoid\left(x\right) = \dfrac{1}{1+e^{-x}}$

During training, samples of the conditional distribution are obtained from the Bernoulli distribution since the neurons are binary. Hence, the layers are also referred to as Bernoulli layers [35], [43], [10].

While RBM with binary neurons works well, it is limited to learn datasets that are binary. However, many applications involve data that are non-binary [44]. Therefore its application is rather limited [33], [45].

2.3.2 Softmax Neurons

Softmax is a normalized exponential function, which generalizes the sigmoid function to represent multiple classes [46],[47]. Neurons with activation defined by the softmax function are known as softmax neurons. They have multiple threshold levels in neuron activation. Hence, this type of neurons can represent *k* multiple states with $k \geq 2$. As the visible layer has direct interaction with the input, conventionally softmax units are used in the visible layer of RBM for applications with non-binary inputs. The hidden layer still consists of only binary units [33].

For RBM with softmax neurons in the visible layer, the conditional probability distributions of visible neurons given binary hidden nodes are based on the multinomial distribution as [48]:

$$p\left(v_i^{\,k}=1\,|\,h\right)=\frac{\exp\left(a_i^{\,k}+\sum_{j=1}^{n}w_{ij}^{\,k}h_j\right)}{\sum_{l=1}^{k}\exp\left(a_i^{\,l}+\sum_{j=1}^{n}w_{ij}^{\,l}h_j\right)} \tag{2.18}$$

Since the hidden neurons are binary, the conditional distribution of hidden neurons is given by

$$p\left(h_j=1\middle|V\right)=sigmoid\left(b_j+\sum_{i=1}^{m}\sum_{k=1}^{K}v_i^k w_{ij}^{\,k}\right) \tag{2.19}$$

where

$$V=\begin{Bmatrix}1, & v_i\in\left(k\neq 0\right)\\ 0, & v_i=0\end{Bmatrix}$$

For $k>2$, the definition of the energy of a neuron will need to be modified, otherwise it will keep increasing with *k*. The energy can be defined as

$$E(v,h)=-\sum_{j=1}^{n}\sum_{i=1}^{m}\sum_{k=1}^{K}W_{ij}^{\,k}h_j v_i^{\,k}-\sum_{i=1}^{m}\sum_{k=1}^{K}v_i^{\,k}a_i^{\,k}-\sum_{j=1}^{n}h_j b_j \tag{2.20}$$

with individual weights denoted by $W_{ij}^{\,k}$ and visible bias by $a_i^{\,k}$ for each state, and *m* and *n* represent the number of visible and hidden nodes respectively.

Since conditional distribution samples are taken from multinomial distribution [33, 48] instead of Bernoulli distribution of Binary RBM, Softmax neurons are also known as multinomial neuron [49].

An alternative softmax RBM has binary visible layer. It is used as encoders to encode *k*-dimensional inputs in a single unit [48]. Though visible layer is represented in binary, encoding is performed with respect to *k* digit observations and *n* nodes. For example, if

we have six binary visible neurons and $k=3, n=2$, then visible layer is encodes as 2 discrete neurons each with 3 digits. Thus, *V* has *n* discrete states and its conditional distribution given the hidden states can be expressed in the form of softmax units. That is,

$$p(v \mid h) = \prod_{i=1}^{n} p(v^i \mid h) \tag{2.21}$$

Can be expanded as

$$p\left(v^i = e_k \mid h\right) = \frac{\exp\left(a_i e_k + \sum_{j=1}^{n} w_{ij}{}^k h_j\, e_k\right)}{\sum_{k=1}^{K} \exp(a_i e_k + \sum_{j=1}^{n} w_{ij} h_j\, e_k)} \tag{2.22}$$

Hidden layer is same as binary RBM [50]. Since all the visible and hidden nodes are essentially binary, the energy of this RBM is the same as binary RBM.

Since the visible layer can only represent a finite number of values, both of these softmax RBM models are not able to represent continuous-valued datasets efficiently.

2.3.3 Discussions

Softmax neurons are logical extensions to binary ones for discrete values. However, sampling becomes more difficult as the number of states increases [48], with more noise introduced in the sampling process [11]. In addition, the number of parameters also increases linearly with the number of states [51],[33], increasing the cost of training. Recently, **theoretical** analysis based on KL Divergence shows that **increasing the number of states in the visible units increases the maximum approximation error [19].** Although it showed promise in a word-counting application [50], these disadvantages ultimately lead to a lack of interest on softmax units for RBMs. Instead, research has been focused more on RBMs with neurons that are continuous rather than discrete, at least for the visible layer.

2.4 RBMs for Continuous data

The visible layer is the connection between an RBM and the external world. Since many practical applications have data that are continuous, it is reasonable to consider RBMs with continuous visible neurons. Accordingly, several different continuous RBMs have been proposed. Gaussian neurons have been introduced in the visible layer to represent continuous valued inputs [11]. These types of visible units have been combined with different types of hidden units [6], [13], [50], [27]. RBMs with Gaussian visible neurons are generally referred to as Gaussian RBMs.

2.4.1 Gaussian RBM

The first Gaussian RBM is introduced by adding Gaussian noise to the binary visible units [11]. This model use probability values as instead of samples from the Gaussian distribution as input. Though it works well, it is a rough estimation method. In [6], the energy of a binary RBM is modified to,

$$E(v,h) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij} + \sum_i \frac{v_i^2}{\sigma_i^2} \tag{2.23}$$

with the addition of the last term on the right-hand-side of the equation. Here, σ_i is variance of the input data. This creates Gaussian distributed visible neurons instead of sigmoid neurons. It has been shown that this Gaussian RBM classify and predict inputs better than Binary RBMs.

Another Gaussian RBM is proposed in [52]. In this model, the energy function is given by

$$E(v,h) = \sum_i \frac{v_i^2}{2\sigma_i^2} - \frac{1}{\sigma_i^2}\left(\sum_{i \in visible} a_i v_i + \sum_{j \in hidden} b_j h_j + \sum_{i,j} v_i h_j w_{ij}\right) \tag{2.24}$$

Compared with Equation(2.23), the activation energy is scaled by the variance. This scaling has the effect of reducing the mean square error between the input and the output. Hence, it results in better reconstruction than the previous Gaussian RBM [53]. The conditional distributions of the visible and hidden units are,

$$p(v \mid h) = \mathcal{N}\left(a_i + w_{ij} h_j, \sigma^2\right)$$
$$p(h \mid v) = sigmoid\left(\frac{1}{\sigma^2}\left(b_j + \sum_i v_i w_{ij}\right)\right) \tag{2.25}$$

Where $\mathcal{N}(\cdot)$ is Gaussian density function. This thesis refers this model as Hinton RBM.

2.4.2 Gaussian Bernoulli RBM

The Gaussian Bernoulli RBM (GBRBM) proposed in [13] has Gaussian distributed visible units and Bernoulli distributed hidden units. The energy function of this model is given by,

$$E(v,h) = \sum_i \frac{(v_i - a_i)^2}{2\sigma_i^2} - \left(\sum_{j \in hidden} b_j h_j + \sum_{i,j} \frac{v_i}{\sigma_i} h_j w_{ij}\right) \tag{2.26}$$

Unlike previous models, the variance only applies to the Gaussian visible units. This also provides mathematical convenience in calculating(2.27). The corresponding conditional probabilities are given by,

$$p(v \mid h) = \mathcal{N}\left(b_i + \sigma_i h_j w_{ij}, \sigma_i\right)$$
$$p(h \mid v) = sigmoid\left(\frac{v_i}{\sigma_i} w_{ij} + b_j\right) \tag{2.27}$$

In all these models, the variance is assumed to be constant throughout the training. This constant variance leads to noisy results if it is too high. Alternatively, training typically gets stuck in poor local minima when it is too small [54]. One solution is to estimate σ using maximum likelihood estimation of sigma values, similar to other parameters of the model. The learning rules therefore include σ as follows:

$$
\begin{aligned}
\Delta\sigma_i &= \varepsilon_\sigma \left(E_{\text{data}} \left(\frac{(\mathrm{v_i} - a_i)^2}{\sigma_i^3} - \sum_j h_j . \frac{w_{ij} v_i}{\sigma_i^2} \right) - E_{\text{model}} \left(\frac{(\mathrm{v_i} - a_i)^2}{\sigma_i^3} - \sum_j h_j . \frac{w_{ij} v_i}{\sigma_i^2} \right) \right) \\
\Delta w_{ij} &= \frac{\left(E_{data}\left(v_i h_j\right) - E_{\text{mod}el}\left(v_i h_j\right) \right)}{\sigma_i} \\
\Delta a_i &= \left(\frac{E_{\text{data}}(\mathrm{a_i}) - E_{\text{model}}(\mathrm{a_i})}{\sigma_i^2} \right) \\
\Delta b_j &= \left(E_{\text{data}}(\mathrm{b_j}) - E_{\text{model}}(\mathrm{b_j}) \right)
\end{aligned}
\tag{2.28}
$$

Although the GBRBM has advantages, learning the standard deviation increases the complexities in training. There are also practical numerical problems with two different scaling factors (σ^3 and σ^2) used in the learning rule for σ and leads to improper gradient update. This concern is addressed in a more recent Gaussian RBM called the improved Gaussian Bernoulli RBM.

2.4.3 Improved Gaussian Bernoulli RBM

The Improved GBRBM (IGBRBM) [10] has an energy function given by,

$$
E(v,h) = \sum_i \frac{(v_i - a_i)^2}{2\sigma_i^2} - \left(\sum_{j \in hidden} b_j h_j + \sum_{i,j} \frac{v_i}{\sigma_i^2} h_j w_{ij} \right) \tag{2.29}
$$

Which is the same as that for GBRBM except the scaling factor in the last term is σ^2 instead of σ. The corresponding conditional probability distributions become

$$
\begin{aligned}
p(v \mid h) &= \mathcal{N}\left(b_i + h_j w_{ij}, \sigma_i^2 \right) \\
p(h \mid v) &= sigmoid\left(\frac{v_i}{\sigma_i^2} w_{ij} + b_j \right)
\end{aligned}
\tag{2.30}
$$

and the learning rules are

$$
\begin{aligned}
\Delta\sigma_i &= \varepsilon_\sigma\left(E_{\text{data}}\left(\frac{(\mathrm{v_i}-a_\mathrm{i})^2}{\sigma_i^{\,3}}-2\sum_j h_j.\frac{w_{ij}v_i}{\sigma_i^{\,3}}\right)-E_{\text{model}}\left(\frac{(\mathrm{v_i}-a_\mathrm{i})^2}{\sigma_i^{\,3}}-2\sum_j h_j.\frac{w_{ij}v_i}{\sigma_i^{\,3}}\right)\right)\\
\Delta w_{ij} &= \frac{\left(E_{data}\left(v_i h_j\right)-E_{model}\left(v_i h_j\right)\right)}{\sigma_i^{\,2}}\\
\Delta a_i &= \left(\frac{E_{\text{data}}(\mathrm{a_i})-E_{\text{model}}(\mathrm{a_i})}{\sigma_i^{\,2}}\right)\\
\Delta b_j &= \left(E_{\text{data}}(\mathrm{b_j})-E_{\text{model}}(\mathrm{b_j})\right)
\end{aligned}
\tag{2.31}
$$

Simulation results show that reduces in the learned weights; noise is reduced compared with GBRBM.

Chapter 3

Gaussian Discrete Restricted Boltzmann Machine

The power of ANN lies in the nonlinearity of its neurons. While continuous Gaussian neurons provide a better interface with real valued data, it is linear. This is the main reason why RBMs with both Gaussian visible and hidden layers do not work well. Therefore, most of the continuous RBM models described in Chapter 2 have a binary hidden layer. The theoretical analysis presented in [19] shows that reconstruction error can be reduced by increasing the number of hidden states. However, this analysis has not been backed up by the performance of an actual RBM model with a continuous Gaussian visible layer and a discrete hidden layer that is non-binary. It is the aim of this research to do so.

This chapter presents the new Gaussian Discrete RBM (GDRBM) which is an extension of the IGBRBM [10]. The rest of this Chapter describes the architecture and the training algorithm of GDRBM, followed by simulation results that demonstrate the characteristics of this RBM using some benchmark data.

3.1 Architecture of GDRBM

Similar to other RBMs, GDRBM is a two-layer neural network. The two layers are interconnected by bidirectional symmetric weights. The visible layer neurons have Gaussian states and those in the hidden layer have discrete states. Softmax units, described in Section 2.3.2, are used to represent multiple states of hidden units. The neurons in individual layers share the same bias term. A similar kind of Gaussian–Softmax RBM has previously been proposed in [17]. However, the energy function and the objective of that model are different.

In rest of this chapter, v_i and a_i represent the continuous-valued output of the i-th Gaussian visible neuron and its bias term respectively. h_j and b_j represent the discrete output of the *j*-th hidden neuron and its bias respectively. The symmetric weight between the *i*-th visible and the *j*-th hidden layer is represented by w_{ij}. There are n_v visible neurons and n_h hidden neurons.

3.2 Training Methodology

GDRBM can be trained by the Contrastive Divergence algorithm [1]. The energy function is same as that for IGBRBM which is,

$$E(v,h)=\sum_{i=1}^{n_v}\frac{(v_i-a_i)^2}{2\sigma_i^{\,2}}-\sum_{i=1}^{n_v}\sum_{j=1}^{n_h}w_{ij}h_j\frac{v_i}{\sigma_i^{\,2}}-\sum_{j=1}^{n_h}b_jh_j \tag{2.32}$$

Where $\sigma_i^{\,2}$, represents the variance of the Gaussian distribution. Using Equation(2.32), the conditional probabilities for the visible layer given hidden layer is derived from the Gibbs distribution in the same way as IGBRBM, which are given by

$$p(v=v_i|h)=N\left(v\,|\,a_i+\sum_j h_jw_{ij}\ ,\sigma_i^{\,2}\right) \tag{2.33}$$

Where N () represents Gaussian density function with mean $\left(v\,|\,a_i+\sum_j h_jw_{ij}\right)$ and variance $\sigma_i^{\,2}$.

Given that the joint probability distribution $p(v,h)$ of visible, hidden units is

$$p(v,h)=\frac{1}{z}e^{-E(v,h)}$$

and the marginal probability of visible units $p(v)$ is

$$p(v)=\frac{1}{z}\sum_h e^{-E(v,h)}$$

The conditional probability of hidden layer given visible layer is derived as follows:

$$p(h\,|\,v)=\frac{p(v,h)}{p(v)}=\frac{e^{-E(v,h)}}{\sum_h e^{-E(v,h)}} \tag{2.34}$$

The probability of single hidden neuron in the hidden layer, expressed in terms of the energy function(2.32) , is given by

$$p\left(h_j \mid v\right)=\frac{e^{h_j\left(b_j+\sum_{i=1}^{n_h} w_{ij}\frac{v_i}{\sigma_i}\right)}}{\sum_h e^{h_j\left(b_j+\sum_{i=1}^{n_h} w_{ij}\frac{v_i}{\sigma_i}\right)}} \tag{2.35}$$

Assuming the hidden units have $\left(n+1\right)$ discrete states, the value of h_j can vary from 0 to *n*. Equation (2.35) can then be written as

$$p(h_j=h \mid v)=\frac{e^{h(b_j+\sum_{i=1}^{n_h} w_{ij}\frac{v_i}{\sigma_i})}}{\sum_{h_j=0}^{n} e^{h_j(b_j+\sum_{i=1}^{n_h} w_{ij}\frac{v_i}{\sigma_i})}} \tag{2.36}$$

Where $h\in\left[0,n\right]$. The state of hidden node at a given time is sampled from the categorical distribution, i.e.

$$p\left(h \mid v\right)=categorical\left(b_j+\sum_{i=1}^{n_h} w_{ij}\frac{v_i}{\sigma_i}\right) \tag{2.37}$$

3.2.1 Maximum Likelihood Learning

The log likelihood function of this model is given by

$$\ln L\left(\theta \mid v\right)=\ln\left(\sum_h e^{-E(v,h)}\right)-\ln\left(\sum_{v,h} e^{-E(v,h)}\right) \tag{2.38}$$

A gradient descent based algorithm is used to find the maximum of the negative likelihood. The derivative of the log likelihood with respect to the parameters θ is given by

$$\frac{\partial \ln L\left(\theta \mid v\right)}{\partial\theta}=\sum_h p(h \mid v)\frac{\partial E\left(v,h\right)}{\partial\theta}-\sum_{v,h} p\left(v,h\right)\frac{\partial E\left(v,h\right)}{\partial\theta} \tag{2.39}$$

In this equation, the first term on the right-hand-side represents the expected value of the energy function under conditional distribution of hidden variables given the training data. The second term represents the expected value of the energy function of the model distribution [19].

θ Consists of four sets of parameters: $w_{ij}, a_i, b_j, \sigma_i$. The learning rule for each parameter is obtained by differentiating Equation (2.32) with respect to that parameter and then substituting into Equation(2.39). This process is illustrated below for the weights w_{ij}. In this case, the first term becomes

$$\sum_h p(h|v)\frac{\partial E(h,v)}{\partial w_{ij}} = \sum_{h_j} p(h_j|v) h_j v_i \tag{2.40}$$

With discrete hidden states $h_j = 0,1,2,\ldots,n$, we have

$$\begin{aligned}\sum_h p(h\,|\,v)\frac{\partial E(h,v)}{\partial w_{ij}} &= p(h_j = 0\,|\,v)v_i(0) + p(h_j = 1\,|\,v)v_i(1) \\ &\quad + p(h_j = 2\,|\,v)v_i(2) + \ldots + p(h_j = n\,|\,v)v_i(n) \\ &= 0 + p(h_j = 1|\,v)v_i + 2p(h_j = 2\,|\,v)v_i + \ldots + np(h_j = n\,|\,v)v_i\end{aligned} \tag{2.41}$$

Similarly the second term of (2.39) is expressed as

$$\begin{aligned}\sum_{v,h} p(v,h)\frac{\partial E(h,v)}{\partial w_{ij}} &= \sum_v \sum_h p(v)\, p(h\,|\,v)\; h_j v_i \\ &= \sum_v p(v) \sum_h p(h\,|\,v)\; h_j v_i \\ &= \sum_v p(v)\left(p(h_j = 1|\,v)v_i + \ldots + np(h_j = n\,|\,v)v_i \right)\end{aligned} \tag{2.42}$$

For the whole training set D, the derivative is obtained by taking the mean of the gradient of all data [39]. Thus,

$$\frac{1}{l}\sum_{v\in D}\frac{\partial \ln L(\theta\,|\,v)}{\partial w_{ij}} = -\frac{1}{l}\sum_{v\,/\,D}\left[E_{p(h|v)}\left(\frac{\partial E(h,v)}{\partial w_{ij}}\right) + E_{p(v,h)}\left(\frac{\partial E(h,v)}{\partial w_{ij}}\right)\right] \tag{2.43}$$

Where l is the size of the training set, $E_{p(h|v)}\left(\frac{\partial E(h,v)}{\partial w_{ij}}\right)$ and $E_{p(v,h)}\left(\frac{\partial E(h,v)}{\partial w_{ij}}\right)$ is the mean of the gradient in the $p(h|v)$ phase and the $p(v,h)$ phase respectively.

Differentiating (2.32) partially with respect to w_{ij}, we get $\left(\frac{\partial E(h,v)}{\partial w_{ij}}\right) = v_i h_j$. By substituting this into (2.43) , we have

$$\frac{1}{l}\sum_{v\rceil D}\frac{\partial \ln L(\theta|v)}{\partial w_{ij}} = \frac{1}{l}\sum_{v\rceil D}\left[\mathrm{E}_{p(h|v)}(v_i h_j) - \mathrm{E}_{p(h,v)}(h_j v_i)\right] \tag{2.44}$$

Thus the learning rule with respect to weight is obtained as,

$$\frac{\partial \ln L(\theta|v)}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{2.45}$$

In above equation, $\langle v_i h_j \rangle_{data}$ is called the positive phase, $\langle v_i h_j \rangle_{model}$ is called the negative phase.

The gradients with respect to a_i, b_j, σ_i can be derived in a similar way, giving us

$$\frac{\partial \ln L(\theta|v)}{\partial a_i} = \frac{1}{\sigma_i^2}(\langle v_i \rangle_{data} - \langle v_i \rangle_{model}) \tag{2.46}$$

$$\frac{\partial \ln L(\theta|v)}{\partial b_j} = \langle h_j \rangle_{data} - \langle h_j \rangle_{model} \tag{2.47}$$

$$\frac{\partial \ln L(\theta|v)}{\partial \sigma_i} = \frac{1}{\sigma_i^3}\left(\left\langle (v_i - b_i)^2 - 2\sum_j v_i h_j w_{ij} \right\rangle_{data} - \left\langle (v_i - b_i)^2 - 2\sum_j v_i h_j w_{ij} \right\rangle_{model}\right) \tag{2.48}$$

From (2.42) it is clear that calculating the expectations of the data over large training datasets are difficult. The CD-1 algorithm can be used to approximate the expectations using Gibbs sampling.

The parameter update rules are given by

$$w_{ij} = w_{ij} + \frac{1}{\sigma_i^2}\left(\left\langle \mathrm{v}_i\ \mathrm{h}_j \right\rangle_{\text{data}} - \left\langle \mathrm{v}_i\ \mathrm{h}_j \right\rangle_{\text{model}}\right) \tag{2.49}$$

$$b_j = b_j + \left(\left\langle h_j \right\rangle_{data} - \left\langle h_j \right\rangle_{\text{model}}\right) \tag{2.50}$$

$$a_i = a_i + \frac{1}{\sigma_i^2}\left(\left\langle v_i \right\rangle_{\text{data}} - \left\langle v_i \right\rangle_{\text{model}}\right) \tag{2.51}$$

$$\sigma_i = \sigma_i + \frac{1}{\sigma_i^3}\left(\left(\left(v_i - a_i\right)^2 - 2\sum_j v_i h_j w_{ij}\right)_{data} - \left(\left(v_i - a_i\right)^2 - 2\sum_j v_i h_j w_{ij}\right)_{model}\right) \tag{2.52}$$

3.3 Performance Analysis Using MNIST Datasets

The learning abilities and characteristics of GDRBM are evaluated using the publicly available MNIST handwritten image dataset [24]. This dataset contains 60000 training sets and 10000 test sets. Each image is a handwritten digit between 0 and 9. They are binary (black and white) images of 28×28 pixels. To provide batch gradient analysis, the training sets are divided into 600 batches each containing 100 cases. Simulations are performed using MATLAB.

In these simulations, we used hidden neurons with 4 states, i.e. $n = 3$. Thus we have following conditional probabilities:

$$p\left(h_j = 0 \mid v\right) = \frac{1}{1 + e^{\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)} + e^{2\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)} + e^{3\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)}} \tag{2.53}$$

$$p\left(h_j = 1 \mid v\right) = \frac{1}{1 + e^{-\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)} + e^{\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)} + e^{2\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)}} \tag{2.54}$$

$$p\left(h_j = 2 \mid v\right) = \frac{1}{1 + e^{-\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)} + e^{-2\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)} + e^{\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)}} \tag{2.55}$$

$$p\left(h_j = 3 \mid v\right) = \frac{1}{1 + e^{-\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)} + e^{-2\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)} + e^{-3\left(b_j + \sum_{i=1}^{n} w_{ij}\frac{v_i}{\sigma_i^2}\right)}} \tag{2.56}$$

The following parameters are used in this experiment: the learning rate for the weights is 10^{-3}, bias is$10^{-8}, \sigma = 2^{-5}$, the range of weight cost and momentum is (0.5, 0.8). The network is trained using CD -1 algorithm for 50 epochs.

3.3.1 Reconstruction Error

Reconstruction error reflects how well the network has learned the training data. It is measured by Mean Square Error (MSE) between the input data and the reconstructed data. As the GDRBM is a probabilistic network, the reconstruction error varies in each trial. Hence, the average MSE values of 10 trials are used. The following results show the effects of the number of training epochs, the number of hidden nodes and the learning rate on the reconstruction error.

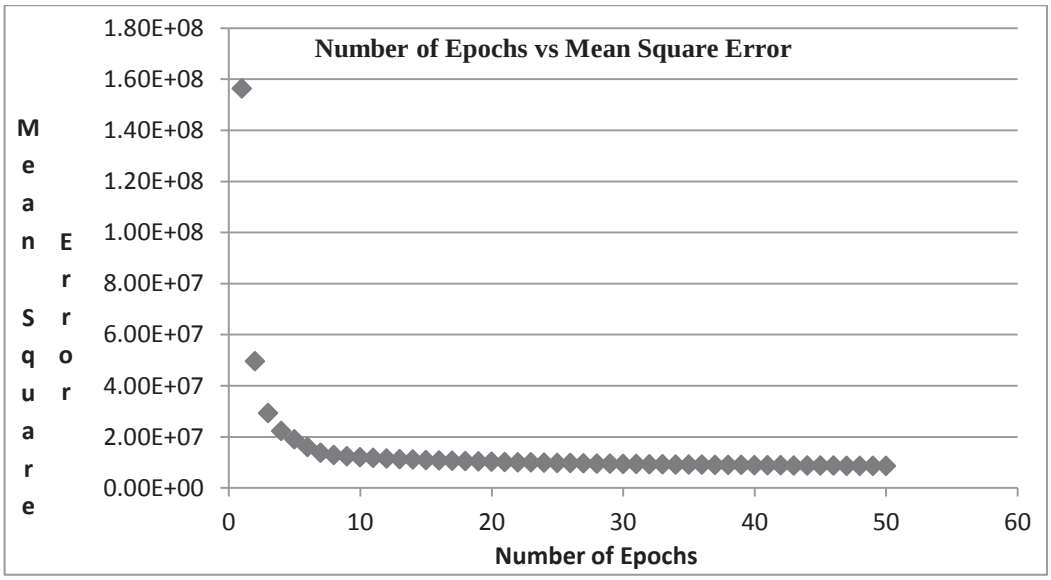


Figure 3.1 Number of Epochs vs Mean Square Error

First, the network is trained with the number of epoch varying from 1 to 50. The results, plotted in Figure 3.1, show that the average MSE decreases exponentially as the number of epochs is increased from 1 to 10. Thereafter, it reduces slowly and remains almost constant between 45 and 50 epochs.

Figure 3.2 shows that the above trend holds as the size of the hidden layer changes. For a network that has been trained for 50 epochs, the size of the hidden layer has no significant effect on the reconstruction error with 50 training epochs as shown in Table

3. 1. Interestingly, for this particular dataset, the MSE is higher for 200 to 600 hidden nodes compared with 100 and 700 or more hidden nodes.

The effects of the learning rate are shown in Figure 3.3. The performance of the binary RBM is sensitive to learning rate changes. The results here show that this is also true for GDRBM. Learning rates of 10^{-3} and 10^{-4} gives the best performance. The network weights do not converge well for too small a learning rate. For a learning rate of 10^{-2}, the MSE after 50 epochs is 3.52×10^{22} which is not shown in Figure 3.3 since it is much larger than the other values.

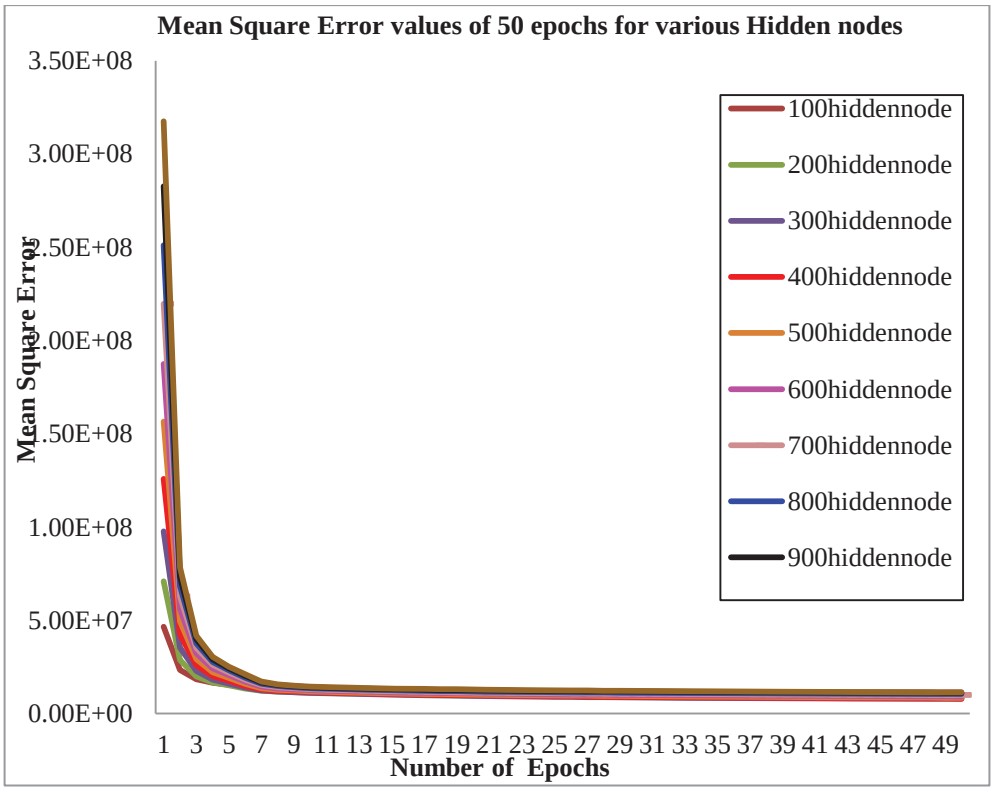


Figure 3.2 Number of epochs vs Mean Square Error

Hiddennodes	100	200	300	400	500	600	800	1000
MSE	1.14E+07	8.54E+06	7.90E+06	8.04E+06	8.53E+06	9.22E+06	1.05E+07	1.14E+07

Table 3. 1 MSE for various Hidden nodes

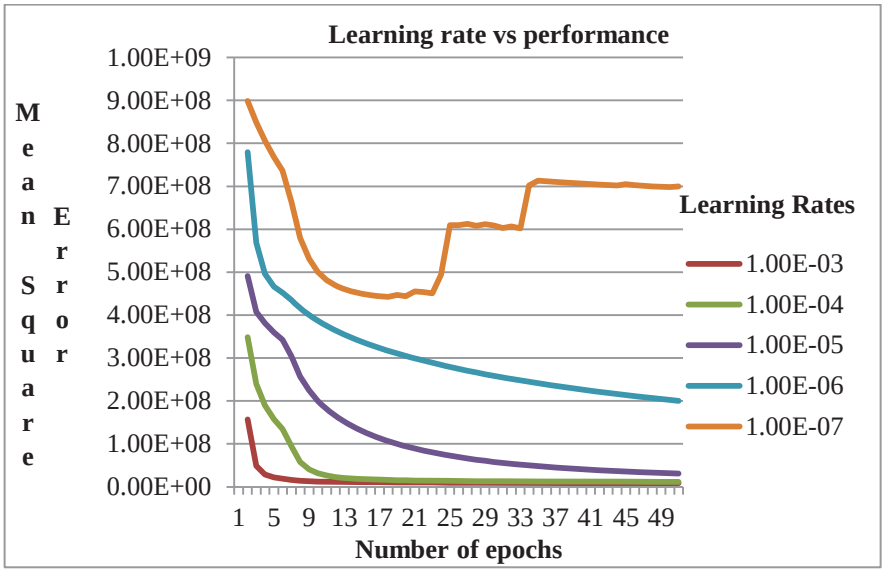
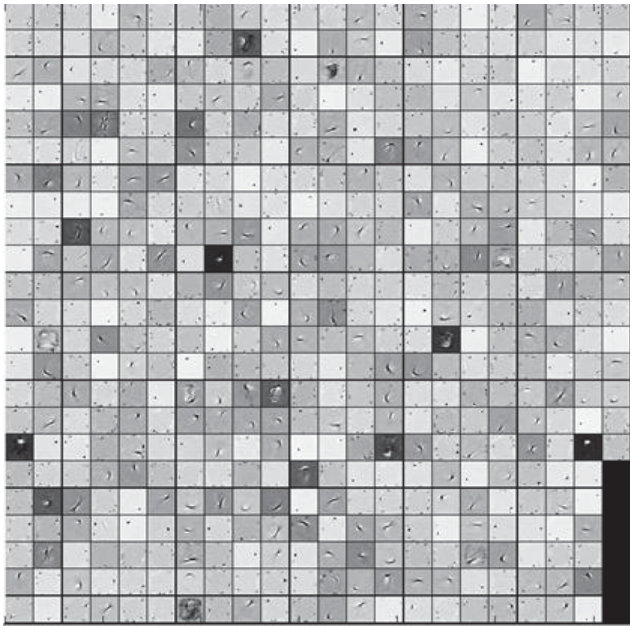


Figure 3.3 Performance of GDRBM in different Learning Rate

In summary, a GDRBM with 500 hidden nodes, learning rate of 10^{-3} and 50 epochs of training works well for this set of data.

3.3.2 Visualization of Weight Distribution

The weights of the connections between the visible and the hidden layer for 500 hidden nodes and 50 epochs of training have values between -0.91 and +0.4. For each hidden node, there are $28 \times 28 = 784$ connections to the visible layer for 28×28 pixel input images. The weights are normalized to values between 0 and 1 and displayed as intensity values in Figure 3.4. The weight distribution is consistent with other traditional RBMs [55], [56]. However, some hidden neurons in GDRBM learn more local features.



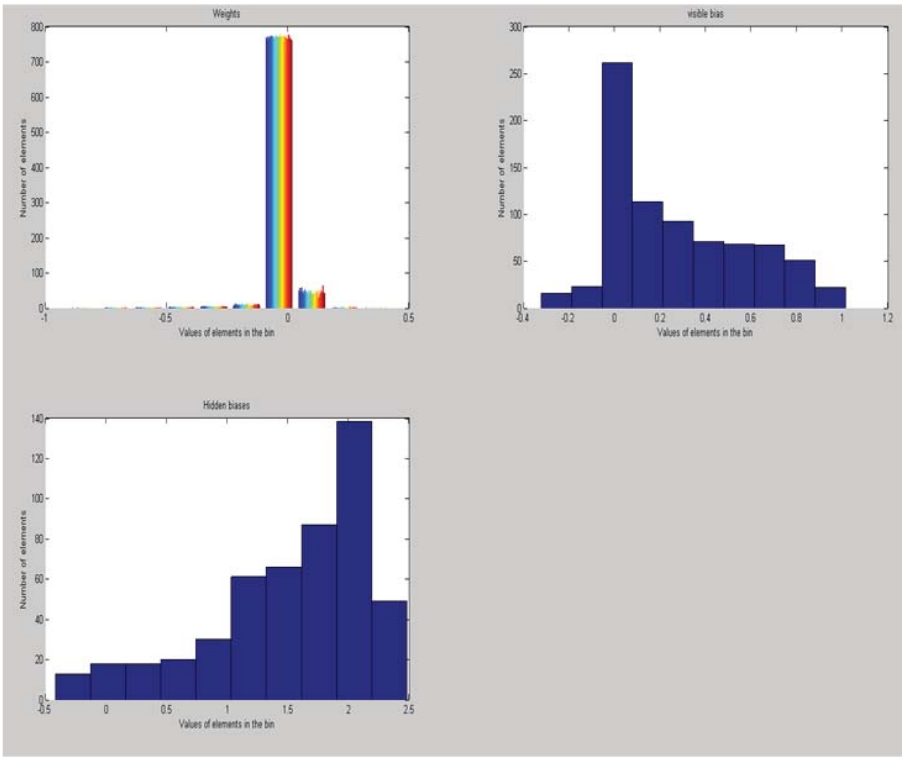
In figure, each square in an image represents 28*28 image of single hidden neuron. Black pixels represents 0, White pixels represents 1, Grey pixels represents values in between 0 to 1.

Figure 3.4 Learned weights after 50 epochs for GDRBM

3.3.3 Histogram Analyses

Hidden layer bias values provide more information on the contribution of hidden neurons. For example, if the hidden bias has large positive values, then the corresponding neurons of bias are always on. On the other hand, if it has large negative values, then they do not contribute much to the encoding of the input [10].

The histograms of visible layer bias, hidden layer bias and weights are shown in Figure 3.5. It can be observed that most of the hidden neuron biases are positive and approximately only twenty neurons are negative. Hence, very few hidden neuron do not contribute much to the encoding of the inputs. The visible neuron biases are also mostly positive but have smaller values compared with the hidden ones.



In Figure, x axis represents values of elements in each bin. Y axis represents number of elements in each bin. Each color in the bar represents a single group of elements.

Figure 3.5 Histogram of weights, visible bias and hidden bias

The distribution of weights is fairly uniform within a narrow range which is expected from our discussions in Section 3.3.2. This shows that the encoding of information is evenly spread out over the entire network which is one of the characteristics of a neural network.

3.3.4 Visualization of Visible Nodes

While the reconstruction error gives us an indication of the encoding and recall ability of the network, it is important in this case to see how well the inputs are reconstructed by the network. In Figure 3.6, the top row shows the input from the MNIST dataset and the bottom row shows an example of the reconstructed images in the visible layer. Fifteen samples are shown. While there is some noise in the reconstructed outputs, in general, it shows that GDRBM learn and reconstruct the input reasonably well.

In figure, top row is sample of input data given in visible layer. Bottom row represents reconstructed data after 50 epochs of training

Figure 3.6 Visualization of input and reconstructed image

3.3.5 Visualization of Hidden Layer Probabilities

If a network is trained well, the hidden layer probability values should be evenly distributed. Figure 3.7 shows the hidden probabilities of two networks. The top one has initial weight of 0.1 and it shows more white pixels which indicate that more white neurons are turned on. This implies that the network could not converge enough.

Decreasing the initial values to 0.001 gives a hidden probability image with more grey pixels as shown in bottom figure in Figure 3.7. Further reduction of the initial weight does not provide any further improvements.

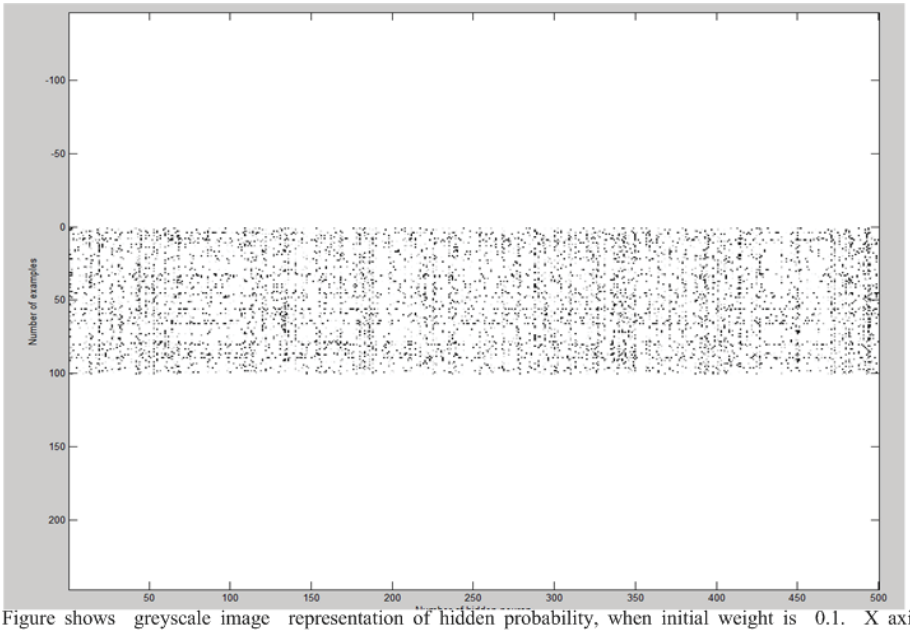


Figure shows greyscale image representation of hidden probability, when initial weight is 0.1. X axis represents Number of hidden neuron and y axis represents number of examples. In Figure probability of 100 examples of 500 hidden nodes after 50 epochs of training are shown as a pixel image. In this, each column shows probability value of one neuron for different examples. Each row represents probability value of one example for different hidden neurons. Here, probability of 0 represents black pixel, 1 represents white pixel. Gray represents values in between (0 to 1).

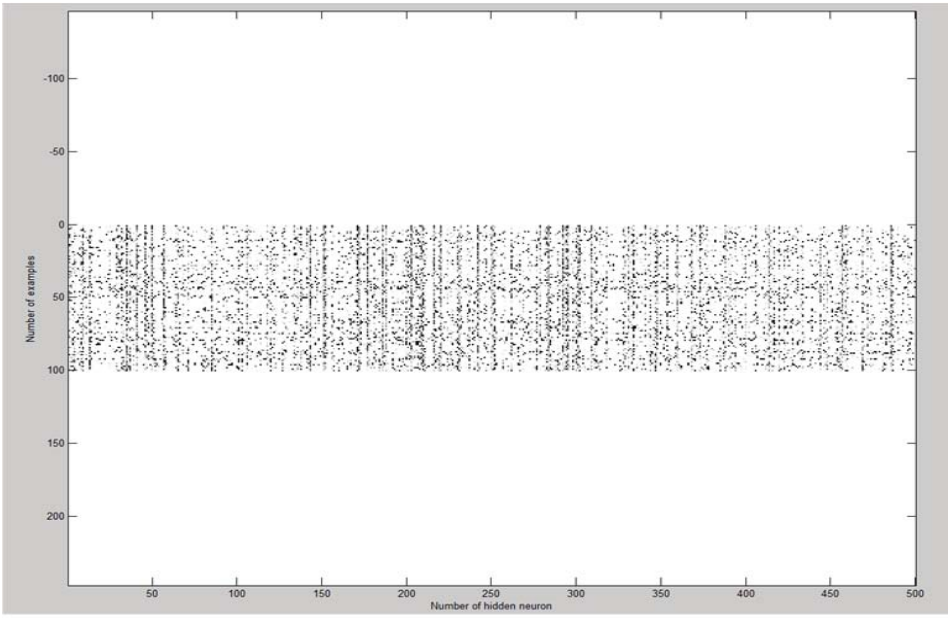


Figure 3.7 Greyscale image of hidden probability in GDRBM

3.3.6 Comparison of GDRBM with Other Gaussian RBMs

The Performance of GDRBM is now compared with three other RBMs: Hinton RBM, [11], GBRBM [13], and IGRBM [10] discussed in Section 2.4. They are trained using similar parameters with the CD-1 algorithm. Unless specified otherwise, all models have 500 hidden nodes, learning rate of 10^{-3} learning rates, $\sigma = 10^{-8}$ and momentum between 0.5 and 0.9.

Number of Epochs	Mean Square Error			
	Hinton RBM	GBRBM	IGBRBM	GDRBM
10	1.23E+07	1.23E+07	1.23E+07	1.20E+07
20	1.12E+07	1.12E+07	1.12E+07	1.02E+07
30	1.05E+07	1.05E+07	1.05E+07	9.40E+06
40	9.88E+06	9.87E+06	9.86E+06	8.88E+06
50	9.26E+06	9.25E+06	9.24E+06	8.53E+06

Table 3.2 Reconstruction Error Comparison

Table 3.2 shows the reconstruction error for each model with respect to the number training epochs. The performances of the difference GRBMs are very much the same.

		Number of Epochs				
	MSE	10	20	30	40	50
200 Hidden node	Hinton RBM	1.22E+07	1.15E+07	1.11E+07	1.07E+07	1.03E+07
	GBRBM	1.22E+07	1.15E+07	1.11E+07	1.07E+07	1.04E+07
	IGBRBM	1.22E+07	1.15E+07	1.11E+07	1.07E+07	1.04E+07
	GDRBM	1.13E+07	9.80E+06	9.07E+06	8.73E+06	8.54E+06
400 Hidden node	Hinton RBM	1.20E+07	1.10E+07	1.04E+07	9.74E+06	9.17E+06
	GBRBM	1.20E+07	1.10E+07	1.04E+07	9.70E+06	9.11E+06
	IGBRBM	1.20E+07	1.11E+07	1.04E+07	9.77E+06	9.20E+06
	GDRBM	1.15E+07	9.78E+06	8.92E+06	8.39E+06	8.04E+06
600 Hidden node	Hinton RBM	1.27E+07	1.15E+07	1.08E+07	1.01E+07	9.48E+06
	GBRBM	1.27E+07	1.15E+07	1.08E+07	1.02E+07	9.56E+06
	IGBRBM	1.27E+07	1.15E+07	1.09E+07	1.02E+07	9.53E+06
	GDRBM	1.26E+07	1.08E+07	1.00E+07	9.55E+06	9.22E+06
800 Hidden node	Hinton RBM	1.34E+07	1.21E+07	1.14E+07	1.07E+07	9.99E+06
	GBRBM	1.34E+07	1.21E+07	1.14E+07	1.07E+07	9.99E+06
	IGBRBM	1.31E+07	1.18E+07	1.11E+07	1.04E+07	9.71E+06
	GDRBM	1.37E+07	1.19E+07	1.12E+07	1.08E+07	1.05E+07
1000 Hidden node	Hinton RBM	1.39E+07	1.25E+07	1.18E+07	1.11E+07	1.04E+07
	GBRBM	1.39E+07	1.25E+07	1.18E+07	1.11E+07	1.04E+07
	IGBRBM	1.39E+07	1.25E+07	1.18E+07	1.12E+07	1.04E+07
	GDRBM	1.46E+07	1.29E+07	1.22E+07	1.17E+07	1.14E+07

Table 3.3 Variation in MSE with different number of hidden nodes

Then, Table 3.3 shows the effect of varying the number of hidden nodes on the various GRBMs. Again, the performances are very similar. It is interesting to note that for any number of hidden nodes, the performance of GDRBM trained for 10 epochs is

approximately the same as that or other GRBMs trained for 20 epochs of other GRBMS. Hence GDRBM learns faster than other GRBMs. Furthermore, for smaller number of hidden nodes (200 and 400), GDRBM can represent the same data with half the size of the hidden layer and less training iterations compared with other GRBMs. However, the standard CD-1 training algorithm does not work well with GDRBM with a larger hidden layer.

Histograms of biases and weights for different GRBMs are plotted in Figure 3.8. Though all three GRBMs are simulated in same environment and learn similarly, histograms of each model are distinct. The weights of GDRBM are distributed in the interval (-0.2, 0.2) while that of the other three GRBMs are distributed in the range of (-1,1).

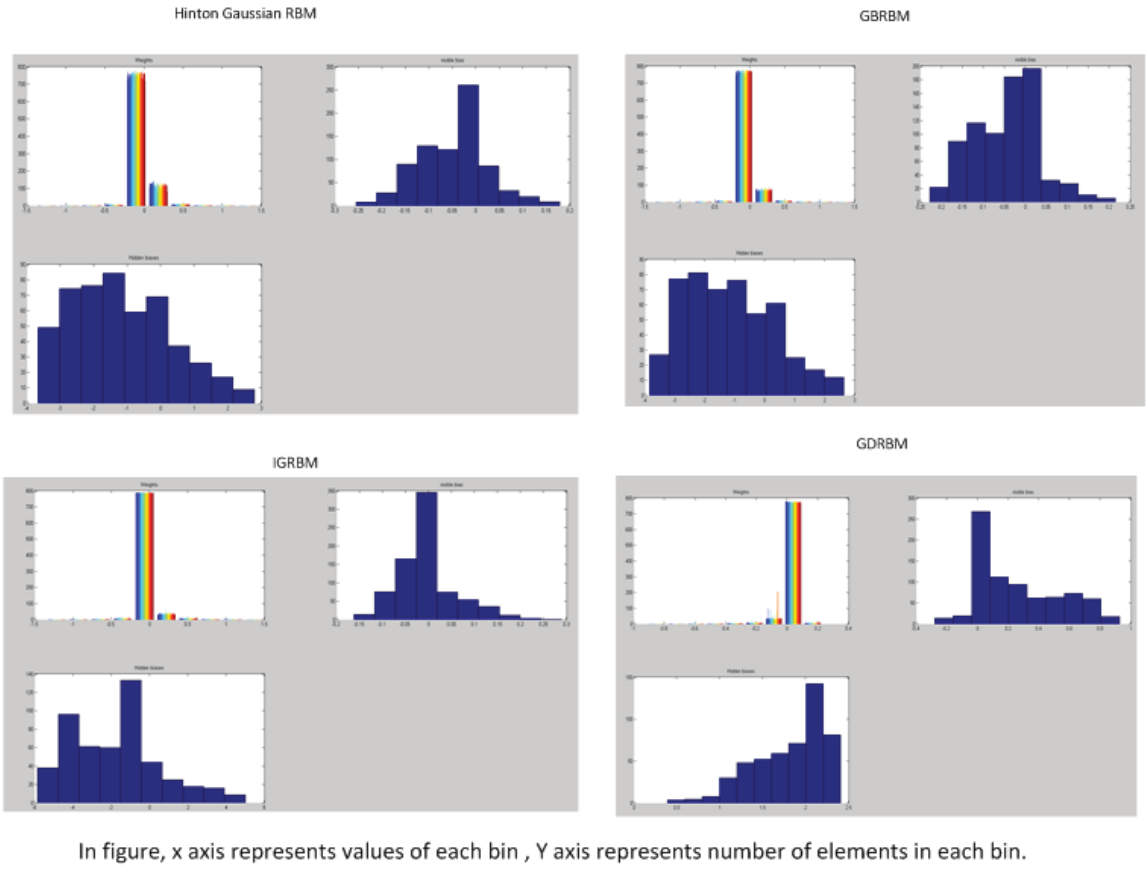


Figure 3.8 Histogram of GRBMs

Finally, the filters of various GRBMs are compared in Figure 3.9. GDRBM's filters are similar to Hinton's binary RBM. The other two GRBMs have hidden node features that are different.

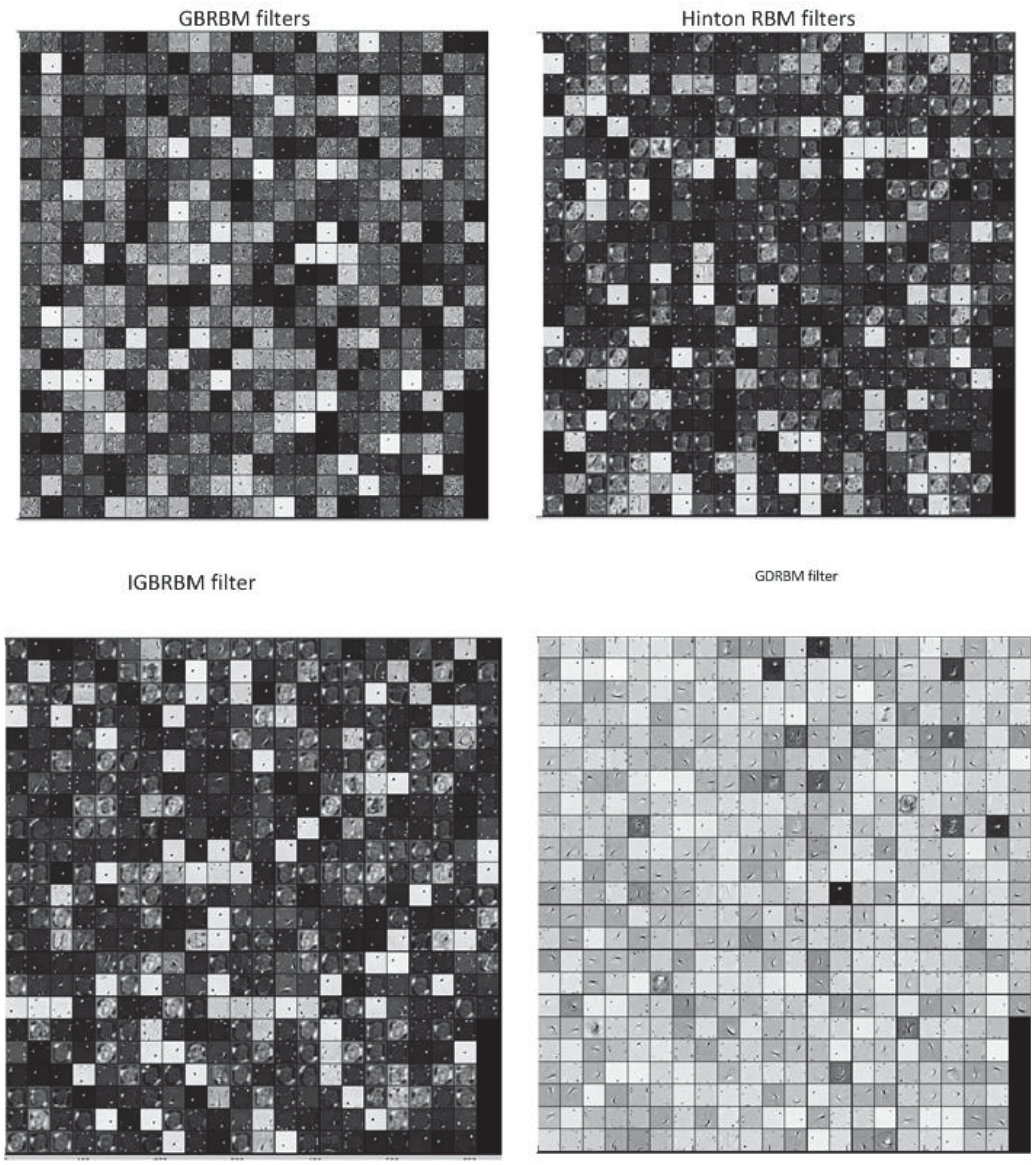


Figure 3.9 Filters of Gaussian RBMs

3.4 Performance of GDRBM with the CBCL dataset

The next sets of experiments are performed with the CBCL dataset [25]. It consists of a wide range of images including face images, street scene images, pedestrian images and car images. Here we used the face image dataset. This dataset consists of 2429 grey-scale images with a size of 19×19 pixels.

3.4.1 Experiments

The GDRBM is trained using the CD-1 algorithm as before. The CBCL training data are organized into 49 mini-batches, each consisting of 50 examples. Since the images are 19×19 pixels in size, the visible layer has 361 neurons. The size of the hidden layer is 256.

The network is trained for 50 epochs with default initial parameters as in the previous Section. The results were noisy with relatively large reconstruction error. So the amount of training is gradually increased to 1000 epochs. The average MSE is plotted against the number of training epochs in Figure 3. 10. This indicates that the amount of training GDRBM required for grey-scale images is much higher than that for binary images in the MNIST dataset.

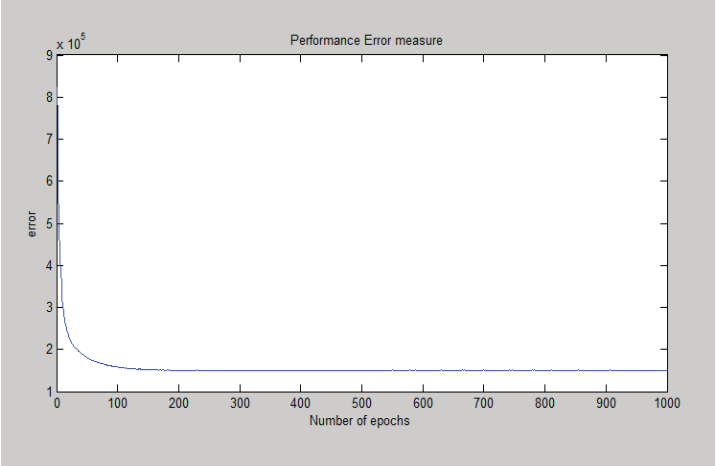
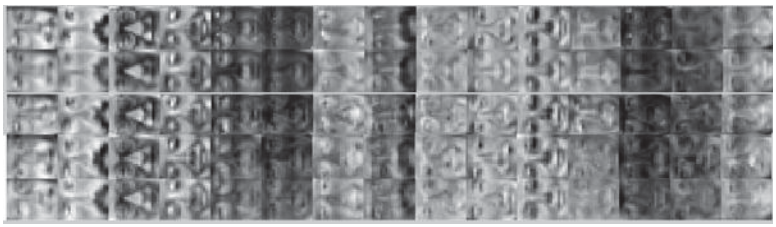


Figure 3.10 Mean Square Error for CBCL Datasets

Examples of the reconstructed images obtained after 50, 500, 1000 and 1500 training epochs are shown in Figure 3.11. After 50 epochs, while the reconstructed faces can be discerned, some sharp features such as glasses are not reconstructed well. These features are reconstructed better after 500 epochs. However compared to the binary handwritten images, these face images are more noisy.



In Figure, top row represents samples of input data in visible layer, second, third, fourth and fifth row represents reconstructed image at 50 epochs, 500 epochs, 1000 epochs and 1500 epochs.

Figure 3.11 Visualization of greyscale images

The histograms of the weights, visible bias and hidden bias are shown in Figure 3.12. The weights range from -0.1 to 0.1. Visible and hidden biases are in the small positive range.

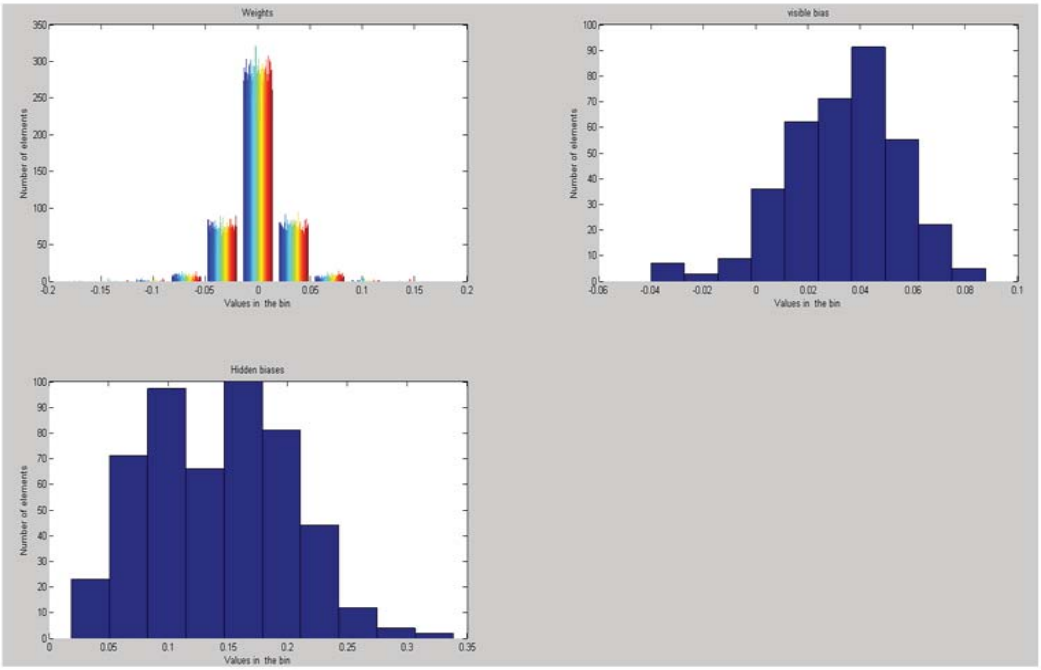


Figure shows, histograms of parameters taken at the end of 50 epochs of training. Each parameter histogram illustrates the number of elements in each bin. Different colors represent different groups of elements.

Figure 3.12 Histogram for CBCL Dataset

Finally, the learned filters of GDRBM are shown in Figure 3.13. Though the filters learn the features of the faces, they are very noisy. Unlike in learning the MNIST datasets, each hidden node learns the whole input (instead of part of the input) from the dataset.

Figure shows, 256 hidden nodes of learned GDRBM after 1000 epochs. In this, each rectangle represents one hidden neuron. Values in the hidden neuron represents the weights value between that hidden neuron with all the visible neuron.

Figure 3.13 Filters of GDRBM in CBCL Datasets

3.5 Summary

This chapter introduced a new RBM variant - GDRBM. The CD-1 training algorithm is adapted for this network. Its performance is analyzed in detail using the MNIST dataset and CBCL datasets and results are discussed. We also compared the performance of GDRBM with three other RBMs, namely, the Hinton RBM, GBRBM and IGRBM.

Results show that the performance of GDRBM is similar to other GRBMs. The advantage of GDRBM is that with multiple discrete hidden states, we can use fewer hidden nodes compared with a GRBM with binary hidden nodes. The amount of training in order to achieve similar performance is also reduced.

Chapter 4
Application of GDRBM to Deep Learning

A major recent application of RBM is in Deep Learning. In this chapter, GDRBM will be applied to a deep learning architecture known as the deep autoencoder.

4.1 Deep Learning ANNs

The ANNs discussed in previous chapters have only one hidden layer. Some neural network architectures make use of more hidden layers. Typically, ANNs with a maximum of three layers are known to have shallow architectures [57]. Shallow architectures require a large number of hidden nodes to represent highly varying complex input functions. Also, they require large sets of training examples to capture the input variations. These two requirements increase the computational complexity [70] of training as well as difficulty of inference from hidden units. This leads to alternative models known deep learning architectures [6],[58].

Deep architectures typically have three or more hidden layers. The idea of deep architecture is to automatically learn complex higher level features by combining low level features. Thus each level in the multilevel architecture provides different representation of the input data [57]. The number of hidden nodes per layer is reduced by having multiple hidden layers. Hence deep networks are computationally more efficient than shallow architectures [59].

One of the main problems with deep architectures is that they are difficult to train. Hinton et.al [4], [60] introduced an effective greedy learning algorithm that combined unsupervised and supervised training. Since then, research in deep architectures has increased. They have now been applied to a number of difficult real-world tasks including pattern recognition, prediction, feature extraction and classification [71]. The most popular feed forward deep architectures are Deep Belief Network [4], Deep Autoencoder [11] and Deep Boltzmann Machines [5].

4.1.1 Deep Belief Networks

Deep Belief Network (DBN) [4] is a multi-level architecture as shown in Figure 4.1. The top two layers are RBMs and the remaining layers are Sigmoid Belief Nets. It has been shown that both kinds of networks learn in a similar manner during the training phase. This structural advantage leads to new greedy way of training deep network that achieves fast and efficient inference.

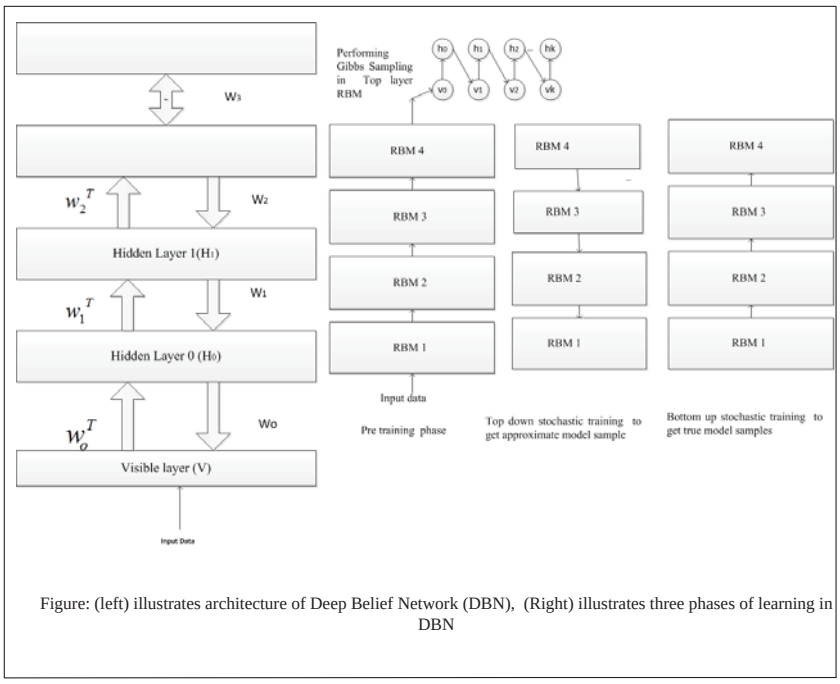


Figure: (left) illustrates architecture of Deep Belief Network (DBN), (Right) illustrates three phases of learning in DBN

Figure 4.1 Deep Belief Network and its training methodology

This algorithm trains the network in two steps. First, each layer is individually pre-trained using maximum likelihood approximation methods such as the CD-*k* discussed earlier. After that, fine-tuning is performed by top-down supervised training to improve the generalization of the network. The training procedure is illustrated in Figure 4.1, with the algorithm given in Algorithm 4.1

.

Greedy Training algorithm

1. Unsupervised Pre training Phase : Train one layer at a time
   1. v_i = input dataset
   2. Find w_1 , take samples of hidden layer 1 and give as input to layer 2.
   3. Keep w_1 constant and Initialize $w_2 = w_1T$
   4. Find w2, Take samples from hidden layer 2 and give as input to layer 3.
   5. Keep w_1 and w_2 constant and initialize $w_3 = w_2T$.
   6. Find w_3, Take samples from hidden layer 2 and give as input to layer 3.
   7. Repeat this for all layers of Deep network
2. Fine tuning phase:
   1. From top two layers, perform alternative Gibbs sampling and obtain a sample .
   2. Pass the sample down to the previous layer and find corresponding probability in the layer. This gives, approximate sample of the model.
   3. Repeat this step until reach visible layer.
   4. Then true samples are obtained by passing the samples again from bottom to top layer.

Note: Gibbs Sampling and calculation of w_i (weights in each layer) is discussed in Chapter 2, section 2.2.1

Algorithm 4.1 Greedy algorithm for DBN

This greedy algorithm has two main advantages. Firstly, the top RBM provides initializing parameters for the lower layers through greedy training rather than using random initialization parameters. Secondly, layer-wise unsupervised pre-training provides a hypothesis; it could initialize the parameters near to energy minimum and leads to generalized representations [6], [61]. In addition, this reduces the complexity and improves the speed of training. DBN and its variants are outperforming other deep architectures in many real time applications such as image recognition, classification and speech recognition [4, 6, 50].

4.1.2 Deep Autoencoder

In this multilayer neural network, the number of neurons in a hidden layer is always smaller than its previous layer. This property leads to an accurate reconstruction of the input. This structure is similar to encoding and decoding of digital circuits. Hence it is

known as the autoencoder [62], [63]. Autoencoders with multiple hidden layers is known as Deep Autoencoder [11]. The difference between a typical autoencoder and a deep autoencoder is shown in Figure 4.2. A deep autoencoder is constructed by stacking conventional auto encoders [6] or stacking RBMs [11] one upon another. In the following, we assume that RBMs are used.

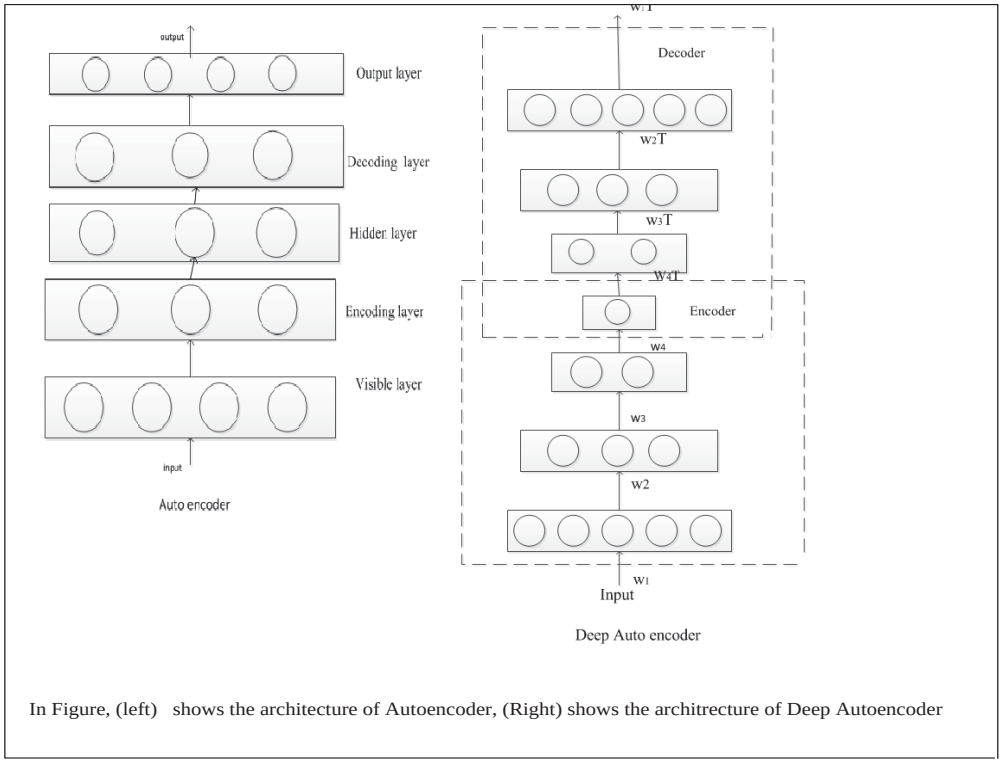


Figure 4.2 Architecture of Autoencoder and Deep autoencoder

Training of the deep autoencoder follows a greedy method similar to DBN with a few modifications. Training is organized into three phases as illustrated in Figure 4.3. The first phase involves pre-training of each RBM using a CD algorithm. Then in next phase, pretrained hidden layers are disclosed as encoder and decoder as shown in the figure. Finally, the trained model is fine-tuned by back propagation algorithm [64], [65], [66] to minimize the cross entropy error between the input and the output to obtain the deterministic states of the neurons.

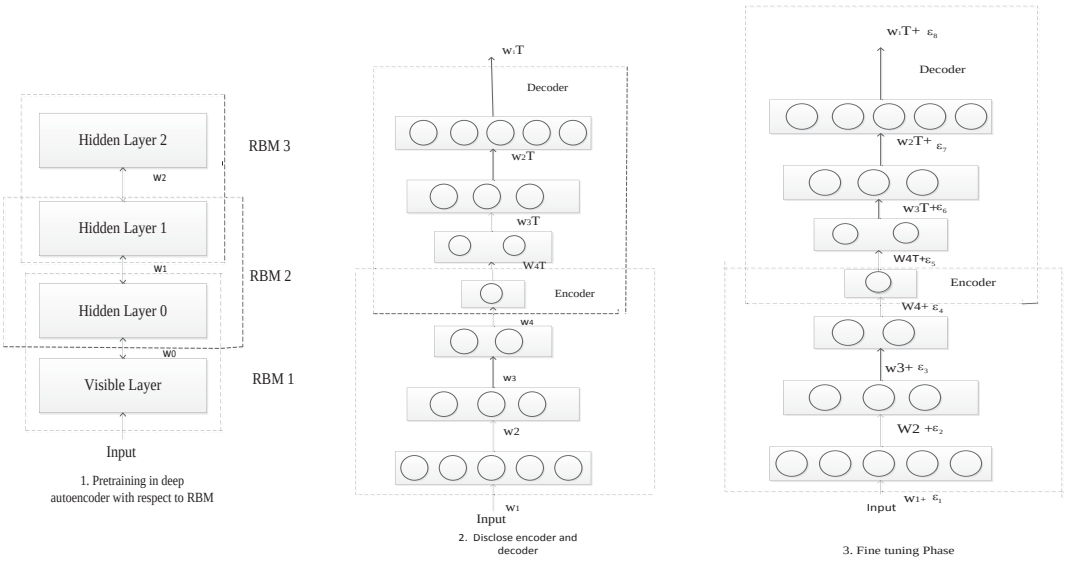


Figure 4.3 The three training phases of a Deep autoencoder

4.2 GDRBM for Deep Autoencoder

Now we apply the GDRBM developed in Chapter 3 to the deep autoencoder architecture. As shown in Chapter 3, the GDRBM is able to reconstruct the inputs better than other Gaussian RBMs. So we hypothesize that the performance of a deep GDRBM autoencoder is able to reconstruct data better than other RBM based autoencoder.

4.2.1 Architecture and Training Details

A four layer deep autoencoder is constructed with a visible layer of 28×28 neurons and four hidden layers of sizes 500, 250, 120 and 30 neurons respectively. The architecture is shown in Figure 4.4. The lowest layer where the inputs are connected to is the Gaussian visible layer of a GDRBM. However, the second layer cannot be another GDRBM as the output of the hidden layer of the first GDRBM is discrete. Hence an RBM with discrete input and binary hidden layers, referred to as the discrete binary RBM, is used. This is followed by a binary RBM. Finally the output of the network is provided by the IBGRBM which is an IGBRBM with the visible and hidden layers reversed.

Simulations are performed using the MNIST datasets. In order to provide batch gradient analysis, training sets are split into 600 batches each containing 100 cases. During fine tuning, the 600 mini-batches are combined to form 60 large batches.

The binary RBMs are trained using a learning rate of 0.1, and momentum in the range of 0.5-0.9. The GDRBM is trained using a learning rate of 10^{-3}, and a sigma learning rate of 10^{-8}. In pre-training, all RBMs are trained for 20 epochs using the CD-1 algorithm. Fine tuning is performed for 200 epochs similar to traditional deep autoencoder [67].

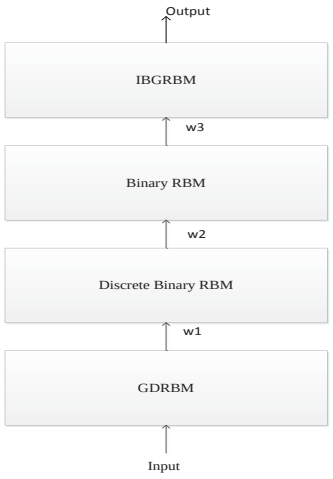


Figure illustrates, Deep Neural Network designed with different variants of RBM. GDRBM: Gaussian visible, Discrete hidden neuron RBM, DBRBM: Discrete visible and Binary hidden RBM, Binary visible and Binary hidden RBM, IBGRBM: Bernoulli visible and Gaussian RBM.

Figure4.4 Architecture of GDRBM Deep Autoencoder

4.2.2 Results

The reconstruction results of the GDRBM-based and the binary deep autoencoders are shown in Figure 4.5.

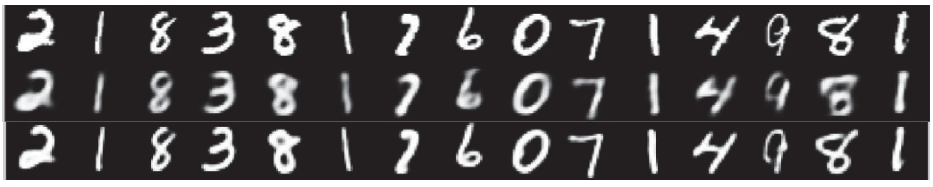


Figure visualizes samples from MNIST Datasets in top row, second row shows reconstructed samples from GDRBM Deep autoencoder , third row shows reconstructed samples from Binary Deep autoencoder.

Figure 4.5 Visualization of Reconstruction in Deep autoencoder

While the proposed GDRBM deep architecture is able to reconstruct the inputs reasonably well, the amount of reconstruction error remains quite high. A comparison of the change in average MSE over the 200 epochs of fine-tuning is plotted in Figure 4.6. It can be observed that the initial error of the GDRBM deep autoencoder is much larger than the traditional deep autoencoder. This initial error is reduced significantly after the first few epochs of fine-tuning.

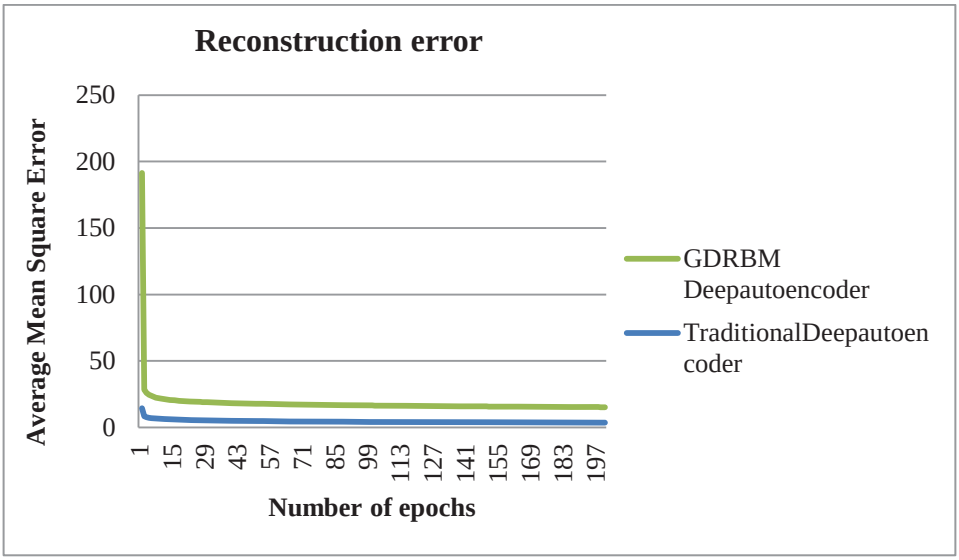


Figure 4. 6 Performance of Deep autoencoders

It is well known that RBMs are very sensitive to the training parameters such as learning rates and momentum. The large error exhibited after the pre-training of the GDRBM deep autoencoder shows that this training phase has not been conducted properly. A better selection of training parameters may help reduce the initial error. Furthermore, the second layer consists of an RBM with discrete visible states. It has been shown that increased visible states could lead to increased error.

Chapter 5
Conclusions and Future Works

In this thesis, a new RBM known as the GDRBM has been developed. It has a Gaussian visible layer and a non-binary discrete output layer. The conditional probabilities of the hidden layer as well as the CD-1 training algorithm are modified for this particular network. Performance results using the MNIST and CBCL benchmark datasets show that the performance of a GDRBM with 4-state hidden neurons is approximately the same as that of other Gaussian RBMs with binary hidden neurons when the size of the hidden layer is doubled. Thus the hypothesis that increasing the number of hidden states is able to increase the representation power of a GRBM with the same number of hidden neurons has been shown to be true. Hence GDRBM has the advantage that it provides a Gaussian layer to represent continuous valued inputs and a discrete layer that is able to store internal representations more compactly than binary ones. The later advantage implies that the number of neurons required in the hidden layer can be smaller.

The GDRBM is also used as a layer for a deep autoencoder for the first time. It is also shown that this autoencoder is able to reconstruct the inputs reasonably well. However, the reconstruction error is higher than traditional deepautoencoders. Further research will be needed to investigate what is the best way to use GDRBM in a deep autoencoder and the best training parameters to use.

Other future works include developing methods to determine the right training parameters for each application. This problem arises for any ANN. A possibility is to develop adaptive methods for GDRBM similar to [10].

As discussed in section 3.4.6, GDRBM has some neurons which do not learn any useful features from the input dataset. This can be reduced by introducing sparsity in the hidden layer of GDRBM. The human brain introduces sparsity through an “attention mechanism” to select useful information from the environment [68],[69]. Adopting attention mechanisms in GDRBM could optimize the resulting network.

Lastly, further research will be needed to determine if GDRBM could be used for deep learning architectures. Particularly, the filters of this model must be analyzed to find interesting features learnt by this variant of deepautoencoder [72].

Appendix-1

Mathematical Calculations in IGBRBM

For IGBRBM, Energy is defined as,

$$E(v,h)=\sum_{i=1}^{n_v}\frac{(v_i-a_i)^2}{2\sigma_i^{\,2}}-\sum_{i=1}^{n_v}\sum_{j=1}^{n_h}w_{ij}h_j\frac{v_i}{\sigma_i^{\,2}}-\sum_{j=1}^{n_h}b_jh_j$$

To find conditional distribution of visible given hidden, it must be derived from below steps,

From definition of conditional distribution we know,

$$p(v\,|\,h)=\frac{p(v,h)}{p(h)}$$

By substituting numerator and Denominator from Boltzmann distribution $p(v,h)=\frac{1}{z}e^{-E(v,h)}$, we get

$$p(v\,|\,h)=\frac{e^{-E(v,h)}}{\int_v e^{-E(v,h)}dv}$$

$$=\frac{e^{-\sum_i\left(\frac{(v_i-a_i)^2}{2\sigma_i^2}-\sum_j\frac{w_{ij}v_ih_j}{\sigma_i^2}\right)+\sum_{j=1}^{n_h}b_jh_j}}{\int_v e^{-\sum_i\left(\frac{(v_i-a_i)^2}{2\sigma_i^2}-\sum_j\frac{w_{ij}v_ih_j}{\sigma_i^2}\right)+\sum_{j=1}^{n_h}b_jh_j}dv}$$

$$= \frac{\prod_i e^{-\left(\frac{(v_i-a_i)^2}{2\sigma_i^2}-\sum_j \frac{w_{ij}v_i h_j}{\sigma_i^2}\right)+\sum_{j=1}^{n_h} b_j h_j}}{\int_v e^{-\sum_i\left(\frac{(v_i-a_i)^2}{2\sigma_i^2}-\sum_j \frac{w_{ij}v_i h_j}{\sigma_i^2}\right)+\sum_{j=1}^{n_h} b_j h_j} dv}$$

Rewrite the denominator with respect to product of expert model similar to numerator we get,

$$= \frac{\prod_i e^{-\left(\frac{(v_i-a_i)^2}{2\sigma_i^2}-\sum_j \frac{w_{ij}v_i h_j}{\sigma_i^2}\right)+\sum_{j=1}^{n_h} b_j h_j}}{\prod_i \int_v e^{-\left(\frac{(v_i-a_i)^2}{2\sigma_i^2}-\sum_j \frac{w_{ij}v_i h_j}{\sigma_i^2}\right)+\sum_{j=1}^{n_h} b_j h_j} dv}$$

Denominator is simplified as,

$$\prod_i \int_v e^{-\left(\frac{(v_i-a_i)^2}{2\sigma_i^2}-\sum_j \frac{w_{ij}v_i h_j}{\sigma_i^2}\right)+\sum_{j=1}^{n_h} b_j h_j} dv = \prod_i \int_v e^{-\left(\frac{1}{2\sigma_i^2}({v_i}^2-2v_i a_i+{a_i}^2)-\sum_j \frac{w_{ij}v_i h_j}{\sigma_i^2}\right)+\sum_{j=1}^{n_h} b_j h_j} dv$$

$$= \prod_i e^{\left(\frac{-{a_i}^2}{2\sigma_i^2}\right)+\sum_{j=1}^{n_h} b_j h_j} \int_v e^{\frac{1}{2\sigma_i^2}\left(-{v_i}^2+2v_i a_i\right)+\sum_j \frac{w_{ij}v_i h_j}{\sigma_i^2}} dv$$

$$= \prod_i e^{\left(\frac{-{a_i}^2}{2\sigma_i^2}\right)+\sum_{j=1}^{n_h} b_j h_j} \int_v e^{\frac{1}{2\sigma_i^2}\left(-{v_i}^2\right)} e^{v_i\left(\frac{a_i}{\sigma_i^2}+\sum_j \frac{w_{ij}h_j}{\sigma_i^2}\right)^2} dv$$

Integrating the above equation with respect to v, we get,

$$= \prod_i e^{\left(\frac{-{a_i}^2}{2\sigma_i^2}\right)+\sum_{j=1}^{n_h} b_j h_j} e^{\frac{\sigma_i^2\left(\frac{a_i}{\sigma_i^2}+\sum_j \frac{w_{ij}h_j}{\sigma_i^2}\right)^2}{2}} \left(\sqrt{2\sigma_i^2\pi}\right)$$

$$= (\sigma_i \sqrt{2\pi}) \prod_i e^{\frac{1}{2}\left(\sum_{j=1}^{n_h} w_{ij} h_j\right)^2 + \sum_j b_j h_j + \frac{a_i w_{ij} h_j}{\sigma_i^2}}$$

Dividing numerator and denominator,

$$p(v \mid h) = \frac{\prod_i e^{-\left(\frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_j \frac{w_{ij} v_i h_j}{\sigma_i^2}\right) + \sum_{j=1}^{n_h} b_j h_j}}{(\sigma_i \sqrt{2\pi}) \prod_i e^{\frac{1}{2}\left(\sum_{j=1}^{n_h} w_{ij} h_j\right)^2 + \sum_j b_j h_j + \frac{a_i w_{ij} h_j}{\sigma_i^2}}}$$

$$= \prod_i \frac{1}{(\sigma_i \sqrt{2\pi})} e^{-\left(\frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_j \frac{w_{ij} v_i h_j}{\sigma_i^2}\right) + \sum_{j=1}^{n_h} b_j h_j - \left(\frac{1}{2}\left(\sum_{j=1}^{n_h} w_{ij} h_j\right)^2 + \sum_j \left(b_j h_j + \frac{a_i w_{ij} h_j}{\sigma_i^2}\right)\right)}$$

Simplifying the above equation, p (v, h) is,

$$p(v \mid h) = \prod_i \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{1}{2\sigma_i^2}\left(v_i - \left(a_i + \sum_{j=1}^{n_h} w_{ij} h_j\right)\right)^2}$$

The above equation is a density function of Gaussian distribution with mean $v_i - \left(a_i + \sum_{j=1}^{n_h} w_{ij} h_j\right)$ and variance σ_i^2.

Conditional distribution of hidden given visible is derived as follows,

$$p(h \mid v) = \frac{p(v,h)}{p(v)}$$

By substituting Boltzmann Distribution, we get

$$p(h \mid v) = \frac{e^{-E(v,h)}}{\sum_h e^{-E(v,h)}}$$

$$= \frac{e^{-\sum_i \left(\frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_j \frac{w_{ij} v_i h_j}{\sigma_i^2} \right) + \sum_{j=1}^{n_h} b_j h_j}}{\sum_h e^{-\sum_i \left(\frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_j \frac{w_{ij} v_i h_j}{\sigma_i^2} \right) + \sum_{j=1}^{n_h} b_j h_j}}$$

Rewriting the above equation in terms of product of expert model, we get

$$= \frac{\prod_j e^{-\sum_i \left(\frac{(v_i - a_i)^2}{2\sigma_i^2} \right)} e^{\frac{w_{ij} v_i h_j}{\sigma_i^2} + b_j h_j}}{\prod_j \sum_h e^{-\sum_i \left(\frac{(v_i - a_i)^2}{2\sigma_i^2} \right)} e^{\left(\frac{w_{ij} v_i h_j}{\sigma_i^2} + b_j h_j \right)}}$$

$$p(h \mid v) = \prod_j \frac{e^{\frac{w_{ij} v_i h_j}{\sigma_i^2} + b_j h_j}}{\sum_{h=h_j} e^{\left(\frac{w_{ij} v_i}{\sigma_i^2} + b_j \right) h_j}}$$

$$p(h \mid v) = \prod_j p(h_j \mid v)$$

For binary neuron, h_j=0, 1. Hence p (h_j=1) is calculated as follow,

$$p(h_j = 1) = \frac{e^{\frac{w_{ij} v_i}{\sigma_i^2} + b_j}}{e^{\left(\frac{w_{ij} v_i}{\sigma_i^2} + b_j \right) * 1} + e^0}$$

To find gradient update rules, differentiate energy with respect to parameters (w_{ij}, b_i, c_j).

$$\frac{\partial E(v,h)}{\partial w_{ij}} = \frac{v_i h_j}{\sigma_i^2}$$

$$\frac{\partial E(v,h)}{\partial a_i} = \frac{v_i}{\sigma_i^2}$$

$$\frac{\partial E(v,h)}{\partial b_j} = h_j$$

$$\frac{\partial E(v,h)}{\partial \sigma_j} = \frac{(v_i - a_i)^2}{2}\left(\frac{-2}{\sigma_i^3}\right) + w_{ij} h_j v_i \left(\frac{-2}{\sigma_i^3}\right)$$

By simplifying above equation, gradient of sigma is obtained as,

$$\frac{\partial E(v,h)}{\partial \sigma_j} = \left(\frac{(v_i - a_i)^2}{\sigma_i^3}\right) + w_{ij} h_j v_i \left(\frac{-2}{\sigma_i^3}\right)$$

By applying this gradient in positive and negative phase of below log likelihood gradient equation,

$$\frac{1}{l}\sum_{v \in D} \frac{\partial \ln L(\theta \mid v)}{\partial w_{ij}} = -\frac{1}{l}\sum_{v \not\in D}\left[E_{p(h|v)}\left(\frac{\partial E(h,v)}{\partial w_{ij}}\right) + E_{p(v,h)}\left(\frac{\partial E(h,v)}{\partial w_{ij}}\right)\right]$$

We obtain the update rule as follows,

$$\Delta\sigma_i = \varepsilon_\sigma\left(E_{\text{data}}\left(\frac{(\mathrm{v_i} - a_i)^2}{\sigma_i^3} - \sum_j h_j . \frac{w_{ij} v_i}{\sigma_i^2}\right) - E_{\text{model}}\left(\frac{(\mathrm{v_i} - a_i)^2}{\sigma_i^3} - \sum_j h_j . \frac{w_{ij} v_i}{\sigma_i^2}\right)\right)$$

$$\Delta w_{ij} = \frac{\left(E_{data}\left(v_i h_j\right) - E_{\text{mod}el}\left(v_i h_j\right)\right)}{\sigma_i}$$

$$\Delta a_i = \left(\frac{E_{\text{data}}(\mathrm{a_i}) - E_{\text{model}}(\mathrm{a_i})}{\sigma_i^2}\right)$$

$$\Delta b_j = \left(E_{\text{data}}(\mathrm{b_j}) - E_{\text{model}}(\mathrm{b_j})\right)$$

Bibliography

[1] G.E.Hinton, "Training Products Of Experts by Minimizing Contrastive Divergence," *Neural computation,* vol. 14, no.8, pp. 1771-1800, Aug.2002.

[2] G.E.Hinton, D.H.Ackley and T.J.Sejnowski, "Boltzmann Machines:Constraint Satisfaction Networks that Learn," CMU and JHU, PA and Baltimore, Rep.no. CMU-CS-84-119,May.1984.[online]. Available:http://www.csri.utoronto.ca/~hinton/absps/bmtr.pdf

[3] D.H. Ackley and G.E.Hinton, "A Learning Algorithm for Boltzmann Machines*," *Cognitive science,* vol. 9, no.1, pp. 147-169, Jan.1985.

[4] G. E. Hinton, S. Osindero and Y.W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural computation,* vol. 18, no.7, pp. 1527-1554, Jul.2006.

[5] R. Salakhutdinov and G. E. Hinton, "Deep Boltzmann Machines," in *Proc.12th Int.Conf.AISTATS.*, vol.5. JMLR: W&CP 5, Florida, USA, Apr.2009, pp. 448-455.

[6] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy Layer-wise Training of Deep Networks," in *Proc.Annu.NIPS.Conf.,* vol.19. Advances in Neural Information Processing Systems, Vancouver, Dec.2006, pp. 153-160.

[7] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans.PatternAnal. Mach. Intell,* vol. 35, no.8, pp. 1798 – 1828, Aug.2013.

[8] A.R.Mohamed and G.E.Hinton, "Phone Recognition Using Restricted Boltzmann Machines," in *Proc. IEEE Int. Conf. ASSP*, Dallas, TX , Mar.2010, pp. 4354-4357.

[9] G. Hinton, "A Practical Guide to Training Restricted Boltzmann Machines," *Momentum*, vol.9, no.1, pp.926, Aug.2010.

[10] K.H.Cho, A. llin, and T. Raiko, "Improved Learning of Gaussian-Bernoulli Restricted Boltzmann Machines," in *Proc.ICANN.,* Springer Berlin Heidelberg, Finland, Jun.2011, pp. 10-17.

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

[12] H. Chen and A. F. Murray, "Continuous Restricted Boltzmann Machine with an implementable training algorithm," in *Vision, Image and Signal Processing, IEE Proceedings,* vol. 150, no.3, pp. 153-158, Jun.2003.

[13] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," MSc Thesis, *C.Sc.Dept., Univ of Toronto.*, Rep.no 1., vol.4, Apr. 2009. [online].Available:http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220&rep=rep1&type=pdf.

[14] N. Wang, J. Melchior, and W. Laurenz, "An Analysis of Gaussian-Binary Restricted Boltzmann Machines for natural images," in *Proc.ESANN*, Burgues, Belgium, Apr.2012, pp. 287-292.

[15] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural networks*, vol. 2, pp. 53-58, Aug.1988.

[16] H. Murao and S. Kitamura, "Incremental State Acquisition for Q-learning by Adaptive Gaussian soft-max neural network," in *Proc.IEEE.Int.Symp.*,CIRA,ISAS.,ISIC, Gaithersburg, MD., Sep.1998, pp. 465-470.

[17] K. Sohn, D. Y. Jung, H. Lee, and A. O. Hero, "Efficient Learning of Sparse, Distributed, Convolutional Feature Representations for Object Recognition," in *Proc.IEEE.ICCV*, Barcelona, Nov.2011, pp. 2643-2650.

[18] R. Salakhutdinov and G. E. Hinton, "Replicated Softmax: an Undirected Topic Model," in *Proc.Annu.conf.NIPS*, vol.22, Advances in Neural Information Processing Systems 2009, pp. 1607-1614.

[19] G. Montufar and J. Morton, "Discrete Restricted Boltzmann Machines," *arXiv preprintarXiv:1301.3529,*Apr.2014.[online].Available:http://arxiv.org/abs/1301.3529.

[20] I. J. Myung, "Tutorial on Maximum Likelihood Estimation," *Journal of mathematical Psychology,*vol.47,no.1,pp. 90-100, Mar.2003.

[21] S. V. N. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. P. Murphy, "Accelerated Training of Conditional Random Fields with Stochastic Gradient Methods," *Proc.23rd Int. conf. Machine learning*. ACM, 2006, pp. 969-976.

[22] T. Tieleman, "Training Restricted Boltzmann Machines using approximations to the likelihood gradient," in *Proc.25th Int. conf. Machine learning*. ACM, pp. 1064-1071, 2008.

[23] T. Tieleman and G. Hinton, "Using fast weights to improve persistent contrastive divergence," in *Proc 26th Annu. Int. Conf. Machine Learning*, Montreal, CA, pp. 1033-1040, 2009.

[24] C. C. Yann LeCun, Christopher J.C. Burges. *THE MNIST DATABASE of handwritten digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[25] MITCenterforBiologicalandComputationalLearning. *CBCL FACE DATABASE #1* [Online]. Available: http://cbcl.mit.edu/software-datasets/FaceData2.html.

[26] S.Haykin, "Stochastic Methods Rooted in Statistical Mechanics" in *Neural networks and Learning machines*, 3rd ed. Newyork:Prentice Hall, 2009, ch.11, sec.11.7, pp 598- 604.

[27] K.H.Cho, "Improved Learning Algorithms for Restricted Boltzmann Machines," Master's.thesis, CSE, Aalto Univ, FL, 2011.

[28] T. J. Sejnowski, "Higher-order Boltzmann machines," in *Proc.Conf.Neural networks for computing., AIP*, Snowbird, Utah, 1986, vol.151, no.1, pp. 398-403,

[29] G. E. Hinton, "Products of experts," in *Proc.* 9th *ICANN*, Edinburgh, 1999, vol. no. 1, pp. 1-6.

[30] G. E. Hinton and T. J. Sejnowski, "Optimal perceptual inference," in *Proc.IEEE. conf. Computer Vision and Pattern Recognition*, IEEE Newyork, 1983, pp. 448-453.

[31] S. Borman, "The expectation maximization algorithm-a short tutorial," Oct 2004.[online].Available:http://ftp.csd.uwo.ca/faculty/olga/Courses/Fall2006/Papers/EM_algorithm.pdf .

[32] L. K. Saul and M. I. Jordan, "Boltzmann Chains and Hidden Markov Models," in *Advances in neural information processing systems, U.S.A:The MIT press*, 1995, part.5, pp. 435-442,

[33] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted Boltzmann machines for collaborative filtering," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 791-798.

[34] E. K. Berndt, B. H. Hall, and R. E. Hall, "Estimation and Inference in Nonlinear Structural Models," in *Annals of Economic and Social Measurement.*, NBER, 1974, *Vol. 3, no. 4*. pp. 103-116.

[35] A.R. Mohamed, G. E. Dahl, and G.E.Hinton, "Acoustic Modeling Using Deep Belief Networks," *IEEE Trans. Audio, Speech,Language Process*,vol.20, no.1, pp. 14-22, Jan.2012.

[36] G. Desjardins, et.al, "Tempered Markov chain Monte Carlo for Training of Restricted Boltzmann Machines," in *Proc.*13th *Int. Conf. AISTATS*, Sardinia, Italy, vol.9. JMLR:W&CP, 2010, pp. 145-152.

[37] J. Besag, "Efficiency of Pseudolikelihood Estimation for Simple Gaussian Fields," *Biometrika,* vol. 64, no.3, pp. 616-618, Dec.1977.

[38] B. M. Marlin, K. Swersky, B. Chen, and N. D. Freitas, "Inductive Principles for Restricted Boltzmann Machine Learning," in *Int.Conf.AISTATS*, 2010, Sardinia, Italy, vol.9. JMLR:W&CP, pp.509-516.

[39] A. Fischer and C. Igel, "An Introduction to Restricted Boltzmann machines," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, Applications*, vol.7441, pp.14-36. Springer Berlin Heidelberg, 2012.

[40] M. A. Carreira-Perpinan and G. E. Hinton, "On Contrastive Divergence Learning," in *Proc.10th Int. workshop. AISTATS*, Barbados, 2005, pp. 33-40.

[41] Y. Bengio and O. Delalleau, "Justifying and Generalizing Contrastive Divergence," *Neural Computation*, vol. 21, no.6, pp. 1601-1621, Jun.2009.

[42] B. Walsh, "Markov Chain Monte Carlo and Gibbs Sampling," MIT Lecture Notes,Apr.2004.[online].Available:http://web.mit.edu/~wingated/www/introductions/mcmc-gibbs-intro.pdf.

[43] A.R. Mohamed, T. N. Sainath, G. Dahl, B. Ramabhadran, G. E. Hinton, and M. A. Picheny, "Deep belief networks using discriminative features for phone recognition," in *IEEE Int. Conf. Acoustics, Speech and Signal Processing*, Prague, 2011, pp. 5060-5063.

[44] L. Prechelt, "Proben1: A set of neural network benchmark problems and benchmarking rules," Univ. Karlsruhe, Germany, Tech.Rep.21/9, Sep.1994.

[45] Y. W. Teh and G. E. Hinton, "Rate-coded Restricted Boltzmann Machines for Face Recognition," in *Proc.Annu.conf.NIPS*, vol.13, Advances in Neural Information Processing Systems, 2000, pp. 908-914.

[46] G. Bouchard, "Efficient Bounds for the Softmax Function, Applications to Inference in Hybrid Models," in *NIPS*, British Coloumbia, CA, Dec.2007. [online]. Available: http://eprints.pascal-network.org/archive/00003498/.

[47] M. Welling, M. Rosen-Zvi, and G. E. Hinton, "Exponential Family Harmoniums with an Application to Information Retrieval," in *Proc.Annu.conf.NIPS*, vol.17. Advances in neural information processing systems, 2004, pp. 1481-1488.

[48] G. E. Dahl, R. P. Adams, and H. Larochelle, "Training Restricted Boltzmann Machines On Word Observations," in *Proc. 29th Int. conf. Machine Learning*, Scotland, UK, 2012. [online]. Available: *arXiv preprint arXiv:1202.5695.*

[49] N. Srivastava and R. Salakhutdinov, "Multimodal Learning with Deep Boltzmann Machines," in *Proc.Annu.conf.NIPS,* vol.25. Advances in Neural Information Processing Systems, 2012, pp. 2222-2230.

[50] R. Salakhutdinov, "Learning Deep Generative Models," Ph.D.thesis, Dept.C.S., Univ .Toronto, 2009.

[51] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain, "Neural Probabilistic Language Models," in *Innovations in Machine Learning*, vol.194. Springer Berlin Heidelberg, 2006, pp. 137-186.

[52] C. Ekanadham, "Sparse Deep Belief Net Model for Visual area V2," U.G.thesis, Symbolic systems program, Standford University, Jun.2007.

[53] H. Lee, C. Ekanadham, and Ng.Andrew, "Sparse Deep Belief Net Models for Visual Area V2," *Proc.Annu.conf.NIPS*, vol.21. Advances in Neural Information Processing Systems, 2008, pp. 873-880.

[54] N. Le Roux, N. Heess, J. Shotton, and J. Winn, "Learning a Generative Model of Images by Factoring Appearance and Shape," *Neural Computation,* vol. 23, no.3, pp. 593-650, Mar.2011.

[55] T. D. Nguyen, T. Tran, D. Phung, and S. Venkatesh, "Learning Parts-based Representations with Nonnegative Restricted Boltzmann Machine," in *Proc. Workshop and conf. ACML*, 2013, vol.29 . JMLR, pp. 133-148.

[56] J. Yosinski and H. Lipson, "Visually Debugging Restricted Boltzmann Machine Training with a 3D Example," in *29th Int. Conf.Machine Learning: Workshop.Representation Learning*, Scotland, UK, 2012.

[57] Y. Bengio, "Learning Deep Architectures for AI," *Foundation and Trends in Machine. Learing*, vol. 2, no.1, pp. 1-127, 2009.

[58] P. Cisek, T. Drew, and J. Kalaska, "On the challenge of learning Complex functions," in *Progress in Brain Research :Computational Neuroscience: Theoretical Insights into Brain Function*, 1st ed, Oxford, UK: Elsevier, 2007, p.521.

[59] N. Le Roux and Y. Bengio, "Representational Power of Restricted Boltzmann Machines and Deep Belief Networks," *Neural Computation*, vol. 20, no.6, pp. 1631-1649, Jun.2008.

[60] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The Difficulty of Training Deep Architectures and The Effect of Unsupervised Pre-training," in *Int.Conf. AISTATS*. 2009, Florida, USA, vol.5. JMLR : W&CP , pp. 153-160.

[61] G. E. Hinton, "Learning Multiple Layers of Representation," *Trends in Cognitive Sciences,* vol. 11, no.10, pp. 428-434, Oct.2007.

[62] D. Pokrajac and A. Lazarevic, "Applications of Unsupervised Neural Networks in Data Mining," in *7th Seminar. Neural Network Applications in Electrical Engineering. NEUREL 2004,* pp. 17-20.

[63] D. DeMers and G. Cottrell, "Non-linear Dimensionality Reduction," *Advances in neural information processing systems,* pp. 580-580, 1993.

[64] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Cognitive modeling,* MIT, pp.213-220, 2002.

[65] R. Hecht-Nielsen, "Theory of the Backpropagation Neural Network," in *Int.Joint.conf. Neural Networks.1989,* Washington, USA, vol.1, pp. 593-605 .

[66] G. E. Hinton, "How Neural Networks Learn from Experience," *Scientific American,* vol. 267, pp. 145-151, 1992.

[67] G. Hinton, "Training a Deep Autoencoder or a Classifier on MNIST digits," [online].Available:http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html.

[68] C. E. Connor, H. E. Egeth, and S. Yantis, "Visual Attention: Bottom-Up Versus Top-Down," *Current Biology,* vol. 14, no.19, pp. R850-R852, Oct.2004.

[69] L. Zhiqing, S. Zhiping, L. Zhixin, and S. Zhongzhi, "Image Reconstruction by Sparse Coding and Selective Attention," in *2nd Int.Congr.Image,signal Process,* Tianjin, 2009., IEEE, pp. 1-5.

[70] E. J. Teoh, K. C. Tan, and C. Xiang, "Estimating the Number of Hidden Neurons in a Feedforward Network Using the Singular Value Decomposition," *IEEE Trans.Neural Networks, ,* vol. 17, no.6, pp. 1623-1629, Nov.2006.

[71] I. Arel, D. C. Rose, and T. P. Karnowski, "Deep Machine Learning-A New Frontier in Artificial Intelligence Research," *IEEE Mag.Computational Intelligence ,* vol. 5, no.4, pp. 13-18, Nov 2010.

[72] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing Higher-Layer Features of a Deep network," Dept. IRO, Univ. Montréal, Tech. Rep.1349,Jun. 2009.