Pattern Recognition of Arabic/Persian Handwritten Digits using Adaptive Boosting, Neural Networks and Deep Boltzmann Machines

PATTERN RECOGNITION OF ARABIC/PERSIAN HANDWRITTEN DIGITS USING ADAPTIVE BOOSTING, NEURAL NETWORKS AND DEEP BOLTZMANN MACHINES

BY

SARAH ALI A. ALATTAS, B.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF MATHEMATICS & STATISTICS

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Master of Science (2017) McMaster University

(Mathematics & Statistics) Hamilton, Ontario, Canada

TITLE:	Pattern Recognition of Arabic/Persian Handwritten Digits using Adaptive Boosting, Neural Networks and Deep Boltzmann Machines
AUTHOR:	Sarah Ali A. Alattas McMaster University, Canada
SUPERVISOR:	Dr. Paul D. McNicholas
NUMBER OF PAGES:	viii, 40

To my parents, Nehad & Ali

and

To my family: Talal, Rital and Rima

Abstract

Classification of handwritten numerals has captured the attention of the statistical and machine learning community. A common statistical approach is ensemble learning, which combines basic classifiers to produce one powerful predictor. Another popular method is neural networks, which is a non-linear two stage statistical model inspired by the biological neural networks. In this thesis, neural networks (nnet), adaptive boosting (AdaBoost), and deep Boltzmann machine (DBM) algorithms are tested for Arabic/Persian handwritten digits, and their recognition performance is then compared.

Acknowledgements

First, I would like to thank my supervisor Dr. Paul McNicholas for the supervision of the thesis and for providing me with all the information about the field of classification and data science. Secondly, I would like to acknowledge the King Abdullah Scholarship Fund and the Saudi Culture Bureau in Canada. I would also like to thank Dr. Ben Bolker, and Dr. Ridha Khedri with Dr. McNicholas for accepting to be on my examination committee. Moreover, I would like to thank Dr. Yang Tang for the helpful discussions and for sharing her code for generating MCLT binary data. I would like to thank my friends and collogues, who I was fortunate to work with throughout Master's studies. Finally, I wanted to thank my husband, my parents and my siblings for their love and support even from overseas.

Contents

List of Figures

Chapter 1

Introduction

In statistics and machine learning, pattern recognition is the process of applying algorithms to enable a model or a system to interpret or recognize some properties in the pattern. One example of pattern recognition is classification. Classification involves taking a set of observations and using data features as inputs, and assigns these observations into categories. Given a pattern, its recognition/classification may follow from:

- supervised classification in which we need the label of the data to make a classification rule,
- unsupervised classification or clustering, where the true labels of a data does not exit or not used to make the classification rule,
- semi-supervised classification, where both labelled and unlabelled data are used (with equal weights) to generate the rule,
- fractionally-supervised classification, introduced by Vrbik and McNicholas (2015),

where also both labelled and unlabelled data are used; however, the weight or the amount of labelled observations can vary between any level.

For simple patterns, machine learning methods like logistic regression or decision trees can produce accurate results; for complex patterns, neural nets start to outperform other machine learning methods (Friedman *et al.*, 2001). At a very high level of pattern complexity, even a standard neural nets would take excessive time to train, or would simply fail to obtain a respectful accuracy; in this case, deep learning have proved to be the best solution for a highly complicated data (Schmidhuber, 2015).

Classifiers such as boosting, neural networks and deep Boltzmann machines have succeeded in recognising English handwritten digits (LeCun *et al.* (2010) and Salakhutdinov and Hinton (2009)). Boosting is an architecture for combining several learning approaches to achieve a powerful predictor. Neural networks are non-linear statistical models that predict the output after process complex calculation in some latent hidden layers (Friedman *et al.*, 2001). Deep Boltzmann machines, as opposed to neural networks, are generative models (i.e., they attempt to learn the distribution of the data) which consist of many hidden layers and those hidden layers are random variables (Salakhutdinov and Hinton, 2009). More details about these three classifiers will be given in Chapter 2. In Chapter 4, an application using these classifiers will be applied but in the Arabic/Persian handwritten digits.

Chapter 2

Background

2.1 Decision Trees

Decision trees (also known as classification trees) are recursive processes. As shown in Figure 2.1, the process starts at the root where the first step is to select a split that divides a data set into subsets, and each subset is further divided based on other selected splits; additional splits continue in the same style. Using the notation of Friedman *et al.* (2001), for region R_n with k classes and N observations, the probability that an observation is in node n is defined by:

$$\hat{P}_{nk} = \frac{1}{N} \sum_{x_i \in R_n} \mathbb{I}(y_i = k), \tag{2.1}$$

where $\mathbb{I}(y_i = k) = 1$ if $y_i = k$ and $\mathbb{I}(y_i = k) = 0$ otherwise. There are three criteria to select the splits, one is the classification error:

$$1 - \hat{P}_{nk}(n), \tag{2.2}$$

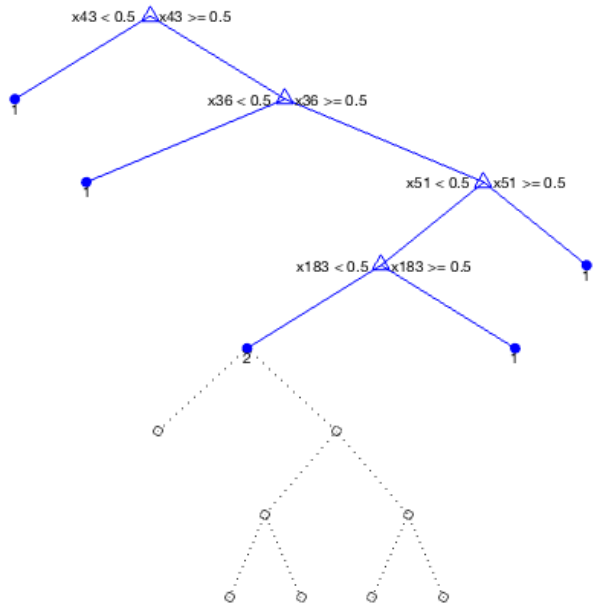


Figure 2.1: A decision tree on simulated dataset with binary splits on each step.

Another measure is Gini index:

$$\sum_{k \neq k'} \hat{P}_{nk} \hat{P}_{nk'}, \tag{2.3}$$

The third measure is the maximum deviance reduction (cross-entropy),Quinlan (1998):

$$-\sum_{k=1'}^{K} \hat{P}_{nk} \log(\hat{P}_{nk'}). \tag{2.4}$$

Friedman *et al.* (2001) stated that all three measurements are similar. To guide the pruning complexity, the misclassification rate is the typical choice. However, to grow the tree, cross-entropy and the Gini index are preferred because they are differentiable and more practical for numerical optimization.

2.2 A Classification Ensemble

Classification ensembles are predictive models which consist of a weighted combination of multiple classification models. Ensemble refers to the general problem of combining many possibly weak learners into one high-quality ensemble predictor. Combining multiple classification models helps to achieve an increase in the predictive accuracy.

2.2.1 Boosting Methods

Boosting is an ensemble methods that combine many classifiers to produce a powerful one by increasing the weight on observations modelled poorly by a classifier and reduce focus on observations that were classified correctly. Boosting can be applied beyond some algorithms such as decision trees or discriminant analysis. One of the most common boosting methods is Adaptive Boosting.

Adaptive Boosting

Adaptive Boosting, or AdaBoost, was first used by Freund *et al.* (1997) and has become the most common boosting method. This algorithm generates a set of classifiers and trains them sequentially. After training, Adaboost combines all the classifiers using some weighted error function. In binary classification, AdaBoostM1 uses the weighted classification error for every learner. For N observations and f_t models, the weighted classification error is given by:

$$\varepsilon_t = \frac{1}{2}\sum_{n=1}^{N} d_n^{(t)} \mathbb{I}\left[y_n \neq f_t(x_n)\right], \tag{2.5}$$

where: f_t is the learned classifier, y_n is the true label, $\mathbb{I}$ is the indicator function and $d_n^{(t)}$ is the weight.

For multiple classes, AdaBoostM2 computes weighted pseudo-loss and uses it as a measure of the classification accuracy. The learning procedure minimises the pseudo-loss error (Eibl and Pfeiffer, 2005).

For N observations and K classes, the weighted pseudo-loss error of a boosting algorithm is :

$$\varepsilon_t = \frac{1}{2} \sum_{n=1}^{N} \sum_{k \neq y_n} d_{n \neq k}^{(t)} \left[1 - f_t \left(x_n, y_n \right) f_t \left(x_n, k \right) \right], \tag{2.6}$$

Note that $f_t \left(x_n, k \right)$, ranging from [0,1], is the confidence of prediction by classifier f_t into class K, $d_{n \neq k}^{(t)}$ are observation weights for the class K at tree t, and y_n is the true class label.

2.3 Neural Networks

Friedman *et al.* (2001) defined neural nets (nnet) as non-linear statistical models that were developed from both Statistics and Artificial Intelligence. An nnet is a two stage learning model; before the inputs model the target, each input has to go through a function of linear combinations to extract features which then model the target. The idea of neural networks was inspired by the human brain (Carpenter, 1989). These networks are highly structured, and have three different layers: the first is the input layer, the last is the output layer, and units that are not in the input or output layer are hidden units. Each neuron in the hidden and output layers has a classifier. The first input units receive the data features of the object passing their output into the

hidden layers. The hidden layers can be described as distorting the input units in a non-linear approach which allows categories to become linearly separable by the output layer (Figure 2.1). If a network has more than one hidden layer, then the process repeats until the last layer is reached — the output layer. Based on the final output score, the object is classified. This process is called Forward Propagation (Schmidhuber, 2015).

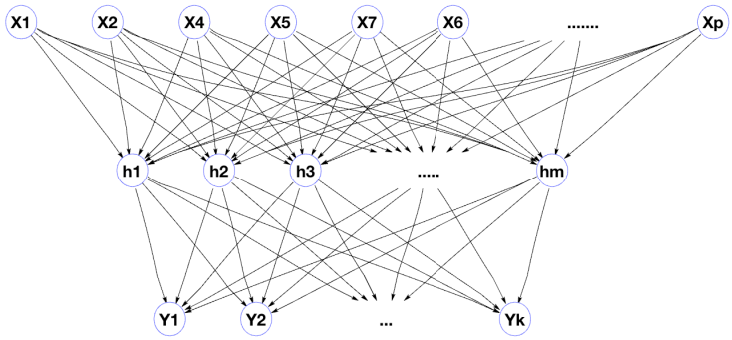


Figure 2.2: A single hidden layer, feed-forward neural network.

For clarity of presentation, we omit the bias parameters below. Consider hidden units $h_1, ..., h_m$, where each h_m is function of a linear combination of $X_1, ..., X_p$., then the target Y_k, which is a linear combination of the h_m can be described as:

$$h_m = \sigma(W_{0m}^1 + W_m^{1'} X), m = 1, ..., M, \tag{2.7}$$

$$T_k = W_{0k}^2 + W_k^{2'} h, k = 1, ..., K, \tag{2.8}$$

$$f_k(X) = g_k(T). \tag{2.9}$$

where $h = (h_1, h_2, ..., h_M)$, $T = (T_1, T_2, ..., T_K)$, $\sigma()$ is any activation function (non-linear transformation), usually the sigmoid activation function: $\sigma(v) = \frac{1}{1+e^{-}v}$, and the function $g_k(T)$ is the softmax function:

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}. \tag{2.10}$$

There are two main unknown parameters in neural networks: weight and bias. We denote the complete set of weights by $\theta = W^1, W^2$, which consists of:

$$\left\{W_{0m}^1, W_m^1; m = 1, 2, ..., M\right\} M(p+1)\text{weights},$$
$$\left\{W_{0m}^2, W_m^2; m = 1, 2, ..., K\right\} K(M+1)\text{weights}.$$

2.3.1 Loss Functions

In the classification paradigm, we can measure the network fit by either squared error or cross-entropy (deviance):

$$R(\theta) = \sum_{k=1}^{K}\sum_{i=1}^{N}(y_{ik} - f_k(x_i))^2, \tag{2.11}$$

$$R(\theta) = -\sum_{k=1}^{K}\sum_{i=1}^{N}(y_{ik}\log(f_k(x_i))). \tag{2.12}$$

As the network is trained, the weight and bias are updated based on the final accuracy result. When the accuracy in the nnet is low, the weight and bias numbers are modified slightly until the accuracy improves. Usually, the global minimizer of $R(\theta)$ is likely to be an overfitted solution. Instead, we tend to minimise $R(\theta)$ in a gradient descent, this procedure is known as back-propagation (Friedman *et al.*, 2001).

2.4 General Boltzmann Machines

General Boltzmann machines (BMs) are generative graphical models that are derived from the statistical physics literature, and they are known as energy-based models (Ackley *et al.*, 1985). The original learning algorithm for BMs requires randomly initialized Markov chains to perform their equilibrium distributions in order to estimate the gradient required for the likelihood objective function. One main issue is that this learning procedure is rather too slow and inefficient (Salakhutdinov and Hinton, 2009). However, some restrictions in the graphical model made learning these models more practical. For example, restricted Boltzmann machines (RBM), where the connection within the same layer is removed, and deep Boltzmann machines (DBM) which are undirected stacks of RBMs.

2.5 Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBMs) were first introduced by Smolensky (1986) under the name *harmonium*. An RBM is a special case of BMs Hinton (2010). RBMs are entirely undirected; unlike nnet (Figure 2.2), the edges in an RBM have no direction (as shown in Figure 2.3). Structurally, an RBM is a stochastic neural net with two layers: a layer of visible units and a layer of hidden units. Each unit in the first layer is connected to every unit in the following layer, but there is no connection between units within the same layer. This restriction makes the inference and learning easier because it makes the visible and hidden units conditionally independent given the other units.

The main objective in RBMs is to reconstruct the input; therefore, it does not require a labelled data set, which is useful for dealing with many real-world applications as most are unlabelled (Hinton *et al.*, 2006).

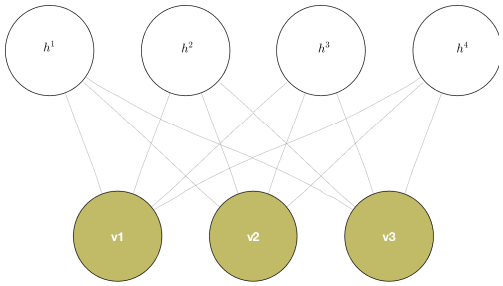


Figure 2.3: The structure of an RBM model with undirected connections between the layers.

RBMs are energy-based models, their joint probability distribution is given by its energy function:

$$P(\mathbf{v}, \mathbf{h}) = \frac{1}{Z(\theta)} e^{-E(\mathbf{v},\mathbf{h};\theta)}, \tag{2.13}$$

where the energy function of a joint configuration of the visible and hidden units $(\mathbf{v}, \mathbf{h})$ is given as:

$$E(\mathbf{v}, \mathbf{h}) = -a^T\mathbf{v} - b^T\mathbf{h} - v^T W\mathbf{h}. \tag{2.14}$$

Note that $Z(\theta)$ is the normalizing constant known also as partition function:

$$Z(\theta) = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)}. \tag{2.15}$$

$\theta = \{W, a, b\}$ are the parameters that we want to estimate. Furthermore, for a two-layer RBM model, the probability that the model is assigned to a visible vector $\mathbf{v}$

defined as a Gibbs distribution:

$$P(\mathbf{v}) = \frac{1}{Z(\theta)} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)}. \tag{2.16}$$

2.5.1 Conditional Distributions

Because of the bipartite structure, the inference of $P(h|v)$ and $P(v|h)$ are straightforward. The conditional distributions are derived as :

$$\begin{aligned} P(\mathbf{v}|\mathbf{h}) &= \frac{P(\mathbf{v},\mathbf{h})}{P(\mathbf{h})} \\ &= \frac{1}{P(h)} \frac{1}{Z(\theta)} \exp\left\{a^T\mathbf{v} + b^T\mathbf{h} + v^T W\mathbf{h}\right\} \\ &= \frac{1}{Z'(\theta)} \exp\left\{a^T\mathbf{v} + \mathbf{v}^T W\mathbf{h}\right\} \\ &= \frac{1}{Z'(\theta)} \exp\left\{\sum_{i=1}^{n} a_i v_i + \sum_{i=1}^{n} v^T W_{j,i} h_i\right\} \\ &= \frac{1}{Z'(\theta)} \prod_{i=1}^{n} \exp\left\{a_i v_i + v^T W_{j,i} h_i\right\}. \end{aligned} \tag{2.17}$$

Now, we can write the conditional distribution over the binary visible units as:

$$\begin{aligned} \tilde{P}(v_i = 1, |\mathbf{h}) &= \frac{\tilde{P}(v_i = 1|\mathbf{h})}{\tilde{P}(v_i = 0|\mathbf{h}) + \tilde{P}(v_i = 1|\mathbf{h})} \\ &= \frac{\exp\left\{a_j + \sum_j W_{ij} h_j\right\}}{\exp\{0\} + \exp\left\{a_j + \sum_j W_{ij} h_j\right\}} \\ &= \sigma(a_j + \sum_j W_{ij} h_j). \end{aligned} \tag{2.18}$$

Similarly, the conditional distribution over the hidden units h_j given the visible units is:

$$P(h_j = 1, |\mathbf{v}) = \sigma(b_j + \sum_i v_i W_{ij}). \tag{2.19}$$

2.5.2 Training RBMs

Training RBMs can be achieved by maximising the likelihood, we can also derive the log-likelihood with respect to a weight W_{ij}:

$$\begin{aligned}
\frac{\partial \log(P(\mathbf{v};\mathbf{h},\theta))}{\partial W_{ij}} &= \frac{\partial}{\partial W_{ij}} \log \frac{1}{Z(\theta)} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)} \\
&= \frac{\partial}{\partial W_{ij}} \log \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)} - \frac{\partial}{\partial W_{ij}} \log \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)} \\
&= -\sum_{\mathbf{h}} \frac{e^{-E(\mathbf{v},\mathbf{h};\theta)}}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)}} \frac{\partial E(\mathbf{v},\mathbf{h};\theta)}{\partial W_{ij}} + \sum_{\mathbf{v},\mathbf{h}} \frac{e^{-E(\mathbf{v},\mathbf{h};\theta)}}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)}} \frac{\partial E(\mathbf{v},\mathbf{h};\theta)}{\partial W_{ij}} \\
&= -\sum_{\mathbf{h}} p(h|v) \frac{\partial E(\mathbf{v},\mathbf{h};\theta)}{\partial W_{ij}} + \sum_{\mathbf{v},\mathbf{h}} p(h,v) \frac{\partial E(\mathbf{v},\mathbf{h};\theta)}{\partial W_{ij}}.
\end{aligned} \tag{2.20}$$

Now taking the derivative of (2.14) w.r.t $W_{i,j}$, (2.20) becomes:

$$\begin{aligned}
\frac{\partial \log(P(\mathbf{v};\mathbf{h},\theta))}{\partial W_{ij}} &= \sum_{\mathbf{h}} p(h|v) h_j v_i - \sum_{\mathbf{v}} p(v) \sum_{\mathbf{h}} p(h|v) h_j v_i \\
&= E_{Pdata} - E_{Pmodel}.
\end{aligned} \tag{2.21}$$

As stated in Hinton (2010), E_{Pdata} in 2.21 denotes to the expectation sufficient statistics with respect to the training data; this part can be computed in close form. Using the result of (2.19), we can take $v_i h_i$ as an unbiased sample. However, the second

term, E_{Pmodel}, is intractable because it involves the sum over all possible configurations of the visible and hidden units which are exponentially configurations.

2.5.3 Contrastive Divergence Learning

Introduced Hinton (2002), contrastive divergence learning (CD) is an objective function that minimises the computational costs when Markov chain Monte Carlo methods (MCMC) is used with exponential configurations. The idea of using CD with RBMs is to approximate E_{Pmodel} by running a Gibbs sampling in two steps: we first sample the $P(h_j = 1, |\mathbf{v})$ and update all h_j, and then, sample $P(v_i = 1, |\mathbf{h})$ and update all v_i. A Gibbs chain can run for few iteration and often one Gibbs sampling is sufficient for E_{Pmodel} (Carreira-Perpinan and Hinton, 2005). Note that the estimate of CD is biased but generally this bias is very small (Fischer and Igel, 2014).

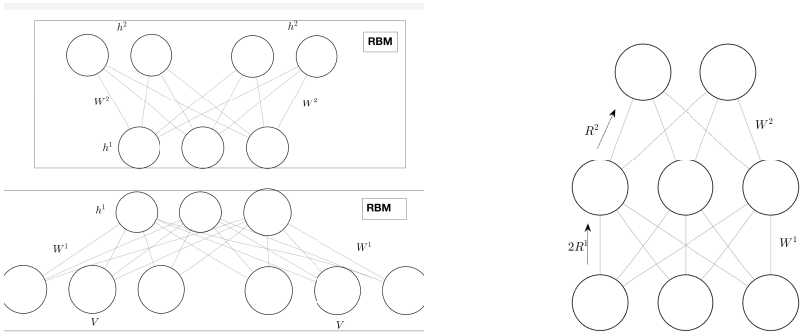
Then, for RBMs with CD, the change in a weight becomes:

$$W_t = W_{t-1} - \alpha_{t-1} \frac{\partial \log(P(v, h; \theta))}{\partial \theta}, \tag{2.22}$$

Where α is the learning rate and $\frac{\partial \log(P(v,h;\theta))}{\partial \theta} = E_{Pdata} - E_{Pgibbs_n}$ (Hinton, 2010).

2.6 Deep Boltzmann Machines

Introduced by Salakhutdinov and Hinton (2009), a deep Boltzmann machine is a type of Markov random field. The structure of a DBM consists of pre-training a stack of modified RBMs that are then composed to create DBM. All layers form an RBM with undirected connections, also with no within-layer connections.



(a) (b)

Figure 2.4: a: Pre-training two RBMs that will compose to create DBM with one visible layer (bottom) and two hidden layers. b: The graphical model for a DBM with recognition models: All connections between layers are undirected but with no within-layer connections.

Using the notation of Salakhutdinov and Hinton (2009), and omitting the bias term, a DBM with two hidden layer has a joint probability given by:

$$P(\mathbf{v}, \mathbf{h^1}, \mathbf{h^2}) = \frac{1}{Z(\theta)} \exp(-E(\mathbf{v}, \mathbf{h^1}, \mathbf{h^2}; \theta)), \tag{2.23}$$

where $\mathbf{v} \epsilon \{0,1\}^D$ is the set of binary visible units; $\mathbf{h} \epsilon \{0,1\}^D$ is the set of hidden units, and the model parameters are $\theta = \{W^1, W^2\}$, which represent the interaction between visible-to-hidden units and the interaction between hidden-to-hidden units. The energy function is given by:

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2) = -\mathbf{v}^T W^1 \mathbf{h}^1 - \mathbf{h}^T W^2 \mathbf{h}^2. \tag{2.24}$$

When pretraining RBMs to creat DBMs, Salakhutdinov and Hinton (2009) advocate doubling the inputs and tying the visible-to-hidden weights in for the first and the last RBMs layer, as shown in figure 2.4a. The modified RBM solves the

double-counting problem when top-down and bottom-up influences are subsequently stacked. As in RBMs, the bipartite structure of DBMs makes the units within a visible or hidden layer conditionally independent given the units of the other layers. The conditional distributions are derived as :

$$P(h_j^1 = 1, |\mathbf{v}, \mathbf{h^2}) = \sigma(\sum_i W_{ij}^1 v_i + \sum_k W_{jk} h_j^2), \tag{2.25}$$

$$P(v_i = 1, |\mathbf{h^1}) = \sigma(\sum_j W_{ij}^1 h_j), \tag{2.26}$$

$$P(h_k^2 = 1, |\mathbf{h^1}) = \sigma(\sum_j W_{ik}^2 h_i). \tag{2.27}$$

As with RBMs, the derivative of the log-likelihood w.r.t parameter W^1, as shown in Salakhutdinov and Hinton (2009), is the difference between two expectations :

$$\frac{\partial \log P(\mathbf{v}, h; \theta)}{\partial W^1} = E_{Pdata}[\mathbf{v}\mathbf{h}^{1^T}] - E_{P\theta}[\mathbf{v}\mathbf{h}^{1^T}]. \tag{2.28}$$

The first term denotes to the expectation sufficient statistics with respect to the training data, and the second term denotes to the expectation sufficient statistics with respect to the model parameters. However, unlike with RBMs, neither of these expectations are available in close form.

2.6.1 Variational Inference

Salakhutdinov and Hinton (2009) introduced a variational approach for approximating the E_{Pdata}. The idea of variational inference is to approximate an intractable

distribution using another tractable one. In the context of DBMs, the E_{Pdata} approximating can be done using mean-field inference. The mean field approximation is a variational inference, where the approximating distribution $Q_\mu(\mathbf{h}|\mathbf{v})$ is the fully factorial distribution (Goodfellow *et al.*, 2016).

For a two sets of hidden units, the fully factorial distribution can be defined as:

$$Q^{MF}(\mathbf{h} \mid \mathbf{v}; \mu) = \prod_{j=1}^{F_1} \prod_{k=1}^{F_2} q(\mathbf{h}_j^1) q(\mathbf{h}_k^2). \tag{2.29}$$

where $\mu = \{\mu^1, \mu^2\}$ are the mean-field parameters with $q(h_i^l = 1) = \mu_i^l = 1$ for $l = 1, 2$.

$$\mu_i^1 = \sigma\left(\sum_j W_{i,j}^1 v_j + \sum_j W_{i,j}^2 h_j^2\right), \tag{2.30}$$

$$\mu_i^2 = \sigma\left(\sum_j W_{i,j}^2 h_j^1\right). \tag{2.31}$$

To solve these fixed-point equations, we basically iterate (2.30) and (2.31) to minimize the Kullback-Leibler (KL) divergence between the mean-field posterior $Q_\mu(\mathbf{h}|\mathbf{v})$ and the $P(\mathbf{v}, h; \theta)$. For details, see (Salakhutdinov and Hinton, 2009).

Then, Salakhutdinov and Larochelle (2010) proposed a new algorithm to speed the mean field approach. They use a separate recognition model to initialize the values of the latent variables in all layers. This approximate algorithm allows the parameters of all layers to be optimized jointly and to be more practically for large datasets. The idea is in using a separate set of recognition weights $Q^{rec} = \{R^1, R^2\}$ as shown in Figure 2.4b. During learning, given an input vector, the recognition parameters are used to provide an initial guess $\nu = \{\nu^1, \nu^2\}$ of the fully factorized approximating distribution: for each training example set. Finally, the recognition

weights are updated to minimize the Kullback–Leibler (KL) divergence between the mean-field posterior Q^{MF} and the factorial posterior Q^{rec};

$$KL(Q^{MF} \parallel Q^{rec}) = -\sum_i \mu_i \log \nu_i - \sum_i (1-\mu_i)\log(1-\nu_i) + Const, \tag{2.32}$$

Using back-propagation, we can efficiently calculate the gradient of this KL with respect to the recognition weights θ^{rec}:

$$Q^{rec}(\mathbf{h} \mid \mathbf{v}; \mu) = \prod_{j=1}^{F_1} \prod_{k=1}^{F_2} q^{rec}(\mathbf{h}_j^1) q^{rec}(\mathbf{h}_k^2), \tag{2.33}$$

where $q^{rec}(h_i^l = 1) = \nu_i^l$ for $l = 1, 2$. For further details see (Salakhutdinov and Larochelle, 2010).

After training, DBMs can recognize the patterns in the data, and each layer ends up learning the input structure. In the end, a DBM may require only a small set of labels to fine-tune and make minor changes to the weights and biases which helps the model to increase the accuracy (Salakhutdinov and Hinton, 2009).

Chapter 3

Methodology

For an efficient pattern recognition, our task is to apply multiple learning algorithms (to choose among) and to set their parameters (to tune the models).

3.1 Model Assessment and Selection

When we train models, we need to make sure that our trained model has a good capability to classify a new independent data set. First, the performance of multiple models is evaluated to choose the final model (model selection). After choosing the final model, we assess its prediction error on a test data (performance assessment). Then we compare the performance to other methods.

As the performance evaluation criteria, the widely used error rate measure is chosen.

$$\text{error rate} = 1 - \frac{1}{N}\sum_{n=1}^{N}\mathbb{I}(y_n = f(x_n)), \tag{3.1}$$

where y_n is the true label and $f(x_j)$ is the learned classifier.

3.2 Cross validation

Cross validation (CV) is a well-known technique for estimating prediction error or what Friedman *et al.* (2001) called "the expected extra-sample error". This error can estimate the model performance on a new data set which assesses that a model is generalizing the data without overfitting. An over-fitting occurs when the learning algorithm focuses on the training set mostly and compromises the generalization ability.

The expected extra-sample error of cross validation can be defined as:

$$Err = E[L(Y, \hat{f}(X)], \tag{3.2}$$

which is the average generalization error of a classifier $\hat{f}(X)$ on an independent test set from the joint distribution of X and Y.

In CV, the data are divided into k subsets, each subset called a fold. the number of observations in each fold is usually equal. Then the learning algorithm is applied k times where every time the union of all subsets is used for training except one fold that is used as a test set (Friedman *et al.*, 2001).

3.3 Stratification

A stratified sampling or stratification is a process of partitioning the data into multiple homogeneous subsets. One common stratification is: a training set, a validation set, and a test set. First, the training set to fit the models; then a validation set to predict the model error and to prevent overfitting; and a test set for assessment of the generalization error of the final chosen model.

The choice whether to apply cross validation or stratification relies on the situation. If we are in a "data rich" situation, then we can simply apply a stratification to the data set; otherwise, CV is the better choice (Friedman *et al.*, 2001).

In this paper, we can make the sample size large in the simulation section, and the real data set used in Chapter 4 is over 100K large (Khosravi and Kabir, 2007). Accordingly, we will use a validation set without the need of CV.

3.4 Some Issues in Training an Ensemble Tree

3.4.1 Selecting the Number of Trees and Splitters

There are three parameters to tune:

- Numbers of trees (N): assigning too many trees results in an overfitted solution. The best way to avoid this is by cross validation.

- The Learning rate λ: Also known as the shrinkage parameter, the shrinkage controls the contribution of each decision tree by scaling its contribution by $0 < \lambda < 1$ factor. This helps to reduce overfitting.

  Note that minimal shrinkage λ will need a larger value of N to achieve good accuracy while greater λ requires less trees .

- Number of Splitters: these control the complexity of the boosted ensemble. Often a single split works effectively (James *et al.*, 2014).

3.5 Some Issues in Training Neural Networks and DBM

3.5.1 Selecting the Number of Hidden Units and Layers

Hidden units enable the model to capture the nonlinearity in the data set, so with few hidden units, the model may be unable to achieve this. However, without an appropriate regularization, using more parameters than training cases will typically cause overfitting (Hinton, 2010).

A reasonable number of hidden units as suggested by Heaton (2008) is somewhere more than the outputs and no more than twice the number of inputs.

A choice of the number of hidden layers also has no rule but experimentation. One could start with a single layer, and increase the hidden units until the desirables performance is achieved. If not, then an additional layer helps with appropriate regularization.

3.6 Simulated Data

To illustrate the accuracy of classifying a highly dimensional binary data into latent groups , we generated dataset from a Mixture of Latent Trait Models with Common Slope Parameters (MCLT) Tang *et al.* (2015). We simulated two different datasets with 6000 samples each. Two-dimensional multivariate normal distributions are considered as latent variables. We took D=20 and 200 to test different levels of data dimensionally. For each dataset we split the data into three sets: training, validating and test sets. 50% of the data was selected randomly as a training set, 25% as a

validation set and the remaining 25% as a test set.

The first dataset consists of two latent variables, the first component has $\mu = (0, 0)$ and the second has $\mu = (3, 3)$. We created an ensemble of boosted classification trees. Because we have only two groups, we will set the maximum number of splitters per tree to be 4. We will tune the learning rate to minimize the number of learning cycles. We also added two classification trees: a stump decision tree (i.e., a tree with a single split), and a deep tree where the minimum observations per node is one observation. We can conclude the boosted tree gave better performance than single trees. The selected model has 415 trees and a learning rate=0.1. This model achieves an error rate of 0.0023 on the test set.

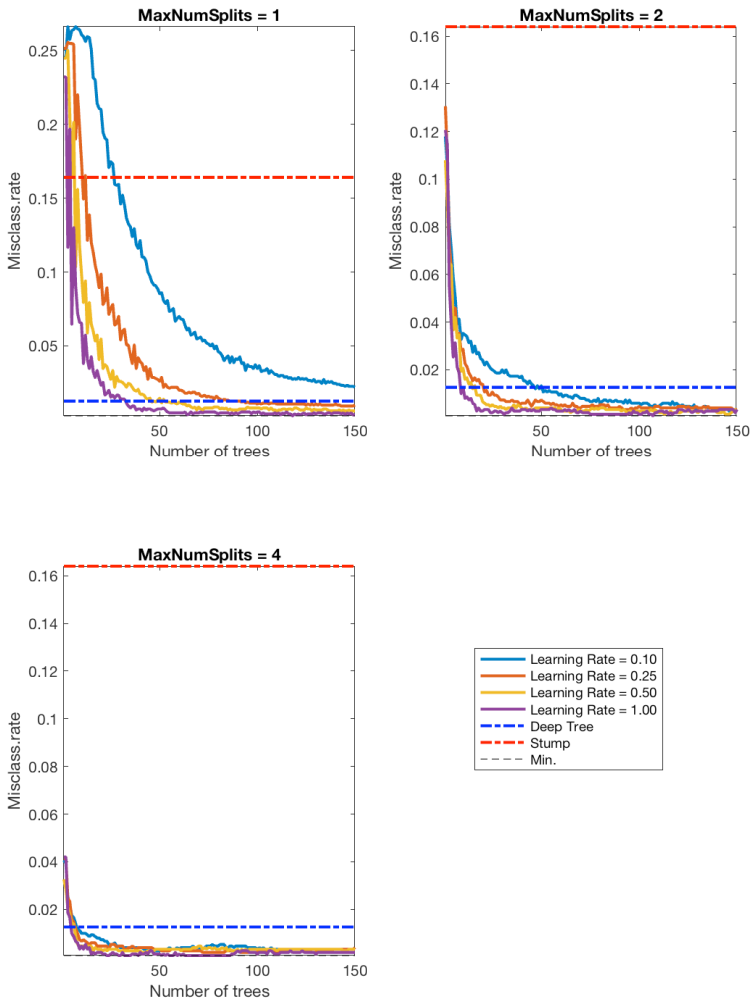


Figure 3.1: The misclassification rate of MCLT data as the number of trees and splitters increase.

Moreover, we trained two nnet with one hidden layer. The first nnet had 10 hidden units and the other had 50 hidden units. The neural network pattern recognition tool (Demuth *et al.*, 2008) was used to train the two neural networks. The performance

is measured in term of the misclassification rate on the test set; the 10-hidden-units achieves an error rate of 0.002; while the 50-hidden-units achieves an error rate of 0.001.

Using the publically available code (Larochelle, 2010), we trained two two-layer DBMs each consisting of a binary visible layer followed by two binary hidden layers. The DBM is with 100 units in the first and second hidden layer. The other is with 200 hidden units in both layers. Following Salakhutdinov and Larochelle (2010), the data is subdivided into 600 mini-batches, and after each mini-batch, the weights are updated. The RBM for the pretraining requires 50 epochs over the training set. And a DBM training was done for 150 epochs. Both 200-200 net and 500-500 net had a perfect classification performance on the test set.

Then, we repeated the experiment with four two-dimensional latent variables. The first component has mean $\mu_1 = (0, 1)'$, the second has mean $\mu_2 = (1, 1)'$; the other two have means $\mu_3 = (3, 3)'$ and $\mu_4 = (8, 8)'$. We started with an ensemble of boosted classification trees using AdaboostM2. Our selected model has 239 trees with a maximum of 81 splitters per tree and Learning Rate = 0.1. This model has an error rate of 0.2107 on the test set.

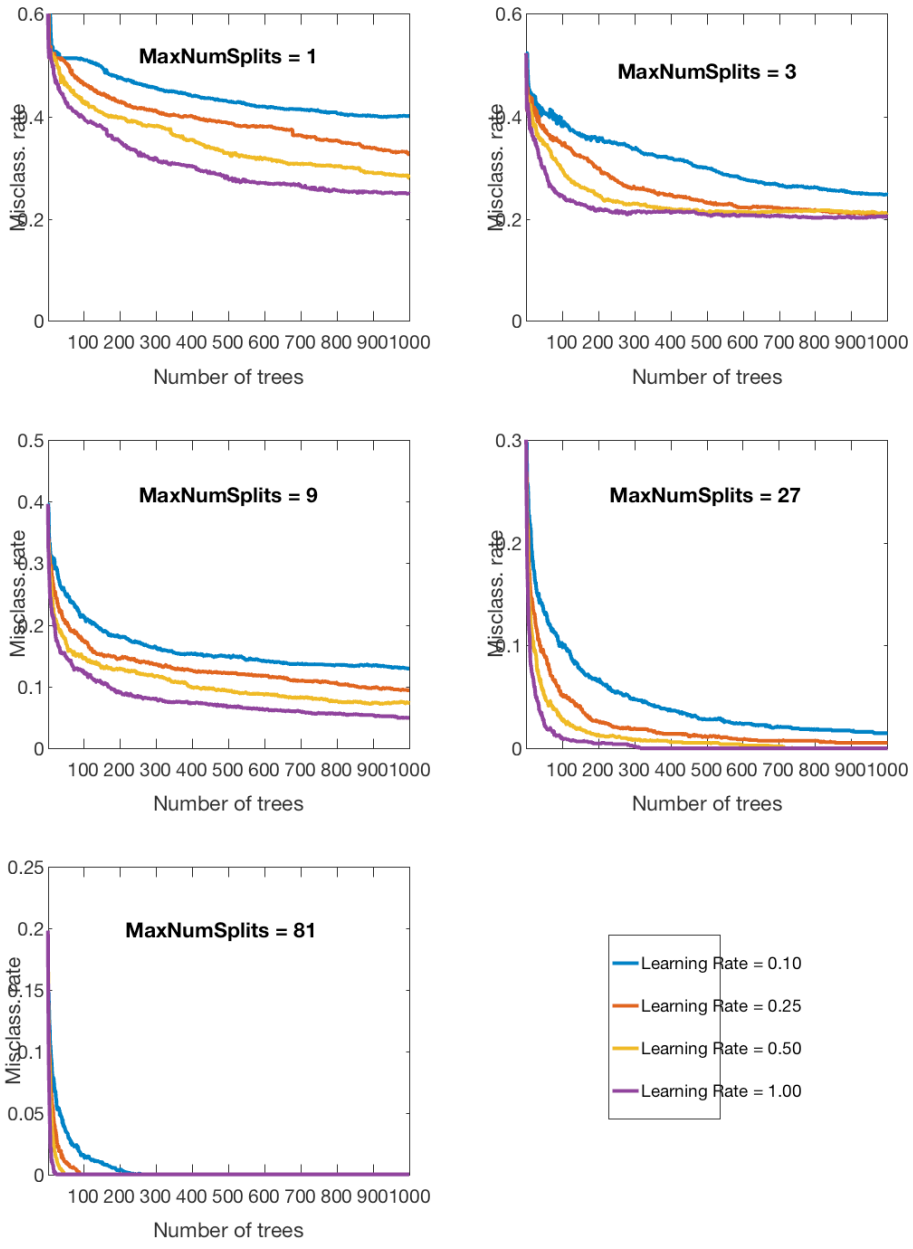


Figure 3.2: The misclassification rate as the number of trees and splitters increase.

Moreover, we trained three two-layer feed-forward neural networks with 10, 50 and 500 hidden units. The performance is measured in term of the misclassification rate on the test set; the 10-hidden-units achieves an error rate of 0.204; the 50-hidden-units achieves an error rate of 0.207; while the 500-hidden-units achieves an error rate of 0.246. Finally, we tested two two-layers DBMs, one with 200 hidden units in the two layers and another with 100-100 hidden units. The the 100-100 net achieves 0.1627 while 200-200 hidden units achieves an error rate of 0.1593.

Table 3.1: A summary of the performances of selected models.

Number of groups G	Method	error rate	
		Trainig set	test set
G= 2	AdaboostM1	0.0027	0.0023
	10 hidden units nnet	0	0.002
	50 hidden units nnet	0	0.001
	100-100 DBM	0	0.001
	200-200 DBM	0	0
G= 4	AdaboostM2	0.228	0.2107
	10 hidden units nnet	0.101	0.204
	100 hidden units nnet	0.109	0.207
	500 hidden units nnet	0.092	0.212
	100-100 DBM	0.0753	0.1627
	200-200 DBM	0.0637	0.1593

In conclusion, with a total of two groups, all methods performed well. However, with more than two groups the algorithms started to fail. Increasing the number of

hidden units did improve the accuracy in the first experiment. However, when we generated 4 groups, we did not notice a significant improvement as the number of hidden units increases.

Chapter 4

Arabic/Persian Handwritten Digits Recognition

Recognition of HUDA data set (Arabic/Persian handwritten digits) is performed using DBMs, nnet, and Boosting methods. This is the largest data set of Arabic hand digits which is created by Khosravi and Kabir (2007). The data set itself consists of a 60000 training set, a 20000 test set, and a 22,352 remaining set. We will train and test our classifiers using the same training and test subsets, and we will choose the first 22000 digits in the remaining set as a validation set. Each of the images in the data set contains of a digit (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). In this analysis, we will classify the test images into one of the 10 digit classes. We will also compare the error rate of the three classifiers. To analyse the images, these training and test sets were grey-scaled, so all samples have binary units. Since the images come in different sizes, we normalize and unify the size of each picture to be 28×28 pixels as a preprocessing step. This means there are 784 pixels in each picture, and 85 columns, as the first column denotes the real number the image represents.

All corresponding computations were performed using Matlab software supported by SHARCNet, a network of high performance computers. The Wobbie cluster is a contributed network of 149 computers, and a total of 1456 computing cores. All tests were performed on Wobbie cluster with 8 to 16 cores.

4.1 Discriminative Performance

4.1.1 Boosting Ensemble

We created an ensemble of classification trees using AdaBoostM2 with specifying that each tree should be split a maximum of five times. The ensemble achieves a misclassification rate of around 0.08 after accumulating about 150 weak learners. Figure 4.1 shows the behaviour of the cross-validated misclassification rate as the number of trees in the ensemble increases. Table 4.1 shows the classification performance of the training set for three different values of tree size and learning rate.

Table 4.1: The error rate of adaBoostM2 with different parameters.

	Shrinkage value		
	.001	0.1	1
25 trees	0.4026	0.2930	0.1559
50 trees	0.3411	0.2505	0.1235
150 trees	0.3392	0.1526	0.0824

With maximum of five splitters, the number of trees N, and λ learning rate that yields the lowest misclassification rate is $N = 150$ and $\lambda = 1$. Then, a final predictive model is created based on these parameters and the entire training data. This model

has an error rate of 0.2183 on the test set.

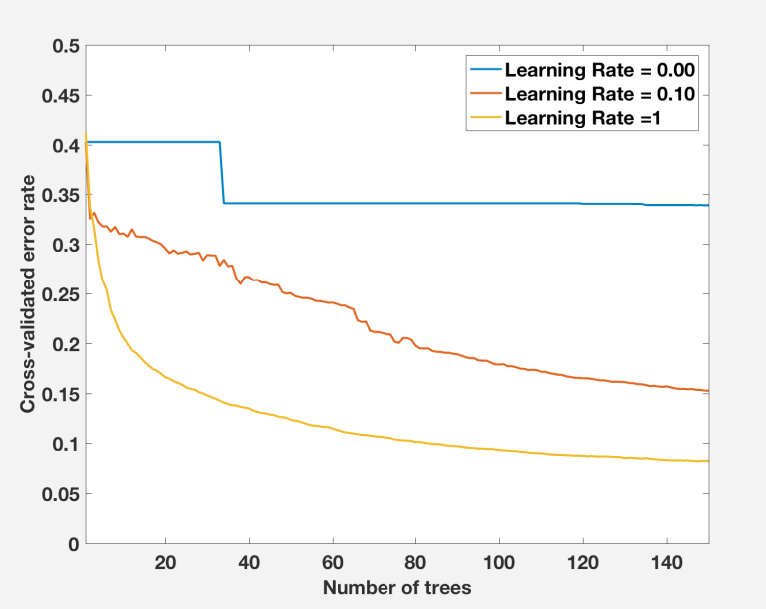
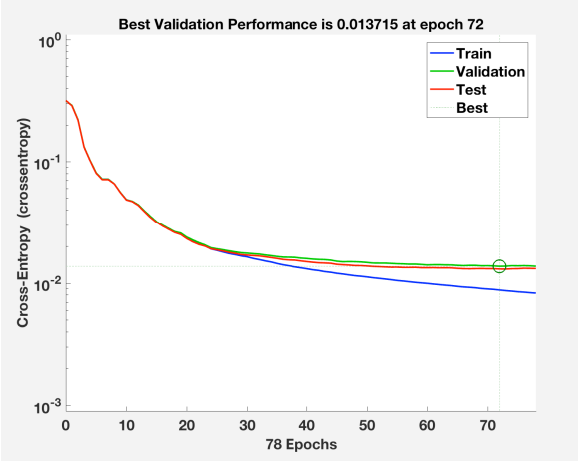


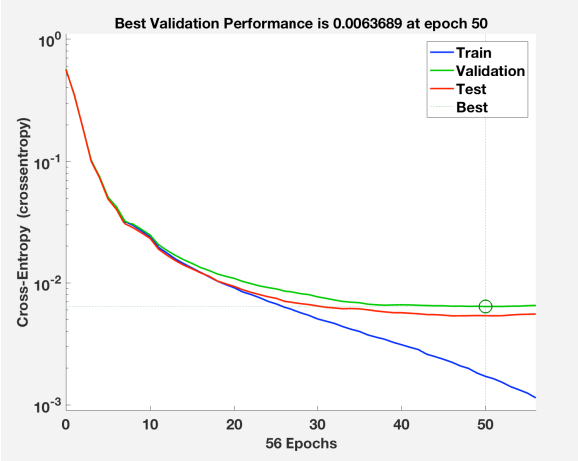
Figure 4.1: The misclassification rate as the number of trees increases.

4.1.2 Neural Network

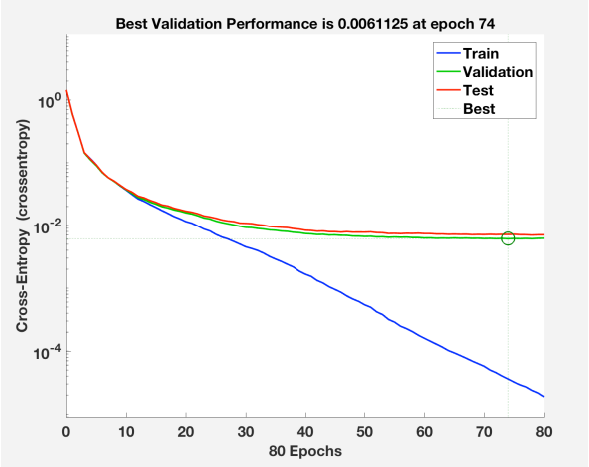
The standard network implemented here is a two-layers feed-forward network; the first, a hidden layer that contains the sigmoid transfer function; while the next layer (the output layer) is a softmax transfer function. Using neural network pattern recognition tool (Demuth *et al.*, 2008), we train three neural networks: one with 10 hidden units; another with 50 hidden units, and a third with 500 hidden units. Figure 4.2 shows the performance of the networks. Performance is measured in terms of cross entropy error of (2.12), and presented in log scale. It rapidly decreases as the number of hidden units increase. The overall output is very accurate; the 10-hidden-units achieves an error rate of 0.037; the 50-hidden-units achieves an error rate of 0.014; while the 500-hidden-units achieves an error rate of 0.0096.



(a)



(b)



(c)

Figure 4.2: The classification performance of nnet with different hidden units: (a) 10 hidden units, (b) 50 hidden units, and (c) 500 hidden units.

4.1.3 Deep Boltzmann Machine

Using the the publically available code (Larochelle, 2010), we trained two two-layer DBMs each consisting of a binary visible layer followed by two binary hidden layers. The DBM is with 500 units and 1000 hidden units in the first and second hidden layer respectively. The other is with 1000 hidden units in both layers. Following Salakhutdinov and Larochelle (2010), the data is subdivided into 600 mini-batches and after each mini-batch the weights are updated. For the stochastic approximation algorithm, we used five-Gibbs updates. Pretraining of DBMs was done with 200 epochs over the training set. The overall DBM training was done for 500 epochs.

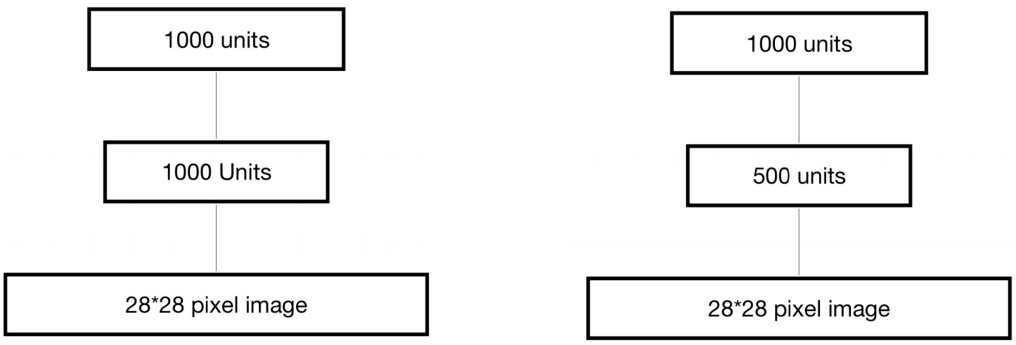


Figure 4.3: Two deep Boltzmann machines used in the experiment.

Finally, after fine-tuning, the 500–1000 hidden units DBM achieves an error rate of 0.0107 on the test set; and the 1000–1000 hidden units DBM achieves an error rate of 0.009.

DBM with 1000-1000 units outperforms nnet and Boosting in our experiment. To our knowledge, DBM with 1000-1000 units has the best error rate achieved for the Huda data set. This is compared to .0381 obtained by K-nearest neighbour (KNN), 0.0097 obtained by Convolutional Neural Networks, and 0.0188 by random forest

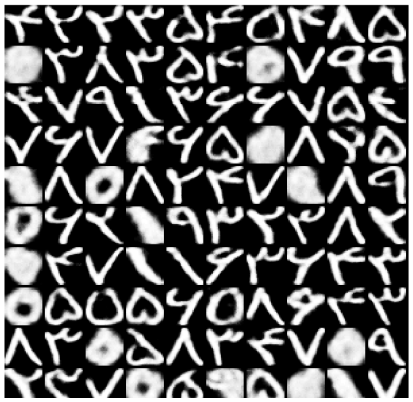
algorithm (Zamani *et al.*, 2015).

4.2 Generative Performance

4.2.1 Generating the Digits using DBMs

As mentioned previously, DBMs end up learning the full input structure. As in Salakhutdinov and Larochelle (2010), we generated samples from the DBM model by randomly initializing all binary states and running the Gibbs sampler for 100,000 steps. Figure 4.4 shows two digit-samples; one from our real handwritten digits, and the other from the generated ones. The images shown are the probabilities of the visible units given the states of the hidden units. We can see the generated samples mimic the real handwritten digits.

(a) Training Samples



(b) 500-1000 DBM

Figure 4.4: Digits samples from the training set and digits generated from DBM.

Chapter 5

Conclusions and Future Work

This thesis demonstrated different approaches on a pattern recognition for Arabic/Persian digits. The data set had 60,000 training samples, 22,000 validation samples and 20,000 test samples. Three supervised classifiers were used: Boosted ensemble trees, an architecture for combining many classifiers. The boosting techniques implemented here was Adaptive boosting founded by Freund and Schapire (1997). This algorithm trains learners sequentially based on pseudo-loss errors. The second method was two-layer feed-forward nueral network. It consists of a hidden layer that contains the sigmoid transfer function; while the next layer (the output layer) is a softmax transfer function. The other method was deep Boltzmann machine DBM (Salakhutdinov and Hinton, 2009), which is an undirected graphical models that consists of composed modified RBMs. The DBM models presented in this paper only work in the binary visible units,(i.e. variables are valued from 0 to 1) but other types of unit can be used for dealing with data that is not well-modeled by binary (or logistic) visible units Hinton (2010).

On the simulated data, the results showed that all classifiers had similar performances, the standard nnet and DBM had zero error rate when the number of groups were two. On the real data, however, we were able to achieve a very good classification accuracy with a one layer neural network; a less than 0.01 error rate was achieved by two hidden layer DBM with 1000 hidden units in each layer. Furthermore, AdaboostM2 achieved a 0.2183 classification accuracy.

From this analysis, we could conclude that an nnet with one hidden layer was the more practical algorithm. Compared to DBM, an nnet with 500 hidden units contained around 50% less parameters and performed the second best results. However, if these digits were to be used in a finical matter such as recognition of cheque deposits, any small improvement will have larger benefits, and hence DBMs are helpful even with their relatively expensive computational cost.

Future work will include using other types of processing the data. Demonstrating other ensemble methods such as Gentle AdaBoost and linear programming boost (Schapire and Singer (1999) and Demiriz *et al.* (2002)). Although Deep Boltzmann machine was very accurate, investigation to unsupervised learning methods such as deep belief networks is suggested (Hinton *et al.*, 2006). Furthermore, a combination of ensemble techniques and feature selection considered in Saeys *et al.* (2008) could be applied to performed the boosting tree AdaBoostM2 to achieve faster and more effective performance.

Bibliography

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive science*, **9**(1), 147–169.

Capinera, J. L. (2008). *Encyclopedia of entomology*. Springer Science & Business Media.

Carpenter, G. A. (1989). Neural network models for pattern recognition and associative memory. *Neural Networks*, **2**(4), 243–257.

Carreira-Perpinan, M. A. and Hinton, G. E. (2005). On contrastive divergence learning. In *Aistats*, volume 10, pages 33–40.

Demiriz, A., Bennett, K. P., and Shawe-Taylor, J. (2002). Linear programming boosting via column generation. *Machine Learning*, **46**(1), 225–254.

Demuth, H., Beale, M., and Hagan, M. (2008). Neural network toolbox6. *Users Guide*, pages 37–55.

Eibl, G. and Pfeiffer, K.-P. (2005). Multiclass boosting for weak classifiers. *Journal of Machine Learning Research*, **6**(2), 189–210.

Fischer, A. and Igel, C. (2014). Training restricted boltzmann machines: An introduction. *Pattern Recognition*, **47**(1), 25–39.

Freund, Y., Schapire, R. E., Singer, Y., and Warmuth, M. K. (1997). Using and combining predictors that specialize. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 334–343. ACM.

Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*, volume 1. Springer series in statistics New York.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. url http://www.deeplearningbook.org.

Heaton, J. (2008). *Introduction to neural networks with Java*. Heaton Research, Inc.

Hinton, G. (2010). A practical guide to training restricted boltzmann machines. *Momentum*, **9**(1), 926.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, **14**(8), 1771–1800.

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

James, G., Witten, D., and Hastie, T. (2014). An introduction to statistical learning: With applications in r.

Khosravi, H. and Kabir, E. (2007). Introducing a very large dataset of handwritten farsi digits and a study on their varieties. *Pattern Recognition Letters*, **28**(10), 1133–1141.

Larochelle, H. (2010). *pretraining, learning and finetuning DBM with recognetion model*. url$http://www.cs.toronto.edu/\ larocheh/code/dbm_recnet.tar.gz$; retrieved on May 19th, 2017.

LeCun, Y., Cortes, C., and Burges, C. J. (2010). Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, **2**.

Quinlan, J. R. (1998). Miniboosting decision trees. *Journal of Artificial Intelligence Research*, pages 1–15.

Saeys, Y., Abeel, T., and Van de Peer, Y. (2008). Robust feature selection using ensemble feature selection techniques. *Machine Learning and Knowledge Discovery in Databases*, pages 313–325.

Salakhutdinov, R. and Hinton, G. (2009). Deep boltzmann machines. In *Artificial Intelligence and Statistics*, pages 448–455.

Salakhutdinov, R. and Larochelle, H. (2010). Efficient learning of deep boltzmann machines. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 693–700.

Schapire, R. E. and Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, **37**(3), 297–336.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, **61**, 85–117.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory; cu-cs-321-86.

Tang, Y., Browne, R. P., and McNicholas, P. D. (2015). Model based clustering of high-dimensional binary data. *Computational Statistics & Data Analysis*, **87**, 84–101.

Vrbik, I. and McNicholas, P. D. (2015). Fractionally-supervised classification. *Journal of Classification*, **32**(3), 359–381.

Zamani, Y., Souri, Y., Rashidi, H., and Kasaei, S. (2015). Persian handwritten digit recognition by random forest and convolutional neural networks. In *IEEE Machine Vision and Image Processing (MVIP), 2015 9th Iranian Conference*, pages 37–40.