Classification with Deep Belief Networks

HussamHebbo
Jae Won Kim

Table of Contents

Table of Figures

1 Introduction

The machine learning is widely used to solve practical problems by learning from a given input. The core values of machine learning are to retain a good representation of data and create a generalized algorithm to predict unseen data properly. There are different algorithm types of machine learning, such as supervised learning, semi-supervised learning, and unsupervised learning; supervised learning produces a classifier or regression function from labeled data, semi-supervised learning make use of both labeled and unlabeled data, and unsupervised learning exploits unlabeled data. [10] Besides, each algorithm has various approaches to solve the problem. In this report, we are focusing in neural network approach to solve a classification problem.

2 Neural Networks

Neural networks are modeled similar to human brain's biological neural networks. [13] In the same manner as central nervous systems, neural network consists of an interconnected group of nodes (neurons). Each node receives inputs from other nodes and the weights between nodes adapt so that the whole network learns to perform useful computations. There are several types of neural networks structures with corresponding learning algorithms.

2.1 Perceptron

A perceptron neural network is one of the earliest and simplest types of neural networks. [10] The network takes a vector of feature activations converted from the raw input vector and learns the associated weights, in order to calculate a single scalar quantity for decision unit. If the quantity of decision unit is above some threshold, then the input vector can be classified as the target class. Thus, the perceptron is an algorithm for supervised classification. The learning algorithm is simply adapting weights by minimizing the error between the desired output and the actual output. If the data is linearly separable, then the learning algorithm will converge. This simple network has many limitations, such as hand-coded feature units are expensive and its single-layer structure makes it hard to learn a complex model.

2.2 Backpropagation

A backpropagation neural network is a multi-layer neural network and it overcomes the limitations of a single-layer neural network, Perceptron. Unlike single neural networks, multi-layer neural networks can create internal representations and learn different features in each layer. [13]

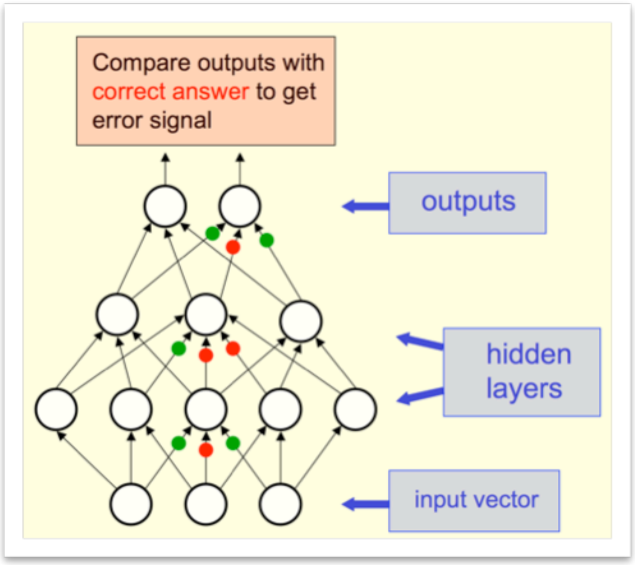


Figure 1: The structure of backpropagation neural network.

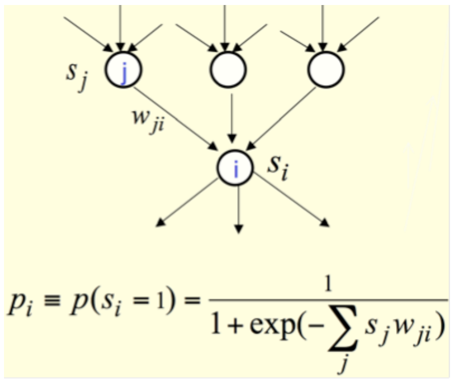
Backpropagation algorithm is developed to train multi-layer neural networks by computing error derivatives with respect to hidden activities and updating weights accordingly. Storing more features in neural networks and its relatively simple method of learning give great advantages for backpropagation neural networks to perform classifications; however, requiring labeled training data and the possibility of converging to a local minimum are the limitations of backpropagation neural networks. In fact, Support Vector Machines (SVM) has a simpler and faster learning method and its performance of classification is better than backpropagation neural networks.

3 Deep Belief Networks

In order to overcome the limitation of earlier neural networks, professor Geoffrey Hinton introduces Deep Belief Networks. Deep Belief Networks (DBN) consists of two different types of neural networks – Belief Networks and Restricted Boltzmann Machines. In contrast to perceptron and backpropagation neural networks, DBN is unsupervised learning algorithm.

3.1 Belief Networks

A basic belief network is composed of layers of stochastic binary units with weighted connections. In addition, the network is acyclic graph that allows us to observe what kinds of data the belief network believes in at the leaf nodes. [10]
The goal of a belief network is to infer the states of the unobserved stochastic binary units and adjusting the weights between these units so that the network can generate similar to the observed data. The stochastic binary units in belief networks have a state of 0 or 1 and the probability of becoming 1 is determined by a bias and weighted input from other units. The probability equation for these units is as follows:



$$p_i \equiv p(s_i = 1) = \frac{1}{1 + \exp(-\sum_j s_j w_{ji})}$$

Figure 2: The structure of belief network and its probability equation

The problem of learning weights in belief networks is the difficulty of obtaining the posterior distribution that has ‘explaining away’ issue. In ‘explaining away’, two independent hidden units can become dependent when there is an effect that those units can both influence. It is also called Conditional dependence where the occurrence of either of hidden units can explain away the occurrence of the unit connected from both hidden units. Furthermore, if belief networks have multi-layered neural networks, then the posterior distribution depends on the prior and likelihood of upper hidden layers and there are numerous ways of possible configurations of these layers. Therefore, Hinton proposes an idea of learning one layer at a time and restricting the connectivity of stochastic binary units, in order to make the learning efficient and simple. [2]

3.2 Restricted Boltzmann Machines

Boltzmann Machine is a stochastic recurrent neural network with stochastic binary units and undirected edges between units. Unfortunately, learning for Boltzmann machines is impractical and has a scalability issue. As a result, Restricted Boltzmann Machine (RBM) has been introduced [10],which has one layer of hidden units and restricts connections between hidden units. This allows for more efficient learning algorithm [6].The structure of RBM is depicted in the following figure:

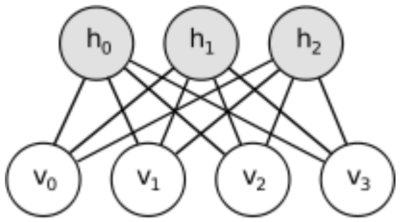


Figure 3: The structure of RBM

Given these configurations, probability distributions over hidden and/or visible units are defined in terms of the energy function:

$$P(v,h) = \frac{1}{Z}\exp\big(-E(v,h)\big) \tag{1}$$

Where Z:

$$Z = \sum_{v,h} \exp\,(-E(v,h)) \tag{2}$$

Then, the maximum likelihood learning algorithm can train the network by simply alternating between updating all the hidden units in parallel and all the visible units in parallel:

$$\frac{\partial \log P(v)}{\partial w_{ij}} = \langle v_i h_j \rangle_0 - \langle v_i h_j \rangle_\infty \tag{3}$$

To fasten the learning for a RBM, contrastive divergence algorithm is used and the general idea is to update all the hidden units in parallel starting with visible units, reconstruct visible units from the hidden units, and finally update the hidden units again. The learning rule is:

$$\Delta w_{ij} = \langle v_i h_j \rangle_0 - \langle v_i h_j \rangle_1 \tag{4}$$

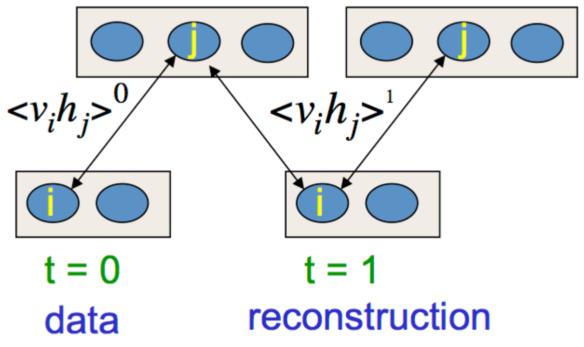


Figure 4: Contrastive Divergence algorithm for RBM

3.3 Deep Belief Networks

As Deep Belief Networks (DBN) name indicates, it is multi-layer belief networks. [1]Each layer is Restricted Boltzmann Machine and they are stacked each other to construct DBN. The first step of training DBN is to learn a layer of features from the visible units, using Contrastive Divergence (CD) algorithm. [7]Then, the next step is to treat the activations of previously trained features as visible unites and learn features of features in a second hidden layer. Finally, the whole DBN is trained when the learning for the final hidden layer is achieved.

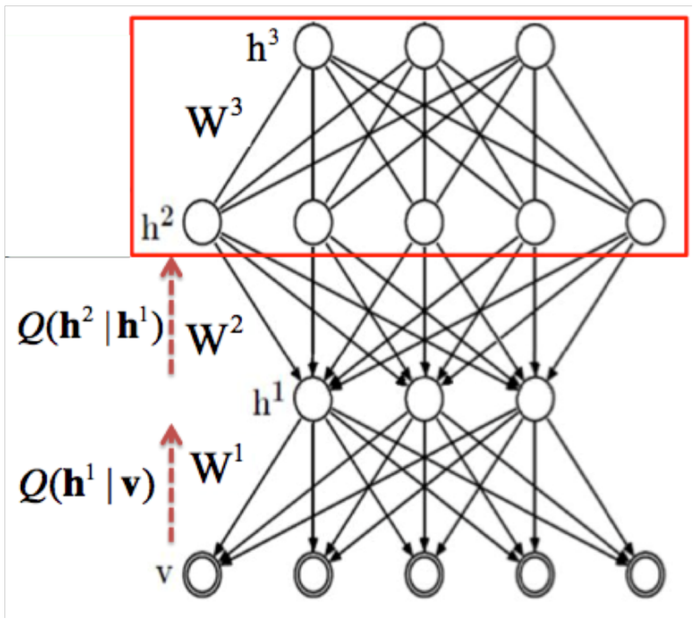


Figure 5: Greedy learning for DBN

This simple greedy learning algorithm works for training DBN. This is because that training RBM using CD algorithm for each layer looks for the local optimum and the next stacked RBM layer takes those optimally trained values and again look for the local optimum. At the end of this procedure, it is likely to get the global optimum as each layer consistently trained to get the optimum value. [4]

4 Implementation

Since MATLAB can easily represent visible layer, hidden layers and weights as matrixes and efficiently execute algorithms, we choose to implement DBN in MATLAB. In addition, we choose the MNIST handwritten digits to perform calculation so that we can compare the performance against other classifiers, listed in online [12].(http://yann.lecun.com/exdb/mnist/)

4.1 Dataset

The MNIST dataset is a dataset for handwritten digits. It consists of 60,000 training examples and 10,000 testing examples of digits. The handwritten digits are from 0 to 9 and have different shapes and positions for each image. However, they are normalized and centered in 28x28 pixels. Furthermore, all these images are labeled.

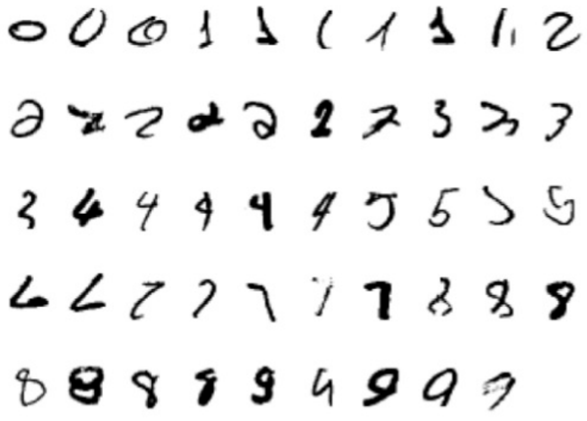


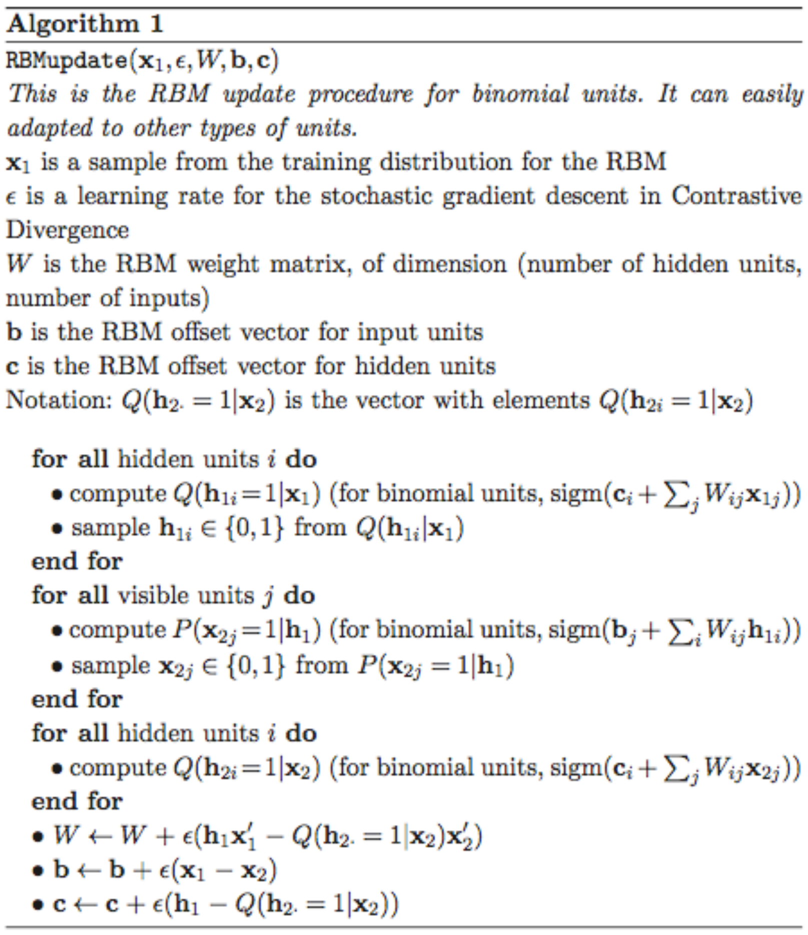
Figure 6: Examples of MINST handwritten dataset

4.2 Mini Batches

There are three different techniques to decide how often the weights are updated – online, full-batch and mini-batch. Online learning updates weights after each training data instance; thus, it takes more computation time to complete the learning compared to other techniques. On the other hand, full-batch runs a full sweep through the training data and updates weights, however, it is impractical to run full batch learning for a big dataset, such as 60,000 training samples in MNIST. Mini-batch divides a dataset into small chunks of data and performs the learning for each chunk. This method allows matrix-matrix multiplies in software programming, which takes less computation time and more efficient on GPUs. [11] Therefore, mini-batch learning is applied for our implementation. The training set has been divided into 600 mini-batches composed of 100 samples, 10 samples for each digit. Likewise, the testing set is divided into 100 mini-batches.

4.3 Implementation

As the DBN stacks many layers of RBMs, implementing a DBN requires training each layer of RBM. The theoretical parts of RBM and DBN training are explained in the previous section and now we introduce those algorithms in steps. The next algorithm shows the steps for Contrastive Divergence to train RBM. [11]



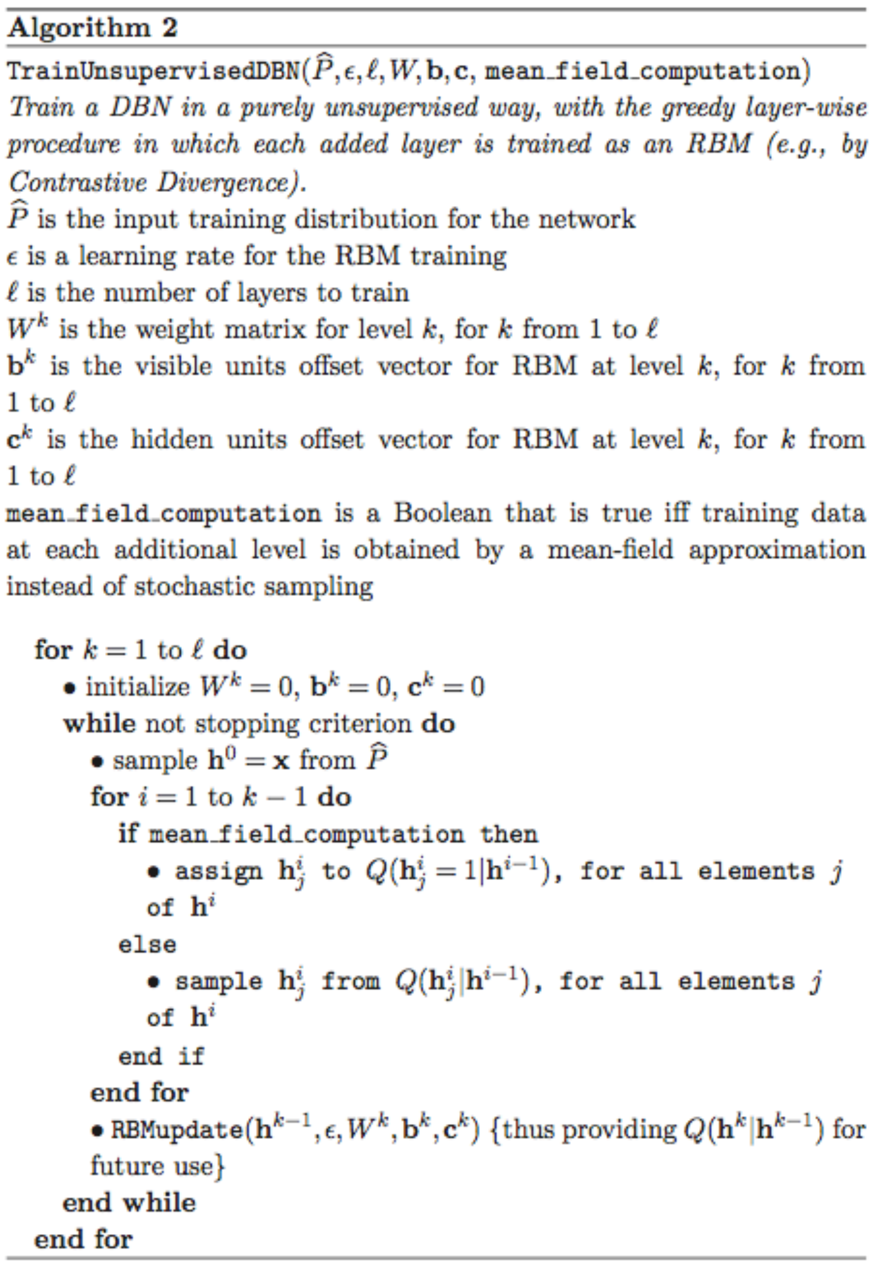
Algorithm 1

```
RBMupdate($\mathbf{x}_1, \epsilon, W, \mathbf{b}, \mathbf{c}$)
This is the RBM update procedure for binomial units. It can easily adapted to other types of units.
$\mathbf{x}_1$ is a sample from the training distribution for the RBM
$\epsilon$ is a learning rate for the stochastic gradient descent in Contrastive Divergence
$W$ is the RBM weight matrix, of dimension (number of hidden units, number of inputs)
$\mathbf{b}$ is the RBM offset vector for input units
$\mathbf{c}$ is the RBM offset vector for hidden units
Notation: $Q(\mathbf{h}_{2\cdot} = 1|\mathbf{x}_2)$ is the vector with elements $Q(\mathbf{h}_{2i} = 1|\mathbf{x}_2)$

  for all hidden units $i$ do
    • compute $Q(\mathbf{h}_{1i}=1|\mathbf{x}_1)$ (for binomial units, $\mathrm{sigm}(\mathbf{c}_i + \sum_j W_{ij}\mathbf{x}_{1j})$)
    • sample $\mathbf{h}_{1i} \in \{0,1\}$ from $Q(\mathbf{h}_{1i}|\mathbf{x}_1)$
  end for
  for all visible units $j$ do
    • compute $P(\mathbf{x}_{2j}=1|\mathbf{h}_1)$ (for binomial units, $\mathrm{sigm}(\mathbf{b}_j + \sum_i W_{ij}\mathbf{h}_{1i})$)
    • sample $\mathbf{x}_{2j} \in \{0,1\}$ from $P(\mathbf{x}_{2j}=1|\mathbf{h}_1)$
  end for
  for all hidden units $i$ do
    • compute $Q(\mathbf{h}_{2i}=1|\mathbf{x}_2)$ (for binomial units, $\mathrm{sigm}(\mathbf{c}_i + \sum_j W_{ij}\mathbf{x}_{2j})$)
  end for
  • $W \leftarrow W + \epsilon(\mathbf{h}_1\mathbf{x}_1' - Q(\mathbf{h}_{2\cdot} = 1|\mathbf{x}_2)\mathbf{x}_2')$
  • $\mathbf{b} \leftarrow \mathbf{b} + \epsilon(\mathbf{x}_1 - \mathbf{x}_2)$
  • $\mathbf{c} \leftarrow \mathbf{c} + \epsilon(\mathbf{h}_1 - Q(\mathbf{h}_{2\cdot} = 1|\mathbf{x}_2))$
```

Figure 7: RBM learning algorithm

For training RBM, we first randomly initialize the units and parameters. Then, there are two phases in Contrastive Divergence algorithm – positive and negative. During the positive phase, the binary states of the hidden units are determined by calculating the probabilities of weights and visible units. Since it is increasing the probability of training data, it is called positive phase. On the other hand, the negative phase decreases the probability of samples generated by the model. A complete positive-negative phase is considered as one epoch and the error between generated samples by the model and actual data vector is calculated at the end of the iteration. Finally, weights are updated by taking the derivative of the probability of visible units with respect to weights, which is the expectation of the difference between positive phase contribution and negative phase contribution. The full update rule with momentum and learning rate is implemented in MATLAB after error calculation.
The follwing algorithm is a greedy learning algorithm to train the whole DBN. [11]



Algorithm 2

`TrainUnsupervisedDBN`($\widehat{P}, \epsilon, \ell, W, \mathbf{b}, \mathbf{c}$, `mean_field_computation`)

Train a DBN in a purely unsupervised way, with the greedy layer-wise procedure in which each added layer is trained as an RBM (e.g., by Contrastive Divergence).

$\widehat{P}$ is the input training distribution for the network

ϵ is a learning rate for the RBM training

ℓ is the number of layers to train

W^k is the weight matrix for level k, for k from 1 to ℓ

$\mathbf{b}^k$ is the visible units offset vector for RBM at level k, for k from 1 to ℓ

$\mathbf{c}^k$ is the hidden units offset vector for RBM at level k, for k from 1 to ℓ

`mean_field_computation` is a Boolean that is true iff training data at each additional level is obtained by a mean-field approximation instead of stochastic sampling

```
for $k = 1$ to $\ell$ do
  • initialize $W^k = 0$, $\mathbf{b}^k = 0$, $\mathbf{c}^k = 0$
  while not stopping criterion do
    • sample $\mathbf{h}^0 = \mathbf{x}$ from $\widehat{P}$
    for $i = 1$ to $k - 1$ do
      if mean_field_computation then
        • assign $\mathbf{h}^i_j$ to $Q(\mathbf{h}^i_j = 1|\mathbf{h}^{i-1})$, for all elements $j$ of $\mathbf{h}^i$
      else
        • sample $\mathbf{h}^i_j$ from $Q(\mathbf{h}^i_j|\mathbf{h}^{i-1})$, for all elements $j$ of $\mathbf{h}^i$
      end if
    end for
    • RBMupdate($\mathbf{h}^{k-1}, \epsilon, W^k, \mathbf{b}^k, \mathbf{c}^k$) {thus providing $Q(\mathbf{h}^k|\mathbf{h}^{k-1})$ for future use}
  end while
end for
```

Figure 8: DBN learning algorithm

Training the DBN is achieved by greedy learning that trains one RBM at a time and continues until the last RBM. The visible layer of RBM is a mere copy of the data vector that has (28x28 pixels=) 784 units and a bias. Besides, the visible layer has undirected connections with the hidden layer that has a default value of 500 units. In MATLAB, greedy learning algorithm for DBN is implemented by simply initializing parameters with previously obtained values. Then, it calls Contrastive Algorithm to train next RBM hidden layer.

In order to perform classification of hand-written images, another layer is added at the end of the last hidden layer. This layer is also connected to the last hidden layer with corresponding weights. For our implementation, there are 10 different classes (hand-written images of 0-9); hence, the last layer contains 10 units. Then, a fine-tuning algorithm can be applied to perform classification by obtaining the highest probability of one unit in the last layer. Then, the weights can be updated according to the feedback and improve the classification. Since the hidden units and weights in each hidden layer are initialized from the pre-training, the fine-tuning algorithm works decently. The proposed algorithm in our work is Backpropagation. The error derivative function in backpropagation updates weights for the future classification and prediction.

5 Results and Discussion

In this section, we will discuss about performances of different settings of DBN. To compare performances among several settings, we have trained DBN with various parameters and structures and computed the results of training and testing errors for each scenario. The training error is calculated during the learning process with 60,000 MINST hand-written digits training dataset and the testing error is obtained when the classification of 10,000 testing dataset is performed. In this work, we focus on the following parameters: the number of layers, the number of units in each hidden layer, the learning rates, the number of epochs in training RBMs, and the initialization of weights and biases. Different settings for each parameter are tested and the best parameter values for classification are discovered. Finally, we conclude with the best parameter settings and structures of DBN for classification of MNIST hand-written digits.

5.1 The number of layers

Currently, there is no absolute answer of how many hidden layers should be stacked for best results. It depends on the types and structures of datasets and it is no exception to our datasets. A few hidden layers can be trained in relatively short period of time, but result in poor performance as the system cannot fully store all the features of training datasets. Too many layers may result in over-fitting and slow learning time. Therefore, in order to examine the best case, three different settings for number of hidden layers are tested and the results are shown in the below graph.

As shown in the graph, more number of layers results in better performance. However, the over-fitting starts to happen in the three hidden layers as the testing error graph gradually increases over the number of epochs. Yet, DBN with three hidden layers performs the best among different settings.

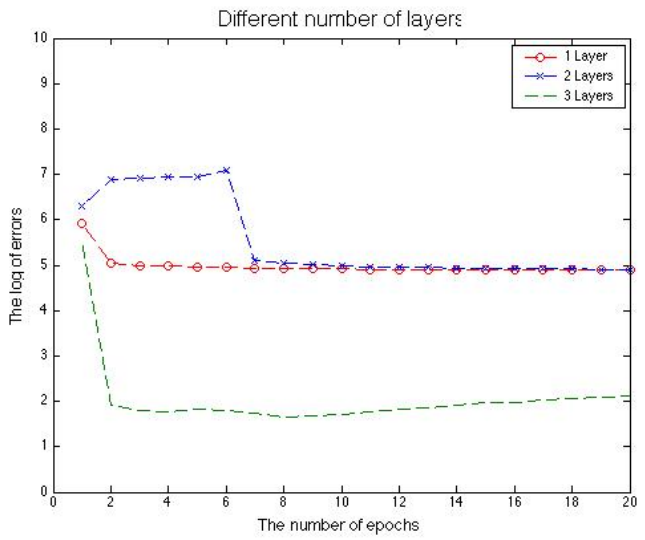


Figure 9: Error graph for different number of hidden layers

5.2 The number of hidden units

The number of hidden units in each layer corresponds to the features of input data stored in the system. Similar to the number of hidden layers, too little or too many hidden units result in slow learning and poor performance. The different number of hidden units of 3-layer DBNs, 250-250-1000, 500-500-2000 and 1000-1000-2000, are trained and the best number of hidden units in each hidden layer is acquired in the following graph.

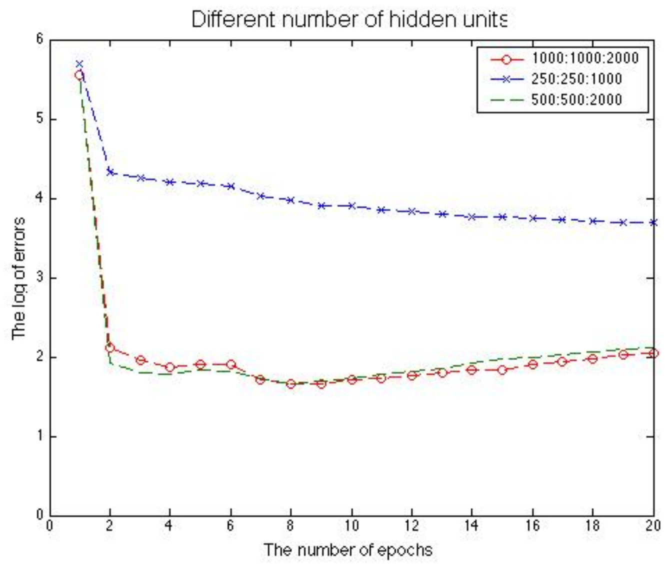


Figure 10: Error graph for different number of hidden units in each layer

As shown in the graph, 250-250-1000 DBN has the highest error and its performance does not improve much until 20 iterations of epochs. For 500-500-1000 DBN and 1000-1000-2000 DBN, their errors are almost identical to each other throughout the testing. However, it requires less time to train 500-500-2000 DBN since there are less hidden units than 1000-1000-2000 DBN. In our system, 500-500-2000 DBN takes approximately an hour faster than the other DBN. In conclusion, 500-500-2000 DBN is the best setting for the classification as it has faster training time and the least errors.

5.3 The learning rate

The learning rate determines how much to update the weights during the training. Having a large value for the learning rate makes the system to quickly learn, but it may not converge or result in poor performance. On the other hand, if the value of learning rate is too small, it is inefficient as it takes too much time to train the system. In DBN, there are three learning rates for weights, a bias for visibly layer, and a bias for hidden layer. We consider only the learning rate for weights in our experiment since it significantly influences the performance compared to other learning rates.

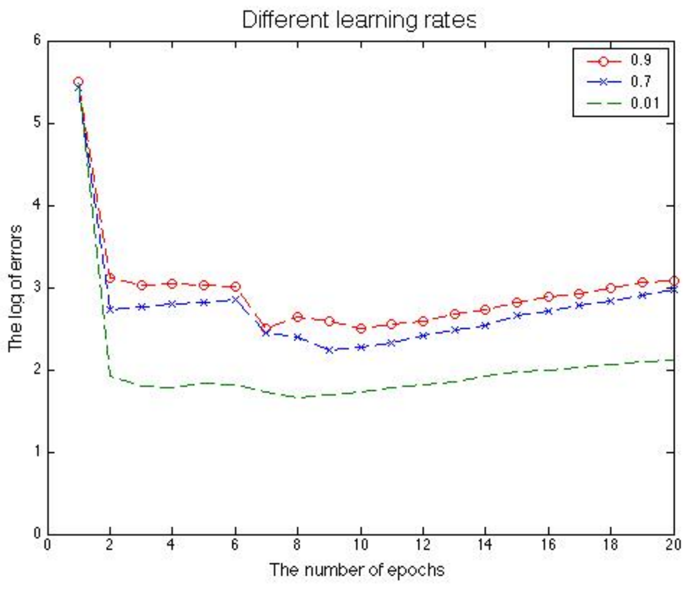


Figure 11: Error graph for different learning rates

The above graph shows errors for 0.01, 0.7, and 0.9 learning rates, which are applied to 500-500-2000 DBN. It clearly shows that the lowest value of learning rate has the least errors. Furthermore, we discover that the training times among those learning rates are not remarkably different in 500-500-2000 DBN, due to an efficient learning algorithm (Contrastive Divergence) for RBMs.

5.4 Number of epochs for training an RBM

Training a RBM with Contrastive Divergence algorithm requires a certain number of iterations to converge into an optimal value. A failure to obtain an optimal value for each RBM results in poor performance of overall system since RBM is a basic building block of DBN. Presumably, it appears running numerous iterations yields better results, but in fact, it does not only take a long time to train, but it also ends up over-fitting the data. As a result, it is important to stop before the over-fitting occurs.

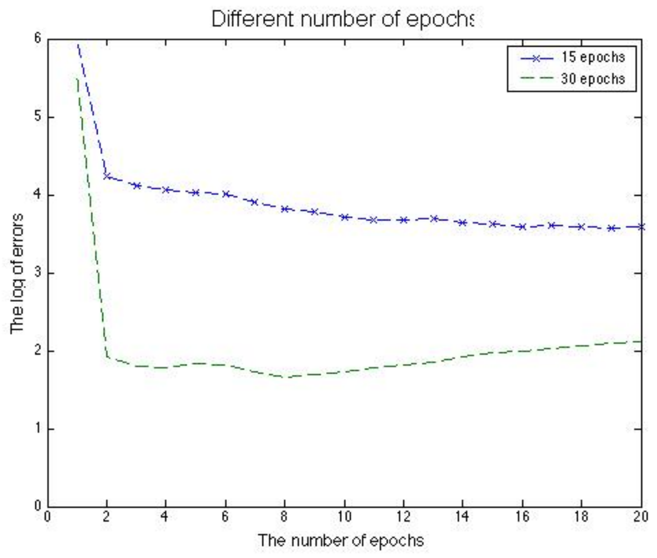


Figure 12: Error graph for different number of epochs

In the error graph, the DBN trained for 15 epochs continuously improves for the duration of 20 epochs; however, it clearly shows it has higher errors than the DBN trained for 30 epochs. The DBN trained for 30 epochs already has much better performance at 2 epochs. Nevertheless, the graph starts to gradually increase, which hints the sign of over-fitting. It concludes that 30 epochs is the best setting as it is just before the over-fitting of MNIST data and has the least errors.

5.5 Weights initialization

Weights initialization has been widely recognized as one of the most effective approaches in speeding up the training of a neural network. In fact, it influences not only the speed of convergence, but also the probability of convergence and the generalization [3]. Using too small or too large values could speed the learning, but at the same time, it may end up performing worse. In addition, the number of iterations of the training algorithm and the convergence time would vary depending on the initialized values. Three different initialization methods have been applied during the training of 30 epochs; small random values between 0 and 1, large random values between 1 and 10, and initializing with zeros. The following graph shows the different obtained results.

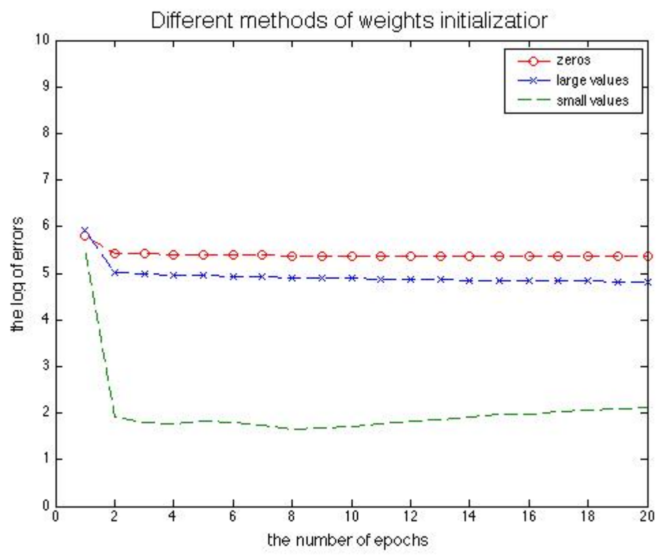


Figure 13: Error graph for different methods of weights initialization

Initializing the weights with small random values between 0 and 1 results the least error as shown in the graph. It is noticeable that the algorithm stuck in a poor local optimum using zeros weights. Starting with the same weights makes the neurons follow the exact same updating rule, and will always end up doing the same. Using large values helps the algorithm to converge faster, however, the results are not as good as using small initial values.

6 Conclusion

In this report, we implement the basic structure of DBN for MNIST hand-written digits classification and discover different behaviors of DBN under varying parameter settings. Due to RBM's unique structure, no connectivity between hidden units in each layer, and its efficient learning algorithm, Contrastive Divergence, all different DBNs in our experiment are trained in a reasonable time frame. Nonetheless, there is best-performed DBN structure for our specific classification task. Its structure is 500-500-2000 hidden units for each layer and small value for learning rates, 0.01. Furthermore, this DBN structure requires 30 epochs of training to acquire the lowest errors. Currently, this is the best result of MNIST hand-written digits classification among all other generalized machine learning.

Another unique characteristic of DBN is its scalability. [2] Depending on the data, DBN's parameters can be adjusted and trained, which results in many possibilities of implementations. It is already occurring in a wide range of domain, such as speech recognition, recommendation and text analysis. [8]

As we have seen different performances with varying parameter settings of DBN, it will be interesting to find out a method to automatically figure out its optimal parameter settings for the input datasets in our future work.

7 References

[1] Arel, I., Rose, D. C. and Karnowski, T. P. (2010).Deep Machine Learning – A New Frontier in Artificial Intelligence Research. Computational Intelligence Magazine, IEEE: Vol. 5, pp. 13-18.

[2] Bengio, Y. (2009). Learning Deep Architectures for AI, Foundations and Trends in Machine Learning: Vol. 2: No. 1, pp. 1-127.

[3] Fernández-Redondo, M., Hernández-Espinosa, C. (2001). Weight Initialization Methods for Multilayer Feedforward. European Symposium on Artificial Neural Networks.pp. 119-124.

[4] Hinton G. E. (2007). Learning multiple layers of representation. Trends in Cognitive Sciences: Vol. 11, No. 10, pp. 428-434.

[5] Hinton, G. E. (2007). To recognize shapes, first learn to generate images. Computational Neuroscience: Theoretical Insights into Brain Function. Elsevier.

[6] Hinton, G. E. (2010). A Practical Guide to Training Restricted Boltzmann Machines.Department of Computer Science; University of Toronto.

[7] Hinton, G. E., Osindero, S. and Teh, Y. (2006). A fast learning algorithm for deep belief nets.Neural Computation, 18.

[8] Mohamed, A., Dahl, G. and Hinton G. E. (2009). Deep Belief Networks for phone recognition. Department of Computer Science; University of Toronto.

[9] Geoffrey E. Hinton online page.www.cs.toronto.edu/~hinton

[10] Neural Networks for Machine Learning online course. Coursera online courses.www.coursera.org/course/neuralnets

[11] Deep Learning website. www.deeplearning.net

[12] The MNIST Database of handwriting digits. yann.lecun.com/exdb/mnist

[13] Pattern Recognition and Machine Learning (2006). Christopher M. Bishop