STA 571 - Deep Neural Network Classifier of Handwritten Digits

Mohammed Reza Sanatkar
reza.sanatkar@duke.edu
Mengke Lian
mengke.lian@duke.edu

Abstract

This course project aims at dealing with hand writing recognition task. Deep neural networks have garnered a lot of attention in the field of machine learning in the recent years. In this project, we implement a multi-layer neural networks to perform the task of handwritten digits. Both neuron network with back propagation (BP) and deep belief network with restricted Boltzmann machine (RBM) are trained in MNIST database. For BP, the error rate on test set is 3.75%, while for RBM the error rate is XXX.

1 Introduction

This course project aims at dealing with hand writing recognition task. Deep neural networks have garnered a lot of attention in the field of machine learning in the recent years. In this project, we implement a multi-layer neural networks to perform the task of handwritten digits. Our project can be divided in the following main steps:

- Implementation of a general class of multi-layer neural networks in C++.
- Implementation of feed-forward and back-propagation algorithm to train such networks as discriminative classifiers.
- Implementation of learning algorithm for RBM to train neural networks as generative classifiers.

2 MNIST Database

The MNIST database of handwritten digits contains a training set of 60,000 examples, and a test set of 10,000 examples.

Each image is transformed from bilevel (black and white) image from NIST database by renormalizing to fit in a 20×20 pixel box while preserving aspect ration, which result into a 20×20 grey-scaled image. And then this 20×20 grey-scaled image is centered in a 28×28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28×28 field.

The database consists of:

- 60,000 training images of handwritten digits as well as their labels
- 10,000 images of handwritten digits as test data

3 Back-Propagation

The goal of any supervised learning algorithm is to find a function that best maps a set of inputs to its correct output. For digit classification, we use the following structure of the neuron network: the first layer is the input, which contains $28 \times 28 = 768$ nodes; the second layer is a hidden layer with 300 hidden nodes; the last layer is the output layer which has 10 nodes corresponding ten digits respectively. Given an image whose digit is i, the desired output t_j for the neuron network is

$$t_i = 1 \text{ and } t_j = -1, \forall j \in \{0, \ldots, 9\}, j \neq i$$

And denote the true output of neuron network as o_j, then the mean-square error is the function we want to minimize:

$$J = \frac{1}{2} \sum_{j=0}^{9} (t_j - o_j)^2$$

Assume we have a neuron network with L layers except input layer (we regard input layer as the 0-th layer), and for each layer there are n_i nodes for $0 \leq i \leq L$, the weight from j-th node in $(i-1)$-th layer to k-th node in i-th layer is denoted as w_{jk}^i for $1 \leq j \leq n_{i-1}, 1 \leq k \leq n_i, 1 \leq i \leq L$. Let $\alpha_j^i(x)$ and $\beta_j^i(x)$ be the activation and output for hidden node j in layer i of the network, and σ be the transfer function at each node:

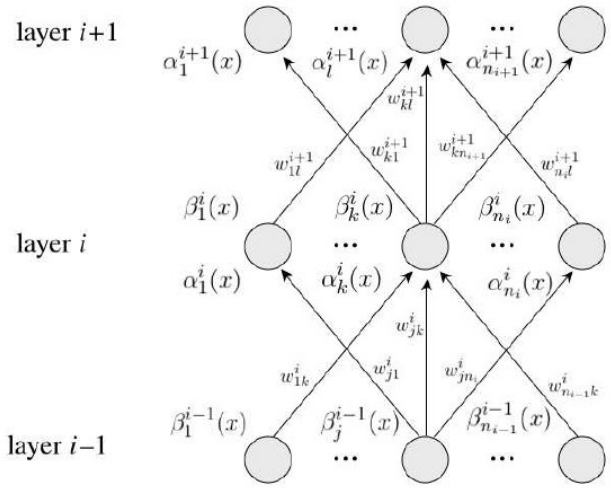

Figure 1: values transferred in back propagation

We have that $o_k = \beta_k^L(x) = \alpha_k^L(x), \beta_k^0(x) = x_k$, and for $1 \leq i \leq L-1$

$$\beta_k^i(x) = \sigma(\alpha_k^i(x)), \quad \alpha_k^i(x) = \sum_{j=1}^{n_i} w_{jk}^i \beta_j^{i-1}(x)$$

$$\frac{\partial J}{\partial w_{jk}^i} = \frac{\partial J}{\partial a_k^i} \frac{\partial a_k^i}{\partial w_{jk}^i} = -\delta_k^i \beta_j^{i-1}(x), \text{ where } \delta_k^i \equiv -\frac{\partial J}{\partial a_k^i}$$

Given the learning rate ℓ, we can determine the update rule for weight w_{jk}^i as

$$\Delta w_{jk}^i = -\ell \frac{\partial J}{\partial a_k^i} \beta_j^{i-1}(x)$$

To determine δ_k^i one uses the chain rule: $\delta_k^L = -(t_k - o_k), \forall 1 \leq k \leq n_L$

$$\delta_k^i = -\frac{\partial J}{\partial a_k^i} = -\sum_{l=1}^{n_{i+1}} \frac{\partial J}{\partial a_l^{i+1}} \frac{\partial \alpha_l^{i+1}}{\partial a_k^i} = \sum_{l=1}^{n_{i+1}} \delta_l^{i+1} \frac{\partial \alpha_l^{i+1}}{\partial a_k^i}, \forall 1 \leq k \leq n_i, 1 \leq i \leq L-1$$

while

$$\frac{\partial \alpha_l^{i+1}}{\partial a_k^i} = \frac{\partial \alpha_l^{i+1}}{\partial \beta_k^i}\frac{\partial \beta_k^i}{\partial a_k^i} = w_{kl}^{i+1}\sigma'(\alpha_k^i(x)), \forall 1 \leq k \leq n_i, 1 \leq l \leq n_{i+1}, 1 \leq i \leq L-1$$

This results in the back propagation equation:

$$\delta_k^i = \sigma'(\alpha_k^i(x)) \sum_{l=1}^{n_{i+1}} w_{kl}^{i+1}\delta_l^{i+1}, \forall 1 \leq k \leq n_i, 1 \leq i \leq L-1$$

Hence, δ_k^i is determined by the deltas δ_l^{i+1} of the next layer. A single presentation of the entire training set is called an epoch. The amount of training is measured by the number of epochs.

Back propagation training algorithm

Input: training samples $\{(x_n, y_n)\}_{i=n}^N$, learning rate ℓ, number of epoches T
Output: trained neuron network with weights w_{jk}^i
Initialize all weights w_{jk}^i uniformly with numbers from $[-1, 1]$;
for $t = 1, \ldots, T$ **do**
  Randomly permute all training samples to change the training order in current epoch;
  for $i = 1, \ldots, N$ **do**
    Feed forward: compute activations $\alpha_k^i(x_n)$ and outputs $\beta_k^i(x_n)$ for all nodes;
    Back propagation: compute δ_k^i recursively with $\delta_k^L = -(t_k - o_k), \delta_k^i = \sum_{l=1}^{n_{i+1}} \delta_l^{i+1} \frac{\partial \alpha_l^{i+1}}{\partial a_k^i}$;
    Update Weights: $w_{jk}^i = w_{jk}^i + \Delta w_{jk}^i = w_{jk}^i - \ell \delta_k^i \beta_j^{i-1}(x)$;
  end
end

4 Restrict Boltzmann Machine

Multi-layer neural networks represent the consistent effort of human beings to mimic the brain. Below, you see the neural networks which is used in this project. This network consists of three hidden layers.

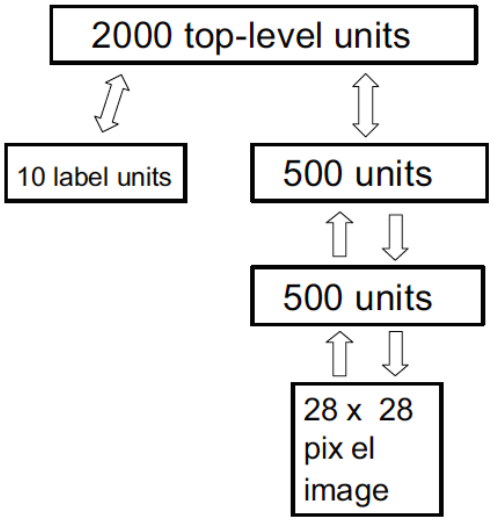

Figure 2: Neural Network implemented as a classifier of handwritten digits [1]

Deep belief networks outperform the traditional neural networks because of taking benefits from RBMs. What is a RBM? RBM is a bipartite undirected inference graph which is used to learn

a multivariate joint distribution satisfying the conditional independence imposed by the bipartite structure of the graph. In deep neural networks, every pair of successive layers is considered as a RBM. The first and most important step of training a deep neural network is to train these RBMs.

First, we train the RBM corresponding to the input layer and the first hidden layer. After learning the generative model for the first RBM, we use this set of learned weights to generate inputs for the second RBM. Having inputs for the second RBM, the set of weights of edges entering the nodes in the second hidden layer are learnt. These steps are repeated for all the other RBMS too.

How to learn a RBM in a reasonable time : Contrastive Divergence Learning Procedure

$$\Delta w_{ij} = \epsilon(< v_i h_j >^0 - < v_i h_j^1)$$

- Start with available data on the visible units
- Compute the hidden nodes in parallel
- Compute the visible units (Reconstruction)
- Update the hidden nodes again

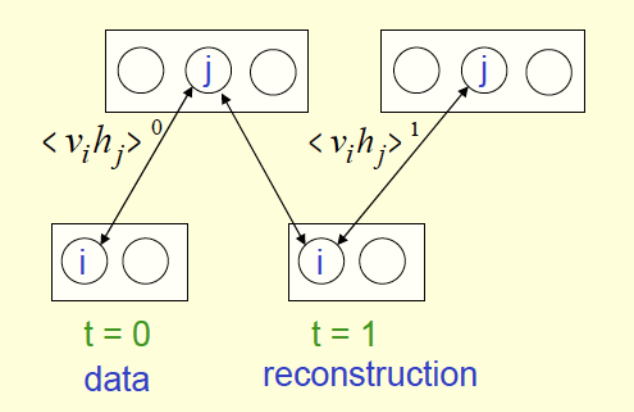

Figure 3: Contrastive Divergence Learning[1]

5 Result

For back propagation method, our model is constructed as following: the first layer is the input, which contains $28 \times 28 = 768$ nodes; the second layer is a hidden layer with 300 hidden nodes; the last layer is the output layer which has 10 nodes corresponding ten digits respectively.

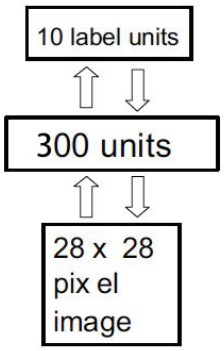

Figure 4: neuron network trained by back propagation used for handwritten digits classification

Also, we chose the learning rate $\ell = 0.02$ and maximum iteration number $T = 100$. Under such setup, the error rate on training set is 0.74% after 100 iteration, while the error rate on testing set if 3.75%. The figure below show the trend of error rate changing with the epoches.

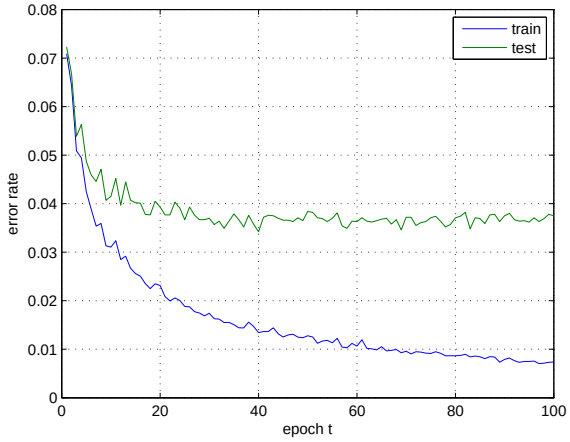

Figure 5: error rate vs. epoch on both training set and testing set

6 Conclusion

For back propagation, each update is guaranteed to lower the individual MSE for the pattern, but it could increase the total error. But in the long run, the total error decreases. However, training a neuron network with a too large epoch number can be dangerous since this may cause overfitting. This phenomenon can be seen from Figure 5, the error rate on training set is decreasing in the long run, but the error rate on testing set stops decreasing after about 40 epoches.

References

[1] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006.

[2] Yann LeCun. Deep learning tutorial. *ICML, Atlanta*, 2013.

[3] Shashi Sathyanarayana. A gentle introduction to backpropagation. 2014.