A Study of Greedy Layer-wise Training on Deep Neural Networks

Tianqi Gao  
Department of Electrical and Computer Engineering, University of Illinois Urbana Champaign, IL 61801, USA  
gao19@illinois.edu

Abstract—Neural Networks are popular a classification tool because of its great representation power. However, training a neural network is not an easy task. Straight forward training method like Error Backward Propagation algorithm does not work very well in many real life applications, especially for networks with a large number of weight parameters. Greedy layer-wise training method which elegantly combines unsupervised training with supervised learning proves to be very handful to speed up the process of convergence when training such a large networks. In the project, the layer-wise training method and Error Backward Propagation algorithm are both implemented for a pattern recognition application. The two training methods perform similarly in terms of training and testing errors; but, layer-wise training present much faster convergence speed.

Keywords—pattern recognition, image processing, neural networks, unsupervised training, supervised training, error backward propagation.

I. INTRODUCTION

Neural Networks are very popular in many pattern recognition applications, such as computer visions, text mining, image or audio signal processing, because it exhibits a great representation power. To realize desired performance in practice, training a neural network is critical; and often the task of training can be difficult. In many real-life applications, a neural network can consist of at least many thousands of weight parameters, and the error function may present local optimal points. Efficiently training a large network to a position which is close to the global optimal is essential for the usage of neural networks.

Error Backward Propagation is a commonly used technique in the training of a network. Backward propagation is based on simple gradient descent algorithm and the chain rule. It starts with random initialization of the parameters and iterates to a finite number of times with updating the parameters according to the derivative of the loss function during every iteration process. Given enough time, it will eventually converge and reach to a local optimal or the global optimal based on the starting point. Error Backward Propagation training is usually conducted multiple times with different initializations in the hope of arriving at the global optimal or a better local optimal. In the case where the error function has many local optimal points, the process of repetition could be tedious and less fruitful. Besides, some loss functions could exhibit a flat area where the gradient of the loss function is very small and thus training could be extremely slow.

For a large network with multiple layers and a great number of parameters, such as convolutional neural network (CNN) or deep belief neural network (DBN), backward propagation can be very slow. A greedy layer-wise training algorithm was proposed to train a DBN [1]. The proposed algorithm conducts unsupervised training on each layer of the network using the output on the $k_{th}$ layer as the inputs to the $k+1_{th}$ layer. Fine-tuning of the parameters is applied at the last with the respect to a supervised training criterion.

This project aims to examine the greedy layer-wise training algorithm on large neural networks and compare the performance with Error Backward Propagation; the comparison would focus on convergence speed of the two algorithm. The rest of the report is organized as follows, Section 2 briefly reviews Error Backward Propagation; Section 3 explains greedy layer-wise training method in details; Section 4 presents the setting for experiments and the results; Section 5 concludes the report.

II. A REVIEW OF ERROR BACKWARD PROPAGATION

First, the notations used in the algorithm are defined as:

$l$: the index for layers  
$W_l$: the weight matrix at $l_{th}$ layer  
$b_l$: the bias at $l_{th}$ layer.  
$x_l$: output nodes at $l_{th}$ layer.  
$a_l$: linear output at $l_{th}$ layer.  
$g$: nonlinear function (hyperbolic tangent).  
$L$: mean square loss.  
$\eta$: learning rate.

The algorithm takes four steps in total [2]:  

1. Initialize the weight matrix $W_l$ and bias $b_l$ for each layer.
2. Compute the output $a_l$ for each layer using the linear output $x_l$.
3. Compute the mean square loss $L$.
4. Update the weight matrix and bias using the chain rule.

The process is repeated until the convergence criterion is satisfied.
1. Forward Propagation
   For each layer, do the following:
   \[ a_{l+1}^{(n)} = b_{l+1} + W_{l+1} x_{l}^{(n)} \]
   \[ x_{l+1} = g_{l+1}(a_{l+1}^{(n)}) \]

2. Error Backward Propagation
   For each layer, do the following:
   \[ \delta_l^{(n)} = \frac{\partial}{{\partial a_l^{(n)}}} \sum_{y} \exp(-\ell^{(n)}) \]

3. Gradient Calculation
   For each layer, do the following:
   \[ \nabla_{W_l} \mathcal{L} = \sum_n \delta_l^{(n)} x_l^{(n)T} \]
   \[ \nabla_{b_l} \mathcal{L} = \sum_n \delta_l^{(n)} \]

4. Gradient Decent
   For each layer, do the following:
   \[ W_l \leftarrow W_l + \eta \nabla_{W_l} \mathcal{L} \]
   \[ b_l \leftarrow b_l + \eta \nabla_{b_l} \mathcal{L} \]

The above equations are in the matrix form.

III. GREEDY LAYER-WISE TRAINING METHOD OF NEURAL NETWORKS

A very efficient training method for neural networks was developed in [1]. The idea is to combine fast and efficient layer-wise unsupervised trainings with supervised trainings on the entire network. The unsupervised trainings are conducted first and are from bottom to top to serve as pre-training; while the supervised trainings fine tunes the network from top to bottom. The pre-trainings treat each layer of the network as a Restricted Boltzmann Machine (RBM) and then train the RBMs using Contrastive Divergence (CD) algorithm; the supervised fine tuning leverages the backward propagation algorithm. This section focuses on the implementation of the pre-training: the first part reviews RBMs and explains the setting of using RBMs as building blocks, and the second part describes CD algorithm.

A. A Review of RBM and the Pre-training Setting

A Boltzmann Machine is a powerful graphical energy-based model to represent complicated distributions. An energy-based model defines probability distributions through an energy function \( p(x) = \frac{e^{-E(x)}}{Z} \). \( E(x) \) is the energy function which depends on the state of the nodes in a Boltzmann Machine; \( Z = \sum_x e^{-E(x)} \) is the partition function which takes the summation of all possible configurations, and it is used as a normalized factor. A RBM is a variant form of Boltzmann Machine with restrictions on nodes connections, as shown in Fig.1. It contains visible nodes \( v \) and hidden nodes \( h \) and the connections of visible-to-visible as well as hidden-to-hidden are forbidden. In another word, visible nodes or hidden nodes are conditionally independent. The energy function of a RBM takes the form of \( E(h,v) = -Wv - Wh - \sum_{i,j} w_{ij} v_i h_j \), where \( W \) represents the weight matrix connecting visible and hidden nodes. The joint distribution of hidden and visible nodes is expressed as \( p(h,v) = \frac{e^{-E(h,v)}}{Z} \), and \( Z = \sum_v e^{-E(h,v)} \); the probability of the network assigned to a visible node vector is \( p(v) = \frac{1}{Z} \sum_h e^{-E(h,v)} \).

During the process of pre-training, each layer of the network is treated as a RBM. The input nodes to that layer are the visible nodes and the output nodes are hidden nodes. In general, the nodes can be binary numbers of real numbers; and, under the case of real numbers, the nodes can take the values of conditional probabilities, which are expressed as \( p(h = 1|v) = \sigma(\sum_i v_i w_{ij}) \) and \( p(v = 1|h) = \sigma(\sum_j h_j w_{ij}) \). The process starts at the bottom layer of the network and goes up layer by layer; every layer in the middle takes the hidden nodes of the previous RBM as the visible nodes.

B. Training a RBM with Contrastive Divergence Algorithm

The purpose of pre-training is to generate a better representation of the training inputs of the network through layer by layer training. To train a RBM with a training set of visible nodes, the probability of the training visible nodes needs to be maximized by adjusting the weights and such probability is \( P(V) = \frac{1}{Z} \sum_v p(v) \). Increasing the probability is equivalent to lower the energy function of each training node. [4] derives the derivative of the log probability of a training token with respect to the weights is \( \frac{\partial \log p(v)}{\partial w_{ij}} = E_{data}[v_j h_i] - E_{model}[v_j h_i] \) where the first term is the
empirical mean and the second the true mean of the modeled distribution. The derivatives then are used with a learning rate as weights updates. Another perspective of training a RBM, as we saw in the lecture, is to minimize the distance between the visible nodes distribution and the hidden nodes distribution, $D(V \| B) = H(V \| B) - H(V)$, would result in the same method of performing gradient ascend. As it is can be easily seen that in ideal optimal scenario, the derivative reaches 0 as the distributions of visible and hidden nodes converge.

The updating rule can be closely approximated by contrastive divergence, which is defined as the difference between two Kullback-Liebler divergences [1]. The first term in the learning equation is an empirical expectation which can be easily computed with initial visible nodes and the hidden nodes generated through randomly initialized weights. CD algorithm calculates the second term of the log probability derivative through Gibbs sampling which is implemented by repeated constructions of hidden and visible nodes, as shown in Fig.2. The algorithm uses the training visible nodes as the initial samples and reconstructs the hidden nodes and visible nodes according to the probabilities of $p(h = 1 | v) = \sigma (\sum_i v_i w_{ij})$ and $p(v = 1 | h) = \sigma (\sum_j h_j w_{ij})$ . As mentioned in the previous section, the visible nodes are conditionally independent given the hidden nodes and the hidden nodes are conditionally independent given the visible nodes, which enables the Gibbs sampling step. The CD algorithm does not wait Gibbs sampling to converge, and one step of the sampling is enough to work very well, which is stated in [3] and also supported in the results shown in the next section. The weights are updated after the Gibbs sampling and the updated weights are then used for reconstruction nodes in the next round of sampling until the weights converge. In this project, the visible nodes are real values and the hidden nodes are binary values except the last reconstruction, as suggest in [4].

1. Randomly initialize the weights in the Network.
2. Train the weights in the first layer RBM by using unsupervised CD algorithm and taking training data as visible nodes.
3. Iterate Step 2 through the network until the last layer.
4. Fine tuning the weights by performing a supervised training on the network through Error Backward Propagation algorithm with training labels.

IV. EXPERIMENTS SET UP AND RESULTS

In this section, we first describe the experiments set up, including the application chosen to be experimented on, the network structure, and the implementation details of Error Backward Propagation algorithm as well as the Greedy Layer-Wise Training. Then results are presented and discussed at the end.

A. Experiments Setup

The application chosen is to classify digits from 0 to 9 by processing images of handwritten digits. The data used in the experiments are selected from MINST database; and the images are 28 by 28 grey-scale images with digits at the center. Some examples of the images are shown in Fig.3. For the sake of simplicity, binary classifications are considered and are conducted on pairs of digits. For each experiment, 1000 training images are randomly selected to form a training set, and another 1000 testing images are also randomly selected for a testing set; both sets contain approximately 500 of each digit chosen in the pair.

![Figure 3 Examples of the dataset.](image)

A 5-layer neural network is used as a classifier which contains an input layer with 28*28 nodes, an output layer with 1 node, and 3 layers in the middle with 392, 98, 48 hidden nodes. The numbers of nodes for each layer are decided by the principle that each layer should contain fewer nodes than the previous layer. Hyperbolic tangent functions are used at the output of each layer, and a slicer with threshold 0 is used at the final output to decide the binary class 1 or -1. All the weights are randomly initialized from 0 to 1. Batch training with all
the training tokens in the set are used for both training method, and mean square error is used as a loss function to be minimized.

For Error Backward Propagation training, learning rate is fixed at $1 \times 10^{-10}$, and stop criteria is either 1000 iterations are finished or the change of the loss function is below some threshold. For layer-wise pre-training, to update weights once, only one step of Gibbs sampling is performed and the weights updating stops until either they converge or iterations of a fixed number is finished. The learning rate for weights updating is set as one half of the rate in Error Backward Propagation. Fine tuning of the entire network is performed as Error Backward Propagation with the setting described before.

B. Results

Fig.4. plots the Loss function versus iterations for training conducted on digits 0 and 1. The blue curve corresponds to Error Backward Propagation, and red one fine tuning after pre-training. From the plot, it is clear that both training methods can eventually drive the loss function to a global optimal. However, the error surface was very flat at the region of the initial weights, and it took the Backward Propagation algorithm quite a long time to move out that flat region then starts to lower the loss at a fast pace. On the other hand, the pre-training was able to move the weights out of the flat error region and allowed the fine tuning to help the loss arrive at the global optimal much quicker.

Figure 4 Loss function versus Iteration.

More curves of the loss functions versus iterations are plotted in Fig.5. They are corresponding to classifications on different digit pairs and different initializations of the weights. Some classification tasks are more difficult than others by nature, for example, classify between 1 and 7 should be harder than classify between 0 and 1; and harder tasks tended to need longer training time. Besides, the random initializations of the weights vectors also contribute to the randomness of convergence time. On average of all the experiments conducted, which is about 100 different trials, pre-training can save about 500 iteration times of Backward

Propagation.

Figure 5 Loss functions versus Iterations for all experiments.

Pre-training enables fast fine tuning and save convergence time while itself takes time to converge. During one round of the pre-training, every node in the network has to be reconstructed once and the reconstruction in total takes approximately twice the amount of computations as forward propagation in the Backward Propagation algorithm; thus, an iteration of pre-training takes fewer computations than an iteration of backward propagation. However, just like CD algorithm does not need to wait for Gibbs sampling to converge, the pre-training too does not have to keep updating the weights until them converge.

An experiment on the relationship of the number of iterations for pre-training and the number of iterations for fine tuning is conducted and the results are plotted in Fig.6. The curves are the loss functions versus iterations for the fine tuning after $x$ iterations of pre-training, while other experiment setting are fixed including the initialization; and $x$ took the values of 1, 25, 50, 75 and 100, as indicated on each of the curves. From the curves, a few interesting observations can be made. First, even though only one time iteration is performed, it could significantly improve the convergence speed of the fine-tuning. Second, the more time spent on pre-training, the more time can be saved for fine-tuning, and however the absolute saving on iteration times decreases.

Figure 6 Loss function convergence comparision.
It is evident that the pre-training helps to improve the learning process from the results; but, it not necessary improves the classification behavior. As indicated in Fig.7., the loss functions are not necessary reduced during the process of pre-training; rather, random fluctuations or idle behaviors are observed. The non-reducing error rates are consistent with the fact that pre-training is unsupervised learning and it involves no information on the labels of training images. But, the pre-training indeed lowers the energy of each layer and increases the log probability of each RBM.

The last plot shows the classification performance of different tasks and different initialization. The red points are for Backward Propagation training, and the blue for greedy layer-wise training; the cross ‘x’ is for training, and the circle ‘o’ is for testing. Clearly, a for a lot cases the classifier was over-trained with almost 0 error rate, which is also indicated as the loss functions are very close to 0. Classifications on testing set shows that greedy layer-wise training mostly can perform as good as Backward Propagation without pre-training; while, in some cases, the latter outperforms the former quite obviously. Either training method performs pretty well for the application chosen.

V. CONCLUSION AND FUTURE WORK

In this project, a layer-wise training which consists pre-training and fine tuning and Error Backward Propagation training are both implemented and experimented for a handwritten digits classification application. From the comparison of the convergence of loss functions, layer-wise training method proves to converge much faster due to the help of unsupervised pre-training. Pre-training allows the randomly initialized weight parameters to move to the “edge” of a global optimal, and then enables a fast drop of the loss function during the process of Backward Propagation fine tuning.

REFERENCES