CSE567

Programming Assignment #1

Handwritten Digits Classification Using Neural Networks

Date: March 4th 2016

Group #32

Gian Pietro Farina (50176742)

Meghana Ananth Gad (50182335)

Ashwin Mittal (50168798)

**Abstract —** The MNIST database of handwritten digits is a standard touchstone of effective image classification algorithms. It is extensively studied and tested by many machine learning techniques. This project aims at implementing Neural Network classification algorithm to recognize 28x28 grayscale handwritten digit image and mark them as one of the digits between 0 and 9.

INTRODUCTION1

IN this project we aim to build a classification model using single hidden layer Neural Network to predict the output for the new incoming data in a batch mode.

I. THE DATASET AND FEATURES

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

The original black and white (bi-level) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. The images were centered in a 28x28 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 28x28 field. This is a relatively simple database for people who want to try machine learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

II. DATA PREPROCESSING

For the dataset, the images are read as 28 \* 28 pixel data that are transformed into image sample vector of size size 784, where each element holds a value between 0 and 255 for different shades to grey. These values are normalized so that they are between 0 and 1.

The training dataset has 60,000 images and it is being used for training the models and validating it. We randomly permute to set and split it into two sets of 50,000 and 10,000; the former set for training while latter for validation.

Further we apply Feature Selection to the training in which we remove the features which have same value for the entire dataset and hence no learning is possible.

III. THE NEURAL NETWORK

A Neural network consists of a hidden layer of *perceptrons* with sigmoid activation functions. There are three different layer types in the neural network: input, output, and hidden layer. The role of the input layer is simply to take in sets of features (one set of feature for each example) and pass them forward to the hidden layer. Each node in a hidden layer takes inputs from every node in the previous layer, sums the inputs up, applies a sigmoid activation function, and passes the result on to every node in the next layer, i.e. the output layer. The input layer is fully connected to the first hidden layer i.e., every node in the input layer passes its information on to every node in the hidden layer and the hidden layer is fully connected to the output layer.

The Neural network learns by iteratively updating the weights according to a back propagation algorithm with stochastic gradient descent. This algorithm first passes a single example, subset of examples, or the full set of examples through the network. In general, using larger sets of examples, called batches or mini-batches, will cause the algorithm to converge more smoothly and directly to a local minimum, while smaller batches will require less computational power but cause the algorithm to converge less smoothly or directly toward a local minimum. In our implementation we used only one example per iteration in order to reduce the computational strain on our CPU. After finishing the forward pass, our code then utilizes the sum-squared error equation:

***J(W) = 1/2(y − o)^2***

where *o* is the observed output of our network, *W* is our weight matrix, and *y* is the actual label. From this equation, we can derive the equations for δ (gradient). We can use the derived formula to calculate the δ’s for each node in the output layer:

***δout ← o(1 − o)(y − o)***

Then we can calculate the δ for each of the hidden layer nodes:

***δ(i) ← Oi(1 − Oi)wi+1,δi+1***

where wi+1, is the weight going from node i to node i+1 and Oiis the output from node i. As we can see, the δ for each node in the hidden layer depends on the δ’s in the next layer and the weight of the edges leaving that node. In other words, this equation propagates the error backwards through the network. Once we have calculated the δ’s, we can use them to update each of the weights in the network according to the following equation:

***wi+1,i ← wi+1,i + αi+1,iδi+1xi+1,i***

Where α is some predefined learning rate between 0.0 and 1.0. Notice that this means that weight updates in layers closer to the input layer will be typically smaller in magnitude than weight updates in layers closer to the output layer. As such, the weight updates are usually smaller and more diffuse at earlier layers in the network. In our FFNN implementation, the above

calculations and updates are performed for each example in our training set (i.e., every example in each epoch). At the end of each epoch, the algorithm calculates the error of the model on a validation set. If the model has done substantially better (as defined by one of our hyper-parameters) than its performance in the previous epoch, the algorithm will run until it no longer improves substantially with each epoch.

IV. HYPER-PARAMETER SELECTION

We ran experiments to find satisfactory values for the number of nodes (m) in the hidden layer and the normalization value (λ) using the grid search technique. For the number for nodes varying from 4 to 20 at intervals of 4, we calculated the accuracy on Test data by setting values of λ between 0 and 1 with step size 0.1. This experiment took around 18 hours. Although it would have taken weeks or months to search the entire hyper-parameter space, and it was possible to find relatively satisfactory hyper-parameters after only a few days of training, hence we did not perform k-cross validation.

The results of the experiments are shown in the charts below, where m stands for number of hidden nodes in the network and λ is regularization co-efficient:

We made the following observations from the above plots:

* As the number of hidden nodes were increased the training, validation and testing accuracies increased for all the tests for different values of λ.
* We also noticed that for cases where number of hidden nodes(m) were set to 4,8,12 best accuracies for test, training and validation sets were noted for higher values of λ(greater than 0.5).
* When the number of hidden nodes were set 16 and 20 the best accuracies were observed for smaller values of λ(less than 0.2).

We got the best accuracy for λ = 0.1 with 20 nodes. So we fixed λ to 0.1 and compared the accuracies obtained when the number of hidden nodes were set to 4, 8, 12, 16 and 20. The comparison for λ = 0.1 for all nodes is represented in following graph:

From the above graph we notice that the best accuracies for training, validation and test are obtained when the number of hidden nodes takes either of the following values 12, 16 and 20. Also for all values ‘m’(number of hidden nodes) greater than 12 the training, validation and test accuracies do not improve by a significant value.

V. CONCLUSION

We observed that the number of hidden nodes in a neural network impact the accuracy of the network. If the number of hidden nodes are set to a very small value (m=4 in our experiment) the performance of the network is not optimal. However, increasing the number of hidden nodes after a certain value does not reflect in significant increase in performance of the network. The value of generalization co-efficient for which we get optimal results varies depending on the number of hidden nodes in the network.

We conclude that for optimal results the number of hidden nodes should be set to a value greater than the number of output nodes in the neural network. Also, the hyper-parameters ‘m’ and ‘λ’ are inversely dependent on each other. When the value of

hidden nodes increases the best accuracies are observed for smaller values of

‘λ’. Hence, smaller values of ‘λ’ must be chosen when the number of hidden nodes are greater than the number of output nodes.

Please note:

* Random initial weights have been computed at the beginning, once and for all. So all the iterations start from the same initial random weights. This should not affect the results much because with enough data we usually converge independently by the initial starting point.
* We conducted the random permutation of the data only once during the pre processing stage and used the same data for all experiments. The final results are highly dependent of this permutation and we might have obtained different results if the permutation were different.

VI. REFERENCES

[1] Andrew Mass’s blog on Neural Network

[2] David Stutz’s blog on Neural Network

[3] Wikipedia on Neural Network

[4] Wikipedia on Backpropagation