Learning to Rank using Linear Regression

CSE-574- Fall 2015

Shaleen Mathur (50170060)

# Introduction

In this project we are implementing and evaluating the different classification algorithms. The objective is to learn how to different classification schemes on a real world data like MNIST. Doing so we want to compare the different available schemes of classification by using them on same data.

# Problem Statement

There are three task in this project

1. Implementation and training of Logistic Regression model on MNIST digital images database. Tune hyper parameters to minimize error.
2. Implementation of single hidden layer neural network, and training it on the MNIST digit images. Tune hyper parameters such as the number of units in the hidden layer
3. Using CNN- Convolutional Neural Network class by [Mihail Sirotenko](http://www.mathworks.com/matlabcentral/profile/authors/977971-mihail-sirotenko). This project provides matlab class for implementation of convolutional neural networks.

# Data Set

In this project, for the training of our classifiers, we will use the MNIST dataset. The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. It is a subset of a larger set available from NIST. This database is available at <http://yann.lecun.com/exdb/mnist/> .

The original black and white (bi-level) images from MNIST 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.

To extract the data from MNIST we used the script prepared by UFLDL, in which they extracted the image in form of matrix and then converted it to a vector by reshaping it(<http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset)>.

# Classification Algorithms

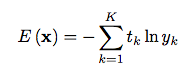
# **Logistic Regression**

It is a Probabilistic Discriminative Model of classification.

We used 1-of-K coding scheme t = [t1, ..., tK ]. for our multiclass classification task. Our multiclass logistic regression model could be represented in the form:

../../../../../../Desktop/Screen%20Shot%202015-12-08%20at%203.3

where the activation ak are given by ak = wkTx + bk. The cross-entropy error function for multiclass classification problem seeing a training sample x would be



where yk = yk (x). The gradient of the error function would be

../../../../../../Desktop/Screen%20Shot%202015-12-08%20at%203.3

You can then use stochastic gradient descent which uses first order derivatives to update

../../../../../../Desktop/Screen%20Shot%202015-12-08%20at%203.4

to find the optimum of the error function and find the solution for wj.

# **Implementation of Logistic Regression**

To implement logistic regression, we wrote a matlab code. Our implementation on abstract scale is gradient descent method. We take each input and train the system, by shifting weights in direction of minimizing errors. Hyperparameters are carefully chosen and then varied to get minimum error.

Following is my step by step tuning of hyperparameters and other programming techniques:

**Step 1**: Constant bias

The bias term in above mentioned equations *bk* is called bias term. It is the value of output if all predictors (or input X) is zero. So, clearly the choice of *bk* will affect performance of the system. We are expecting the weights to be between -1 to +1(very rough estimate).

But the system response was not satisfactory and error was around 20%.

|  |  |
| --- | --- |
| *bk* | Missclassification Rate |
| 1 | 24% |
| 0.5 | 22% |
| **0.1** | **18%** |
| 0.01 | 25% |

So we can say that best performance was at *bk* = 18%.

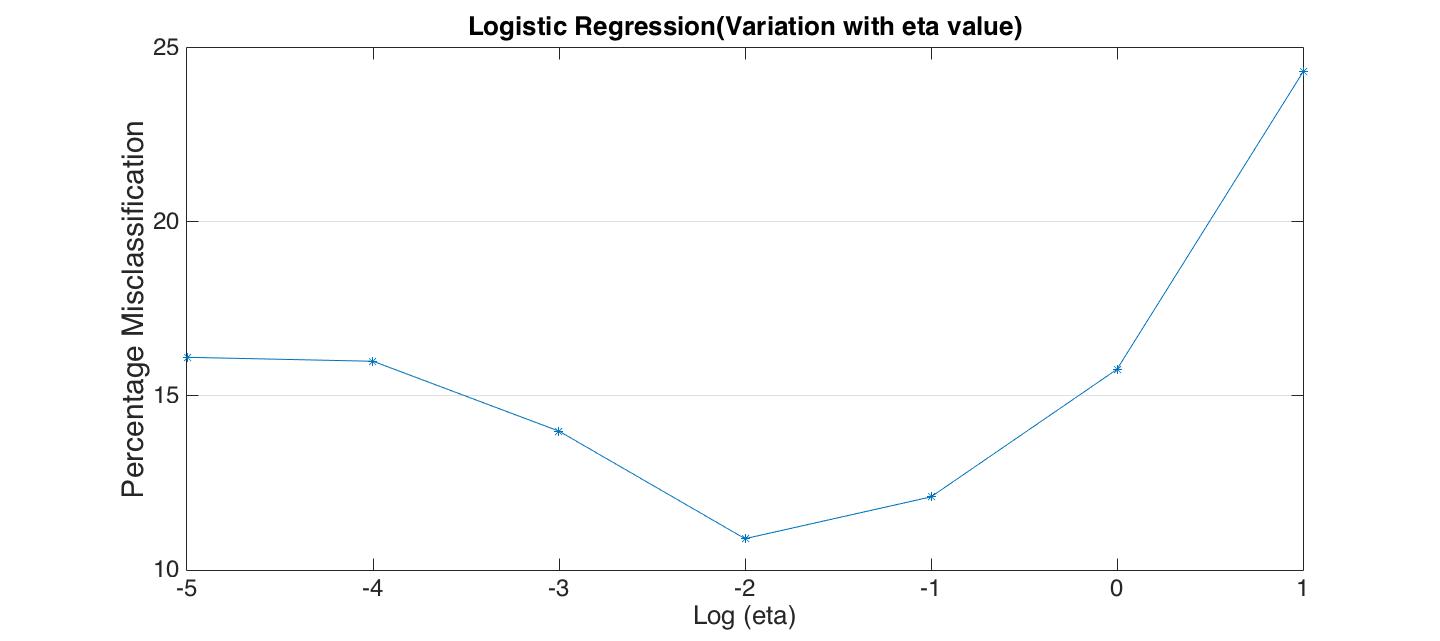
Still this is not satisfactory. So we need to look at other possible measures to reduce error.

**Step 2:** Learn bias also.

Along with learning weights we can also learn the bias, which produces the minimum error. We simply append the bias to the weights vector and append the input images with a one in corresponding location. This drastically reduced the error. The error was reduced right from ~20% to 10%.

**Step 3**: Learning rate tuning.

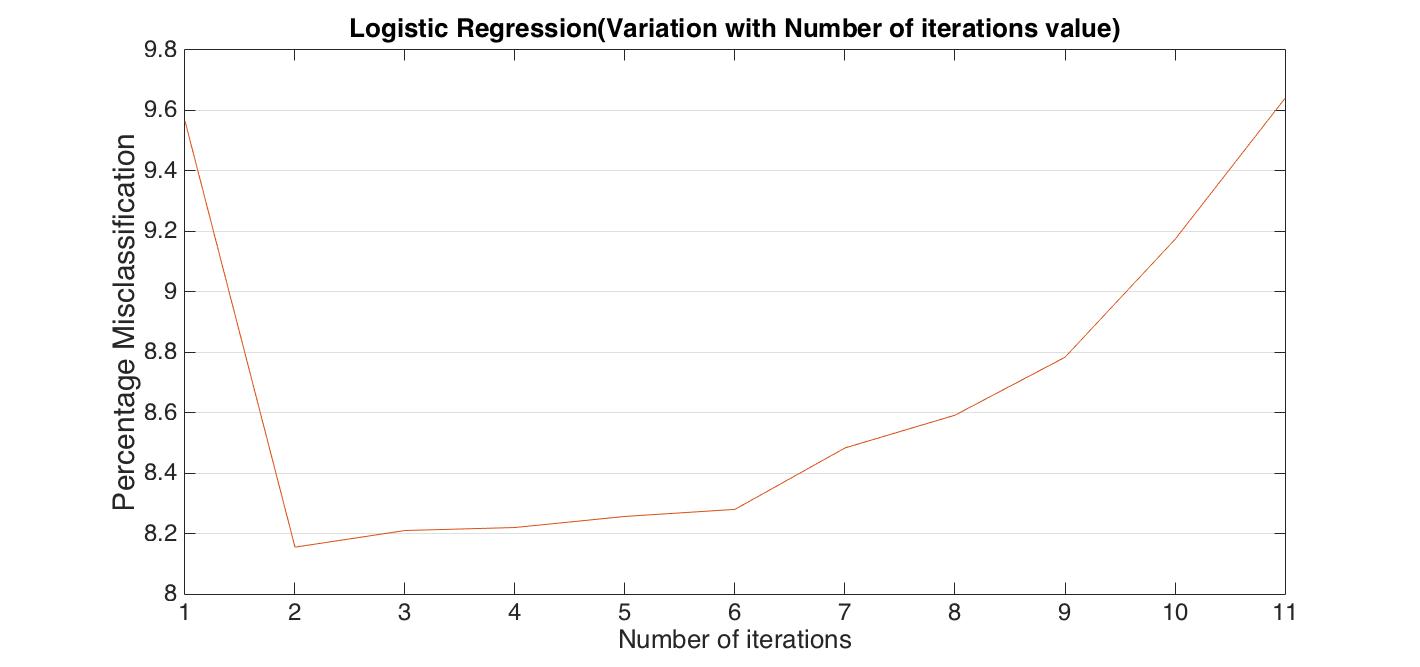
Learning rate is an important parameter, and greatly affect the final error of a system. It controls the magnitude of changes to the parameters. For tuning learning rate we use grid search method.



Misclassification shown in the above graph is misclassification in testing data. We are getting minimum at 0.01, so we fix this value.

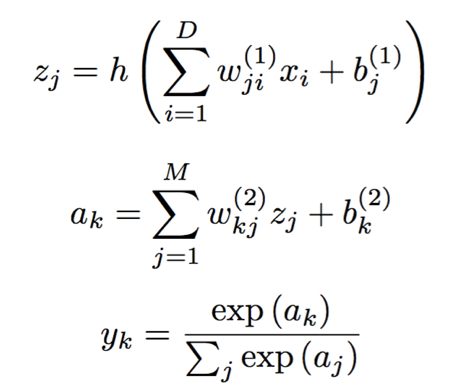
**Step 4:** Number of Iterations:

We sometimes may not have enough data for the error function to reach a minimum in one round or gradient decent training. Or sometime we keep the learning rate so low that we our system is not able to reach minima of error. So we do multiple iterations on the same data.



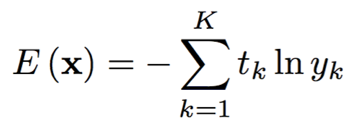
# **Single Layer Neural Network**

We are using a neural network with one hidden layer. Suppose the input layers is denoted by xi and the output is yk. The feed forward propagation is as follows:

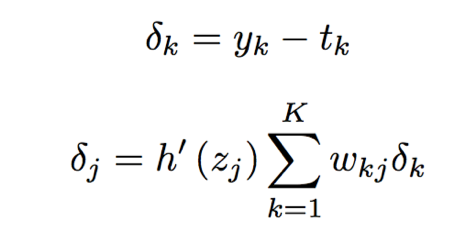


where zj are the activation of the hidden layer and h (·) is the activation function for the hidden layer. You have three choices for the activation function: logistic sigmoid, hyperbolic tangent or rectified linear unit.

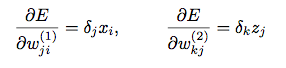
We use cross-entropy error function



where yk = yk (x). The backpropagation is done as follows,



The gradient of the error function would be



Having the gradients, we will be able to use stochastic gradient descent to train the neural network.

../../../../../../Desktop/Screen%20Shot%202015-12-08%20at%204.1

where w is all parameters of the neural network.

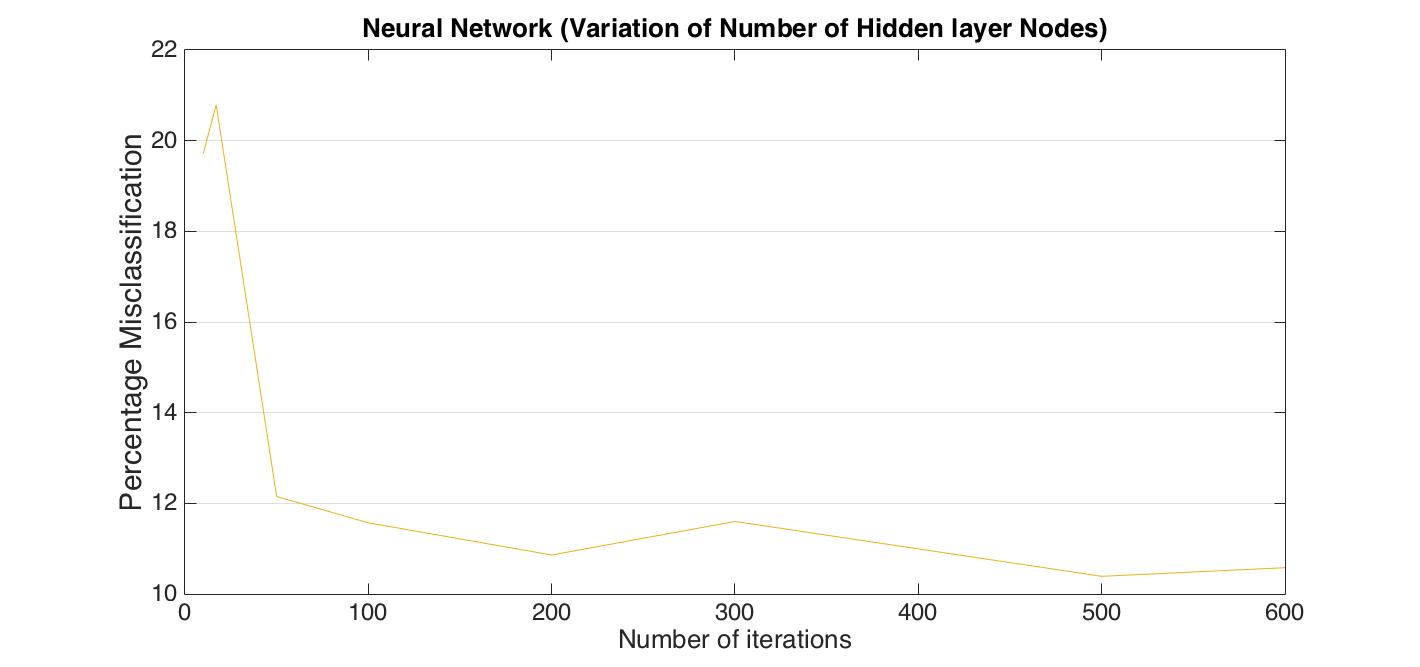
# **Implementation of Single Layered Neural Network**

Single Layered Neural Network was implemented expecting some improvement over neural network performance. We choose single hidden layer because one hidden layer is generally sufficient and there are very few problems more than one hidden layer is required. In single layer neural network, the Hyperparameters are, learning rate, number of nodes in hidden layer, batch size (if using mini batch gradient descent method), bias of weights between input and hidden layer, and bias of weights between hidden layer and output.

Following is my step by step tuning of hyperparameters and other programming techniques:

**Step 1**: Tuning Number of Nodes

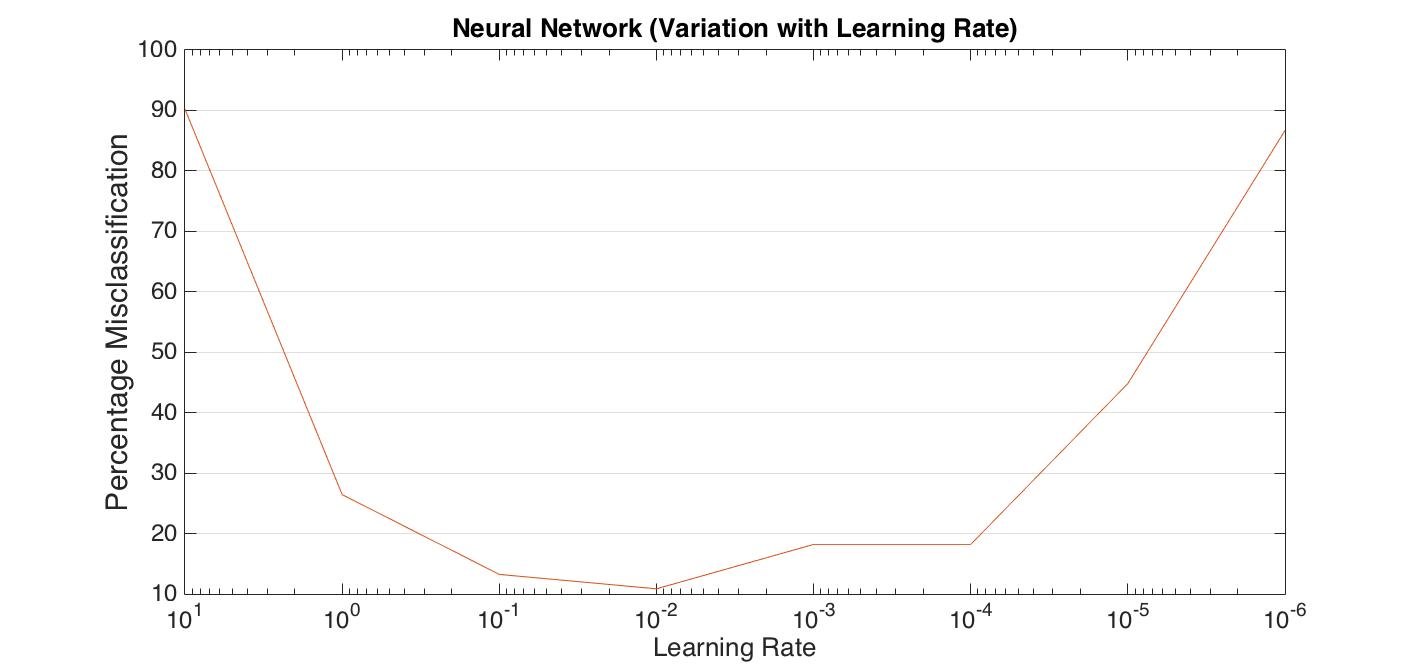
Number of nodes in a neural network is the biggest game changer. Choosing appropriate number of hidden nodes can significantly improve learning. We here changed the number of nodes in hidden layer to see how performance of system improves.



So we fix the number of nodes as 600.

**Step 2**: Learning rate

Learning rate is another important factor to be considered for reducing misclassification rate. But it is better to consider both number of nodes and misclassification rate together, as there can be different optimum learning rate for different number of hidden layer nodes. So we plot a 3D graph comparing learning rate and hidden layer nodes with misclassification rate.

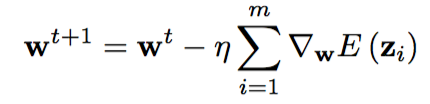


So we fix the value of learning rate as 0.01

**Step 3**: Batch size

So far I considered purely gradient decent method for error calculation. But as it takes longer to run such a code and hence it is not practical to iterate through the same training data more than 5-10 times. So we use mini-batch gradient descent method, which is a mixture of gradient descent and batch methods.

Mini-batch stochastic gradient descent is something between batch gradient descent and stochastic gradient descent. In each iteration of the mini-batch SGD, it samples a small chunk of samples z1, z2, ..., zm from the training data and uses this chunk to update the parameters w:

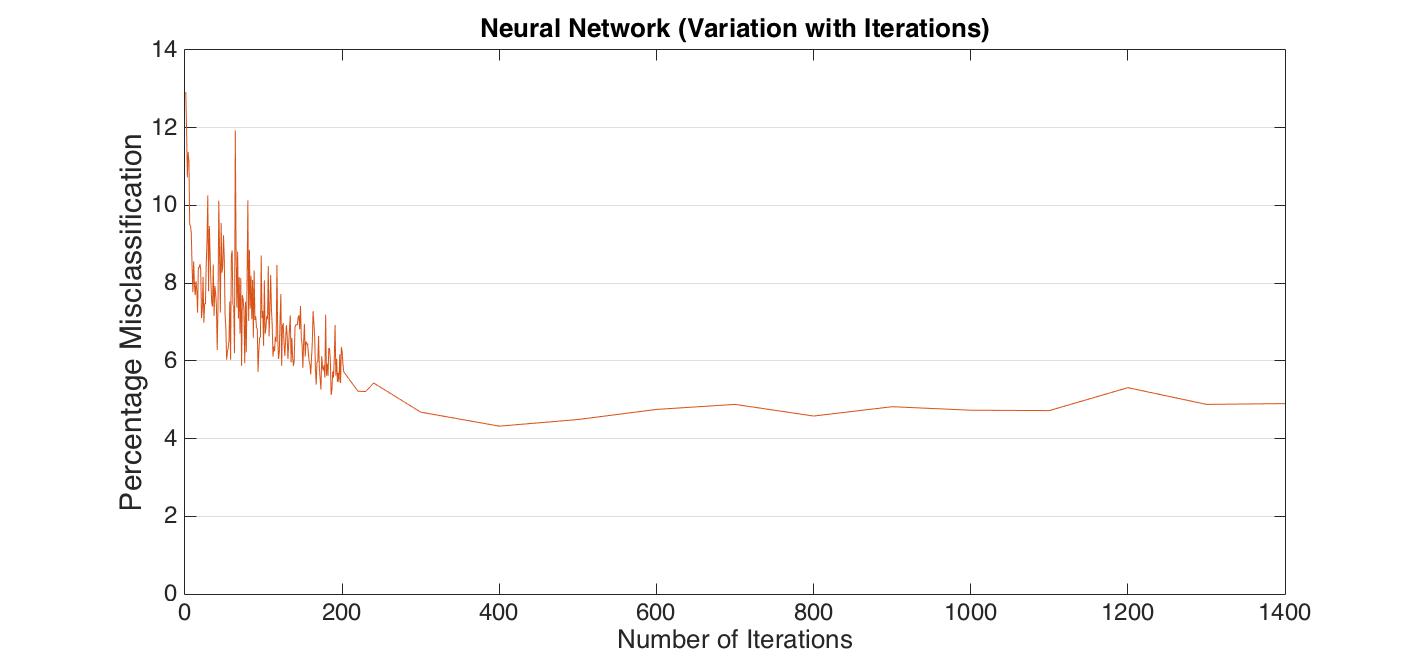


The strength of mini-batch SGD compared to SGD is that the computation of Σim=1∇w*E*(zi) can usually be performed using matrix operation and thus largely out-performs the speed of computing ∇w*E*(zi) individually and updating w sequentially. However, within same computing time, mini-batch SGD updates the weights much more often than batch gradient descent, which gives mini-batch SGD faster converging speed. The choice of mini-batch size m is the tradeoff of the two effects.

Instead of randomly sampling z1, z2 , ..., zm from the training data each time, the normal practice is we randomly shuffle the training set x1,...,xN ,partition it into mini-batches of size m and feed the chunks sequentially to the mini-batch SGD. We loop over all training mini-batches until the training converges.

We kept the batch size as 50.

**Step 4: Number of iterations**



We were getting low error (Minimum of **4.23%**) after 800 iterations

# **Convolutional** Neural Network

Convolutional Neural network improved feed-forward neural network which helps overcome following problems in general feed forward neural network.

* Neural network does not take in to account the spatial structure of the image. Pixels far apart and pixels close together are treated same.
* Neural network is not invariant.
* It is not able to recognize transitions. A number written in left side of image is not same as number written in right side of image.
* It is not able to recognize rotations. It can be the case when the number written in the image is a little bit rotated. There are problems recognizing such images using neural network.
* It is not able to recognize scaling. It may treat a longer six and a more round 6 differently.

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features.[3]

In this project we used an online package to do the task of MNIST image detection. Details of the package is as below:

*CNN - Convolutional neural network class*

*by Mihail Sirotenko*

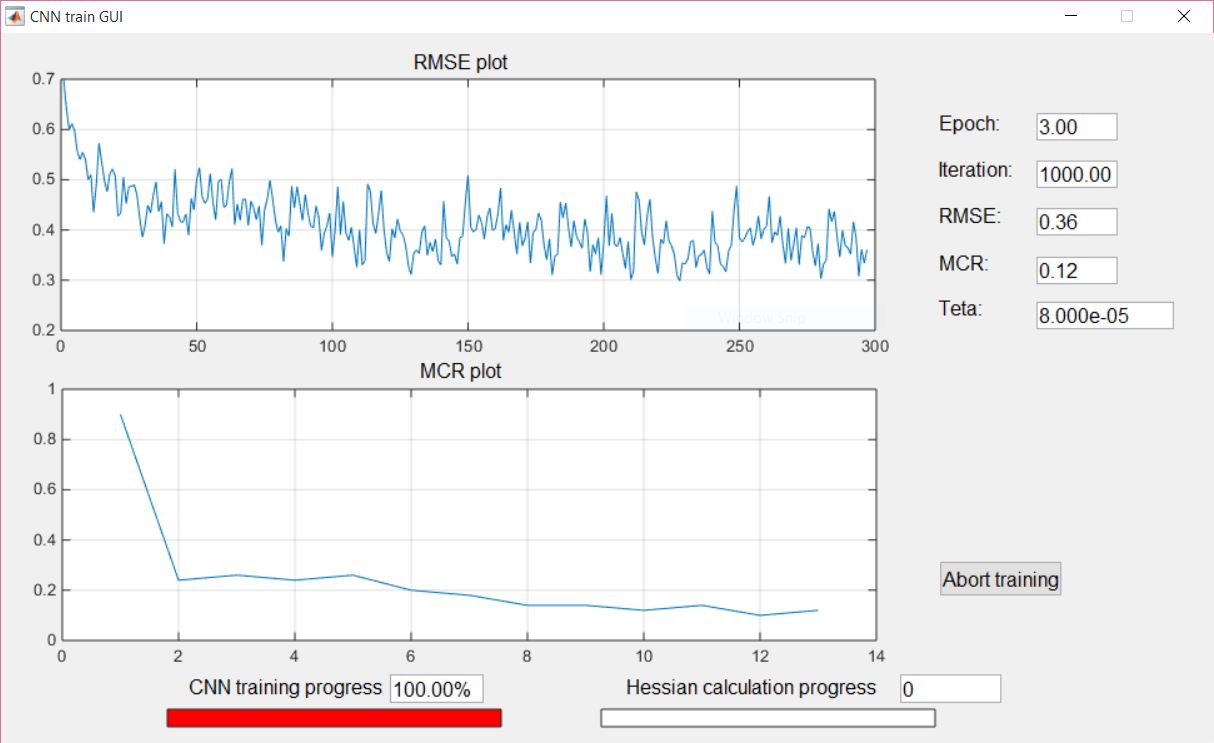
Some of the details of the implementations and default configurations are as follows:

* Number of layers used = 8
* Form the code we can see that the network has 3 subsampling, 3 convolutional layers and 2 fully connected layers which are implemented in following order

Subsampling -> Convolutional -> Subsampling -> Convolutional -> Subsampling -> Convolutional -> Fully Connected -> Fully Connected

* Window size = 5

Following is the output generated after running the code with default configuration:



**References**

[1] Project Description of Project 2: Learning to Rank using Linear Regression. CSE474/574: Introduction to Machine Learning (Fall 2015)

[2] Bishop - Pattern Recognition and Machine Learning.pdf

[3] <http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/>

[4] http://www.mathworks.com/matlabcentral/fileexchange/24291-cnn-convolutional-neural-network-class