**CSE574: Introduction to Machine Learning**

**Project 2: Classification of Handwritten Numerals using Logistic Regression and Neural Networks**

**By-**

**Aniket Deshpande (ad78)**

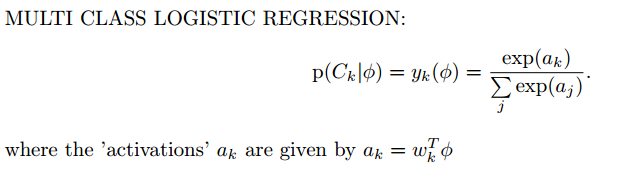
**50133684**

# Introduction:

The objective of the project was to recognize handwritten digits and predict the digit for some test data. This is a classical problem of Classification in Machine Learning. To accomplish this we have used two methods namely:

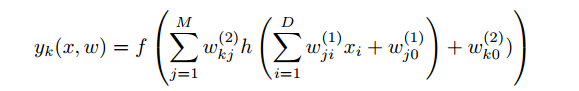
1. Multiclass Logistic Regression

The following formula was used:



1. Neural Networks

We have used the following formula:



Where f (.) is softmax function and h (.) we have taken as tanh function

For input data, I have used the provided GSC features extracted from each of the image.

# Logistic Regression Approach:

Multiclass logistic is implemented using softmax function.



Here we will train the model for classification by minimizing the error. For error function we use the cross – entropy error function which is given as:

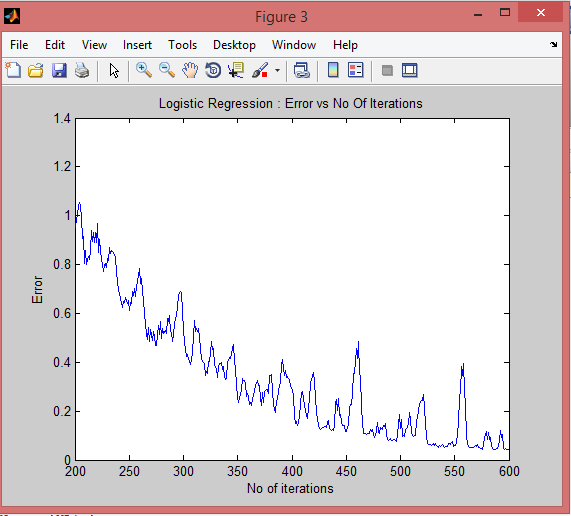


To accomplish this, we use gradient descent algorithm whose formula is given by:



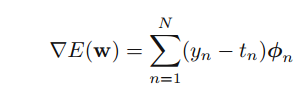
Following steps were taken to implement the above steps: [runfile\_lr.m]

1. Initially, an input matrix X was created by reading the provided .txt files through matlab and appending all of them into a dingle variable X. Then bias parameter is incorporated in this variable. Thus the dimensions of X, in our case, become [(D + 1) x N] where D are the number of features in the provided .txt files and N is the total data for training. A Target matrix T was also created using 1-of-k scheme for our 10 classes. The dimensions of T are [K x N]. [load\_train\_data.m and load\_test\_data.m]
2. The weight vector is then initialized to all ones with dimensions [(D + 1) x K]. After this the parameters of “eta” and “no. of iterations” are set. We initially try for 800 iterations and eta value as 0.2. Following graph is generated for Error vs No of Iteration:



[train\_lr.m]

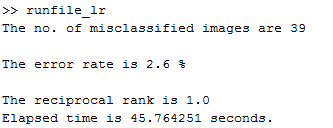
1. Then Y ie output vector was calculated using the above softmax function. The dimensions of Y would be [K x N]. To prevent underflow, a logsumexp() function was written. Then error gradient was calculated using following formula:



Then, the above mentioned gradient descent formula was used to update the weight vector [train\_lr.m]

1. After 600 iterations, the labels were predicted using generated weight vector [test\_lr.m]

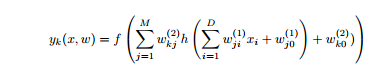
The output after running the runfile\_lr is as follows:



The predicted labels for the net test data are then saved in classes\_lr.txt.

# Neural Networks Approach:

The net Neural Network output function can be written as follows:

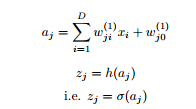


We implement Neural Network with 1 Hidden Layer.

For the first Layer i.e. Input layer (h), we use the tanh function

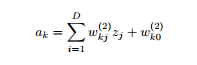
For the output layer(f ), softmax function is used as is the norm in multiclass regression.

The formula for hidden layer is as follows:



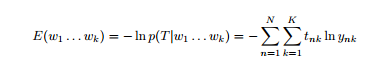
The quantities aj are called activations. Here h(.) is the tanh function we use on the activations to get values of the hidden units zj .

For the output layer, following formula is used



where k = 1 . . . , K and K is the number of outputs, i.e. 10 in our case.

The cross entropy error function used by us is:



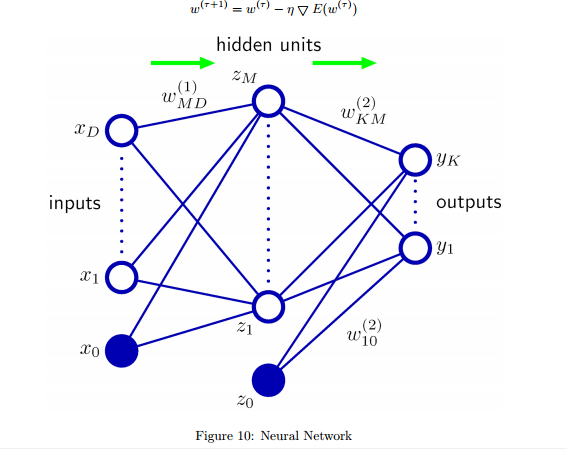
Now to calculate error gradient at hidden layer and output layer, we use error backpropagation. The following formulas are used:







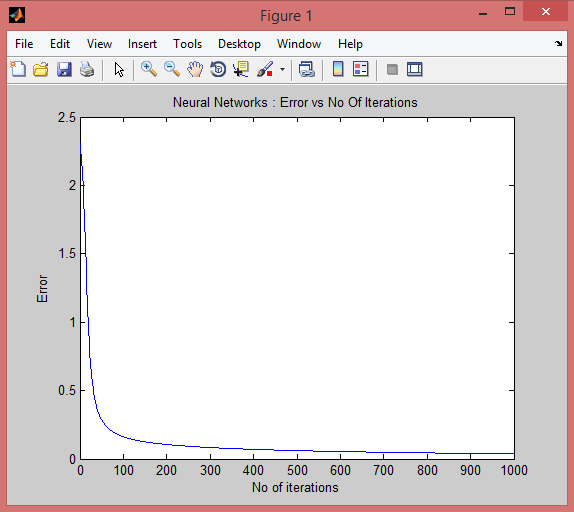
The error gradients are required to calculate the weight matrix at each layer according to following formula of gradient descent:



The model shown above is the standard neural network model that I have implemented in my code.

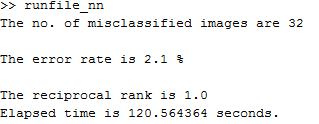
The steps taken to implement the above model are as follows: [runfile\_nn.m]

1. Initially, an input matrix X was created by reading the provided .txt files through matlab and appending all of them into a dingle variable X. Then bias parameter is incorporated in this variable. Thus the dimensions of X, in our case, become [(D + 1) x N] where D are the number of features in the provided .txt files and N is the total data for training. A Target matrix T was also created using 1-of-k scheme for our 10 classes. The dimensions of T are [K x N]. [load\_train\_data.m and load\_test\_data.m]
2. The values for parameters M and learning rate are decided. In our case M is taken as 20 and learning rate as 0.00015. The weight matrices are then initialized to random values using rand() function. The dimensions of the weight matrices are: For hidden layer weight matric ie W1, the dimensions are [(D + 1) x M] and for output layer weight matrix ie W2, the dimensions are [M x K]. The activations, output, delta and updated weight matrices are then calculated using above mentioned formulas. This is done for 1000 iterations. The graph for error vs no of iterations as follows :[train\_nn.m]



1. After these iterations the labels were predicted with the generated weight matrices.

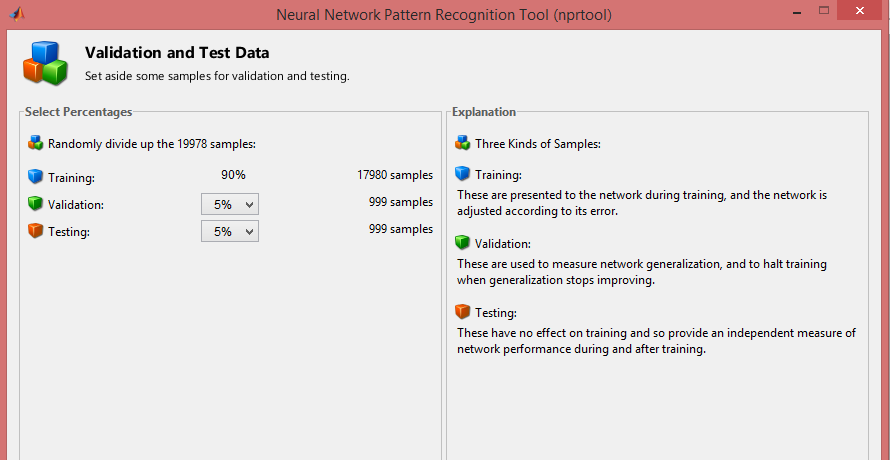
The output after running runfile\_nn.m is as follows:

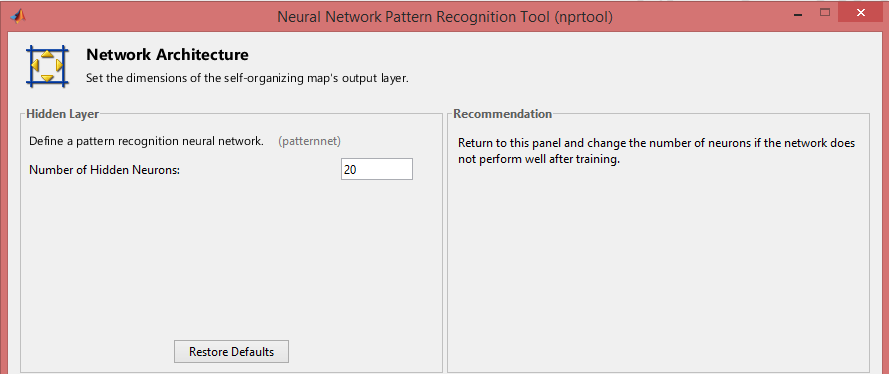


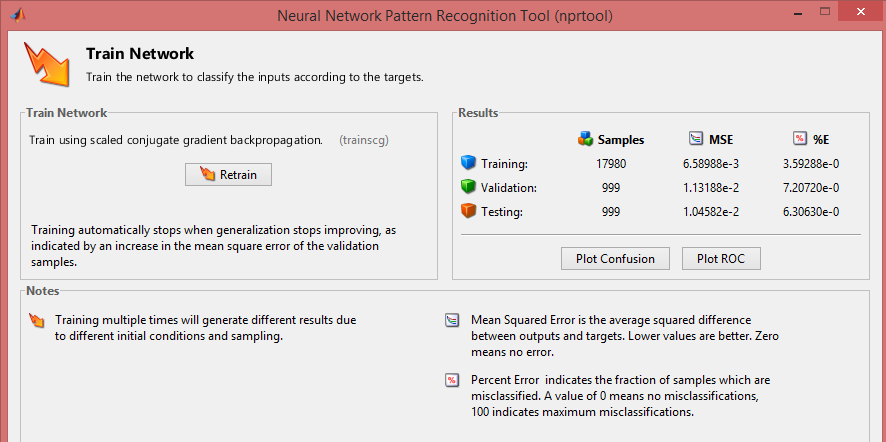
The predicted labels for the net test data are then saved in classes\_nn.txt.

# Comparison with Neural Network Package:

The default MATLAB neural network package is configured as follows:







The different error comparisons are as follows

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No | Model | Error percentage(%) | Reciprocal rank |
| 1 | Logistic Regression | 2.6 | 1 |
| 2. | Neural Network | 2.1 | 1 |
| 3. | Neural Network Package | 6.3 |  |