Project 3

Classification of Handwritten Digits

Submitted by:
Armaan Goyal
50170093
armaango

Logistic Regression

Logistic Regression is a regression model that is used for the purpose of classification. Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution. Thus, it treats the same set of problems as probit regression using similar techniques, with the latter using a cumulative normal distribution curve instead. Equivalently, in the latent variable interpretations of these two methods, the first assumes a standard logistic distribution of errors and the second a standard normal distribution of errors.

Neural Networks

In machine learning and cognitive science, artificial neural networks (ANNs) are a family of models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.

For example, a neural network for handwriting recognition is defined by a set of input neurons, which may be activated by the pixels of an input image. After being weighted and transformed by a function (determined by the network's designer), the activations of these neurons are then passed on to other neurons. This process is repeated until finally, an output neuron is activated. This determines which character was read.

Problem Statement

The objective of the project was to implement and evaluate classification algorithms. The classification task was to recognize a 28 x 28 gray scale handwritten digit image and identify it as a digit among 0, 1, 2, . . . , 9. To perform this, we were required to the following tasks:

1. Implement Logistic Regression, train it on the MNIST digit images and tune hyper parameters.
2. Implement single hidden layer Neural Network, train it on the MNIST digit images and tune hyper parameters such as the number of units in the hidden layer.
3. Use a publicly available convolutional neural network package, train it on the MNIST digit images and tune hyper parameters.

We were advised to use Mini Batch Stochastic gradient descent method for the above algorithms.

Dataset

For the training of our classifiers, we used 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 contains 60,000 training images and 10,000 testing images. The database is also widely used for training and testing in the field of machine learning.

The original black and white (bilevel) 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.

Solution Approach:

The solution approach was divided into several steps based on the tasks we were asked to perform.

Preprocessing of Data:

The data to be used was downloaded in the form of 4 files, 2 each for the training and test sets to be used. The data files had to be processed before using the values provided. A small bit of preprocessing code along with some freely available code from the internet was tweaked and used for performing this preprocessing and required data was extracted from the files and saved in the form of different multidimensional matrices as follows:
Training Images-60000X784 matrix
Training Labels-60000X1 matrix
Testing Images-10000x784 matrix
Testing Labels-10000X1 matrix

These matrices were later saved in a data.mat file, which could be easily loaded and used four Logistic Regression and Neural Networks implementation.

Implementing Logistic Regression:

The following formulae are to be used for the implementation of Logistic regression for our given data set.

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

$$p\left(\mathcal{C}_k|\mathbf{x}\right) = y_k\left(\mathbf{x}\right) = \frac{\exp\left(a_k\right)}{\sum_j \exp\left(a_j\right)}$$

The cross-entropy error function for multiclass classification problem :

$$E(\mathbf{x}) = -\sum_{k=1}^{K} t_k \ln y_k$$

The gradient of the error function would be:

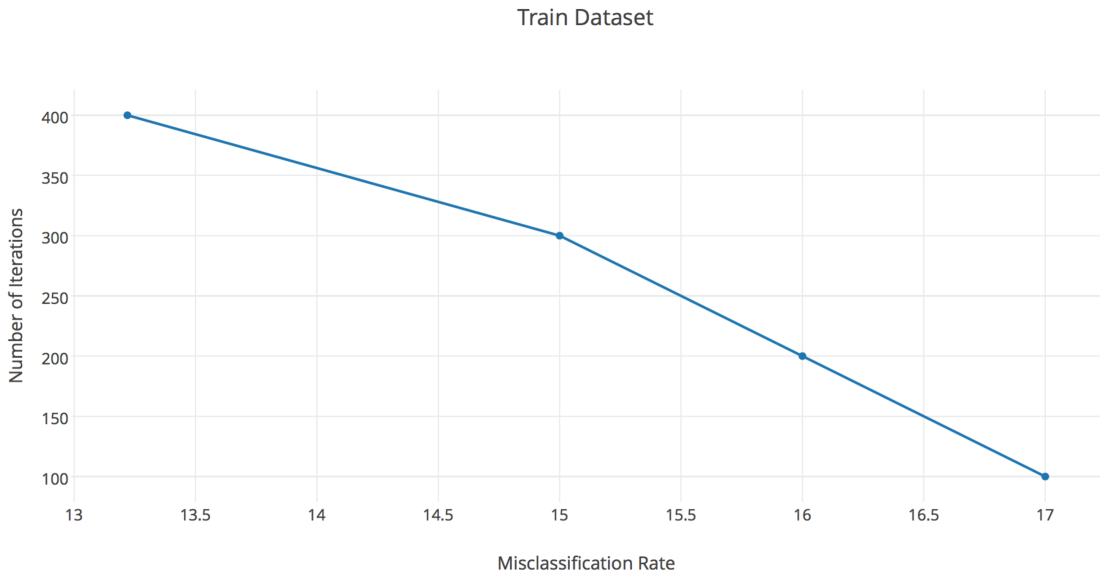
$$\nabla_{\mathbf{w}_j} E(\mathbf{x}) = (y_j - t_j)\,\mathbf{x}$$

Stochastic gradient descent (which uses the first order derivatives) can be used to find the optimum of error function and find the solution for Wj. This takes the form:

$$\mathbf{w}_j^{t+1} = \mathbf{w}_j^t - \eta \nabla_{\mathbf{w}_j} E(\mathbf{x})$$

For this approach, we use a softmax function to classify the data.
For calculating the weight vector we need to add a bias value. I added a column of all ones as the bias value to my initial weights, which were chosen at random.

We use the mini batch stochastic gradient descent method for Logistic Regression. The regression is performed on the training set and then similarly on the test set. After analyzing the error values, I tweaked the values of the hyperparameters like the learning rate and the no of passes. I started with value of eta as 1 and kept decreasing it. After several different iterations for combinations of both I reached at values of 400 passes and learning rate value of 0.01.



I observed that I achieved an error rate of somewhere between 12 - 13% based on the autograder results.

Neural Networks Implementation

We were asked to use a single hidden layer neural network for our implementation.
The input layers are denoted by x_i and the output is y_k. The feed forward propagation is calculated as follows:

$$z_j = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_k^{(2)}$$

$$y_k = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

Where zj is 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.
I used 'tanh' function as my activation function as it was easier to implement and gave me better results.

We use cross-entropy error function:

$$E(\mathbf{x}) = -\sum_{k=1}^{K} t_k \ln y_k$$

The back propagation is done as follows:

$$\delta_k = y_k - t_k$$

$$\delta_j = h'(z_j) \sum_{k=1}^{K} w_{kj} \delta_k$$

Then gradient of the error function is found as:

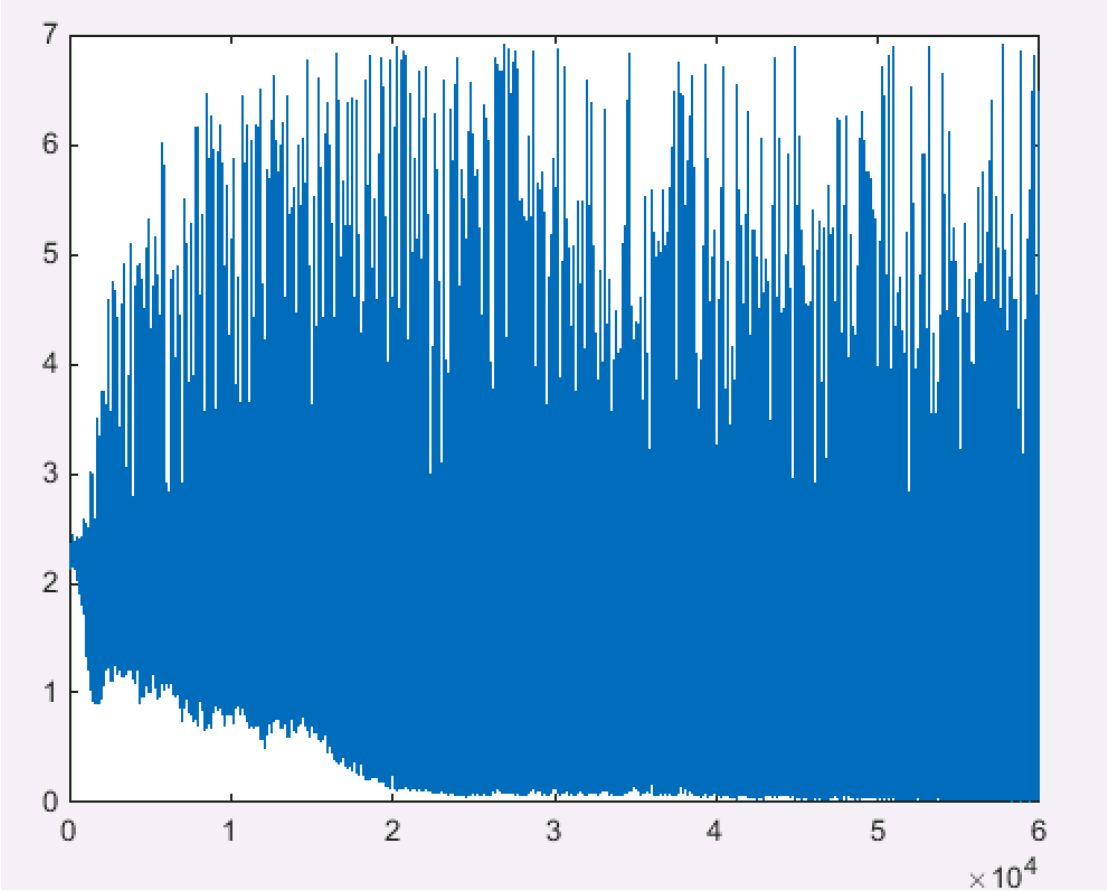
$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

Having the gradients, we were be able to use stochastic gradient descent to train the neural network, which takes the form as earlier:

$$\mathbf{w}_j^{t+1} = \mathbf{w}_j^t - \eta \nabla_{\mathbf{w}_j} E(\mathbf{x})$$

I implemented the above formulae into a short Matlab code and as with the Logistic Regression tried to implement the mini batch stochastic gradient descent method but it gave very high values of error on my local computations. So I implemented the code using Stochastic gradient descent method that runs through each of the training data values to compute the weight vector. After tweaking the values for the hyperparameters I was able to get the error rate to the following values for training and test data.

Error rate for training data ~ 19%
Error rate for test data ~18%



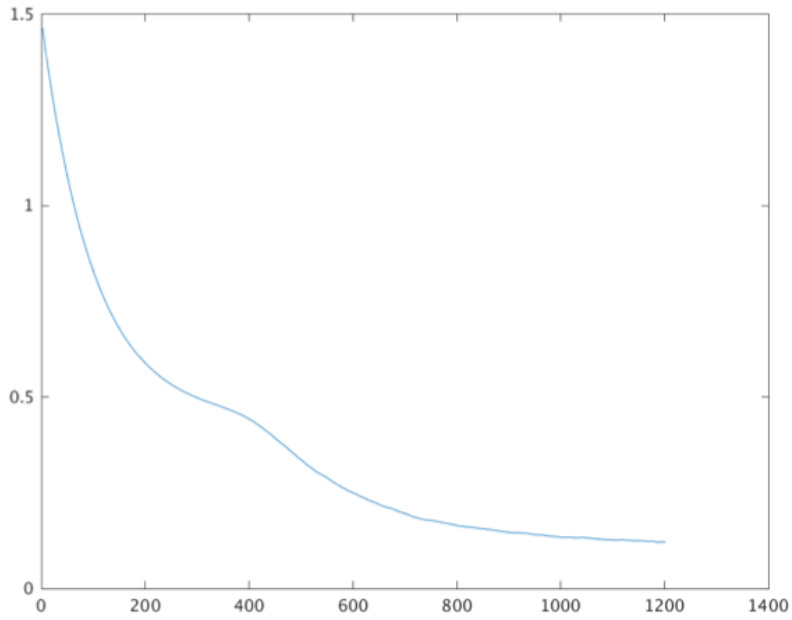
Classification rate plot for Neural network with single layer

Convolutional Neural Network Implementation

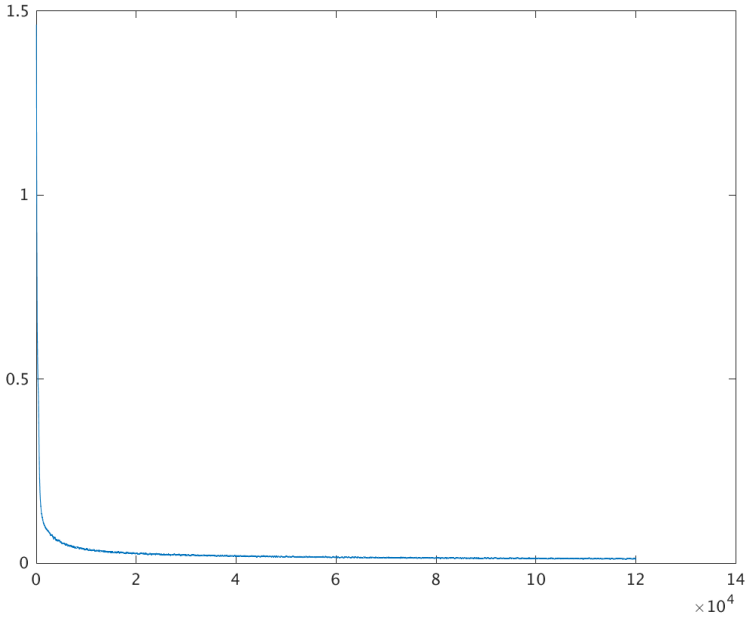
In machine learning, a convolutional neural network is a type of feed-forward artificial neural network where the individual neurons are tiled in such a way that they respond to overlapping regions in the visual field. Biological processes inspired convolutional networks and are variations of multilayer perceptrons designed to use minimal amounts of preprocessing. They have wide applications in image and video recognition.

Convolutional neural networks usually require a large amount of training data in order to avoid over fitting. A common technique is to train the network on a larger data set from a related domain. Once the network parameters have converged an additional training step is performed using the in-domain data to fine-tune the network weights. This allows convolutional networks to be successfully applied to problems with small training sets.

For this part we were asked to use a publicly available Deep Learn toolbox. I used the same to implement this section. The code was also publicly available and was run on the cse metallica server using command line interface. For this first the git repository had to be copied to the metallica server. The code was run in two iterations. First was for 1 epoch or pass through the network and second was for 100 passes through the network.
For both these iterations a Barplot was generated that represents the error rate.



Error rate for 1 epoch



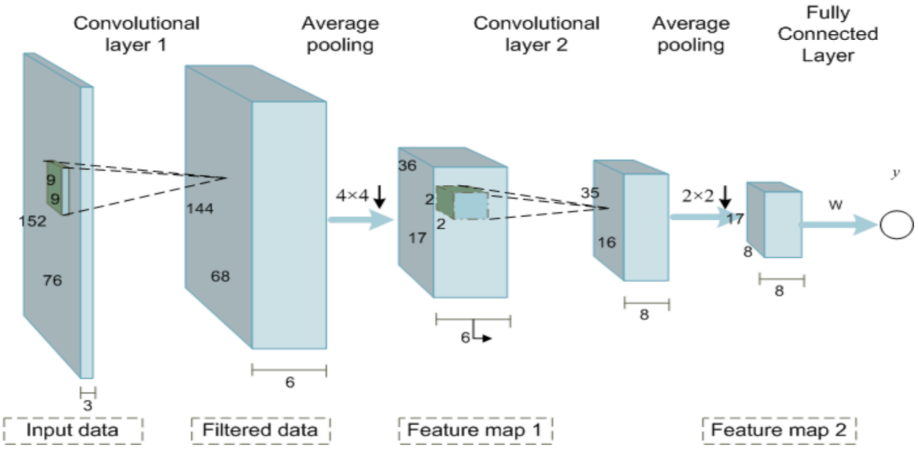
Error rate plot for 100 epochs

It can be seen that the error reduces quickly for 100 epochs and total error also falls from 11 to around 1.2%

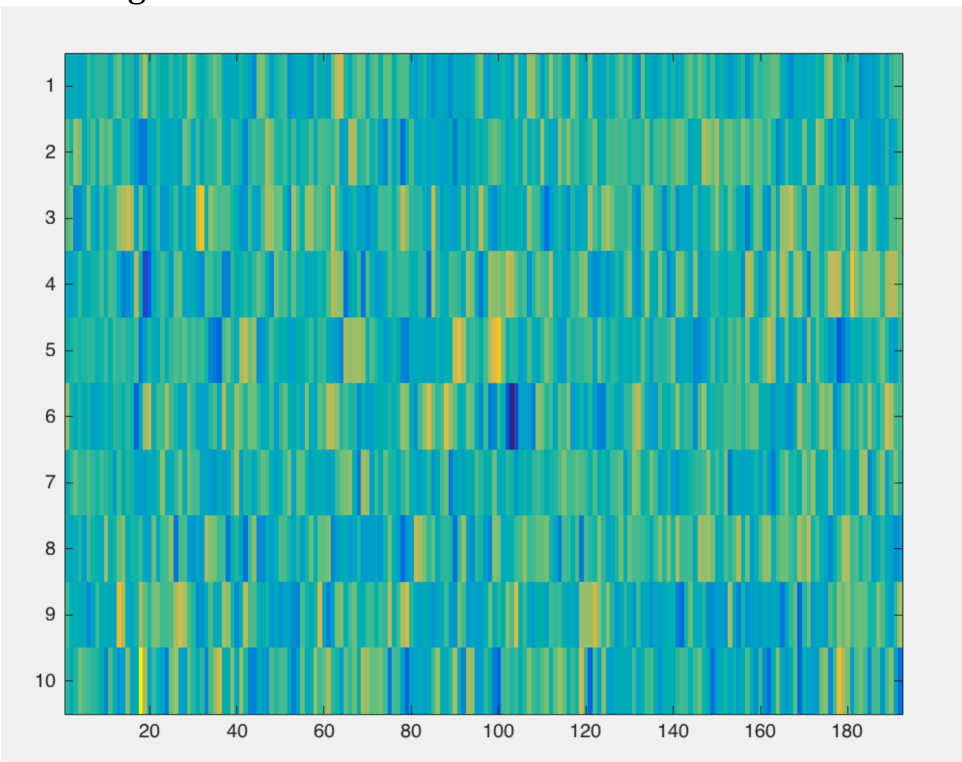
The CNN structure can be divided into several layers as follows:

1. Input layer
2. Convolution Layer
3. sub sampling layer
4. convolution layer
5. subsampling layer

The structure for some sample data can be visualized as below:



The weights calculated can be visualized as below:



Conclusions:

For all three techniques of classification we received different values of misclassification rates.
For Logistic Regression and Neural Networks error rates of around 12 and 18 % were obtained and for Convolutional neural networks an error rate of around 1.2% was obtained. Also it was observed that the computation for Neural and CNN was much quicker than LR for the given data set.

Reference:

The preprocessing code was sourced from
http://www.mathworks.com/matlabcentral/fileexchange/27675-read-digits-and-labels-from-mnist-database

The theory was sourced from
Wikipedia.com and Bishop-Pattern Recognition

Concepts about CNN were sourced from:
https://s3.amazonaws.com/piazza-resources/i48o74a0lqu0/i5jeysuovgq1e9/Tutorial_3_Introduction_to_MatlabToolbox.pdf?AWSAccessKeyId=AKIAJKOQYKAYOBKKVTKQ&Expires=1449708049&Signature=cvSEBCysJ497ZLqToYqH3xuaYRI=