Evaluation and Implementation
Of
Classification Algorithms

Project 3

Abhishek Gautam
#50169657
agautam2@buffalo.edu

Problem Statement

The objective of the project was to implement and evaluate classification algorithms. The classification task was to recognize a 28 x 28 grayscale 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.

Hand writing digit recognition can be formulated by many machine learning algorithms for instance logistic regression, neural networks, expectation maximization algorithms, mixture models, Gaussian processes etc. It is a classic problem and has been solved using many different algorithms already. The new exciting development in the field is the Extraction of features using deep belief networks (DBN). The field of deep learning is famous because it enables automatic feature extraction. Usually, in the field of machine learning, the features that we extract are based on a specific knowledge of the data, i.e., it answers the question of what particular characteristic the particular data has. Deep learning extracts these features on its own. It's powerful and gives extremely good results. This algorithm uses a Markov random field inside the neural network and is found to be an extremely powerful optimizer.

This project report implements two machine learning algorithms namely *logistic regression* and *neural networks* and the details of these are given below:

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 corresponding labels and 10,000 testing images and labels. The database is also widely used for training and testing in the field of machine learning.

Background

Hand written digit recognition is one of the classical problems in Machine learning literature. Simply, put it develops automated machine learning based methods for recognizing handwritten digits and words. This technology was developed about 20-30 years ago and there have been a lot of developments in this particular field of research. One of the important applications of handwritten digit recognition is in United States Postal Service (USPS) where each digit (Pin-code) is automatically scanned and the appropriate area code is retrieved. This saves a lot of time and manual labor.

Handwritten digits recognition is an important problem. The main challenge is in extraction of appropriate features so that digits written in any manner are recognized. Some numerals may be improperly used or written in a cursive manner. There are various ways of writing the same number and it is important for the Machine Learning algorithm to recognize variety in handwriting in order to make a proper and accurate prediction. This involves a significant component of handwriting analysis. The features used here were developed at the Center for Excellence in Document Analysis and recognition (CEDAR) in the University at Buffalo.

Logistic Regression

Suppose we use 1-of-K coding scheme t = [t1, ..., tK] for our multiclass classification task. Our multiclass logistic regression model could be 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 seeing a training sample x would be:

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

The gradient of the error function would be:

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

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:

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

Single Layer Neural Network

Suppose we are using a neural network with one hidden layer. Suppose the input layers are denoted by Xi and the output is Yk. The feed forward propagation is 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 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:

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

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 would be:

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

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

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

Solution Approach

Preprocessing of data

The given dataset was in the form of four different files, two for the training set and the other two for the testing set. I performed some preprocessing on data to make it readably by MATLAB. This was achieved by tweaking the code that was available online to suit our needs.

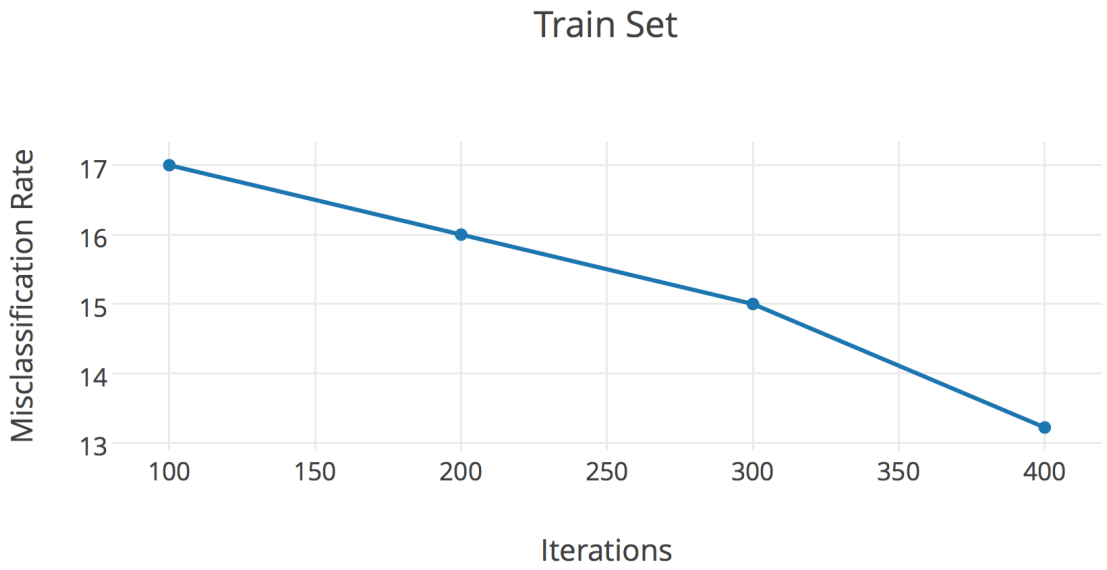
Post preprocessing of data the following format was achieved.

Matrix	Dimension
Training Images	60000X784
Training Labels	60000X1
Testing Images	10000x784
Testing Labels	10000X1

Classification using Logistic Regression and findings

In this approach I used the Mini Batch Logistic Regression method as dictated by the appendix and implemented it on training data and similarly on test data. Based on the error rates I got, I kept on changing the learning rate parameter and the number of iterations. With increasing number of iterations, the computation became more intensive. An analysis of the misclassification error I got upon tweaking values through several iterations is summarized below:

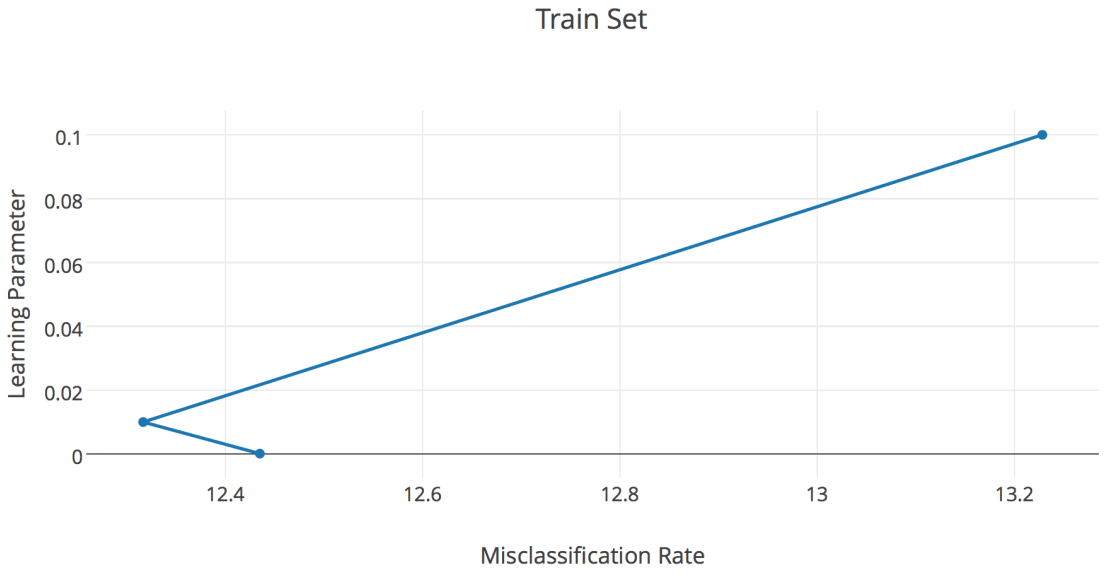
Graph showing relation between misclassification rate and iterations for Train Set.



After this I kept the iterations fixed to 400 and varied the Learning Parameter. Results are shown below:

Learning Parameter	Iterations	Misclassification rate on Train set	Misclassification rate on Test set
1	400	13.2283%	13.24%
0.1	400	12.3167%	11.97%
0.0001	400	12.435%	11.72%

Graph showing relation between misclassification rate and Learning Parameter for Train Set:



Classification using Single Layer Neural Network and findings

For Artificial Neural Networks, we were asked to implement a single hidden layer neural network. The Appendix in the project description dictates the formulae to be used for Neural Network implementation. I implemented the same into a short MATLAB code and as with the Logistic Regression tried to implement the Mini Batch Stochastic Gradient descent method. The computations gave very high values of error. Next, 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 hyper parameters I was able to get the error rates as:

Learning Parameter	Misclassification error Train set	Misclassification error Test set
0.001	19.5383%	18.95%
0.0001	59.6617%	59.56%

Classification using Convolutional Neural Network and findings

In machine learning, a **convolutional neural network** (**CNN**, or **ConvNet**) 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. To implement this part, we were asked to use a publicly available Deep Learning toolbox. I used that package and to implement it, I ran it on the metallica.cse.buffalo.edu server. This was done in two iterations. First iteration had 1 epoch or pass and the second iteration had 100 passes.

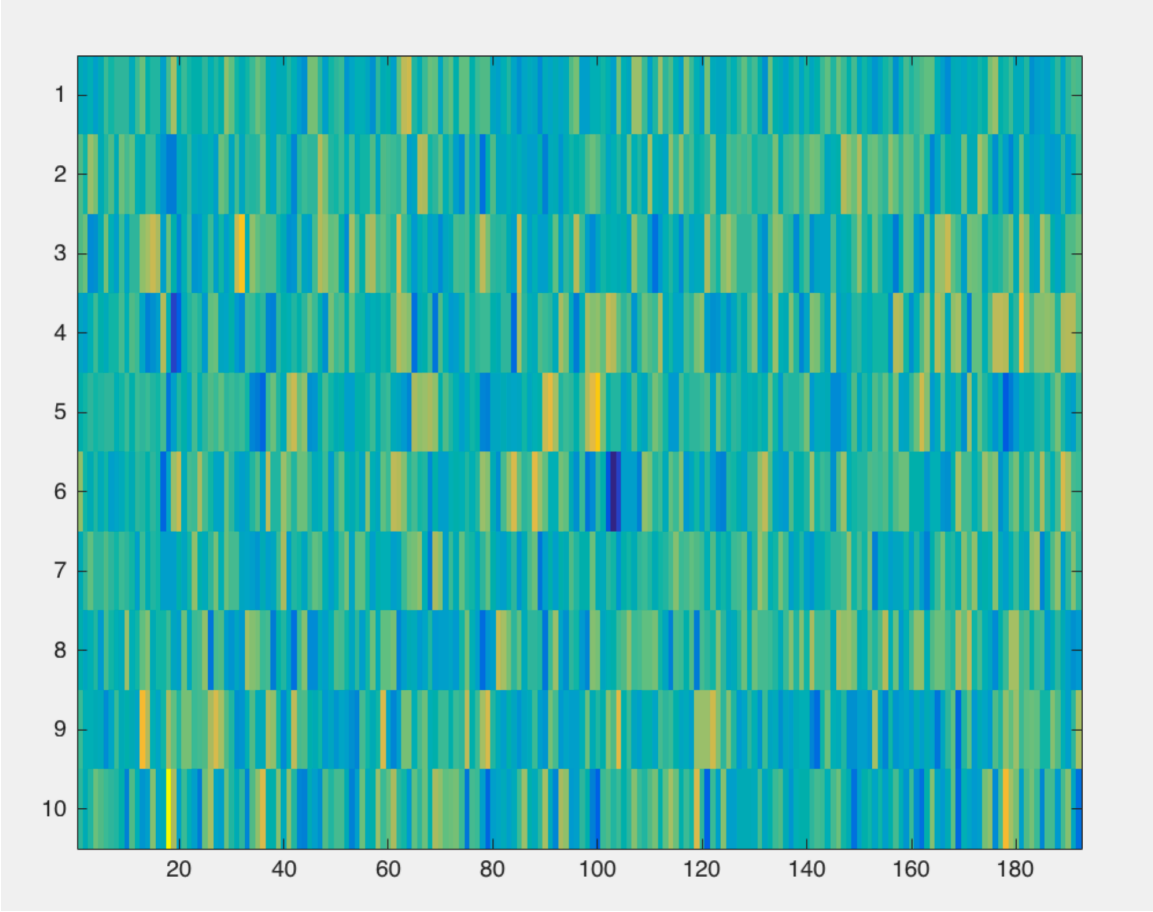
Structure of CNN

In a *convolutional neural network* data and functions have additional structure. The data $\mathbf{X}1,\ldots,\mathbf{X}n$ are images, sounds, or more in general maps from a lattice1 to one or more real numbers. In particular, since the rest of the practical will focus on computer vision applications, data will be 2D arrays of pixels. Formally, each $\mathbf{X}i$ will be a $M\times N\times K$ real array of $M\times N$ pixels and K channels per pixel. Hence the first two dimensions of the array span space, while the last one spans channels. Note that only the input $\mathbf{X}=\mathbf{X}1$ of the network is an actual image, while the remaining data are intermediate *feature maps*.
The second property of a CNN is that the functions Fl have a *convolutional structure*. This means that Fl applies to the input map $\boldsymbol{X}l$ an operator that is *local and translation invariant*. Examples of convolutional operators are applying a bank of linear filters to $\mathbf{X}l$.

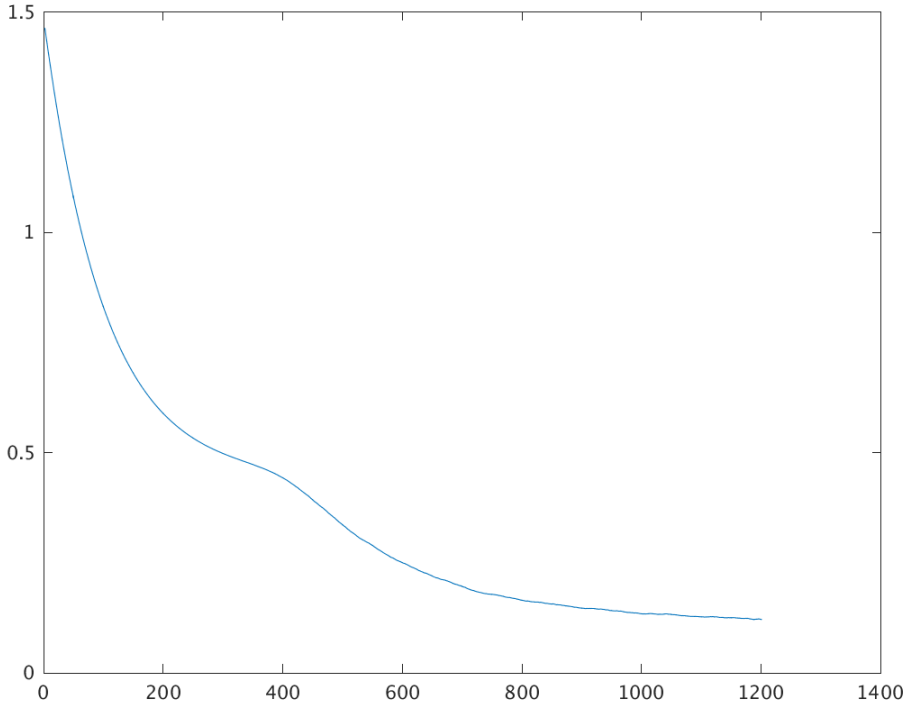
Observations on CNN how well it performs

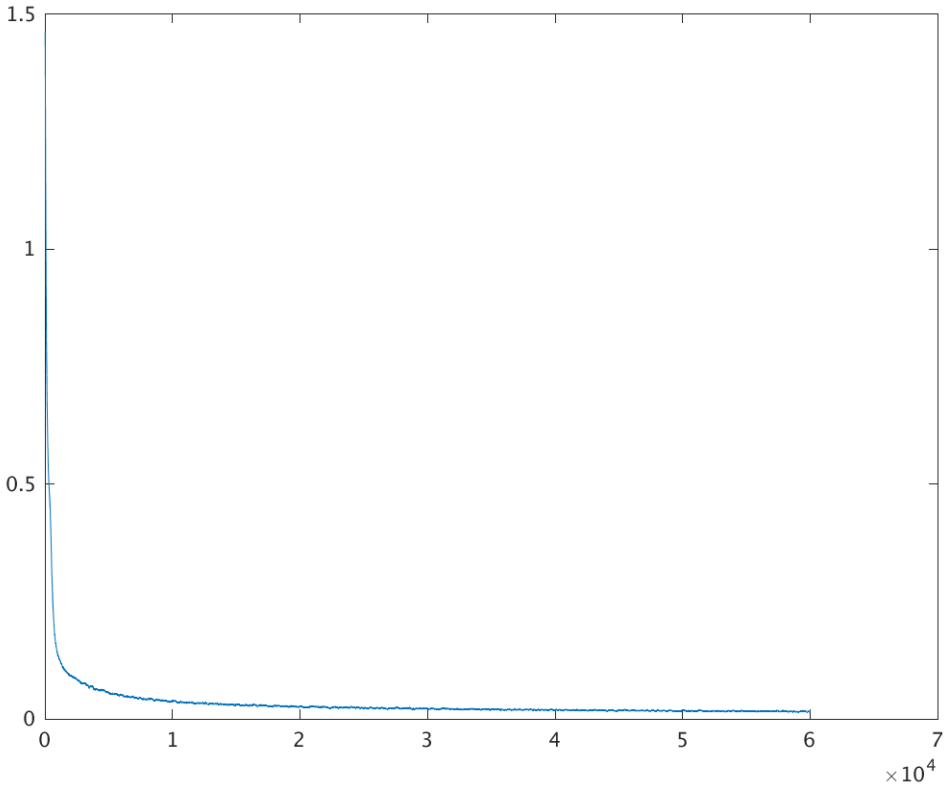
1. Fully connected convolutional layers lead to an exploding number of network connections and weights, making training of big and deep CNNs for hundreds of epochs impractical even on GPUs.
2. Deeper nets require more computation time to complete an epoch, but we observe that they also need fewer epochs to achieve good test errors.
3. The network with 4 instead of 7 hidden layers reaches 4.71%, 1.58%, and 0.68% after one, three and seventeen epochs, achieving a test error below 0.50% after only 34 epochs. This shows once more that deep networks, contrary to common belief, can be trained successfully by back- propagation.
4. Despite the numerous free parameters, deep net- works seem to learn faster (better recognition rates after fewer epochs) than shallow ones.
5. Misclassification rate with 1 epoch was nearing 11% and with 100 epochs it was nearing 1%

A plot for Visualization of weights



The graphs for RMS error are shown below for 1 epoch and 50 epochs respectively:





Sources and References

1. en.wikipedia.org
2. https://plot.ly/plot
3. http://yann.lecun.com/exdb/mnist/
4. http://www.mathworks.com/matlabcentral/
5. http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset
6. http://people.idsia.ch/~juergen/ijcai2011.pdf
7. http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html