Numeric Character Recognition using Multilayer Perceptron

Mukhil Azhagan Mallaiyan Sathiaseelan
Electrical and Computer Engineering Department
University of Florida
Gainesville, U.S.A
mukhil@outlook.com

***Abstract*—Training and Testing a Multilayer Perceptron on Numeric Character recognition using a MNIST dataset. The Architecture is taken as both one and two Hidden Layer Neural Network and tested for various Processing Elements. Stochastic Gradient descent is used as learning algorithm, using both with and without using momentum update to the learning rate.**

Keywords—Multilayer Perceptron; Stochaistic Gradient Descent ; Momentum; Numeric Character Recognition;

I. INTRODUCTION

Artificial Neural Networks(ANN) are one of the most versatile learning systems available today, that are able to extract meaningful features themselves and train its weighting parameters, so as to reach convergence, which is generally more accurate than the other techniques currently available. This welcoming characteristic of an ANN, makes it a semi-parametric classifier. While the user has to have knowledge of the number of Processing elements to use or the shape of the interconnections(topology), the user does not specify the model of the data distribution, the ANN conveniently solves this for us. Over the years, the Machine Learning community has even gone as far to say that ANN are black-boxes, but given a good enough understanding of ANN and, their activation functions in each layer, one is able to predict the inner workings of the ANN, how it separates the input space and how the parameters are calculated by the ANN.

Research in the field has led to many different variations in ANN. Some modify the topology, some modify the interconnectivity and some the activation functions. These have given rise to many types such as recurrent neural networks, convolutional neural networks etc. In this problem however, we will use a simple two Hidden Layer ANN, the detailed architecture of which will be discussed in *Section III.*

The Problem for discussion in the paper is the classification of Numeric Character Recognition. The dataset is obtained from Modified National Institute of Standards (MNIST) database, which is a commonly used database consisting of Handwritten digits [3] . 60000 samples are used for Training and Validation, while 10000 samples are used for Testing. Various techniques such as Normalization and Early stopping during cross-validation are used, the details of which will be discussed in *Section III* and *Section V*

The discussion will be as follows, Section II will discuss Artificial Neural Networks and Perceptron in detail. Section III will discuss about the architecture used for the present Problem, about the Topology and hyper parameters used. Section IV will present various Figures, Tables and Plots to compare the Performance and Accuracy. Section V will be dedicated to the discussion of Obtained results and a detailed explanation of what is obtained.

II. METHODOLOGY OF A MULTILAYER PERCEPTRON

A. The Processing Element

This is the most fundamental block in an ANN. The Processing Element(PE) has an activation function that takes the weighted (*w*) sum of inputs(*x*), and gives out an output *y*. A block diagram that shows a McCulloch-Pitts Processing Element,[1] which has a signum function as it's Activation function,

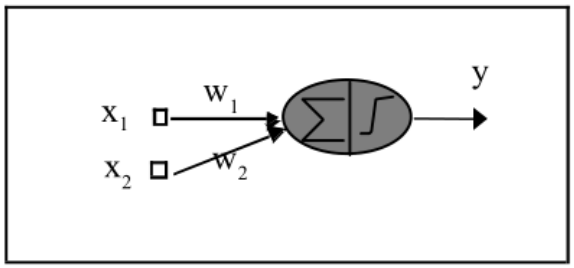

Figure 1: 2 input, 1 output McCulloch-Pitts PE

As can be seen, the inputs are weighted and are passed through a summation element. the output of which is sent to an Activation function. Given by,

$$y = f(net) = f\left(\sum_i x * w_i + b\right) \quad \rightarrow Eq.1$$

Where f is the activation function .The strong point of ANN is the ability for this activation function to be Nonlinear, while a linear function can also be used, it makes the presence of the Activation function redundant as the weights themselves could do that better. This Non-linear function determines the separation surfaces in the Input space. An Artificial Neural Network consists of many such Processing Elements.

B. Perceptron

The is a fully connected set of inputs and output processing elements. The earliest of this was the Rosenblatt's perceptron Which consisted of a *d* input , *m* output layer (*d* and *m* are set based on the problem to be solved) . The perceptron is described in *Figure 2*,

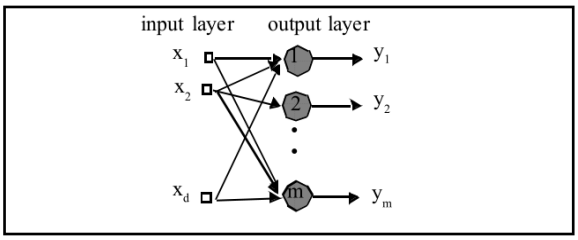

Figure 2: A general no hidden layer perceptron

As can be seen, the input and output layers are fully connected, with Eq.1 and after each output is obtained (the forward propagation), the weights *w* in the link, which are not mentioned in the Figure, but exist are modified by an update equation (the back propagation), given by

$$\Delta w = -\eta \left(\frac{\partial J}{\partial w}\right) = \eta * \varepsilon_p * f'(net) * x_p \qquad \rightarrow Eq.2$$

For any PE (p) given by *Eq.1*. This is also called the delta rule , which here is the Stochastic Gradient descent (SGD) algorithm . Where *w* is the weight, as mentioned η is the step size for the learning rate and f is any activation function.

However, the 2 layer (i.e) Input and Output Layer perceptron, has noticeable disadvantages, the most evident of which, is its inability to classify a feature space into more than two classes , and also only using a linear discriminant function (When used with a sigmoid function such as *logist* function or the *tanh* function) . We then include one more layer in between the input and the output, called the Hidden Layer. With this we move into multilayer perceptrons.

C. Multilayer Perceptron

A multilayer perceptron has one or more Hidden layers in between , each governed by its own set of parameters of weights and its choice of activation function. An Example Block diagram is given below,

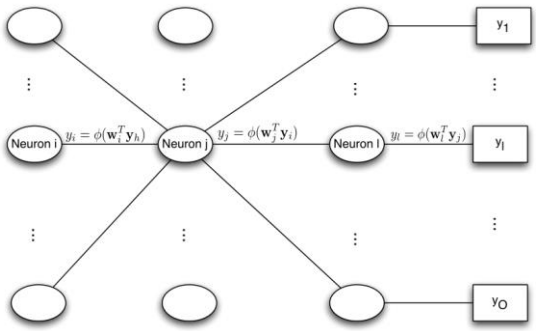

Figure 3: General Diagram of a Multilayer perceptron

Where *y* is the output from each PE, *w* are the weights and ϕ is the activation function.

1) Forward Propagation:

In the forward propagation phase, the input traverses through the network, at each PE following Eq.1 , where x is the input to that PE and y is the output from the PE . Once it reaches the end or the final output layer, It is compared with the labels (the desired response) and Using some Cost function , usually Mean Squared error , all of the weights and biases at each PE are updated , as will be seen in the back propagation.

2) Back Propagation:

The error between the estimated classification and the desired label is used to update the weights and other parameters along the network. This is done with a slight modification to *Eq.2*, using the chain rule, just like in the delta rule, to account for the multiple layers .

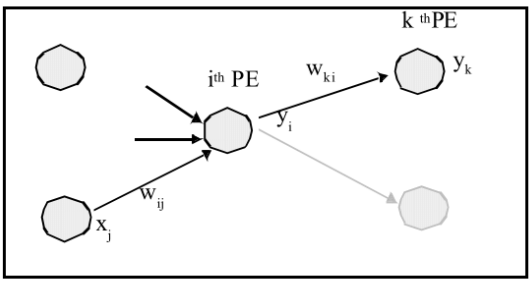

Figure 4: Detail View of a Hidden Layer Network

And the update equation for the weights can be given by

$$w_{ij}(n+1) = w_{ij}(n) + \eta f'\left(net_i(n)\right)\left(\sum_k e_k(n) f'\left(net_k(n)\right) w_{ki}(n)\right) x_j(n) \qquad \rightarrow Eq.3$$

Or shortly using the local area at that PE as,

$$\delta_i(n) = f'\left(net_i(n)\right)\sum_k \delta_k w_{ki}(n) \qquad \rightarrow Eq.4$$

The weight update, the gradient at each iteration from *Eq.2*, then becomes,

$$\Delta w_{ij}(n) = \eta \delta_i(n) x_j(n) \qquad \rightarrow \text{Eq.5}$$

This presents an equation for backpropagation that using the local error at each PE to update the weights to the links arriving to it from its input. This is done using Ordinary Stochastic Gradient Descent(SGD or GD).

In our discussions however, we will also be using a slightly modified gradient descent algorithm called the Stochastic Gradient Descent Momentum (SGDM or just GDM). The modification is that, if the gradient of weights of the next iteration is along the same direction as the current iteration, then the learning rate for that iteration is increased to take a higher step. This way, if there is a consistent gradient convergence will be higher, till it reaches or jumps over a critical point , in which case , the rate will reduce

III. Architecture for Numeric Character Recognition

This Section will cover the important factors and parameters that are required to define the ANN architecture for the given problem

A. Network Layers

In this report, two architectures are used, and they are compared to one another. We use a 1 hidden Layer MLP and a 2 Hidden Layer MLP. Usually, an ANN with more than three layers is called deep learning, and while deep learning is better, problems such a 'Diminishing gradients' etc. are evident when the gradient that is backpropagated , already a small value as it is , in this case , in decimals , is further shared and divided further into smaller values .

B. Number of Processing Elements

The number of Processing elements in each layer plays an important role in the detection of the discriminant surfaces, which directly affect how many different regions are captured in the final output of the ANN. This is a tough problem, as too less PE and there will be undertraining, and too high PE will and there will be computationally expensive and it may not even give the perfect solution due to possible overfitting of data. Measures such Early stopping may have to employed to prevent overtraining.

C. Hyper Parameters

1) Step size:

The weight update in each iteration is governed by the hyper parameter η, which is called the step size. It dictates how the length of the weight update during each step of the gradient descent. A larger η will not converge with precision and have an oscillatory learning rate, while a very small η will take too long to converge, usually multiple epochs, which might increase the computation time many-fold. Finding the right step size for the correct Learning rate is a challenge is every machine learning algorithm. In this case the step size is taken as 0.1 The next hyper parameter deals with trying to minimize this challenge

2) Momentum parameter:

As mentioned briefly in *Section II*, a variation of SGD is using a factor called momentum. This increases Rate of Learning when the gradient direction is the same for consecutive iteration, (i.e) has an accelerating effect of the learning rate. It is given by the following equation,

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \qquad \rightarrow Eq.6$$

Where α is the momentum parameter and dictates how much of an effect the previous weight update has on the speed of the current weight update . When α is made zero , then it becomes normal Gradient Descent. Usually , it is maintained at above 0.5 , in this case ,it is maintained at 0.9 unless mentioned otherwise.

D. Normalization of the Inputs

In most cases of Machine Learning, it is always a good idea to scale and normalize the input space using some technique. Usually, it is normalized to have a zero (0) mean and lie between -1 and 1 . In this case, the values of the input, which are images, typically range between 0 and 255 . This has been normalized to lie between 0 and 1. This has been done so that activation functions whose ranges are usually between -1 and 1 can pick these values up and work with them without achieving saturation. The Hyperbolic tangent function is given below,

$$hyperbolic \quad f = \tanh(\alpha net) \qquad \rightarrow Eq.7$$

E. Activation Function Used

The activation function used here is a sigmoid function. Specifically, a Hyperbolic Tangent function is used. While the logist function is also another sigmoid function, the logist function varies from 0 to 1 in the y axis, which does not capture properly the variance in the negative region of the feature space .While the Hyperbolic tangent(tanh) ranges from -1 to 1 , thus this is used in this report . Both for the hidden layers and the output layer.

F. Mini Batches:

The use of minibatches in the training of the function gives a nice trade off between the advantages of Batch Learning vs Online Learning. Here for GD the minibatch is taken as 128 , for GDM it is 64.

G. Feature Extraction

The MNIST dataset has 60000 images for training, and 10000 images for testing. Each of these images have a size of 28x28. This would cause 784 input elements and when multiplied with the number of elements in the first and second layers, the number of variable parameters such as weights and biases would be a very high value for a low sample of 60000 images. Thus it is important to reduce the input space. Two methods can be used.

1) Down-sampling:

In this report, the entire dataset has been down-sampled by a factor of 2 , which would lead to images of size 14x14 .This brings down the input elements to 196 which is comparatively better than a high value of 784.

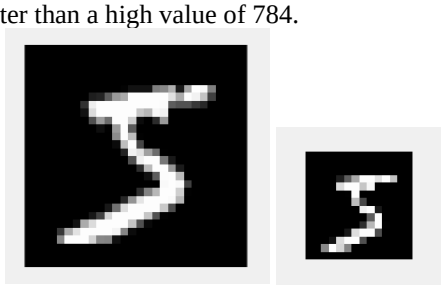

Figure 5: Down-sampling the handwritten number 5, left is the actual 28x28 scaled for reference, right is the 14x14 down-sampled image , also scaled correspondingly for reference.

While another down-sampling could be done to make it 7x7, it was seen to produce worse results and it was not visually decipherable either, thus leaving that option out.

2) Principal Component Analysis(PCA):

It may be the case that even the down-sampled data of 14x14 has too many features or components, One may wish to further optimize the data by removing a few unimportant components. A few evident vectors that provide no information are, the outer rows an columns of the images as they are almost always a part of the background , as the actual number in the image is centered around the center rows and columns , as evident in *Figure 5*. Thus, doing PCA and picking the eigen vectors with that give the most meaningful data is done to improve the feature space.

For this report the PCA is done to retain 95 components for a 1 Layer MLP and 91 components for 2 MLP

IV. Results

This Section presents plots of the two different architectures mentioned in III and shows learning curve and Class confusion matrix for the best values. MATLAB's *nprtool* is used to run the neural network. The parameters are varied across a learning rate of 0.1, using GD and GDM using Performance function as Mean Squared Error (MSE).

Note: Class 1,2,3,4,5,6,7,8,9 denote numbers 1 to 9 respectively, but Class 10 denotes the number 0.

Other Techniques such as Early stopping, where the code has included a part to monitor Validation Fails. As the validation fails increase, the Network terminates early.

A. One Hidden Layer MLP

The results are run for 1000 epochs and the values of Learning curve and the Receiver Operating Characteristic for the best number of PE and when run on the Testing data.

Accuracy -Using Down-sampling		
Number of PE	Using GD	Using GDM
30	64.2	65.8
50	71.4	69.0
80	75.5	79.2
100	80.9	80
120	81.6	83.2
160	62.7	65.6

Table 1: Accuracy comparison using down-sampling

When done with down-sampling to a 14x14 image, the best value is obtained at around 120 PE in the hidden layer.

The below results are for 120 PE in the hidden layer the Class confusion matrix.

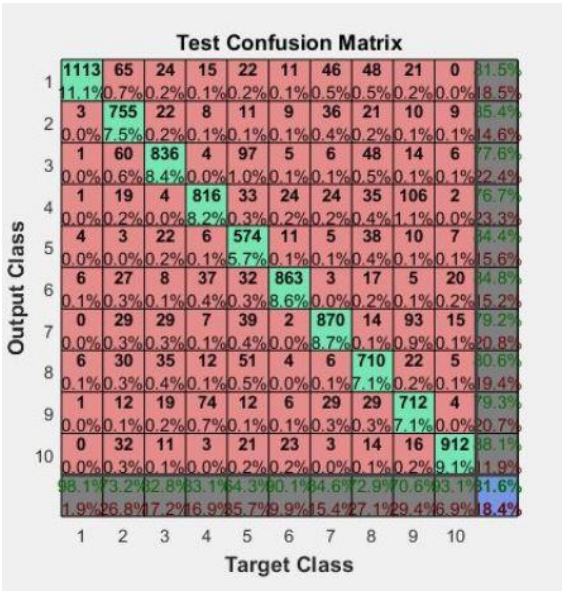

Figure 6: Class confusion matrix on Test Data using GD

When using GD, *Figure 6.* details the class confusion matrix

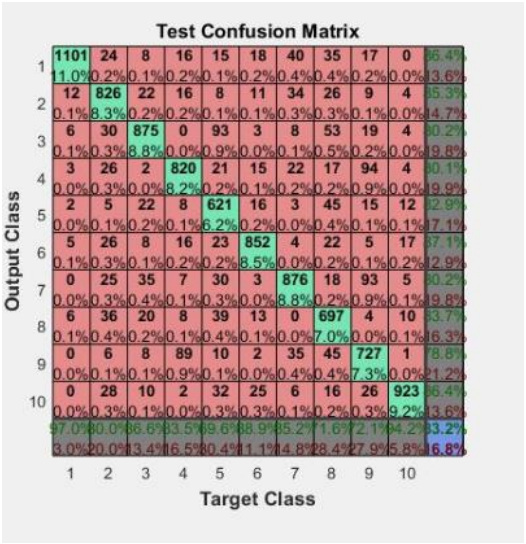

Figure 7: Class confusion matrix of Test data using GDM

Figure 7 discusses the class confusion matrix when done using GDM (Gradient Descent with Momentum)

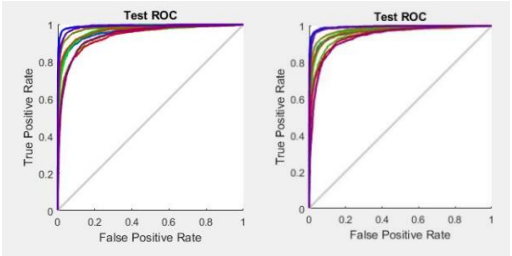

Figure 8: ROC of Test data on GD(left) and GDM (right)

The above *Figure 8* , discusses the ROC for both the GD and GDM , as can be seen , As long as the GD is close to the left top of the graph , the accuracy is good . The accuracy can be calculated by determining the area under the curve. This Area Under the curve is readily available in the Bottom right part of the Class Confusion Matrix. This value is already an average of the Accuracy of each class, so it can be seen as a group average of the Area under the curve of the entire set of classes.

Accuracy-Using PCA		
Number of PE	Using GD (in %)	Using GDM(in %)
20	75.1	73.6
40	81.0	75.6
60	87.2	82.1
80	87.2	85.2
100	88.8	87.0
120	79.6	80.6

Table 2: Accuracy comparison using PCA

Below are the class confusion matrix and plots given for PCA tested on the Test data.

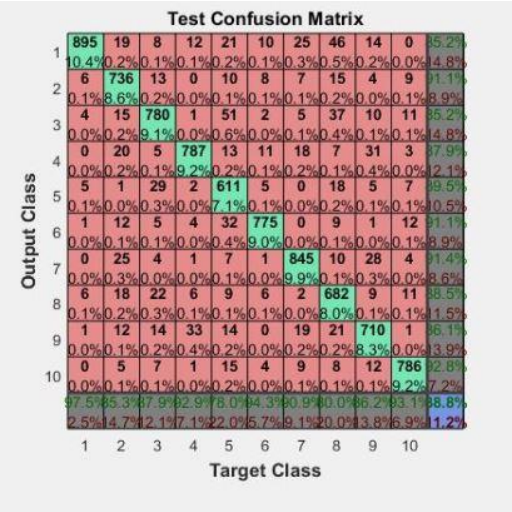

Figure 9: Class Confusion matrix for GD

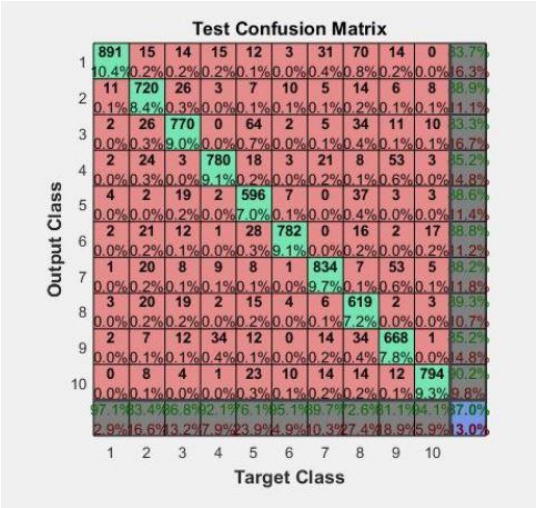

Figure 10 : Class Confusion matrix for GDM

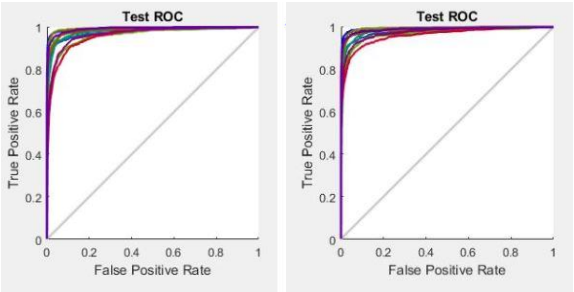

Figure 11 : The ROC plot for GDX(left) and GD(right)

Figure 11, Similarly describes the Accuracy for all 10 classes, But Using PCA with 85 components extracted from the data input of 28x28.

B. Two Hidden Layers

For the case of two hidden layers it can be seen that the Performance function of Mean Squared Error does not perform well even for 2000 epochs , For this case , this part will use the 'Cross Entropy Function' and Compares between GD and GDM.

The data set has been down-sampled to 14x14 then PCA has been done to retain 95 features. The results have been Tabulated for various combinations of PE in Hidden Layer 1 and Hidden Layer 2. It should be noted that due to the ability of GDM to move wider and quicker around the Feature Space , step size is taken to be 0.5 as, it might become necessary due to the problem of diminishing gradients or vanishing gradients in case of multiple hidden layer networks..

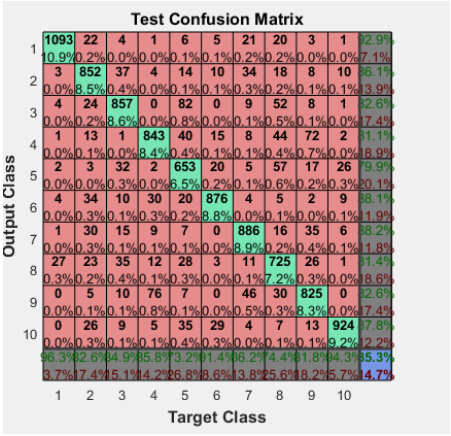

Figure 12 : Confusion Matrix for GD for 30-20 In Hidden Layer

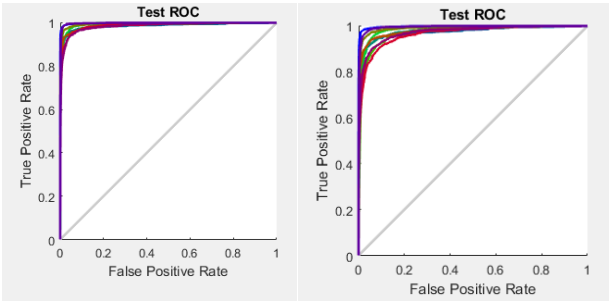

Figure 13: ROC for GDM (left) GD (right) with 30-20 in Hidden layer

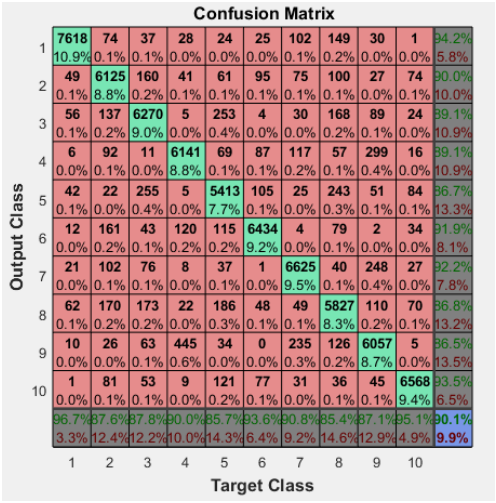

Figure 14 : Class Confusion of GDM with 30-20 Hidden Layer

Accuracy-Using both PCA and Down-sampling (in %)			
PE in First Hidden Layer	PE in second Hidden Layer	Using GD	Using GDM
30	20	85.3	90.1
40	30	82.1	88.1
50	40	77.9	87.5
60	50	72.3	81.4

Table 3: Accuracy comparison using PCA

Table 3 presents the Accuracy of for the combination of PE given in the first two columns of the table to the left. The other columns give the values obtained using Gradient Descent (GD) and Gradient Descent with momentum(GDM).

C. Best Values

For comparison with best values, I have also compared using GDX, which is Gradient descent with momentum and adaptive learning for the step size. The is usually a quicker convergence with better accuracy . This hasn't been discussed in the report because it was not dealt with in the class.

For 1 Layer MLP (100 PE in hidden layer) (in %)			
Algorithm	GD	GDM	GDX
For PCA using 100 PE	88.8	87.0	91.6
For Down-sampling using 120 PE	81.6	83.2	87.8

Table 4: Best value Obtained for 1 Layer MLP

For 2 Layer MLP(30 -20 PE in hidden layer) (in %)			
Algorithm	GD	GDM	GDX
For PCA and Down-sampling together	85.3	90.1	96.2

Table 5: Best values for 2 Layer MLP

Again, it has to be noted that, while better results could have been obtained using Training Function such as GDX and using performance functions like Cross Entropy, and also using a Softmax activation function in Output Layer. All these have given much better values. But these techniques and options have been neglected as these weren't discussed in detail in class. Instead the results have been done using a Hyperbolic Activation Function, using MSE as the Cost or Performance Graph

V. Discussion

The report has been done for two architectures, and both will be discussed under different subheadings.

1) For 1 MLP

As can be seen in *Table 1*, the variation of the accuracy changes as the number of PE changes. The power of the

sigmoidal PE lies in drawing discriminant lines in the feature space.

On comparing the difference between answers of Down-sampling and PCA, PCA has better values than down-sampling because in PCA we have projected the input space into a subspace and extracted only the best components , that cover the maximum variance .This comes to 95 components from the total 784 inputs .

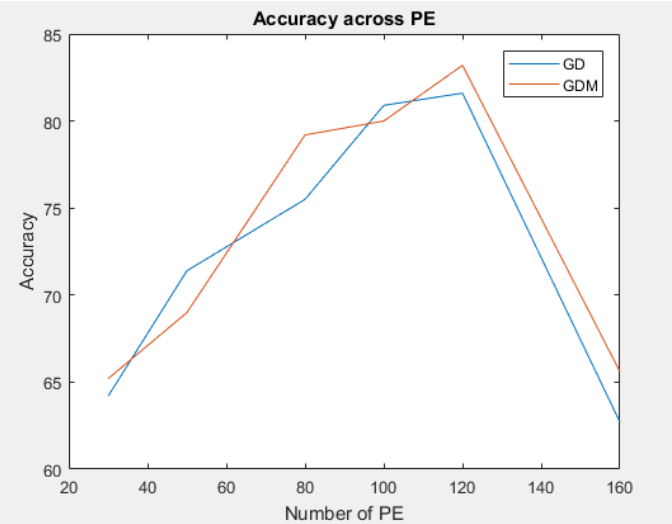

Figure 15: Plot of GD vs GDM using Down-sampling

Within Down-sampling *Figure 15*, when compared amongst GD and GDM, it is to be noted that GDM is expected to perform better for certain values with the same step size . This is not however the case in *Table 2* for PCA , In fact GDM does work better for PCA but with a varied step size . So when it comes to momentum , working with the same step size as for GD does not give better results . In fact ,It can be seen that for a lesser step size , of about 0.05 , GDM gives an accuracy of 89.9 for 100 PE.

The curve in *Figure 15* increases, then decreases because of the fact that when there are less number of PE , there is a lesser number of discriminant lines in feature space. It can be seen analogous to detecting the features using a feature selector . When there are too less features , the accuracy (which is the performance metric) is low and at a point it reaches its maximum value , and reduces down to a low value as PE number increase above a certain amount . This is discussed in Homework 6[] during Feature Selection using Fisher discriminant ratio as the performance metric, this is similar to the problem in Homework 6.

2) 2 Hidden Layer MLP:

In case of the Hidden Layer , *Table 3* displays a better curve for the Accuracy . On comparision with GD and GDM , it can be seen that GDM performs better than GD in almost all cases.

The curve for the accuracy values is not plotted because there are 2 hidden layers with varying PE . However this pattern is visible and The value of accuracy is maximum over 30 -20 PE , its confusion matrix and ROC are discussed in *Figure 12 , 13 ,14* . While it is expected that 2 MLP will work better than 1 MLP , due to the maximum bounds given on epochs as 1000 and the computational complexity 2 MLP is comparitively lesser than that of 1 MLP.

Common Discussion for both 1 MLP and 2 MLP. It can be seen that there is confusion between the number 1 and 7; 3 and 8 ; 8 and 0 ; 4 and 9 . This can easily be explained by the fact that, both these look visually similar, because of the shape, so it is expected , there be a confusion between these. Also, the Results from which the code has been obtained is attached to a pdf file and will be attached along with the file. The 'mainscript.m' includes program to down-sample and do PCA. while the 'nprscript' is the script generated by the nprtool of Matlab. The parameters are changed and modified in this file to achieve the results. While running, just normalize the data properly and the results displayed can be obtained.

VI. Conclusion

In Conclusion, It can be seen that for 100 PE in the hidden layer , 1 layer MLP works better . For 2 Layer MLP 30-20 PE in the hidden layer .it can be seen that GDM works better than GD for general cases , however there may be some differences in certain conditions based on the step size.

Future Work can be done on using various topologies for ANN . There has been research on using various different cost functions and activation functions. Section IV discusses this in brief. The one drawback with ANN is the computational power required and the time needed to train and run bigger datasets . This problem was faced in this report , where there wasn't enough time to work on the data and to reduce the expense of time and computation step size was taken as comparatively large . For Smaller step sizes , it is expected to get better results.

Acknowledgment *(Heading 5)*

I confirm that this assignment is my own work, is not copied from any other person's work (published or unpublished), and has not been previously submitted for assessment either at University of Florida or elsewhere.

References

[1] J. C. Principe, N. R. Euliano, and W. C. Lefebvre, "Neural and AdaptiveSystems: Fundamentals Through Simulation." Wiley: Hoboken, NJ, 2000.I.S. Jacobs and C.P. Bean

[2] Christopher Bishop,Pattern Recognition and Machine Learning, Springer 2006.

[3] https://en.wikipedia.org/wiki/MNIST_database

[4] https://www.safaribooksonline.com/library/view/deep-learning/9781491924570/ch04.html

[5] "L25_Notes_EEL5840_F17.pdf", Elements of Machine Intelligence,University of Florida

[6] “L26_Notes_EEL5840_F17.pdf, EEL 5840”,Elements of Machine Intelligence ,University of Florida

[7] “Lecture notes and Info session,” EEL 5840,Elements of Machine Intelligence , University of Florida.