Multilayer Perceptron with MNIST Dataset

Dylan Stewart
Department of Electrical and
Computer Engineering
University of Florida
Gainesville, Florida 32611
Email: d.stewart@ufl.edu

***Abstract*—In this paper we develop a Multilayer Perceptron (MLP) with 100 Hidden Units to classify the MNIST Dataset. We begin by reviewing the history of Artificial Neural Networks and their ability to solve complex problems. Methods for applying Principal Component Analysis (PCA) and Stochastic Gradient Descent with Momentum are explained to assist in learning the weights to classify the dataset. Activation functions that are compared within our study are then introduced along with their representative equations. Parameters for the network are also discussed along with their intracicies. Methods for parameter selection and training the network are then developed. We lastly discuss the results on the MNIST Dataset based upon several metrics including percent classification and cost function error and elaborate on the complex parameter choices and their effects on building a network. Overall, the results for each activation function perform well on the MNIST test set with each having high classification percentages.**

***Keywords*—Artificial Neural Networks, Multilayer Perceptron, Stochastic Gradient Descent, Momentum, Principal Component Analysis, Classification**

I. Introduction

During the last century, machine learning has been increasingly important in pattern recognition. With advances of artificial neural networks (ANNs), many problems can be addressed today that were unsolvable 50 years ago and previously (such as the XOR problem). One problem that can be addressed by an ANN is handwriting recognition. For many real world applications i.e. mailing packages it is of great importance for machines to recognize handwriting and classify it correctly to mail packages in an orderly fashion. To address this issue, we use the large and complex MNIST dataset (in this case study we use 70,000 images of digits 0-9) and replace hand-crafted feature extraction with a MLP. Before we discuss the specifics of the ANN we developed, it is important to review the advancements of ANNs.

Long before the advancement of present day convolutional neural networks, Rosenblatt's perceptron [1] was proposed as the first model for supervised learning. An extension of McCulloch and Pitts model of a nonlinear neuron [2], the perceptron, figure (1), uses linear combinations of inputs with an externally applied bias [3]. The sum is applied to a hard limiter (activation function) that produces an output of 1 if the sum is positive and -1 if the sum is negative. This element when trained on a two class system can develop a linear decision boundary between classes based on the weights that are updated using gradient descent.

After Minsky and Papert [3] proved the limitations of Rosenblatt's perceptron, Rumelhart, Hinton, and Williams introduced backpropogating error to detect patterns [4]. Most recently, Hornik Stinchcombe, and White showed that non-linear problems can be solved using Multilayer Perceptrons (MLP) [5]. Because of these advancements, we will use a MLP network to attack the MNIST dataset problem of classifying digits. While advancements in ANNs have led researchers to convolutional neural networks and other networks with many hidden layers [6], these will not be discussed in this paper.

The remaining of the report is organized as following: in Section II, the methodology to set up the network and pick parameters is described. Section III contains results from using the different activation functions. In Section IV we discuss the results, and finally in Section V we derive conclusions from the discussion.

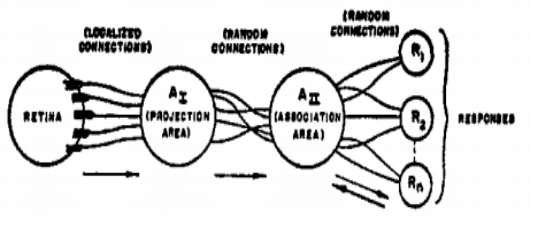

Fig. 1. Rosenblatt's Perceptron

II. Methodology

A. Data and Code

Within this paper the dataset used is the MNIST dataset. For our experiments we chose to use the first 50,000 images of the MNIST dataset for training with the following 10,000 for validation and final 10,000 for testing. All methods of tuning parameters were based solely on the training and validation sets. The test set was used only as a final test to produce results.

The base code for this project was accessed and modified from David Stutz's seminar paper on Neural Networks [10]. The code was edited to fit the specifics of this paper including biases, all plots shown in this paper, confusion matrices, and more.

B. Artificial Neural Network Structure

To understand how the ANN learns to classify the dataset, it is important to know about the structure and operations of an ANN. Figure (2) illustrates a simple representation of an ANN. There is an input layer where the inputs are loaded and

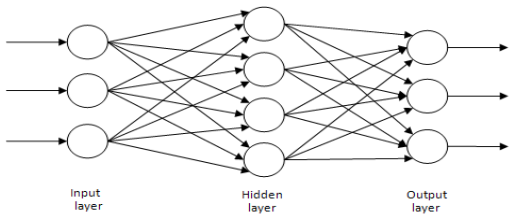

Fig. 2. ANN Architecture

then given weights to the hidden layer. At the hidden layer, each node contains the net of the inputs upon that node and the bias associated with the node. The output of each hidden node is a nonlinear activation function of the inputs with the bias. This is better shown in the equation (1).

$$f_{node} = f(\sum_{i}^{N} x_i w_i + b_{node}) \tag{1}$$

Where N is the number of inputs and node specifies which node of the hidden layer. While this is the function out of a node, when an input is propogated through to the ouput layer, the input to the output layer is a combination of the outputs of the previous layer also evaluated within an activation function with a bias at the specific output node. The function out of an output node can be illustrated in equation (2).

$$f_{node} = f(\sum_{i}^{H} f_i w_i + b_{node}) \tag{2}$$

Where f_i is each hidden node output, w_i is each weight for the hidden node output, and H is the number of hidden nodes. While these functions show how to forward propogate, it is important to understand what happens when the error is evaluated to improve the weights of the ANN. In the end, the weights and biases will provide decision boundaries that classify the dataset.

C. Stochastic Gradient Descent and Momentum

Within ANNs, Stochastic Gradient Descent (SGD) can be used to update the weights and biases to decrease output error. The error function used in our ANN is Mean Squared Error as shown in equation (3).

$$E(w) = \frac{1}{N} \sum_{n=1}^{N} (y(x_n, w) - t_n)^2 \tag{3}$$

where $E(w)$ is the error given specific weights for the samples, w are the weights, $y(x_n, w)$ are outputs, t_n are desired labels, and N is the number of samples used in a mini-batch. This error function produces a multi-dimensional error space when relative to many weights. The method of SGD without momentum is to move the weights a small step in the direction of the negative gradient, so that

$$w^{\tau+1} = w^{\tau} - \eta \nabla E(w^{\tau}) \tag{4}$$

where η is the learning rate. After each update, the gradient is calculated for the new weights and the process is repeated. At each iteration, the weights are moved in the direction of greatest rate of decrease of the error. This algorithm does not necessarily lead to good minimums and is inherently reliant upon initialization and the learning rate parameters [8]. This will be further discussed in section IV. In cases where the error surface can be complicated and contain several local minima (see figure 3), momentum within gradient descent can provide a way to escape local minima [9]. Within our network we propagate a random mini-batch of 50 samples (using a different number would change the weight updates) before evaluating the error using a modified SGD that includes momentum (equation 5).

$$\Delta w^{\tau} = \eta \nabla E(w^{\tau}) + \mu \Delta w^{\tau-1} \tag{5}$$

where μ is the momentum parameter $0 \leq \mu < 1$ with higher momentum giving more weight to the previous change in the weights.

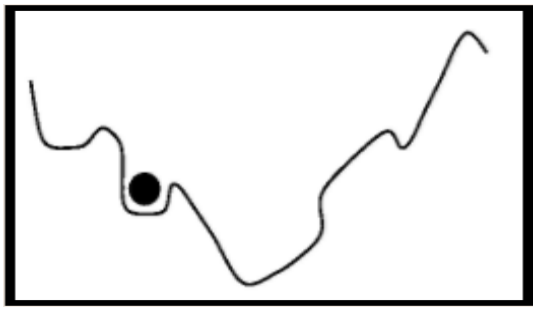

Fig. 3. Complicated Error Surface

Within our ANN, an input is propagated forward then the error is calculated and applied to the weights and biases using stochastic gradient descent with momentum. The update equation for the weights is shown in equation (6).

$$w^{\tau} = w^{\tau-1} - (\eta \delta X_i + \mu \Delta w^{\tau-1}) \tag{6}$$

where X_i is the previous input into the node and $\delta = f'(X_i)e_i$ where e_i is the component of the error between the desired and output response from the previous step for that weight. When momentum is unused, the equation becomes basic SGD. The bias updates for SGD with momentum are quite similar to the weight updates except the input X_i is set to 1 for the biases so the equation will not be illustrated in this paper. The base algorithm of an ANN is provided below.

D. Activation Functions

Excluding the input layer, each node of a neural network contains a nonlinear activation function that applies to the net of the inputs upon the specific node. Although there are many activation functions, we will focus on three in this

Algorithm 1 Artificial Neural Net Algorithm

Input: 784-Dimensional Handwritten Images
Output: Classification Labels
Define number of hidden nodes order, step size for weights, step size for biases, initial biases, initial weights, momentum parameter, batch size, epochs, error function, and activation function

```
for every epoch do
    for every batch do
        Forward propogate input
        Calculate output
        Calculate error
        Update Weights
        Update Biases
    end for
end for
```

paper: Rectified Linear Unit (ReLU) [7], Log-Sigmoid, and the Hyperbolic Tangent.

The Rectified Linear Unit was first introduced by Hahnloser in 2000 [7]. The formula is shown in equation (7).

$$f(x) = max(0, x) \tag{7}$$

The ReLU is the currently the most popular activation function [8] used within ANN applications and has the simplest derivative of the activation functions we will use in this paper.

$$\frac{\partial f}{\partial x} = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

The second activation function we will experiment with is the Log-Sigmoid activation function. The formula is shown in equation (9)

$$f(x) = \frac{1}{1 + \alpha e^{-x}} \tag{9}$$

The parameter alpha can be alternated to produce a different slope for the sigmoid function, however, it will not be varied in this paper. The derivative of the sigmoid is shown in equation (10).

$$\frac{\partial f}{\partial x} = f(x)(1 - f(x)) \tag{10}$$

The last activation function we will experiment with is the hyperbolic tangent function. The formula is shown in equation (11).

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{11}$$

The derivative of hyperbolic tangent is shown in equation (12).

$$\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh^2(x) \tag{12}$$

E. Principal Component Analysis

When the input data is high dimensional, it is a good idea to look into lowering the dimensionality to speed up computation. One technique that is widely used for dimensionality reduction is Principal Component Analysis (PCA), also known as the *Karhunen-Loève* transform. PCA is an orthogonal projection of the data onto a lower dimensional linear space, so that the variance of the projected data is maximized [8]. For our purposes we performed PCA on the input data and found that the first 150 components contained greater than 95 percent of the variance so we used only the first 150 components in our tests. The way we determined this was by summing the first highest 150 eigen values of the covariance matrix and dividing this sum by the total sum of the eigenvalues. The first 150 eigen vectors are then used to project the data to a lower 150 dimensional space. The proof for this method can be found in Bishop's Pattern Recognition Book [8] and is too lengthy to provide in this paper.

F. Parameters

Within ANN several parameters must be selected to train the network. These parameters include learning rates, momentum, number of hidden nodes, epochs, and batch sizes for mini batch learning. Learning rate is used within weight updates and bias updates. If the learning rate is too large for an update, the optimum weight may be missed if it is too small, the weights may take infinite steps to converge and can get stuck at a local optima instead of a global optima. The momentum term must also weight the previous change of the weights and the biases. If the momentum is unused and set to 0 then it is classical SGD, if it is set to some value such as 0.8 then the previous change will effect the next change. The number of hidden nodes must be selected to be able to produce decision boundaries for each class that is input. If there are too few, misclassifications can occur, if there are too many, several will be unused and waste computation time. The amount of epochs is also important, if there are too few of epochs, the network is not trained to classify the dataset. The batch sizes are also important, if the batch sizes are too small, the network will take many more epochs to learn optimal weights, if the batch sizes are too large, the network will train very slowly for a fixed number of epochs.

Each parameter was chosen based on an evaluation metric for the parameter. For the log-sigmoid activation function we varied all parameters for selection except the batch size. For the ReLU activation we varied the learning rates and momentum. Lastly, for the tanh activation function we only varied the momentum parameter. The ReLU and tanh functions are shown to give way to further discussion of parameters but given the time constraints of this paper were not fully evaluated. First we selected the learning rate of the weight updates based on the output weight tracks of the weights for each iteration and run time. We fixed the number of hidden units to 100, the learning rate of the bias updates to 0.03, the batch size to 50, and the momentum term to 0.8, and trained for 5000 epochs. We varied the weight step size from 0.001 to 0.1 in steps if 0.001 and picked the step size that produced smoothly changing weight tracks in figure (4). After selecting the step size for the weight updates, we then alternated the learning rate of the biases over the same range until obtaining relatively smooth weight tracks for the output biases while

also noticing changes of the biases over many samples. These smooth weight tracks indicate the step size is close to optimal because the weights are not jumping to different values but converging to final values gradually. After picking the learning rates based on smooth weight tracks, we then selected the number of hidden units based on error.

To select an optimal number of hidden units, we fixed the learning rates, momentum, batch size, and epochs, and varied the number of hidden units from 10 to 200 in steps of 10 and plotted final MSE for each on the training and validation sets. The reason we selected 10 as a starting point is because there are 10 digits we are classifying so there must be at minimum 10 to decide between them. We ran 10 trials of each number of hidden units and checked when the validation error did not decrease while the training error was decreasing for two standard deviations.

As the network parameters are being decided, the final number of epochs to train is very important. If the network is not trained enough, it will not be able to classify between the data. To pick an optimal number of epochs, we ran the network for increments of 100 epochs and viewed error of the training and validation sets. When the validation error is at a minimum the amount of epochs is substantial. This is also known as early stopping [8] which prevents the network from being overtrained on the training data which will cause the validation error to increase past the early stopping point.

As opposed to online learning where batches are size 1, we selected to do mini batch based on run time. The batch size was chosen to be 50 as a trade off between error and run time.

Lastly, we selected the momentum term for the bias and weight updates. We varied the momentum term fixed the momentum term at 0.8 and varied it by 0.1 to 0.9 or 0.7 based on the weight tracks and classification error to determine the best parameter.

III. Results

Within the results section we will demonstrate plots that led us to pick the specific parameters based on the training and validation sets. We will also include the final results for those specific parameters in a confusion matrix on the test set.

A. Results Log-Sigmoid

As described previously in the methods, we first set about deciding the learning rate for the weights while fixing all other parameters. The step size we chose is shown in figure (3) along with the weight tracks over several iterations. After selecting the step size for the weight updates, we then selected the bias update step size as shown in figure (5). The number of hidden units was then chosen based on minimizing the cost function for the validation set. The plot of the final error for each number of hidden units is shown below in figure (6). We selected 100 hidden units due to the validation error leveling off at higher number of hidden units.

The number of epochs to train was then chosen based on when the validation error minimized for 10 experiments while the training error was decreasing. The means are plotted for each and two standard deviations to show 95 percent confidence. From this plot in figure (7) we selected 5000 epochs due to the validation error leveling off after 5000 epochs. Another plot to demonstrate the effects of using more epochs is shown in figure (8). The two standard deviations are not shown due to their high percentage of overlap causing the

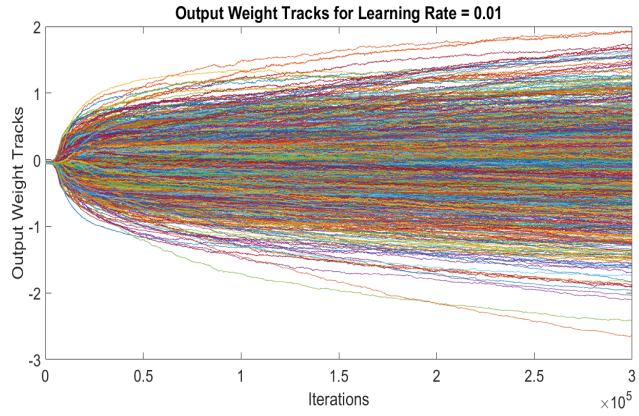

Fig. 4. Output Layer Weight Tracks: 5000 Iterations of 50 Batches

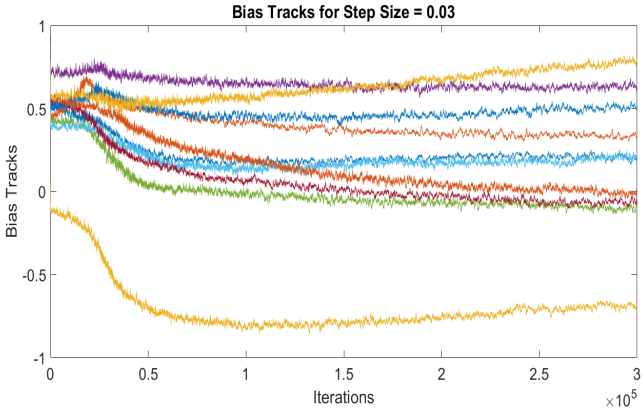

Fig. 5. Output Layer Bias Tracks: 5000 Iterations of 50 Batches

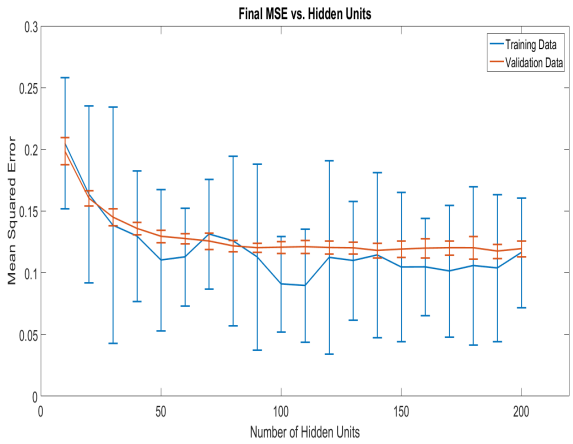

Fig. 6. Final MSE for Training and Validation Sets for 10 experiments

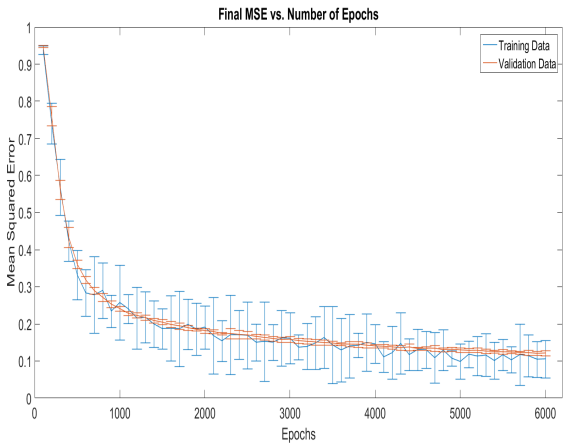

Fig. 7. Final MSE for Training and Validation Sets for 10 experiments at Each Number of Epochs

plot to be indiscernable. From our plots the parameters in table (1) were chosen. The final results on the test set are shown by the confusion matrix in figure (9). Lastly, to show the effects of hidden units and number of inputs, the classification accuracy on the test set with all 784 inputs is shown in figure table (1).

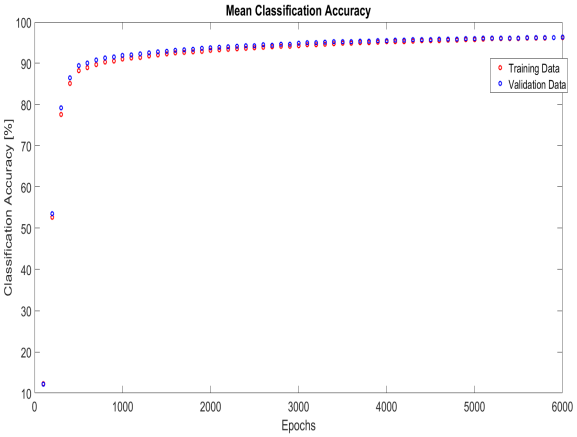

Fig. 8. Classification Accuracy for Training and Validation Sets for 10 experiments at Each Number of Epochs

B. Results ReLU

The same procedure was followed to determine the best parameters for the ReLU. The test confusion matrix is shown in figure (10).

C. Results Tanh

We used the same parameters except for the momentum parameter from the ReLU parameters to have a small test of the tanh activation function. The test confusion matrix is shown in figure (11).

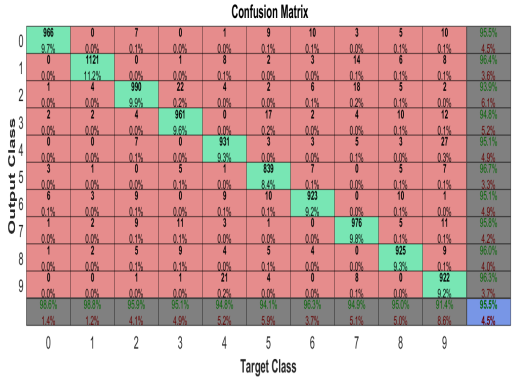

Fig. 9. Confusion Matrix for Log-Sigmoid Parameters on Test Set

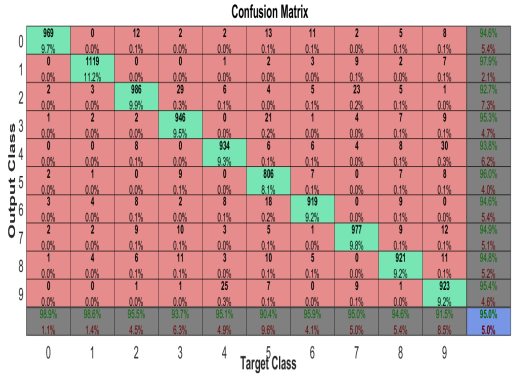

Fig. 10. Confusion Matrix for ReLU Parameters on Test Set

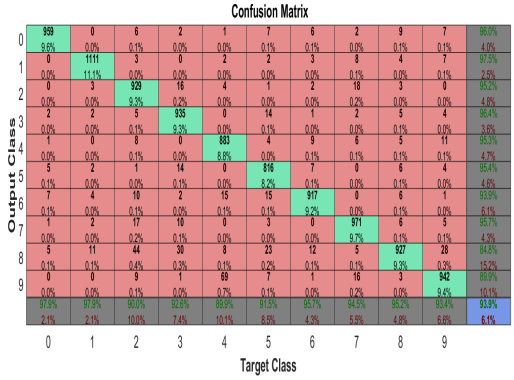

Fig. 11. Confusion Matrix for Tanh Parameters on Test Set

IV. Discussion

It is infeasible in the time we were given to complete these experiments to select truly optimal parameters for all three activation functions based on the amount of influence all parameters have on the final classification and complex intricacies of the networks. However, the parameters selected for each activation function produce sufficient classification percentages (all greater than 90%). Each step of the methodology is influenced highly on the other parameters, this will be discussed further.

When deciding the learning rate based on the output weight

TABLE I
MULTILAYER PERCEPTRON PARAMETERS

Parameter	Log Sigmoid	Log Sigmoid	ReLU	Tanh
Weight Learning Rate	0.01	0.01	0.0003	0.0003
Bias Learning Rate	0.03	0.03	0.005	0.005
Hidden Units	100	100	100	100
Momentum Parameter	0.8	0.8	0.7	0.9
Epochs	5000	5000	5000	5000
Batch Size	50	50	50	50
PCA Components (784 Total)	150	784	150	150
Test Classification	95.5%	90.3%	95%	93.9%

tracks, the weights need to be changing to show learning but they also need to be changing smoothly so that they may not be jumping over a potential optima. When looking at 100 hidden units and 10 output units, the error surface is of much more complexity than dealing with one unit. Initialization of the network can also greatly effect if a good optima is found in the outcome. To deal with this, we randomized the initial weights and ran several tests but in reality it is not likely we found a global optima because SGD is a greedy method. By no means is our learning rate the perfect parameter, a different learning rate would just require more or less epochs and similarly several tests to verify the selection as a good choice. The momentum did not seem to effect the weight tracks much and will be discussed more in the bias track portion of the discussion.

Selecting the bias update parameter was tricky because similarly to the weight update we want the biases to learn quickly but not jump around and miss an optima. After experimenting with momentum for the bias, the momentum showed significant improvements for the bias tracks. The final bias parameter was selected to allow the majority of the biases to stabalize around a small range while a couple of the biases would continue changing regardless of how many epochs we ran the network for. The momentum was kept constant for both the weight and bias updates because it did not effect the weight updates very much compared to the bias updates. As mentioned previously, these parameters are not the only parameters that would work for the network because with a different learning rate, more or less epochs could be used to reach optimal biases.

We needed to run several experiments for each parameter and specifically the number of hidden units. Due to the randomness of the initializations of the network (weights, biases, and random sampling) it was necessary to look at the validation error and training error after training for several numbers of hidden units several times. At a certain point (after 100 hidden units) the validation error would have higher variance and then increase in the output error. Due to these factors we selected not to use more than 100 hidden units. As with every other parameter we have discussed so far, this is not necessarily the optimal parameter because with a higher number of hidden units we could use more epochs to train those units. However, with a high number of hidden units, eventually there will be units that go unused. If biases were unused, a high number of hidden units would be useful to account for those biases, but because we used biases and the biases were updated in SGD with momentum, it eliminates the need for such a high number of hidden units.

We investigated many epochs to train for early stopping. After running several experiments (for the same reasons listed previously) and looking at the final error for the training and validation datasets we noticed the validation error does not decrease very much after 5000 epochs. Shortly after 5000 epochs the validation error will increase. In order to not overtrain the network, we selected to stop the training at this amount of epochs. As with every other parameter in the network, this does not necessarily work for every other set of parameters. If there were a higher number of hidden units, the amount of epochs to train the network would increase, similarly if there were less hidden units, it may take less epochs to train also. This amount of epochs also depends highly on the batch size.

When training a network with such a high number of parameters to vary, it is of good practice to fix certain parameters that do not have as high of influence on all other parameters. For this paper, we selected not to vary the batch size and to use mini-batch learning with 50 samples propagated at a time. If we had decided to use only 1 sample at a time, the amount of epochs to learn would arguably be more to obtain the same amount of learning. This would be considerably closer to online learning in the case of updating for each sample. If the batches were greater it would take less epochs, but would be closer to batch learning.

Although it would be interesting to vary the number of components chosen to use to train and test, we decided to use the 150 components that contained the greatest variance. When selecting this number, PCA was performed and the sum of the variances of the first 150 contained greater than 95% of the total variance of all components. Due to this, 150 components seems like a reasonable number of components to use given that it is significantly less than the possible 784 input components from the data set and leads to a much faster trained network. While a smaller number could be used, it is a much easier argument to keep 95% or greater variance that a smaller amount of variance. Like other parameters, 150 is not necessarily the magic number of components to use, however, along with the other parameters we used it was a component that led to a satisfactory classification of the data set. As shown in the results section, if the same number of hidden units of 100 is used for 784 input components, the results are not as proficient. This is intuitively understood based on the amount of learning that must occur to map the 784 inputs to a space that can be classified by the net. A higher amount of inputs would require a higher number of epochs and possibly hidden units. This would be an interesting study to dive further into, however, it is infeasible to investigate for this paper due to the time allotted.

Lastly, as most parameters from the log-sigmoid activation

function were fixed for the ReLU and tanh activations, it brings about an interesting caviat of Multilayer Perceptrons. Although they have been shown to be highly sensitive to parameterization, good results (in the authors point of view) were obtained for the ReLU activation function from the modification of three parameters: weight learning rate, bias learning rate, and the momentum parameter. This supports the idea that for this set of experiments on the MNIST dataset, 100 hidden units is sufficient for classifying the dataset and the epochs, batch size, and PCA components were also appropriate. In addition, the only parameter changed from ReLU for the tanh activation was the momentum parameter. This resulted in only a 1.1% decrease of test classification, giving insight as to the ReLU and tanh activation functions similar behavior in these experiments. Overall, any of these three activation functions can be used to classify the dataset and the ReLU and tanh can most definitely have higher classification results if all parameters are tuned for each. Once again, it is important to recognize given the time for the experiments, only the log-sigmoid parameters were tuned in accordance with all methods described in section II.

V. Conclusions

Overall the author proposed a method of training a Multilayer Perceptron with one hidden layer to classify the MNIST dataset with three different activation functions. All three activation functions accurately classified the test set (with all having greater than 90% correct classification). Although these results are satisfactory for the author it must be noted that these results are in no way necessarily the best results possible for each activation function. Based on the amount of influence that each parameter has on the other parameters, it is infeasible to find the absolute best set of parameters to classify the dataset and entirely more likely that the parameters used in this paper convey one of many sets of close to optimal parameters based on classification rate and minimizing the cost function. With the parameters that are used: batch size, learning rates, momentum parameter(s) (if different for biases and weights), number of hidden units, epochs, components used from PCA, there are an infinite amount of possibilities of parameters and it is very difficult to model the dependency that each parameter has on the others.

Honor Statement

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] F. Rosenblatt, "The Perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, 1958

[2] R. Duda, P. Hart, and D. Stork, *Pattern Classification*, New York: Wiley, 2001

[3] S. Haykin, *Neural Networks and Learning Machines*, New Jersey:Prentice Hall, 2009

[4] D. Rumelhart, G. Hinton, and R. Williams, "Learning Representations by back-propogating errors," *Nature*, vol. 323, 1986

[5] K. Hornik, M. Stinchcombe, H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, 1989

[6] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems*, vol. 25, 2012

[7] R. Hahnloser et. al, "Digital Selection and Analogue Amplification Coexist in a Cortex-Inspired Silicon Circuit," *Nature*, vol. 405,

[8] C. Bishop, *Pattern Recognition and Machine Learning*, New York: Springer, 2006

[9] G. Orr, *Momentum and Learning Rate Adaptation*, [online] Williamette.edu, [Accessed 4 Dec. 2017]

[10] D. Stutz, "Introduction to Neural Networks," *Seminar on Selected Topics in Human Language Technology and Pattern Recognition*, 2014