Using Shallow ANN for Handwritten Digit Recognition

Matthew Bouchard  
1. Undergraduate Student, University of North Georgia, Dahlonega, Georgia, U.S.A  
Email Address: mabouc4699@ung.edu

**Abstract – The task of recognizing handwritten characters has proven itself at times to be a difficult one to solve for both humans and computers; however, the aforementioned task appears to be one that can be resolved using current machine learning tools and practices. In this project, I approach the problem by using a shallow artificial neural network which contains only three layers of neurons to solve the problem. In training and configuring the network, I hoped to develop a working solution for identifying handwritten numbers, and I report on the problems and discoveries associated with achieving the solution along the way. The significance of this project lies in the fact that it only enters the development of a shallow artificial neural network. With more complex neural networks, the possibilities are endless with what can be done by using sophisticated networks like convolutional neural networks. The goal of this project is to develop a solution for the problem of recognizing handwritten numbers by using the power of a shallow artificial neural network.**

*Keywords – MNIST, Handwritten, Neural Network, Label, Shallow ANN, Training Function, Neuron*

1. INTRODUCTION

As iterated before, recognizing handwritten characters can be a difficult task for humans and computers to manage. The current method that many organizations like that of the United States Postal Service use computer aided software (USPS system is termed HWAI for hand-written address interpretation software) [3]. The integration of this technology and other facets of machine learning makes it possible to solve these complex problems and automate the process to make better use of time; however, the question may still remain regarding how computers can automate these processes and, specifically with this project, how computers recognize handwritten digits when even humans have difficulty reading them within certain applications? The answer lies within the concept of machine learning called neural networks.

1. INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS

Neural networks (NN) mark a substantial bridge between computer science and the human brain. This bridge is reflected in the design of these NN in that neurons in a NN receive input, perform calculations on the input, and output using certain parameters, though these parameters are obviously set by the developer whereas it is automatic in the human brain. To elaborate further, the most basic NN is a simple perceptron. A perceptron is single-layer NN that classifies information using the basic elements of a NN including input values, weights, biases, summation function(s), and activation function(s) [5]. Input values represent the data inserted into the network, the weights are initialized values that are multiplied to the input values before they enter a summation function, the bias value(s) are added to all the weighted input values before being inserted into the summation function which aids in the accuracy of the neural network output, and, lastly, an activation function operates on the weighted sum to output / classify the input data. To correlate, the NN that is utilized within this project is a shallow artificial neural network (ANN). ANN’s can be described as multi-layer perceptrons in that they use the same architecture as the perceptron with exception to having multiple layers (for this shallow ANN, there is three: input, hidden, and output) [4].

Shallow ANN’s have the computational ability to solve a great array of problems including as is the case of a study performed by Al-Omari, Putra Sumari, Sadik A, et al. where they developed a NN that could identify handwritten Arabic characters [2]. Similar efficacy is hoped to be reached with this project in identifying handwritten digits using three training functions with neuron counts of 40 and 60.

1. IMPLEMENTATION DETAILS

The images and labels used in this project are first extracted from a .rar file revealing the training and testing data sets and labels. They are loaded into the MATLAB working environment using the loadMNISTLabels and loadMNISTImages functions and reveal the subsequent labels and data sets. The specification of the training data set is 784 x 60000 which represents the 60,000 28x28 (784) images available to use for training the neural network. The training labels are saved to a 60000 x 1 matrix which is transposed and converted to a dummyvar matrix for use in training the network later. The test data set is represented by a considerably smaller 784 x 10000 matrix, followed by a test label matrix of size 10000 x 1 matrix (which is also transposed for my specific case). Once the data is loaded in, the size of the training label set is saved to a variable matrix (n) for use in calculating the two portions of the training set. One portion of the matrix is set to train the function while the other portion is used in the training of the network. This is done via multiplying the (n) matrix by decimal value 0.8 and utilizing MATLAB’s built-in floor function. The result is saved to a separate matrix array named (m). This m value is used later as the iteration endpoint in the creation of the data sets for the 80% / 20% training data sets. The preceding operations are executed to satisfy the requirement of reserving 20% of the training data and labels for use in post network training testing for accuracy. After successfully loading in the information necessary for creating the shallow ANN, the next step was to format the data for use in training the network. This process was done by saving the training and testing labels to their own matrices (aids in readability) and setting the label matrices equal to 10 where they equal 0. In executing the following commands:

targets = tr\_labels;

targets(targets == 0) = 10;

targetsd = dummyvar(targets);

targets\_test = ts\_labels;

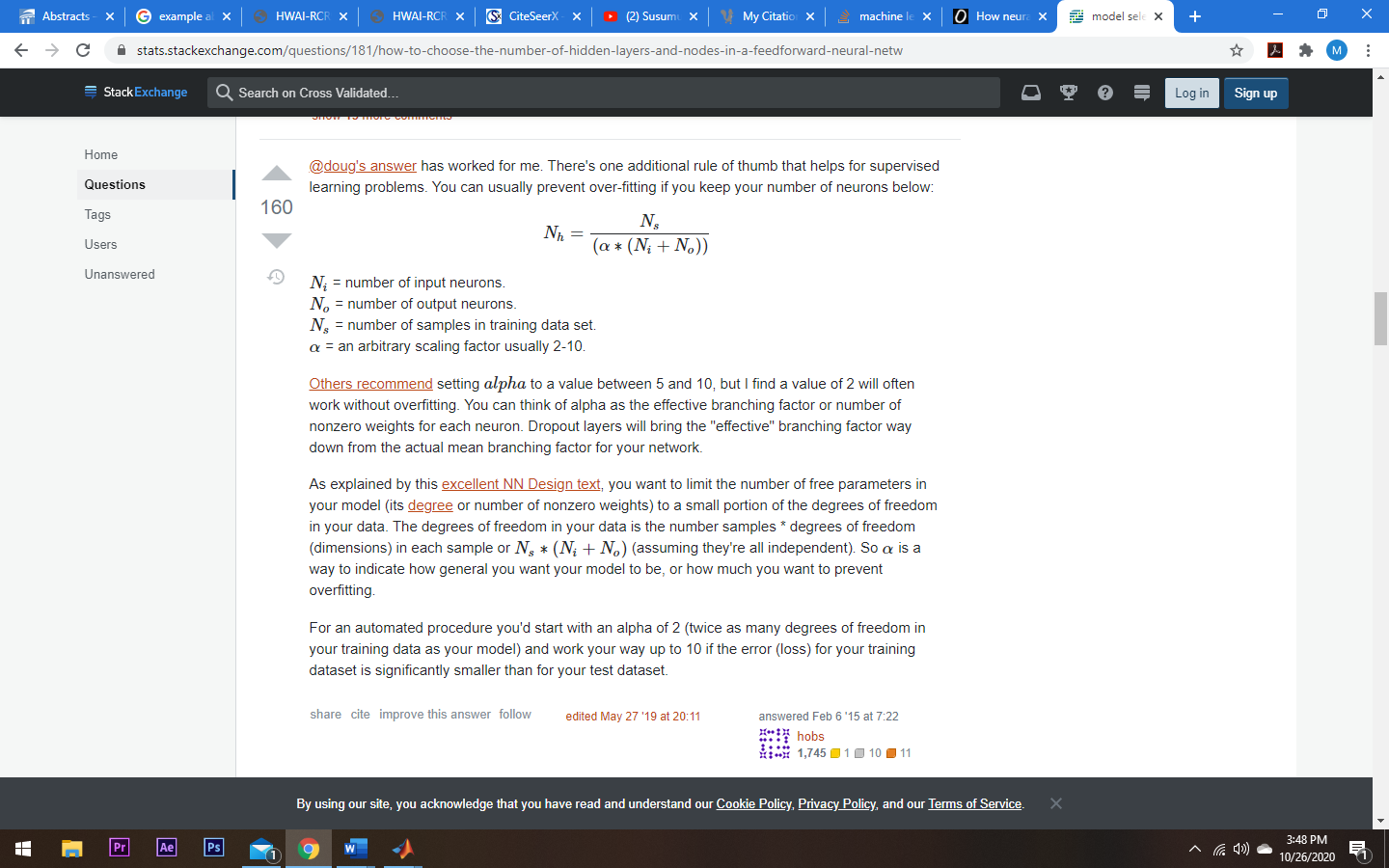
targets\_test(targets\_test == 0) = 10;

targetsd\_test = dummyvar(targets\_test);

losing the 0 values in the proceeding processes is omitted for both the training and testing label sets. The target labels for both sets are then transposed, followed by the creation of matrix arrays x1, x2, y1, and y2. X1 and y1 contain 80% of the training data (images and labels), and x2 and y2 contain 20% of the training data. This finalizes the formatting of the data for the project.

To create the neural network, two essential arrays are created: sweep and models arrays. Following along Loren Shure’s documentation on her experience with a similar project, the sweep matrix is populated with a value for the hidden layer. This is useful in the chance one wishes to make a more complex ANN with multiple hidden layers; however, for this project, it is only populated by one value at a time [1]. The scores matrix of dimensions 1 x length sweep array is populated with zeros of dimension 1 x length of sweep array. Lastly, the models cell array of dimensions 1 x length sweep array is filled with zeros.

The steps used to create this shallow ANN involved selecting a hidden layer value size (equal to the number of neurons within the hidden layer), selecting parameters for the network including the training function, and, finally, training the network. In choosing the size of the hidden layer, the generally understood function used to calculate the optimal number of neurons is represented below:



For this specific case, the number of neurons needed in the hidden layer is equal to 38 (60,000 / (2-10) (784 + 10)), which when rounded up equals 40. To satisfy the requirement of using various values for the hidden layer size, I chose to increment 20 and use 60. Continuing, the only parameter set for the neural network was the training function which was set to scaled conjugate gradient (trainscg), conjugate gradient backpropagation with Powell-Beale restarts (traincgb), and conjugate gradient backpropagation with Polak-Ribiére updates (traincgp).

The accuracy of the network is tested after it has been trained using various means to satisfy the requirement of systematically testing the accuracy of the network; therefore, the script continues with the display of the plot regression values for the initial 80% training data, 20% reserved training data, and the testing data. This is done with the following code:

plotregression(y1, models{1}(x1), 'Initial', y2, p, 'Train', targetsd\_test, models{1}(inputs\_test), 'Test')

The plotregressions for the initial, training, and testing data reveal the R value of each data set comparing the predicted values of running the data through the neural network and the actual label data (true value). The R value is considered strong if it is close to 1 and considered weaker if the resulting value is closer to 0. The results explain how closely the networks predictions lie comparable to the real label values.

The accuracy of the network is tested again using a separate script called testing\_Accuracy.m which calculates a confusion matrix for the initial 80% training, 20% reserved training, and testing data sets. Within the testing\_Accuracy script, the confusion matrix is created for each set of data using the trained network. After, the (per) matrix is utilized to set the True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) values of the confusion matrix. The TP values represent where the predictions of the output of the network for the given data set were correctly predicted and the label value was true. True negative values represent where the predictions of the output of the network for the data set were correctly predicted that the label was false (it was not the correct label). These values are used later to calculate the overall accuracy of the network’s predictions for each data set. The accuracy value cannot be generated until all the columns of the per array (TP, FP, TN, and FN) are summated; therefore, total counter variable matrices (Ex: tp\_total) are created and populated with a single zero value before iterating through a for loop to parse through all elements in per summing the total for the TP, TN, FP, and FN values. Once the for loop terminates, the accuracy of the data set is calculated using the function:

acc = (tp\_total + tn\_total)/(tp\_total + tn\_total + fp\_total + fn\_total);

This additional accuracy calculation enables the use of the confusion matrix to evaluate accuracy in tandem with the plotregression to better understand the accuracy of the network.

Finalizing the script, I set the predicted label values = 0 where they equal 10 (as is to set the predicted label values that would typically equal 10 to 0). Afterwards, a for loop is utilized to parse from 1 through 80 in order to create a sufficient number of subplots for displaying a total of 80 images with the predicted labels p as the title.

figure('Name','Digit Identification','NumberTitle','off');

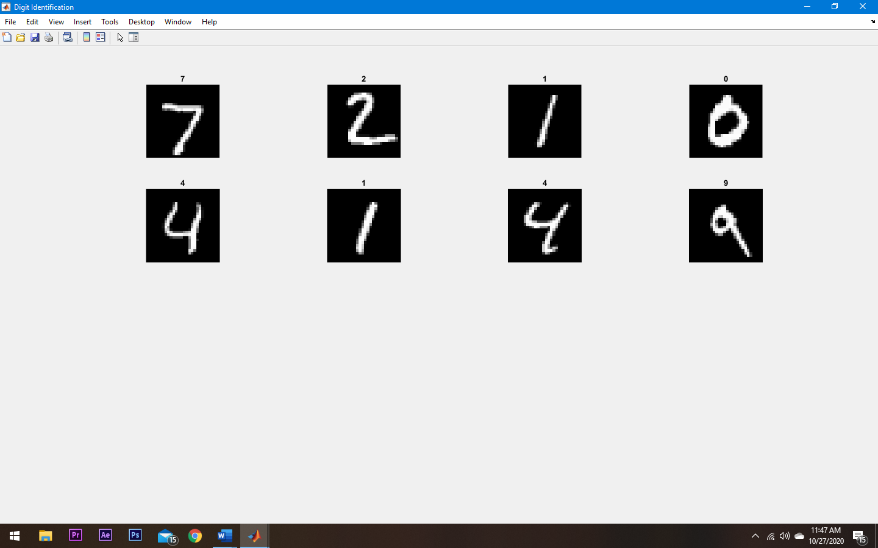
for i = 1:80

subplot(10,8,i)

subimg = reshape(inputs\_test(:,i), 28, 28, []);

imshow(subimg), title(q(i));

end

Adjusting the above code, I performed the same calculations but only display 8 images for visual reference:

The accuracy calculations for each network created using the two hidden layer value parameters of 40 and 60 and the three training functions trainscg, traincgb, and traincgp are discussed in the next section.

1. EXPERIMENTAL RESULTS

It is important to note that the same code was utilized to create the NN’s used for testing. The only parameters that were changed for each network was the HLV (Neuron count) and training functions. For the accuracy readings, the plotregressions were calculated using the trained NN and the three data sets: 80% training data, 20% reserved training data, and the testing data. The confusion matrix accuracy calculation is made using a separate script called testing\_Accuracy.m which, as iterated earlier, calculates a confusion matrix for each data set: 80% training data, 20% reserved training data, and the testing data. The per array created during the calculation of the confusion matrix is used to calculate the accuracy of the specific data set using the function displayed in the previous section.

The first training function tested on the NN was the scaled conjugate gradient with backpropagation (trainscg) with a hidden layer values of 40 and 60. Using the shallow ANN with the trainscg training function and 40 neurons, the resulting R values calculated from the plotregression for the initial 80% training data, 20% reserved, and testing data were 0.9682, 0.9665, and 0.9554. These values represent a strong correlation between the predicted values and the actual label data. Using the confusion matrix created from each data set, the accuracy values generated were 0.9796, 0.9809, and 0.9705. These results indicate that the values of TP and TN were much greater than that of the FP and FN values. Meaning, the network predicts values at a great level of accuracy. Using the shallow ANN with the trainscg training function and 60 neurons, the resulting R values calculated form the plotregression for the initial 80% training data, 20% reserved, and testing data were 0.9822, 0.9845, and 0.9638. These values represent an even stronger correlation between the predicted values and the actual label data compared to using 40 neurons. Using the confusion matrix created from each data set, the accuracy values generated were 0.9890, 0.9902, and 0.9771. These results indicate that the values of TP and TN were greater than those values generated from the network with 40 neurons. This occurrence can be explained with the addition of 20 more neurons as they increase the complexity of the network and the computational costs resulting in greater accuracy.

The second training function tested on the NN was the conjugate gradient with Beale-Powell restarts (traincgb) with a hidden layer values of 40 and 60. Using the shallow ANN with the trainscg training function and 40 neurons, the resulting R values calculated from the plotregression for the initial 80% training data, 20% reserved, and testing data were 0.9685, 0.9696, and 0.9577. These values represent a strong correlation between the predicted values and the actual label data. Using the confusion matrix created from each data set, the accuracy values generated were 0.9796, 0.9825, and 0.9713. These results indicate that the values of TP and TN were much greater than that of the FP and FN values. Meaning, the network predicts values at a great level of accuracy. Using the shallow ANN with the trainscg training function and 60 neurons, the resulting R values calculated form the plotregression for the initial 80% training data, 20% reserved, and testing data were 0.9818, 0.9832, and 0.9657. These values represent an even stronger correlation between the predicted values and the actual label data compared to using 40 neurons. Using the confusion matrix created from each data set, the accuracy values generated were 0.9887, 0.9896, and 0.9781. These results indicate that the values of TP and TN were greater than those values generated from the network with 40 neurons. Again, this occurrence can be explained with the addition of 20 more neurons as they increase the complexity of the network and the computational costs resulting in greater accuracy.

The last training function tested on the NN was the Polak-Ribiére conjugate gradient (traincgp) with a hidden layer values (neuron count) of 40 and 60. Using the shallow ANN with the trainscg training function and 40 neurons, the resulting R values calculated from the plotregression for the initial 80% training data, 20% reserved, and testing data were 0.9685, 0.9692, and 0.9550. These values represent a strong correlation between the predicted values and the actual label data. Using the confusion matrix created from each data set, the accuracy values generated were 0.9797, 0.9806, and 0.9704. These results indicate that the values of TP and TN were much greater than that of the FP and FN values. Meaning, the network predicts values at a great level of accuracy. Using the shallow ANN with the trainscg training function and 60 neurons, the resulting R values calculated form the plotregression for the initial 80% training data, 20% reserved, and testing data were 0.9792, 0.9798, and 0.9644. These values represent an even stronger correlation between the predicted values and the actual label data compared to using 40 neurons. Using the confusion matrix created from each data set, the accuracy values generated were 0.9868, 0.9876, and 0.9776. These results indicate that the values of TP and TN were greater than those values generated from the network with 40 neurons. Again, this occurrence can be explained with the addition of 20 more neurons as they increase the complexity of the network and the computational costs resulting in greater accuracy.

1. CONCLUSION

One can conclude that creating, training, and utilizing a shallow ANN to recognize handwritten digits is feasible and can result in a script that can, within a small margin of error, return a correctly predicted target / label given a set of image data. This same process can be conducted to solve an array of problems within the world around us, and many references have done so with great success. The continuation of research within the realm of machine learning looks to bode well for the advancement of society.

1. REFERENCES

[1] L. Shure, “Artificial Neural Networks for Beginners,” Loren on the Art of MATLAB, 04-Aug-2015. [Online]. Available: https://blogs.mathworks.com/loren/2015/08/04/artificial-neural-networks-for-beginners/?doing\_wp\_cron=1603656637.3436419963836669921875. [Accessed: 25-Oct-2020].

[2] S. A. K, P. Sumari, S. A. Al-Taweel, and A. J. A. Husain, “Digital Recognition using Neural Network,” N/A, 2009. [Online]. Available: https://www.researchgate.net/profile/Sumari\_Putra/publication/40714131\_Digital\_Recognition\_using\_Neural\_Network/links/547fbac20cf2ccc7f8bb056b.pdf. [Accessed: 25-Oct-2020].

[3] S. N. Srihari and E. J. Kuebert, "Integration of hand-written address interpretation technology into the United States Postal Service Remote Computer Reader system," Proceedings of the Fourth International Conference on Document Analysis and Recognition, Ulm, Germany, 1997, pp. 892-896 vol.2, doi: 10.1109/ICDAR.1997.620640.

[4] “Complete Guide to Artificial Neural Network Concepts &amp; Models,” MissingLink.ai, 2020. [Online]. Available: https://missinglink.ai/guides/neural-network-concepts/complete-guide-artificial-neural-networks/. [Accessed: 25-Oct-2020].

[5] DeepAI, “Perceptron,” DeepAI, 17-May-2019. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/perceptron. [Accessed: 26-Oct-2020].