COSC 425 Project 4

Ryan Pauly
University of Tennessee Knoxville

Project Description

The goal of project 4 is to classify spam email based upon a vast range of features related to email content and classifier information to indicate whether an email is spam or not spam. With the help of a artificial multilayer neural network (a neural network consisting of an input, hidden, and an output layer) to train to classify an email as spam or not spam given specific numeric feature values with respect to the email and its contents.

Pre-processing Steps

The data file 'spambase.data' is an enormous comma separated value file consisting of about 4601 rows of data, each row consisting of 57 features not including the classifier. Upon further investigation of the source of the provided data which can be found at the following link: http://archive.ics.uci.edu/ml/datasets/Spambase , it reveals that there is missing data within the dataset. Due to the size of the data, no examples of missing values could be found.

Reading in the data, checks were implemented to find various non-numeric values or blank cells which were then replaced with a Nan and subsequently removed. No rows however were removed which likely means the search failed to find any missing values or the search parameters put in place were insufficient. However, results could be obtained with the data, so the error that exists within the dataset is likely that of a misplaced value, and in this instance is hard to account for other than by simply normalizing the dataset if it even exists at all. Thus, the data is assumed to be vetted and partitioned into the standard 70/10/20 split of training, validation, and testing respectively after the data has been shuffled to ensure a better diversity of different data among the training, validating, and testing partitions.

Description

A neural network is what is to be implemented into project 4 to predict whether an email message is spam or not provided features about the specific email in question. A neural network in many examples is very similar to how the brain function, hence the name it has received. The neural network in this project is a multilayer perceptron, consisting of an input layer, a hidden layer, and an output layer. Within these layers are nodes or neurons, each having an output to the next layer's nodes or neurons.

The following image from a Wikipedia page does an excellent job of illustrating the "horizontal tree" of how a neural network looks visually:

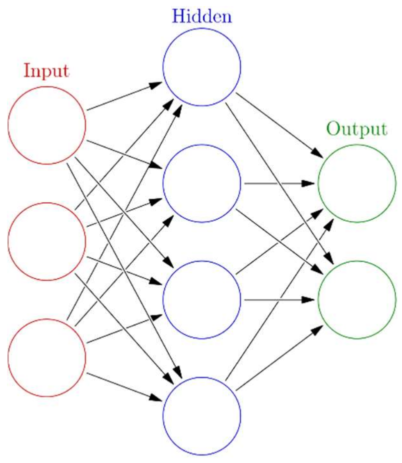


Figure 1. An illustration of a multilayer neural network consisting of an input, hidden, and output layer with several neurons in each layer.

Each of the connections indicated by the arrows in figure 1 are weighted. These weights determine how important or significant a connection is, where a connection with a larger value may be more important in determining a desired output than connections of smaller weight. The hidden layer is essentially a 'black box' which takes in the input data and converts it to try and get the prediction in the output layer. The purpose of a multilayered neural network is to solve the issue of non-linearly separable problems, that is, data that cannot be with a single linear line be classified (by one side being one classification and the other side of the linear line being the other classification). Multilayered neural networks solve this issue and can allow for more than two possible classifications, but in this project, there exists only two classifications, spam email or non-spam email.

To build the multilayered neural network, first all the input connections for each layer are assigned a random weight within the range of 0 to 0.01, a very small number. For example, the hidden layer has input connections from the input layer, all of these are then assigned the random weights within the previously determined random uniform range. The same process is done for the output layer, where all the output connections from the hidden layer are the output layer's input connections. Each node of the hidden and output layer has an output, these outputs are determined by an activation function. The three different types of activation functions for this project include: linear, sigmoid, and SoftMax. These activations have a rippling effect where the activation of one layer can affect the activation of the next layer. That is, the output of the previous activation function becomes the input into the following layer's activation function. This is performed in what is often referred to as forward propagation, which cascades the layer neuron outputs into the next layer neuron's inputs until the final output is achieved. The activation function is performed given a row of information and weights associated with the row's features determined during training. Training a neural network in the case of this lab uses backpropagation, a method which after using random weights for neuron connections, backtracks to update the weights based off the difference of the expected and the predicted output. This error change (the error of the expected output against the predicted output by the neural net based on the weights) is determined by using the gradient descent, which effectively uses the neuron's output in the derivative of the activation function and multiplies it by the expected – the predicted percentage chance. Based upon the error found, the weights for each connection can be updated with the following basic formula:

$$Update = Learning\_rate \times (DesiredOutput - ActualOutput) \times Input \quad (1)$$

Equation 1 represents how weights in the neural network are updated. Notice that if the actual output of the neuron is equivalent to the desired output, that the update becomes 0, meaning if convergence to the correct answer is achieved, the update stops.

Forward propagation creates outputs with an activation function. The activation function forces the output to essentially become a probability based off the activation function's equation. As stated previously, the three activation functions for project 4 include: linear, logistic sigmoid, and SoftMax. The variable x in the following equations represents the weights multiplied by the inputs, all summed together plus the bias.

Linear activation function:

$$f(x) = x$$

Logistic Sigmoid activation function:

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

SoftMax activation function:

$$f_i(\vec{x}) = \frac{e^{x_i}}{\Sigma_{j=1}^{J} e^{x_j}} \quad for\ i = 1, \ldots, J$$

Back propagation is where the error is computed, by traversing backwards through the neural network. The equation for updating the error depends on what layer the error is being computed for. The general idea however is that the adjust based off the error:

$$Back\ Propagation\ Adjustment = [Expected - Actual] * slope(outputs)$$

The pseudocode offered in the textbook best explains how the multilayer neural network algorithm functions:

Multilayer Neural Network Pseudocode

```
Initialize all $v_{ih}$ and $w_{hj}$ to rand$(-0.01, 0.01)$
Repeat
    For all $(\boldsymbol{x}^t, r^t) \in \mathcal{X}$ in random order
        For $h = 1, \ldots, H$
            $z_h \leftarrow \mathrm{sigmoid}(\boldsymbol{w}_h^T \boldsymbol{x}^t)$
        For $i = 1, \ldots, K$
            $y_i = \boldsymbol{v}_i^T \boldsymbol{z}$
        For $i = 1, \ldots, K$
            $\Delta \boldsymbol{v}_i = \eta(r_i^t - y_i^t)\boldsymbol{z}$
        For $h = 1, \ldots, H$
            $\Delta \boldsymbol{w}_h = \eta(\sum_i (r_i^t - y_i^t) v_{ih}) z_h (1 - z_h) \boldsymbol{x}^t$
        For $i = 1, \ldots, K$
            $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \Delta \boldsymbol{v}_i$
        For $h = 1, \ldots, H$
            $\boldsymbol{w}_h \leftarrow \boldsymbol{w}_h + \Delta \boldsymbol{w}_h$
Until convergence
```

Figure 2. Pseudocode representing how to code a multilayer neural network.

The goal with training the neural network is to minimize the error between the expected output and the output calculated by the neural network in instances at a time (online learning). To determine the output, a probability is found which sums to 1. The probability essentially rates in the case of project 4 how likely the output is to be a 0 or a 1 (2 neurons or 2 nodes of the output layer).

In the case of project 4, benign indicates that the email is not spam and malignant indicates that the email is spam. With respect to the benign and malignant results and the actual results of the two classifiers: TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives, this all is best represented in a confusion matrix as defined as the following:

True Class	**Predicted Class**	
	benign	**malignant**
benign	TN	FP
malignant	FN	TP

Figure 3. Confusion Matrix from the project 3 pdf.

Next the metrics that are needed for each prediction of the project: First the accuracy equation, which is very straightforward is defined below:

$$\text{Accuracy} = (\text{TN} + \text{TP}) / (\text{TN} + \text{TP} + \text{FN} + \text{FP}) \tag{5}$$

The next equation is the true positive rate, another straightforward equation:

$$TPR = TP/(TP + FN) \tag{6}$$

Further, the next equation is the positive predictive value:

$$PPV = TP/(TP + FP) \tag{7}$$

Then, the true negative rate, another simple equation:

$$TNR = TN/(TN + FP) \tag{8}$$

Finally, the $F_1\ Score$, which is the harmonic mean, a little more complicated than the previous four equations and it is defined as:

$$F_1\ Score = 2 * PPV * TPR/(PPV + TPR) \tag{9}$$

Finally, we wish to compare how standardizing the data compares to non-standardized data. To standardize the dataset, we perform a calculation which scales the variables down significantly. The goal with standardization is to scale the data down to a mean which equals 0 and a standard deviation equal to 1. To achieve this, we use the z-normalization formula:

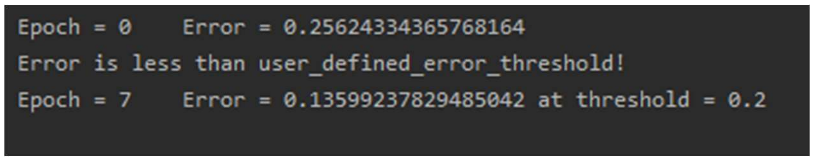
$$z = \frac{X - \mu}{\sigma} \tag{10}$$

X represents the value we wish to standardize, and *Z* is the result of the standardization. We subtract from *X* the mean of *X* or μ, and lastly, we divide it by the standard deviation of *X* or σ.

Analysis

The neural network should perform best on the sigmoid and SoftMax activation functions over the linear function. However, the implemented neural network surprisingly finds that the linear activation method outputs the best accuracy after training is performed with the sigmoid function. The initial neural network is trained with the training dataset with the sigmoid activation function. Epochs and learning rate were adjusted manually as to reduce cross validation time. The error that is computed in the training function is the sum of the squared errors divided by the number of output predictions made by the neural network. This provides a rather large number but one that is flexible to change to best describe how well the neural network is performing. An error threshold is set at about 0.2 to catch and stop the training process if a good training update is found, otherwise the epoch limit is set to 200 epochs. The learning rate used is 0.1, or a 10% of the error delta adjustment for the new weights.

Training received the following error and epoch prints where the print only occurs for every 10 epochs:



```
Epoch = 0    Error = 0.25624334365768164
Error is less than user_defined_error_threshold!
Epoch = 7    Error = 0.13599237829485042 at threshold = 0.2
```

Figure 4. Screenshot of epoch and error percentages for every 10 epochs.

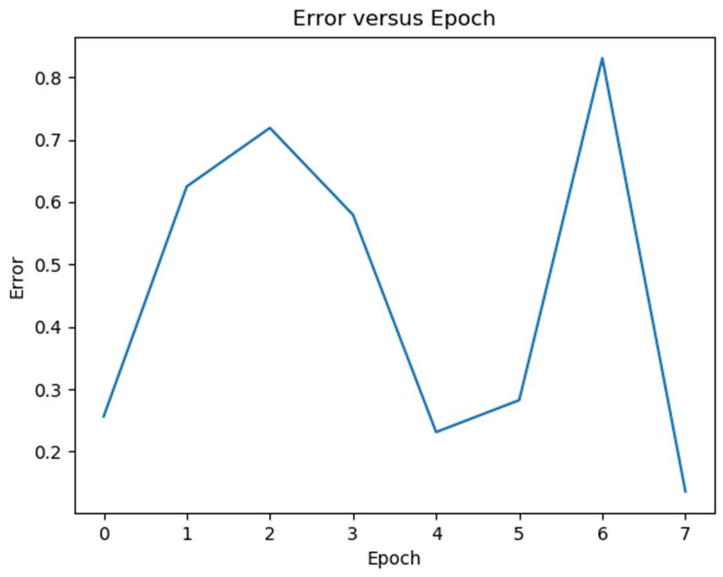
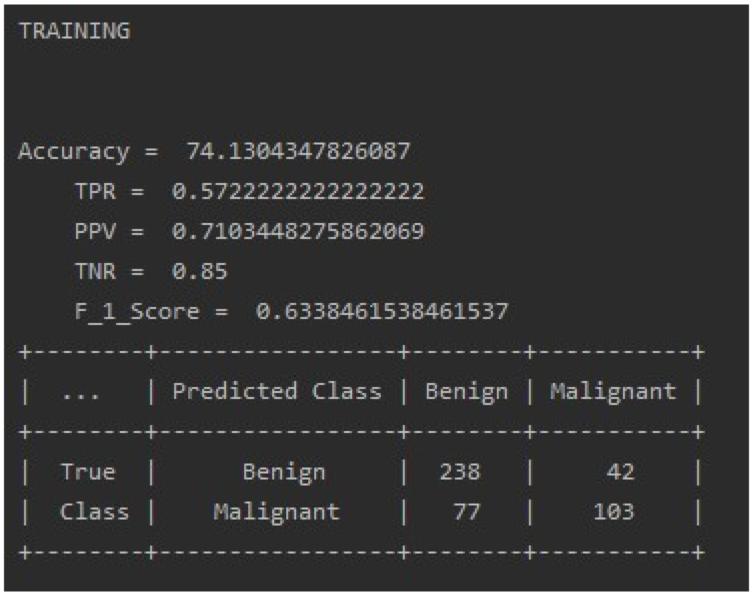


Figure 5. Error versus Epoch graph showing how the neural network is training.

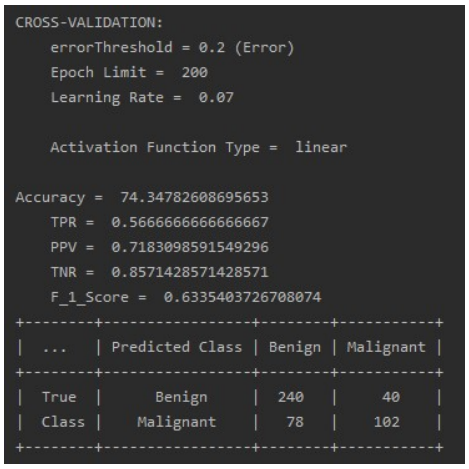
The graph in figure 5 shows that the neural network is doing a poor job at decreasing the error, rather it consistently noisy and fails to converge towards 0. The training then receives the following metric scores:



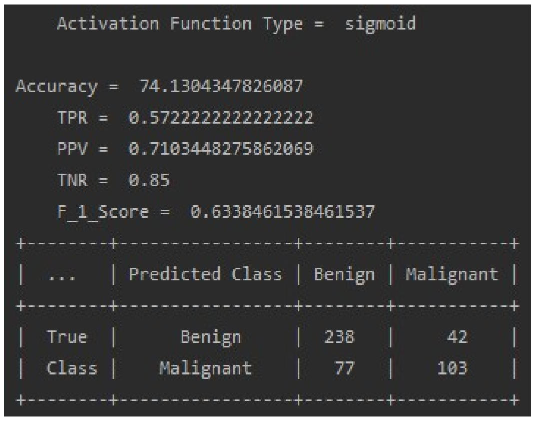
```
TRAINING


Accuracy =  74.1304347826087
    TPR =  0.5722222222222222
    PPV =  0.7103448275862069
    TNR =  0.85
    F_1_Score =  0.6338461538461537
+--------+-----------------+--------+-----------+
|  ...   | Predicted Class | Benign | Malignant |
+--------+-----------------+--------+-----------+
|  True  |      Benign     |  238   |     42    |
|  Class |    Malignant    |   77   |    103    |
+--------+-----------------+--------+-----------+
```

Figure 6. Metrics for the training of the neural network.

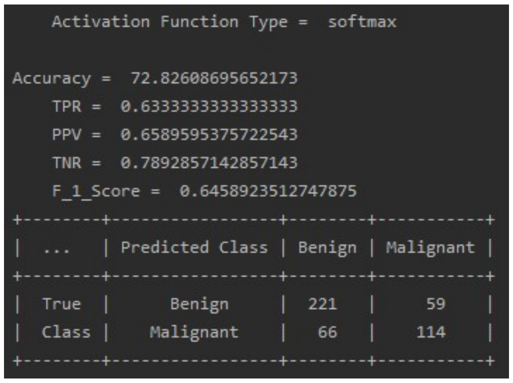
Then next once the training is complete, the cross validation. The cross validation again uses the same learning rate, error threshold, and epoch limit as mentioned before. The cross validation provides the following output metrics:



```
CROSS-VALIDATION:
    errorThreshold = 0.2 (Error)
    Epoch Limit =  200
    Learning Rate =  0.07

    Activation Function Type =  linear

Accuracy =  74.34782608695653
    TPR =  0.5666666666666667
    PPV =  0.7183098591549296
    TNR =  0.8571428571428571
    F_1_Score =  0.6335403726708074
+--------+----------------+--------+----------+
|  ...   | Predicted Class | Benign | Malignant |
+--------+----------------+--------+----------+
|  True  |      Benign     |  240   |    40    |
|  Class |    Malignant    |   78   |   102    |
+--------+----------------+--------+----------+
```



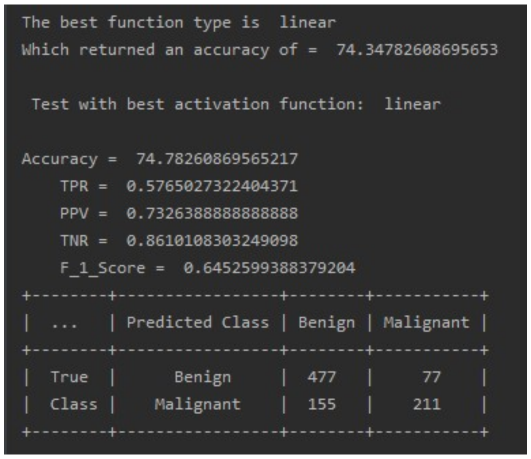
```
    Activation Function Type =  sigmoid

Accuracy =  74.1304347826087
    TPR =  0.5722222222222222
    PPV =  0.7103448275862069
    TNR =  0.85
    F_1_Score =  0.6338461538461537
+--------+----------------+--------+-----------+
|  ...   | Predicted Class | Benign | Malignant |
+--------+----------------+--------+-----------+
|  True  |      Benign     |  238   |    42     |
|  Class |    Malignant    |   77   |   103     |
+--------+----------------+--------+-----------+
```



```
    Activation Function Type =  softmax

Accuracy =  72.82608695652173
    TPR =  0.6333333333333333
    PPV =  0.6589595375722543
    TNR =  0.7892857142857143
    F_1_Score =  0.6458923512747875
+--------+----------------+--------+-----------+
|  ...   | Predicted Class | Benign | Malignant |
+--------+----------------+--------+-----------+
|  True  |      Benign     |  221   |    59     |
|  Class |    Malignant    |   66   |   114     |
+--------+----------------+--------+-----------+
```

Figure 7. Cross Validation metrics.

The cross validation informs that the linear activation function performed the best at out of the three activation functions, just barely beating sigmoid. This however seems incorrect as is likely the result of a mistake with the sigmoid function as well as with the SoftMax function in some capacity. Nevertheless, with the linear activation function determined as the most optimal function, the testing data is then used to forward propagate the trained tree with the linear function and provides the following outcome:

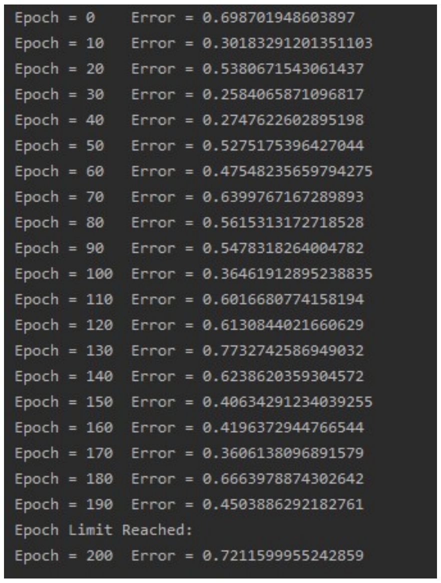


```
The best function type is  linear
Which returned an accuracy of =  74.34782608695653

 Test with best activation function:  linear

Accuracy =  74.78260869565217
    TPR =  0.5765027322404371
    PPV =  0.7326388888888888
    TNR =  0.8610108303249098
    F_1_Score =  0.6452599388379204
+--------+-----------------+--------+-----------+
|  ...   | Predicted Class | Benign | Malignant |
+--------+-----------------+--------+-----------+
|  True  |      Benign     |  477   |     77    |
|  Class |    Malignant    |  155   |    211    |
+--------+-----------------+--------+-----------+
```

Figure 8. Outcome using the testing data to forward propagate with the linear activation function determined as the best during cross validation.

The accuracy was only 74.78 percent accurate to the actual expected outcome, which is an abysmal. Some issues were encountered somewhere with the SoftMax and sigmoid function, but the linear function seemed to perform well for what the function is. Linear should by all measures have been the worst activation function for this sort of problem. Further, the confusion matrix shows that there are many false negatives, about 155 according to figure 8, which means that spam email was falsely identified as non-spam, something critical for an algorithm which attempts to identify spam email.

Now with standardized data:



```
Epoch = 0    Error = 0.698701948603897
Epoch = 10   Error = 0.30183291201351103
Epoch = 20   Error = 0.5380671543061437
Epoch = 30   Error = 0.2584065871096817
Epoch = 40   Error = 0.2747622602895198
Epoch = 50   Error = 0.5275175396427044
Epoch = 60   Error = 0.47548235659794275
Epoch = 70   Error = 0.6399767167289893
Epoch = 80   Error = 0.5615313172718528
Epoch = 90   Error = 0.5478318264004782
Epoch = 100  Error = 0.36461912895238835
Epoch = 110  Error = 0.6016680774158194
Epoch = 120  Error = 0.6130844021660629
Epoch = 130  Error = 0.7732742586949032
Epoch = 140  Error = 0.6238620359304572
Epoch = 150  Error = 0.40634291234039255
Epoch = 160  Error = 0.4196372944766544
Epoch = 170  Error = 0.3606138096891579
Epoch = 180  Error = 0.6663978874302642
Epoch = 190  Error = 0.4503886292182761
Epoch Limit Reached:
Epoch = 200  Error = 0.7211599955242859
```

Figure 9. Error output for every 10 epochs.

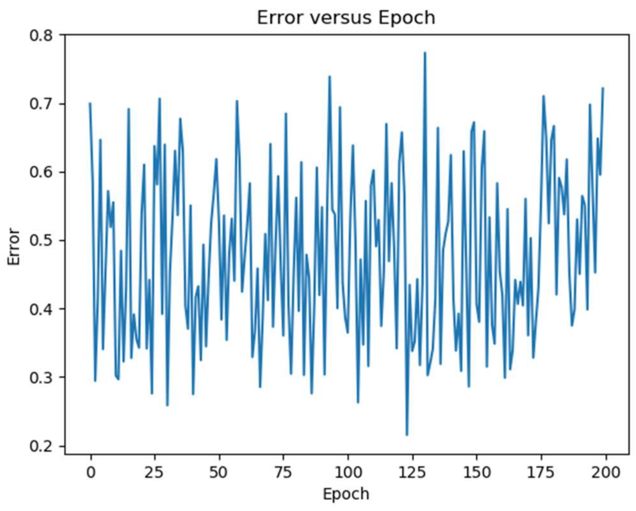
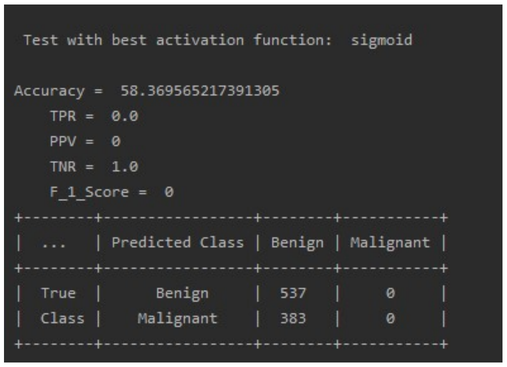


Figure 10. Normalized error versus epoch.

Cross-validation is then performed for the normalized training data trained neural network again using the sigmoid function with the same parameters as the non-normalized version. Cross validation selects the sigmoid function this time over the other two and provides the following output metrics:



```
 Test with best activation function:  sigmoid

Accuracy =  58.369565217391305
    TPR =  0.0
    PPV =  0
    TNR =  1.0
    F_1_Score =  0
+--------+----------------+--------+-----------+
|  ...   | Predicted Class | Benign | Malignant |
+--------+----------------+--------+-----------+
|  True  |      Benign     |  537   |     0     |
|  Class |    Malignant    |  383   |     0     |
+--------+----------------+--------+-----------+
```

Figure 11. Normalized final testing output.

Discussion

At the beginning of project 4, neural networks seemed like a very complicated structure to understand. After project 4, I find that the neural network itself in theory is not difficult to understand. However, the difficulty is with correctly creating and training the neural network, which I believe I have failed to do so correctly. The linear activation function is surprisingly working better than functions designed for non-linearly separable problems such as the one this project aims to solve. I learned a lot in this project, but I am definitely disappointed that I was unable to improve the accuracy of my neural network. Given more time I would likely try a different approach to the project and try to dissect what exactly was causing my neural network to have the issues it was having.

Sources

"Activation Function." *Wikipedia*, Wikimedia Foundation, 17 Nov. 2019, en.wikipedia.org/wiki/Activation_function.

Alpaydin, Ethem. *Introduction to Machine Learning*. third ed., MIT Press, 2014.

"Backpropagation." *Wikipedia*, Wikimedia Foundation, 7 Nov. 2019, https://en.wikipedia.org/wiki/Backpropagation.

"Confusion Matrix." *Wikipedia*, Wikimedia Foundation, 22 Oct. 2019, en.wikipedia.org/wiki/Confusion_matrix. Accessed 4 Nov. 2019. https://en.wikipedia.org/wiki/Confusion_matrix.

"Cross Entropy." *Wikipedia*, Wikimedia Foundation, 24 Oct. 2019, en.wikipedia.org/wiki/Cross_entropy.

"Stochastic Gradient Descent." *Wikipedia*, Wikimedia Foundation, 16 Nov. 2019, en.wikipedia.org/wiki/Stochastic_gradient_descent.