Neural Network Write-up

Homework #4
By: Alexander Polus
3.23.18

Introduction

For this project, we were asked to create a neural network to identify numeric handwriting. The challenge was to differentiate between 1's and 0's, with the training data being an excel sheet where each row was an example, the first cell was the label, and the following cells were each pixel of an image corresponding to its black/white value 0-255.

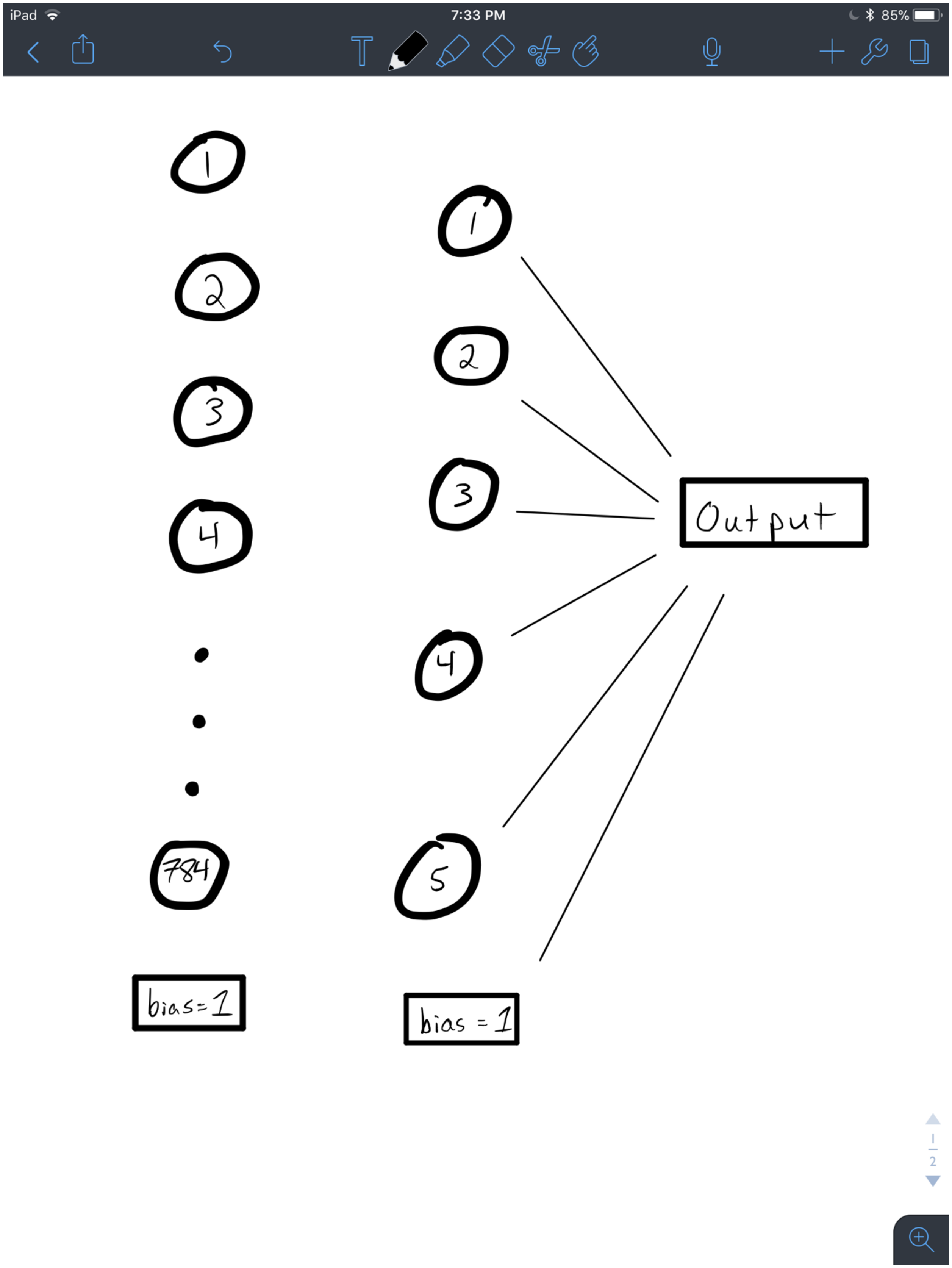
Code Summary

I began my implementation by importing the necessary libraries, most importantly csv and numpy so that I would be able to use efficient matrix multiplication and transposition functions. Next, I validated the usage of the program from the command line to ensure proper behavior.

After that I defined two global values, e and alpha. e was useful in the following functions where I defined my sigmoid function and sigmoid derivate (which I never used but left in because it may be useful for me in the future). The learning rate is perhaps the most important variable in the program, as it defines how fast the values change when I update the weights of the network. I played with the values until I found 0.05 to be the best value for the model.

To train the model, I needed training data, and I used the csv reader to import the excel sheet as a 2D array. From there, I turned every value into a float and made each row of the array a 2D array of the label followed by an array of pixel values.

To implement the network itself, I had a 1x785 matrix for input values and bias which I multiplied by a 784x6 matrix (the input to hidden node weights) which resulted in a 1x6 matrix that I activated with a sigmoid function. Then, I took that value and multiplied it by a 6x1 matrix of hidden node to output weights to get a raw output value, which was activated with the sigmoid function to get my final output. This model is represented by the image on the following page, drawn without the permutations of weights between the input layer and the hidden nodes. The initial values of each weight were initialized to a random value between -1 and 1 which explains the inconsistency of the model accuracy on different runs of the program.


iPad
7:33 PM
85%
1
2
3
4
784
bias=1
1
2
3
4
5
bias = 1
Output
1
2

The TrainWalk() method uses that algorithm to output a decimal value to use to calculate errors and train the model. TestWalk() outputs a 1 if the output value is above 0.5 and a 0 otherwise.

To train the model, I iterated over 10 epochs of the data. For each epoch, I went through the data using stochastic gradient descent and calculated the delta for the output value, followed by the delta values at the hidden nodes. Then, I updated the weights iterating through the normal nodes then doing a separate update with the proper equation for the bias node. Then I did the same for the input layer to hidden layer weights.

After that I imported the testing data formatting it in the same way I did the testing data. Using the TestWalk() function, I predicted each value and compared it to the actual, and outputted the percentage it got correct.

Functionality, Usage, Preconditions and Postconditions

Functionality: This program trains on testing data of pictures of 0's and 1's in order to be able to predict if a picture is a zero or a one (binary classification).

Usage: python3 NeuralNetwork1.py mnist_train_0_1.csv mnist_test_0_1.csv

Preconditions: This program accepts two excel files, a training and test file.

Postconditions: This program outputs the number of testing examples it predicted correctly.

Conclusion

This model predicted 90.92198581560284 % of test examples correctly on the best of 5 runs I performed. The accuracy varies due to the inconsistency in starting value for each node due to the randomization of the values. In the future, it would be more effective to store the weights after a single iteration and use it for testing every time to guarantee consistency if it were to be used for any high-pressure decision.