Reg Gonzalez

Assignment 1 - Report

Summary of Results

Activation Function choice	Beta	Iterations	Learning rate	Number of neurons for hidden layer	Accuracy (using training data)	Accuracy (using testing data)
ReLu	0.90	500	0.10	13	0.849	0.835
Sigmoid	0.90	500	0.10	13	0.723	0.727
Tanh	0.90	500	0.10	13	0.810	0.801
ReLu	0.30	500	0.10	13	0.854	0.867
Sigmoid	0.30	500	0.10	13	0.715	0.749
Tanh	0.30	500	0.10	13	0.810	0.818
ReLu	0.99	500	0.10	13	0.656	0.667
Sigmoid	0.99	500	0.10	13	0.721	0.71
Tanh	0.99	500	0.10	13	0.769	0.774
ReLu	0.90	300	0.10	13	0.817	0.814
Sigmoid	0.90	300	0.10	13	0.644	0.638
Tanh	0.90	300	0.10	13	0.761	0.757
ReLu	0.90	700	0.10	13	0.866	0.86
Sigmoid	0.90	700	0.10	13	0.792	0.796
Tanh	0.90	700	0.10	13	0.856	0.854

Assignment 1 - Report

ReLu	0.90	500	0.01	13	0.432	0.428
Sigmoid	0.90	500	0.01	13	0.231	0.235
Tanh	0.90	500	0.01	13	0.382	0.385
ReLu	0.90	500	1.0	13	0.910	0.893
Sigmoid	0.90	500	1.0	13	0.909	0.893
Tanh	0.90	500	1.0	13	0.921	0.889
ReLu	0.90	500	0.10	6	0.807	0.806
Sigmoid	0.90	500	0.10	6	0.580	0.583
Tanh	0.90	500	0.10	6	0.744	0.725
ReLu	0.90	500	0.10	26	0.875	0.871
Sigmoid	0.90	500	0.10	26	0.804	0.795
Tanh	0.90	500	0.10	26	0.849	0.842

Key:

- *White section* - This is the "control" test. These are the values I initially set for each parameter (Beta, Iterations, Learning rate, and Number of neurons for a hidden layer), and they represent the "standard" for the accuracy of the neural network.
- *Red section* - Change in the 'Beta' parameter (i.e., the parameter that represents a constant value to calculate the velocities needed for gradient descent with momentum)
- *Yellow section* - Change in the 'Iterations' parameter (i.e., the parameter that represents the number of iterations that the neural network goes through)
- *Blue section* - Change in the 'Learning rate' parameter (i.e., the parameter that represents a constant value to help calculate gradient descent with momentum)
- *Green section* - Change in the 'Number of neurons for a hidden layer' parameter (self-explanatory)

Assignment 1 - Report

For each section, I both decreased and increased the value of that respective parameter. For example, in the green section, I kept all the parameters the same as the white section, except for 'Number of neurons for a hidden layer'. The original parameter I set was 13, but I decreased it to 6 and increased it to 26 to see what effect changing that one parameter would have on the accuracy of the neural network.

Discussion of Results

Let's start by first looking at the white section of the table (i.e., the "control" test). As a reminder, the values I set for each parameter in the white section are my "base" parameters. The changes I make to each parameter later on will be relative to those base parameters. The last two columns of the table represent the accuracies. The second to last column is the accuracy for the training data; that is, the accuracy we got when we first built the model. The last column represents the accuracy of the testing data; that is, the accuracy we got when the model was used the model on new, unseen data. From these results, we can see that the use of the ReLu activation function performed the best while the use of the Sigmoid function performed the worst. This might be because ReLu is less susceptible to problems such as the vanishing gradient problem—to which both Sigmoid and Tanh can be prone. The function is also easier to implement and calculate than Sigmoid or Tanh, which might be another reason why it performed better.

In the red section, I changed the value of the *Beta* parameter. This represents a constant that calculates velocities for gradient descent with momentum. Beta has a range between 0 and 1 and generally speaking, a lower beta "forgets" older observations in the data faster. Essentially, having a low beta means that older observations will not contribute much to calculating the gradient descent with momentum while having a high beta means that older observations will be taken into account more. The initial value of beta I set was 0.90, but I decreased it to 0.30 and increased it to 0.99 (while keeping all other parameters the same) to see the effect changing beta had on the accuracies. When I decreased beta, the accuracies didn't change that much for either the training data or testing data. In these cases most of the accuracies actually increased, but not by any significant margin. However, when I increased beta, the accuracy for ReLu (and to a lesser extent, Tanh) changed drastically. The accuracy using the ReLu function decreased by almost 0.20.

In the yellow section, I changed the value of the *Iterations* parameter. This represents the total number of iterations that the neural network will go through while training the model. The initial value was 500. I then decreased this to 300 and increased it to 700, while keeping all other

Assignment 1 - Report

parameters the same. When I decreased the iterations, the accuracies for both the training and testing data decreased when compared to the "control" test (i.e., the white section). This makes sense considering the model had less time to train itself. When I increased the iterations, the accuracies for both the training and testing data subsequently increased. Again, this checks out considering that the model had more time to train when compared to the initial 500 iterations.

In the blue section, I changed the *Learning rate* parameter. This is another constant used to update the weights and biases when performing gradient descent. I initially set this value to 0.1, then decreased it to 0.01, and finally increased it to 1.0, while all other parameters were kept the same. Changing the learning rate perhaps had the biggest effect on the accuracy of the model. When I decreased it to 0.01 the accuracies plummeted significantly. For ReLu, the accuracies for both the training and testing data decreased by ~0.40; for Sigmoid, the accuracies for both the training and testing data decreased by ~0.50; and for Tanh, the accuracies for both the training and testing data decreased by ~0.40. When I increased the learning rate to 1.0, the increase in the model's accuracy was fairly significant. For the first time, the model achieved an accuracy of over 90% (in the case of the training data). When the model used the unseen test data, the accuracies were a bit lower, but still near 90% nonetheless. The drastic decrease and increase in the model's accuracy was made possible because the learning rate is a very important parameter when calculating the weights and biases of the neural network. The learning rate is essentially how "big" or "small" a step to take in the opposite direction of the gradient. It directly influences how we calculate the weights and biases, so it is a critical parameter in our model. Taking too big steps could lead to oscillations around the minimum while taking too small steps could lead to long training times of the network. In our case, taking bigger steps probably led to finding the optimal minimum better than taking smaller steps.

Finally, in the green section, I changed the *Number of neurons for a hidden layer* parameter. This is pretty self-explanatory. Initially, I set this parameter to 13. I then decreased it to 6 and increased it to 26, again, while all other parameters were kept the same. Decreasing the number of neurons in the hidden layer to 6 decreased the accuracies (in both the training and testing data) in all instances; however, the accuracy using the Sigmoid function saw a significantly greater decrease than ReLu or Tanh. Maybe this has to do with the vanishing gradient problem, of which Sigmoid is very susceptible to. Tanh still suffers from this problem, though not as severely as Sigmoid, which might explain why the decrease in Tanh's accuracy wasn't as big as Sigmoid's. Increasing the number of neurons in the hidden layer to 26 increased the accuracies in all instances. This increase might've been due to the network's ability to learn more complex patterns with more neurons.

Assignment 1 - Report

Sample Output

```
C:\Users\regmc\OneDrive\Desktop\CS 6375\Assignment1_Part2>python .\Assignment1_Part2.py
Accuracy of model using training data (ReLu):  0.861
Accuracy of model using testing data (ReLu): 0.84
Accuracy of model using training data (Sigmoid):  0.7005121951219512
Accuracy of model using testing data (Sigmoid): 0.701
Accuracy of model using training data (Tanh):  0.8236829268292682
Accuracy of model using testing data (Tanh): 0.818

C:\Users\regmc\OneDrive\Desktop\CS 6375\Assignment1_Part2>
```

This sample output is from the terminal. You can also run this program through an IDE—I used PyCharm—and you should get a similar-looking output.