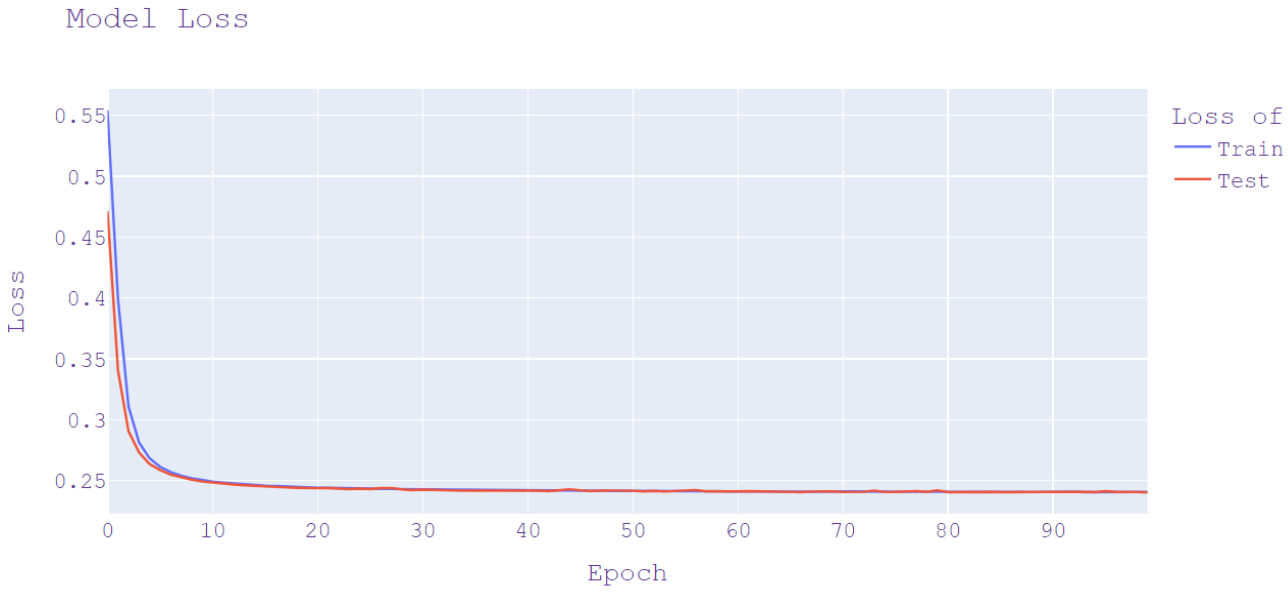


CS 657 - Assignment 4

11.14.2022

Andrick Mercado

Overview

In this assignment, we are asked to create a neural network that implements the logic of a binary half-adder into a neural network that outputs a correct value based on two inputs, all while attempting to optimize it, to produce a valuable solution.

Goals

1. Make a solution that produces optimal solutions.
1. Produce good-looking graphs that illustrate information well.

Structure

For this assignment I took the easy way out with a library initially I attempted a from-scratch solution, but it was quickly that I discovered how hard it was to properly train it and create layers, specifically creating two outputs. Hence why I went with a TensorFlow implementation of Neural Networks, which essentially performs everything for you, and the only thing that needs to be done is to create the training data and set up the NN with custom settings to optimize it.

Data (w/ noise)

For the data, I chose 10,000 samples, mostly because the higher the sample count the higher the overall accuracy of the NN, for creating none binary inputs with the noise I used numpy.random.normal function which returns random values from -0.2 to 0.2 and I added it to create a binary sample of 10,000. Lastly to finalize the data I used train test split to create testing and training data randomly, essentially shuffling it to create more randomness.

Activation functions & Neurons

For activation functions, I used softmax for one layer and sigmoid for the other layer, with the reasoning just being it created better results. For neurons, I used two per layer, mostly because we only have two inputs.

Learning Rate

For the learning rate value, I went for 0.01 because after testing multiple values it created less randomness (the accuracy was more stable).

Epochs

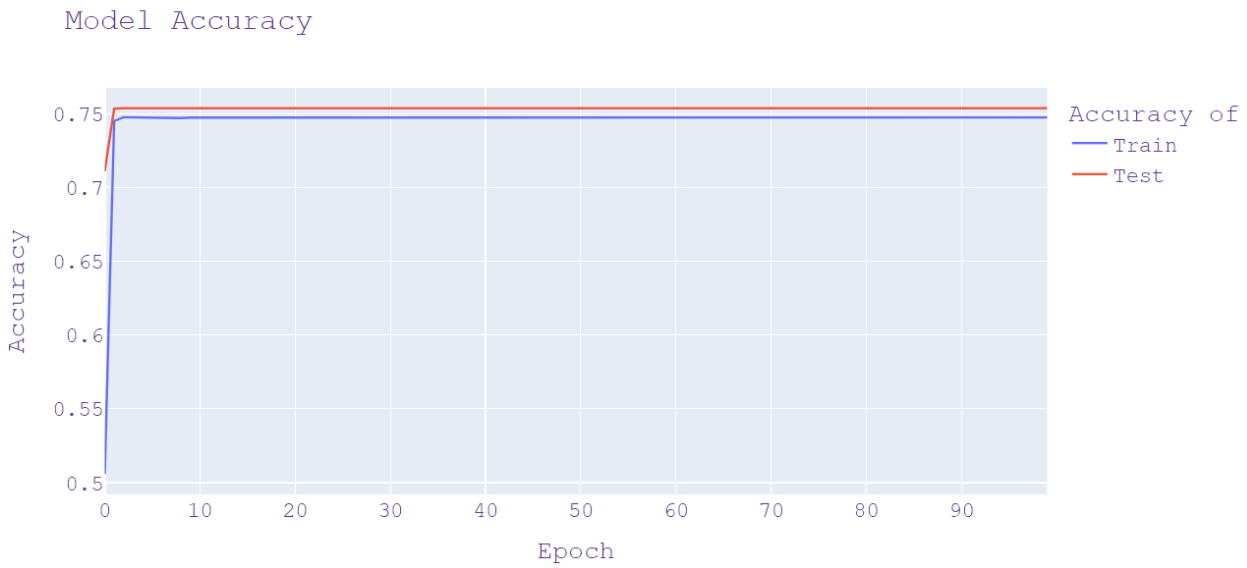
For the epochs amount I went for 100 because the accuracy converged after 10 epochs to .7 to .9 accuracy, and I just wanted to show more values.

Graphs

For the graphs I made them in Jupiter notebooks, it uses the error and accuracy history produced by the NN, to create visualized results, also the graphs are interactable meaning you can see a value at any point from either function as well as zoom in and out..

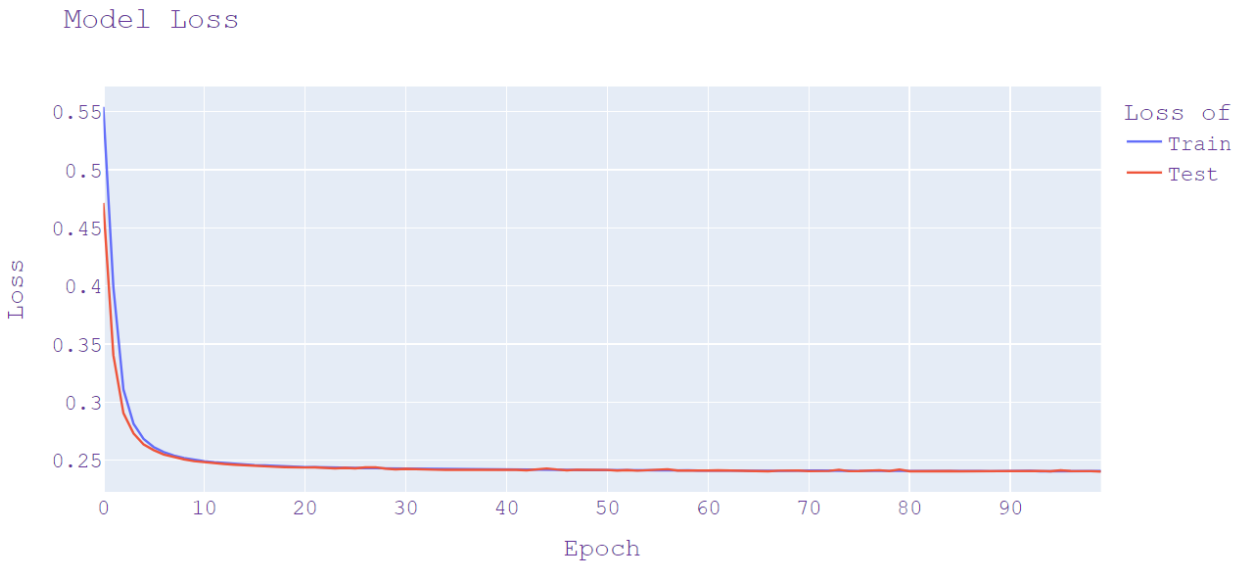
Model Accuracy

Notes: here can be observed the accuracy converges to 0.75 approximately, after creating multiple simulations the highest accuracy I got was 0.85 with my settings.



Model Loss

Notes: here can be observed the loss converges to 0.22 approximately, after creating multiple simulations the lowest loss I got was 0.12 with my settings.



Advantages / Disadvantages

I. Advantages

Produces fairly accurate results with input that has noise

II. Disadvantages

The accuracy is not as high as it can be, thus the NN is not too good

Conclusion

Overall my solution created a good result in terms of its accuracy, despite not being perfect it still creates results that are correct given noisy input, although I did not test values with higher noise than 0.2, the NN seemed to satisfy the problem.

Evaluation of NN

Notes: here is just the average value of loss and accuracy of the NN tested in the Jupyter notebooks.

```
Loss Average: 0.22055640816688538
Accuracy Average: 0.7799999713897705
```