**First test with XOR.**

**Source data:** *XOR.json.*

I built the network using the standard setup for XOR; 2 inputs, 2 hidden nodes and 1 output. I adjusted the learning rate to minimise predictive errors, finding a learning rate of 0.2 to work best.

**Experiments with weight adjustments**

Initially built the MLP using stochastic gradient descent; but had issues while testing with XOR where about half of all tests would get stuck in local minima. This resulted in an output like;

|  |  |
| --- | --- |
| Outcome | Expected |
| [[0.05774859]] | 0 |
| [[0.57647176]] | 0 |
| [[0.96448377]] | 1 |
| [[0.57119892]] | 1 |

And the other half perfect:

|  |  |
| --- | --- |
| Outcome | Expected |
| [[0.01585943]] | 0 |
| [[0.01289982]] | 0 |
| [[0.98725898]] | 1 |
| [[0.987264]] | 1 |

With an error readout as one might expect; decreasing for all data in the second instance and hovering about 0.5 for two of four samples in the 1st instance. I think I would need to have a decreasing learning rate for STG to work effectively.

I then tried Gradient Descent where I updated 25% of the time and played with this figure until I was getting a correct reading out 75% of the time; at 10% of the time. By correct I mean output is close to expected.

**Conclusion**

Final iteration was running with weights updating about 15 of 100 of every iteration through the dataset and a learning rate of 0.2. This seemed to produce the most consistent correct results.

**Testing with Generated Vectors**

**Source data:** *vector.json.*

I imported the model class from NeuralNetwork.py and created an instance with 4 input nodes, 5 hidden nodes and one output node and learning rate of 0.2. I ran through the data for 1000 epochs.

The prediction, expected result and error for each prediction were printed to the file Vectors\_Predict\_*timestamp*.*txt.*

The final prediction and error are given below:

**Results**

|  |  |  |
| --- | --- | --- |
| **Prediction** | **Expected** | **Error** |
| 0.91558071 | 0.9025726658005266 | -0.01300805 |

The errors enjoyed a downward trajectory over the course of each iteration. The first datapoint of the first epoch had an error of -1.139886 which was close to 0 by the 1000th epoch, final error given was 0.07695601.

**Conclusion**

The model performed very strongly. I was obliged to reduce the number of iterations somewhat to produce a result in a reasonable time but at 100 epochs the results remained consistently good with this model.