**Logo

Description automatically generatedIBADULLAH KAHTTANA**

**Report on XOR Solving Neural network using NumPy.**

**Diagram

Description automatically generated**

Generally, in Classification problems a single perceptron (‘line’) is enough for Binary Classification.

In AND & OR gate a single perceptron is enough for correct classification.

In the case of XOR, a single perceptron isn’t enough to correctly classify it.

Diagram, schematic

Description automatically generated

**Input Data**

We are having X and Y training data.

**X** is the testing data for training

**X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])**

The matrix is of 4x2.

**Y** is for the corrected results according to X array

**y = np.array([[0], [1], [1], [0]])**

The matrix is of 4x1.

**Network Parameters**

The Parameter for the XOR Neural network are.

Input Neurons = **2**

Hidden Neurons = 2

Output Neuron = **1**

Learning rate = **0.1**

Epochs = **10000**

**Activation Function**

Diagram

Description automatically generatedI use **Sigmoid(x)** as an activation function which ranges my value between 0 and 1.

**F(x) =** sigmoid(x) **= 1 / 1 + exp**(**-x**)

**Sigmoid\_Derivative** to return derivative of sigmoid.

**Weights and biases**

Initially the **weight** and **biases** are define randomly using **np.random** function

Weights for hidden = **W1**

**W1 = 2** x 2

Weights for output = **W2**

**W2 = 2** x 1

**Biases** are added as an extra axis in the X-input

**Neural Network Training**

**Cost** vs **Iteration**

Chart

Description automatically generatedWe use **Gradient Descent** algorithm to minimize the cost.

* I perform forward propagation to calculate the output of the network for the input data X.
* I then calculate the error between the expected **output(y)** and the actual output **output\_layer\_output(Y^)**.
* Then uses the error to calculate the **deltas** for the **output** and **hidden layers** using the sigmoid derivative.
* Finally, we update the weights using the **deltas** and the **learning rate.**

**Testing the NN**

For Testing neural Network, we have Given inputs ranges with the interval of **0.05** sequentially.

* **input1** =[-0.8, 1.2]
* **input2** =[-0.8, 1.2]

Using **np.arrange** function we arrange the input with a difference of **0.05**

Chart, surface chart

Description automatically generatedWe use forward propagation step to find out the final output for each test input and store result (**y^**)

To draw the output in 3D we import Axes3D

**from mpl\_toolkits.mplot3d import Axes3D**

All the value are plot from a **40 x 40** matrix.

A total of **1600** data points.

Output of the network for the ranges.

**A =[-0.8, 1.2]**

**B =[-0.8, 1.2]**

With the interval of 0.05 sequentially and draw the output in 3D.

Learning Rate = 0.1

We get ideal results showcased in the form of a peak.