SCT Experiment No: 2

Name: Abhishek S Waghchaure

PRN: 1032221714

Dept: FY M Tech DSA(2022-24)

Aim:

Implementing forward feed Artificial Neural Network.

Introduction:

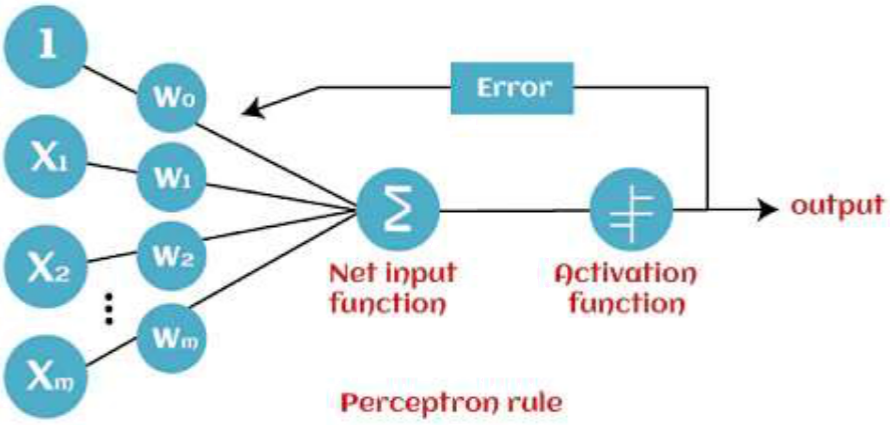
A Feed Forward Neural Network is commonly seen in its simplest form as a single layer perceptron. In this model, a series of inputs enter the layer and are multiplied by the weights. Each value is then added together to get a sum of the weighted input values. If the sum of the values is above a specific threshold, usually set at zero, the value produced is often 1, whereas if the sum falls below the threshold, the output value is -1. The single layer perceptron is an important model of feed forward neural networks and is often used in classification tasks. Furthermore, single layer perceptrons can incorporate aspects of machine learning. Using a property known as the delta rule, the neural network can compare the outputs of its nodes with the intended values, thus allowing the network to adjust its weights through training in order to produce more accurate output values. This process of training and learning produces a form of a gradient descent. In multi-layered perceptrons, the process of updating weights is nearly analogous, however the process is defined more specifically as back-propagation. In such cases, each hidden layer within the network is adjusted according to the output values produced by the final layer.

Dataset:

The neural networks are more useful when the dataset is non-linearly separable and it can not be separated with the help of basics techniques such as linear regession, logistic regression,etc. For this case we will be using a non-linearly separable Dataset (XOR Gate) and we will show that how neural networks can easily classify this data set using feature engineering with the help of different hidden layers.

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

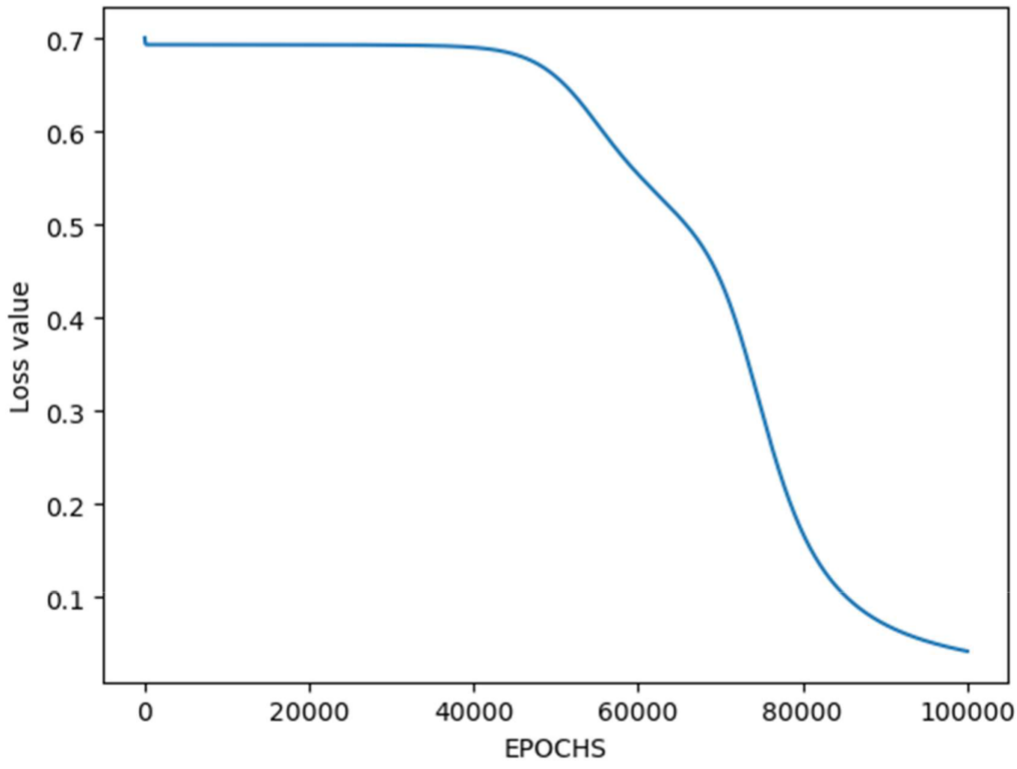
Truth Table for XOR Gate



Perceptron rule

```
    m = X.shape[1]
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    b1 = parameters["b1"]
    b2 = parameters["b2"]

    Z1 = np.dot(W1, X) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = (Z1, A1, W1, b1, Z2, A2, W2, b2)
    logprobs = np.multiply(np.log(A2), Y) + np.multiply(np.log(1 - A2), (1 - Y))

    cost = -np.sum(logprobs) / m
    return cost, cache, A2

# Backward Propagation
def backwardPropagation(X, Y, cache):
    m = X.shape[1]
    (Z1, A1, W1, b1, Z2, A2, W2, b2) = cache

    dZ2 = A2 - Y
    dW2 = np.dot(dZ2, A1.T) / m
    db2 = np.sum(dZ2, axis = 1, keepdims = True)

    dA1 = np.dot(W2.T, dZ2)
    dZ1 = np.multiply(dA1, A1 * (1- A1))
    dW1 = np.dot(dZ1, X.T) / m
    db1 = np.sum(dZ1, axis = 1, keepdims = True) / m

    gradients = {"dZ2": dZ2, "dW2": dW2, "db2": db2,
                 "dZ1": dZ1, "dW1": dW1, "db1": db1}
    return gradients

#  Updating the weights based on the negative gradients def
updateParameters(parameters, gradients, learningRate):
    parameters["W1"] = parameters["W1"] - learningRate * gradients["dW1"]
    parameters["W2"] = parameters["W2"] - learningRate * gradients["dW2"]
    parameters["b1"] = parameters["b1"] - learningRate * gradients["db1"]
    parameters["b2"] = parameters["b2"] - learningRate * gradients["db2"] return
    parameters

#  Model to learn the XNOR truth table
X = np.array([[0, 0, 1, 1], [0, 1, 0, 1]]) # XNOR input Y =
np.array([[1, 0, 0, 1]]) # XNOR output

# Define model parameters
neuronsInHiddenLayers = 2 # number of hidden layer neurons (2)
inputFeatures = X.shape[0] # number of input features (2)
outputFeatures = Y.shape[0] # number of output features (1)
parameters = initializeParameters(inputFeatures, neuronsInHiddenLayers, ou
tputFeatures)
epoch = 100000
```

```
learningRate = 0.01
losses = np.zeros((epoch, 1))

for i in range(epoch):
      losses[i, 0], cache, A2 = forwardPropagation(X, Y, parameters)
      gradients = backwardPropagation(X, Y, cache)
          parameters = updateParameters(parameters, gradients, learningRate)

#  Evaluating the performance
plt.figure() plt.plot(losses)
plt.xlabel("EPOCHS")
plt.ylabel("Loss value")
plt.show()


#  Testing
X = np.array([[1, 1, 0, 0], [0, 1, 0, 1]]) # XNOR input cost, _, A2 =
forwardPropagation(X, Y, parameters) prediction = (A2 > 0.5) *
1.0
#  print(A2)
print(prediction)
```




```
[[0. 1. 1. 0.]]
```

Applications:

While Feed Forward Neural Networks are fairly straightforward, their simplified architecture can be used as an advantage in particular machine learning applications. For example, one may set up a series of feed forward neural networks with the intention of running them independently from each other, but with a mild intermediary for moderation. Like the human brain, this process relies on many individual neurons in order to handle and process larger tasks. As the individual networks perform their tasks independently, the results can be combined at the end to produce a synthesized, and cohesive output.

Conclusion:

Feedforward neural networks are artificial neural networks where the connections between units do not form a cycle. Feedforward neural networks were the first type of artificial neural network invented and are simpler than their counterpart, recurrent neural networks. They are called feedforward because information only travels forward in the network (no loops), first through the input nodes, then through the hidden nodes (if present), and finally through the output nodes.

Reference:

https://builtin.com/data-science/feedforward-neural-network-intro https://brilliant.org/wiki/feedforward-neural-networks/

https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron