Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

Artificial Intelligence and Data Science Department

Subject/Odd Sem 2023-23/Experiment 1

Name : Soham Jadiye	**Class/Roll No. :** D16AD 18	**Grade :**

Title of Experiment : Multilayer Perceptron algorithm to Simulate XOR gate

Objective of Experiment : The objective is to use the Multilayer Perceptron (MLP) algorithm to simulate the XOR gate, a problem that cannot be linearly separated. The goal is to showcase how a simple neural network can learn complex relationships between inputs and outputs through the layers of the network.

Soham Jadiye

Outcome of Experiment : The outcome of this exercise will be a trained MLP model that can accurately mimic the behavior of the XOR gate. This demonstrates the capacity of neural networks to capture nonlinear patterns and solve problems that are not solvable using traditional linear models.

Problem Statement : To implement a multilayer perceptron algorithm to simulate a XOR gate

Description / Theory :

Perceptron:

A perceptron is a fundamental building block of artificial neural networks. It's a simple computational unit that takes multiple input values, applies weights to them, and produces a single output value. Mathematically, the output of a perceptron can be represented as follows:

output = activation_function(sum(input * weight) + bias)

1. Input: An array of input values.
2. Weight: A corresponding array of weights, one for each input.
3. Bias: A constant term added to the weighted sum before passing through the activation function.
4. Activation Function: A function that transforms the weighted sum into the output. Common activation functions include step function, sigmoid, and ReLU.

The perceptron computes a weighted sum of the inputs and biases, applies an activation function to the sum, and produces an output. The activation function introduces nonlinearity, allowing the perceptron to represent more complex relationships between inputs and outputs.

XOR Gate:

The XOR (exclusive OR) gate is a binary logic gate that takes two binary inputs (0 or 1) and produces a binary output (0 or 1). The XOR gate outputs true (1) when the number of true inputs is odd. The truth table for the XOR gate is as follows:

Input A	Input B	Output
0	0	0
0	1	1
1	0	1
1	1	0

The XOR problem is interesting because a single-layer perceptron cannot solve it. The XOR gate's outputs cannot be separated by a single linear decision boundary. In other words, a perceptron can only create a linear separation between classes, and XOR gate's behavior is nonlinear.

To solve the XOR problem, a multilayer perceptron (MLP) with at least one hidden layer is needed. The hidden layer introduces nonlinear transformations that allow the network to capture the XOR gate's behavior. The MLP can learn to create more complex decision boundaries, enabling it to represent the XOR gate's output accurately.

Algorithm/ Pseudo Code / Flowchart (whichever is applicable)

1. Initialize weights and bias to small random values.
2. For each training example (input, target):
   - Compute the weighted sum of inputs and bias.
   - Apply the activation function (often a step function) to the weighted sum to get the predicted class (0 or 1).
   - Calculate the error as the difference between the predicted class and the target class.
   - Update the weights and bias using the learning rate and error: weight = weight + learning_rate * error * input
3. Repeat step 2 for a predefined number of epochs or until the algorithm converges (no misclassified examples).

Program :

In [67]:
```
import numpy as np
import pandas as pd
import tensorflow
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense
```

In [86]:
```
df = pd.DataFrame([[0,0,0],[0,1,1],[1,0,1],[1,1,0]],columns=['x','y','xor'])
```

In [107]:
```
model = Sequential()

model.add(Dense(4,activation='sigmoid',input_dim=2))
# model.add(Dense(2,activation='sigmoid'))
model.add(Dense(1,activation='sigmoid'))
```

In [108]:
```
model.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_29 (Dense)            (None, 4)                 12

 dense_30 (Dense)            (None, 1)                 5

=================================================================
Total params: 17
Trainable params: 17
Non-trainable params: 0
_________________________________________________________________
```

In [111]:
```
optimizer =  keras.optimizers.Adam(learning_rate=0.1)
model.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=['accuracy'])
```

In [112]:
```
model.fit(df.iloc[:,0:-1].values,df['xor'].values,epochs=10,verbose=1,batch_size=1)
```

```
Epoch 1/10
4/4 [==============================] - 0s 3ms/step - loss: 0.0024 - accuracy: 1.0000
Epoch 2/10
4/4 [==============================] - 0s 7ms/step - loss: 0.0017 - accuracy: 1.0000
Epoch 3/10
4/4 [==============================] - 0s 6ms/step - loss: 0.0014 - accuracy: 1.0000
Epoch 4/10
4/4 [==============================] - 0s 8ms/step - loss: 0.0014 - accuracy: 1.0000
Epoch 5/10
4/4 [==============================] - 0s 5ms/step - loss: 0.0014 - accuracy: 1.0000
Epoch 6/10
4/4 [==============================] - 0s 6ms/step - loss: 0.0013 - accuracy: 1.0000
Epoch 7/10
4/4 [==============================] - 0s 5ms/step - loss: 0.0011 - accuracy: 1.0000
Epoch 8/10
4/4 [==============================] - 0s 6ms/step - loss: 9.6853e-04 - accuracy: 1.0000
Epoch 9/10
4/4 [==============================] - 0s 5ms/step - loss: 8.8288e-04 - accuracy: 1.0000
Epoch 10/10
4/4 [==============================] - 0s 6ms/step - loss: 8.2281e-04 - accuracy: 1.0000
```

Out[112]:
```
<keras.callbacks.History at 0x1c81ae99520>
```

In [113]:
```
x_new = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
predictions = [1 if prediction > 0.5 else 0 for prediction in model.predict(x_new)]
for i in range(len(x_new)):
    print(f"Input: {x_new[i]}, Predicted Output: {predictions[i]}")
```

```
1/1 [==============================] - 0s 27ms/step
Input: [0 0], Predicted Output: 0
Input: [0 1], Predicted Output: 1
Input: [1 0], Predicted Output: 1
Input: [1 1], Predicted Output: 0
```

In [ ]:

Results and Discussions : The XOR gate simulation using an MLP illustrates the power of neural networks to handle nonlinearity and learn complex mappings. This simple example highlights the significance of multilayer architectures and appropriate activation functions in solving problems that are beyond the capabilities of linear models. The versatility of neural networks, like the MLP, has contributed to their prominence in modern machine learning and deep learning applications