## EX NO 8. XOR GATE IMPLEMENTATION

### Aim:

To implement multi layer artificial neural network using back propagation algorithm.

### Equipments Required:

1. Hardware – PCs
2. Anaconda – Python 3.7 Installation / Moodle-Code Runner /Google Colab

### Related Theory Concept:

Logic gates are neural networks help to understand the mathematical computation by which a neural network processes its input s to achieve at a certain output. This neural network will deal with the XOR logic problem. An XOR (exclusive OR gate) is a digital logic gate that gives a true output only when both its inputs differ from each other.

The information of a neural network is stored in the interconnections between the neurons i.e. the weights. A neural network learns by updating its weights according to a learning algorithm that helps it converge to the expected output .The learning algorithm is a principled way of changing the weights and biases based on the loss function.

### Algorithm:

1. Import the required libraries.
2. Create the training dataset.
3. Create the neural network model with one hidden layer.
4. Train the model with training data.
5. Now test the model with testing data.

**Program:**

/*

**Program to implement XOR Logic Gate.**

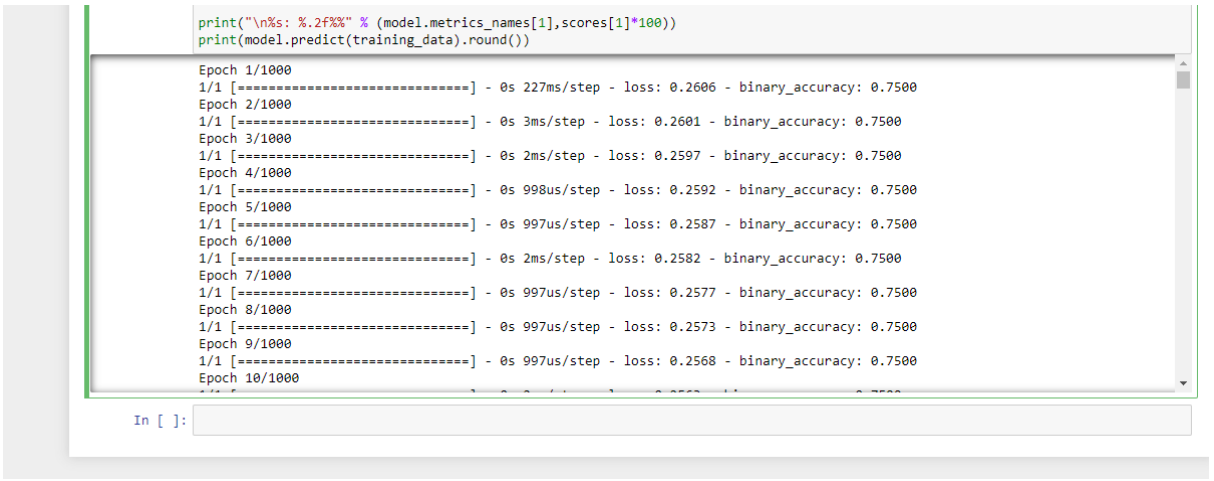
**Developed by :** U. VIVEK KRISHNA

**RegisterNumber :** 212219040180

***/**

```
# XOR Logic Gate Implementation using ANN

import numpy as np from
keras.models import Sequential from
keras.layers.core import Dense

training_data=np.array([[0,0],[0,1],[1,0],[1,1]],"float32")
target_data=np.array([[0],[1],[1],[0]],"float32")

model=Sequential() model.add(Dense(16,input_dim=2,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='mean_squared_error',
                    optimizer='adam',
metrics=['binary_accuracy'])
model.fit(training_data,target_data,epochs=1000)
scores=model.evaluate(training_data,target_data)

print("\n%s: %.2f%%" % (model.metrics_names[1],scores[1]*100))
print(model.predict(training_data).round())
```

## Output:



```
print("\n%s: %.2f%%" % (model.metrics_names[1],scores[1]*100))
print(model.predict(training_data).round())
```

```
Epoch 1/1000
1/1 [==============================] - 0s 227ms/step - loss: 0.2606 - binary_accuracy: 0.7500
Epoch 2/1000
1/1 [==============================] - 0s 3ms/step - loss: 0.2601 - binary_accuracy: 0.7500
Epoch 3/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.2597 - binary_accuracy: 0.7500
Epoch 4/1000
1/1 [==============================] - 0s 998us/step - loss: 0.2592 - binary_accuracy: 0.7500
Epoch 5/1000
1/1 [==============================] - 0s 997us/step - loss: 0.2587 - binary_accuracy: 0.7500
Epoch 6/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.2582 - binary_accuracy: 0.7500
Epoch 7/1000
1/1 [==============================] - 0s 997us/step - loss: 0.2577 - binary_accuracy: 0.7500
Epoch 8/1000
1/1 [==============================] - 0s 997us/step - loss: 0.2573 - binary_accuracy: 0.7500
Epoch 9/1000
1/1 [==============================] - 0s 997us/step - loss: 0.2568 - binary_accuracy: 0.7500
Epoch 10/1000
```

```
In [ ]:
```

## Result:

Thus the python program successully implemented XOR logic gate.