

EX NO 8.

XOR GATE IMPLEMENTATION

Aim:

To implement XOR logic gate using ANN

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 necessary Packages.
2. Set the four different states of the XOR gates.
3. Set the four expected results in the same order.
4. Get the accuracy.
5. Train the model with training data.
6. Test the model with testing data.

Program:

```
/*
```

```
Program to implement ANN by back propagation algorithm.
```

```
Developed by : S.SUDHAKARAN
```

```
RegisterNumber : 212219040155
```

```
*/
```

```
# 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:

```

Epoch 1/1000
1/1 [=====] - 1s 680ms/step - loss: 0.2751 - binary_accuracy: 0.5000
Epoch 2/1000
1/1 [=====] - 0s 7ms/step - loss: 0.2746 - binary_accuracy: 0.2500
Epoch 3/1000
1/1 [=====] - 0s 8ms/step - loss: 0.2742 - binary_accuracy: 0.2500
Epoch 4/1000
1/1 [=====] - 0s 9ms/step - loss: 0.2737 - binary_accuracy: 0.2500
Epoch 5/1000
1/1 [=====] - 0s 7ms/step - loss: 0.2733 - binary_accuracy: 0.2500
Epoch 6/1000
1/1 [=====] - 0s 9ms/step - loss: 0.2729 - binary_accuracy: 0.2500
Epoch 7/1000
1/1 [=====] - 0s 6ms/step - loss: 0.2725 - binary_accuracy: 0.2500
Epoch 8/1000
1/1 [=====] - 0s 9ms/step - loss: 0.2721 - binary_accuracy: 0.2500
Epoch 9/1000
1/1 [=====] - 0s 5ms/step - loss: 0.2717 - binary_accuracy: 0.2500
Epoch 10/1000
1/1 [=====] - 0s 6ms/step - loss: 0.2713 - binary_accuracy: 0.2500
Epoch 11/1000
1/1 [=====] - 0s 10ms/step - loss: 0.2709 - binary_accuracy: 0.2500
Epoch 12/1000
1/1 [=====] - 0s 5ms/step - loss: 0.2705 - binary_accuracy: 0.2500
Epoch 13/1000
1/1 [=====] - 0s 6ms/step - loss: 0.2702 - binary_accuracy: 0.2500
Epoch 14/1000
1/1 [=====] - 0s 13ms/step - loss: 0.2698 - binary_accuracy: 0.2500
Epoch 15/1000
1/1 [=====] - 0s 5ms/step - loss: 0.2695 - binary_accuracy: 0.5000
Epoch 16/1000
1/1 [=====] - 0s 5ms/step - loss: 0.2692 - binary_accuracy: 0.5000
Epoch 17/1000
1/1 [=====] - 0s 5ms/step - loss: 0.2689 - binary_accuracy: 0.5000
Epoch 18/1000
1/1 [=====] - 0s 5ms/step - loss: 0.2685 - binary_accuracy: 0.5000
Epoch 19/1000
1/1 [=====] - 0s 5ms/step - loss: 0.2682 - binary_accuracy: 0.5000
Epoch 20/1000

```

```

Epoch 985/1000
1/1 [=====] - 0s 6ms/step - loss: 0.0358 - binary_accuracy: 1.0000
Epoch 986/1000
1/1 [=====] - 0s 11ms/step - loss: 0.0357 - binary_accuracy: 1.0000
Epoch 987/1000
1/1 [=====] - 0s 14ms/step - loss: 0.0356 - binary_accuracy: 1.0000
Epoch 988/1000
1/1 [=====] - 0s 35ms/step - loss: 0.0355 - binary_accuracy: 1.0000
Epoch 989/1000
1/1 [=====] - 0s 47ms/step - loss: 0.0354 - binary_accuracy: 1.0000
Epoch 990/1000
1/1 [=====] - 0s 22ms/step - loss: 0.0353 - binary_accuracy: 1.0000
Epoch 991/1000
1/1 [=====] - 0s 23ms/step - loss: 0.0353 - binary_accuracy: 1.0000
Epoch 992/1000
1/1 [=====] - 0s 24ms/step - loss: 0.0352 - binary_accuracy: 1.0000
Epoch 993/1000
1/1 [=====] - 0s 17ms/step - loss: 0.0351 - binary_accuracy: 1.0000
Epoch 994/1000
1/1 [=====] - 0s 10ms/step - loss: 0.0350 - binary_accuracy: 1.0000
Epoch 995/1000
1/1 [=====] - 0s 24ms/step - loss: 0.0349 - binary_accuracy: 1.0000
Epoch 996/1000
1/1 [=====] - 0s 20ms/step - loss: 0.0348 - binary_accuracy: 1.0000
Epoch 997/1000
1/1 [=====] - 0s 9ms/step - loss: 0.0347 - binary_accuracy: 1.0000
Epoch 998/1000
1/1 [=====] - 0s 16ms/step - loss: 0.0346 - binary_accuracy: 1.0000
Epoch 999/1000
1/1 [=====] - 0s 17ms/step - loss: 0.0345 - binary_accuracy: 1.0000
Epoch 1000/1000
1/1 [=====] - 0s 10ms/step - loss: 0.0344 - binary_accuracy: 1.0000
1/1 [=====] - 0s 186ms/step - loss: 0.0344 - binary_accuracy: 1.0000

binary_accuracy: 100.00%
[[0.]
 [1.]
 [1.]
 [0.]]

```

Result:

Thus the python program successfully implemented XOR logic gate.