```
#Basic Neuron Model
import numpy as np
import random
def sigmoid(x):
  return 1/(1+np.exp(-x))
def sigmoid_der(x):
  return x*(1-x)
def relu(x):
  return np.maximum(0,x)
def relu_der(x):
  return np.where(x>0,1,0)
np.random.seed(0)
weights=np.random.rand(2)
bias=np.random.rand()
ip=np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])
op=np.array([[0],
              [1],
              [1],
              [0]])
ip_layer=2
hidden_layer=2
op_layer=1
weights_input_hidden = np.random.rand(ip_layer, hidden_layer)
bias_hidden = np.random.rand(1, hidden_layer)
weights_hidden_output = np.random.rand(hidden_layer, op_layer)
bias_output = np.random.rand(1,op_layer)
for i in range(10000):
  hidden_layer_input=np.dot(ip, weights_input_hidden) + bias_hidden
  hidden_layer_output=sigmoid(hidden_layer_input)
  output_layer_input=np.dot(hidden_layer_output,
weights_hidden_output) + bias_output
  predicted_output=sigmoid(output_layer_input)

  #backtracking
  error=op-predicted_output
  d_predicted_output=error*sigmoid_der(predicted_output)
  error_hidden_layer=d_predicted_output.dot(weights_hidden_output.T)
  d_hidden_layer=error_hidden_layer*sigmoid_der(hidden_layer_output)

  #updating
  weights_hidden_output +=
hidden_layer_output.T.dot(d_predicted_output)*0.1
  bias_output += np.sum(d_predicted_output, axis=0, keepdims=True)*0.1
  weights_input_hidden += ip.T.dot(d_hidden_layer)*0.1
  bias_hidden += np.sum(d_hidden_layer, axis=0, keepdims=True)*0.1

for i in range(len(ip)):
```

```python
    print(f"for {ip[i]} - predicted output:{predicted_output[i]} &
actual output:{op[i]}")
```

```
for [0 0] - predicted output:[0.06648959] & actual output:[0]
for [0 1] - predicted output:[0.93672484] & actual output:[1]
for [1 0] - predicted output:[0.93667574] & actual output:[1]
for [1 1] - predicted output:[0.06969297] & actual output:[0]
```