Lab Assignment Group 8

Code:

```
import numpy as np

#Activation sigmoid
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

#Derivative of sigmoid
def sigmoid_derivative(x):
    return x * (1 - x)

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

np.random.seed(42)

input_dim = 2
hidden_dim = 2
output_dim = 1

weights_input_hidden = 2 * np.random.random((input_dim, hidden_dim)) - 1
weights_hidden_output = 2 * np.random.random((hidden_dim, output_dim)) - 1

biases_hidden = np.zeros((1, hidden_dim))
biases_output = np.zeros((1, output_dim))

learning_rate = 0.1
num_epochs = 10000

for epoch in range(num_epochs):
    # Forward propagation
    hidden_layer_input = np.dot(X, weights_input_hidden) + biases_hidden
    hidden_layer_activation = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_activation, weights_hidden_outpu
t) + biases_output
    output_layer_activation = sigmoid(output_layer_input)

    # Backpropagation
    error = y - output_layer_activation
    output_layer_delta = error * sigmoid_derivative(output_layer_activation)

    hidden_layer_error = output_layer_delta.dot(weights_hidden_output.T)
```

```
    hidden_layer_delta = hidden_layer_error * sigmoid_derivative(hidden_layer
_activation)

    weights_hidden_output += hidden_layer_activation.T.dot(output_layer_delta
) * learning_rate
    biases_output += np.sum(output_layer_delta, axis=0, keepdims=True) * lear
ning_rate

    weights_input_hidden += X.T.dot(hidden_layer_delta) * learning_rate
    biases_hidden += np.sum(hidden_layer_delta, axis=0, keepdims=True) * lear
ning_rate

test_input = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
hidden_layer_output = sigmoid(np.dot(test_input, weights_input_hidden) + bias
es_hidden)
predicted_output = sigmoid(np.dot(hidden_layer_output, weights_hidden_output)
+ biases_output)

print("Predicted Output:")
print(predicted_output)
```

```
Predicted Output:
[[0.0961913 ]
 [0.89393519]
 [0.89410922]
 [0.08557778]]
```

```
test_input
```

```
array([[0, 0],
       [0, 1],
       [1, 0],
       [1, 1]])
```

```
y
```

```
array([[0],
       [1],
       [1],
       [0]])
```

```
final_preditions = [1 if predict >= 0.5 else 0 for predict in predicted_outpu
t]
```

Output:

```
final_preditions
```

```
[0, 1, 1, 0]
```