```
import numpy as np

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

def sigmoid_derivative(x):
  return x * (1 - x)

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

gate = input("Enter gate (AND / OR / XOR):
").strip().upper()

if gate == "AND":
  y = np.array([[0],
          [0],
          [0],
          [1]])
elif gate == "OR":
  y = np.array([[0],
          [1],
          [1],
          [1]])
elif gate == "XOR":
  y = np.array([[0],
          [1],
          [1],
          [0]])
else:
  print("Invalid gate! Defaulting to XOR.")
  y = np.array([[0],
          [1],
          [1],
          [0]])

np.random.seed(42)
input_layer_neurons = X.shape[1]
hidden_layer_neurons = 2
output_neurons = 1
W1 =
np.random.uniform(size=(input_layer_neurons,
hidden_layer_neurons))
b1 = np.random.uniform(size=(1,
hidden_layer_neurons))
W2 =
np.random.uniform(size=(hidden_layer_neurons,
output_neurons))
b2 = np.random.uniform(size=(1,
output_neurons))

epochs = 10000
learning_rate = 0.1
for epoch in range(epochs):
  hidden_input = np.dot(X, W1) + b1
  hidden_output = sigmoid(hidden_input)

  final_input = np.dot(hidden_output, W2) + b2
  final_output = sigmoid(final_input)

  error = y - final_output
  d_output = error *
sigmoid_derivative(final_output)

  error_hidden = d_output.dot(W2.T)
  d_hidden = error_hidden *
sigmoid_derivative(hidden_output)

  W2 += hidden_output.T.dot(d_output) *
learning_rate
  b2 += np.sum(d_output, axis=0,
keepdims=True) * learning_rate
  W1 += X.T.dot(d_hidden) * learning_rate
  b1 += np.sum(d_hidden, axis=0,
keepdims=True) * learning_rate

  if epoch % 2000 == 0:
    print(f"Epoch {epoch}, Loss:
{np.mean(np.square(error))}")

print(f"\nFinal Predictions for {gate} gate:")
print(np.round(final_output, 3))
```