```
import numpy as np
import matplotlib.pylab as plt
plt.style.use('seaborn')
```

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

```
def computecost(y, yhat):
  return 0.5 * np.sum((yhat - y) ** 2)


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


def sigmoid_prime(x):
  return sigmoid(x) * (1 - sigmoid(x))
```

```
 def train(epochs, alpha):
  cost = [0 for x in range(epochs)]
  w1 = np.random.random((2, 2))
  b1 = np.random.random((1, 2))
  w2 = np.random.random((2, 1))
  b2 = np.random.random()
  for e in range(epochs):
    for i in range(len(x)):
      # forward propagation
      yin1 = np.dot(x[i], w1) + b1
      y1_predict = sigmoid(yin1)
      yin2 = np.dot(y1_predict, w2) + b2
      y2_predict = sigmoid(yin2)

      cost[e] += computecost(y[i], y2_predict)

      # backward propagation
      y2_error = y[i] - y2_predict
      y2_delta = y2_error * sigmoid_deriv(y2_predict)

      w2 += alpha * np.dot(y1_predict.reshape((2, 1)),
                           y2_delta.reshape((1, 1)))
      b2 += alpha * np.sum(y2_delta, axis=0, keepdims=True)

      y1_error = np.dot(y2_delta, w2.T)
      y1_delta = y1_error * sigmoid_deriv(y1_predict)

      w1 += alpha * np.dot(x[i].reshape((2, 1)), y1_delta)
      b1 += alpha * np.sum(y1_delta, axis=0, keepdims=True)

  return w1, b1, w2, b2, cost
```

```
def main():
```

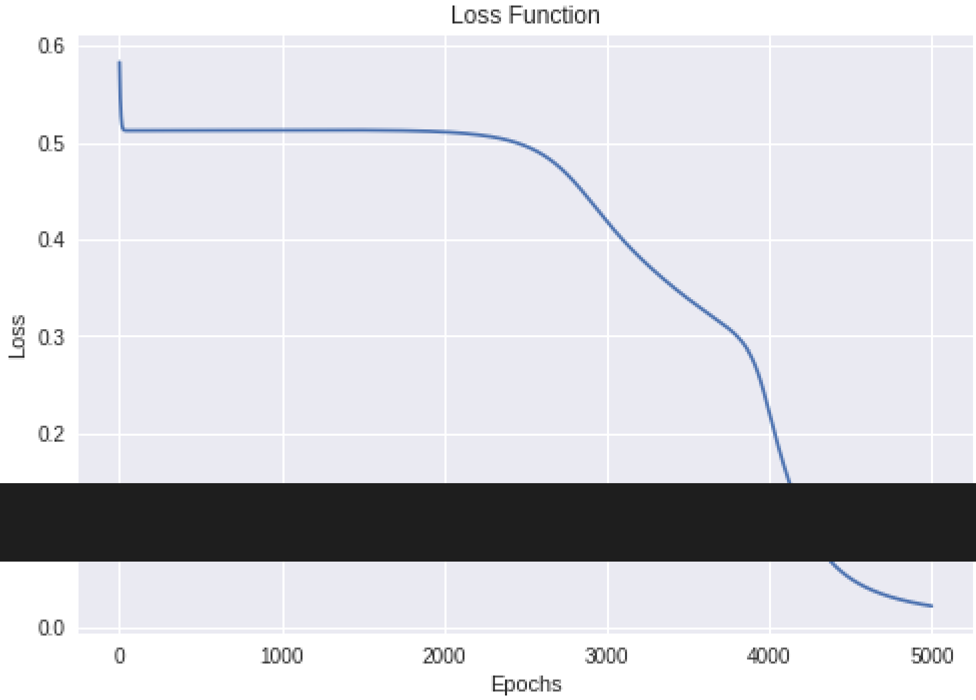
```
  alpha = 0.2
  epochs = 5000
  w1, b1, w2, b2, cost = train(epochs, alpha)
  plt.plot(cost)
  plt.title('Loss Function')
  plt.xlabel('Epochs')
  plt.ylabel('Loss')
  plt.show()

  print("\n\t\tWeights\n")
  print("First layer:")
  print(w1)
  print("Second layer:")
  print(w2)

  print("\n\tBias\n")
  print(f"First layer: {b1}")
  print(f"Second layer: {b2}")


  print("\n\tPredictions\t\n")
  for i in range(2):
    for j in range(2):
      y1 = sigmoid(np.dot([i, j], w1)+b1)
      y2 = sigmoid(np.dot(y1, w2) + b2)
      print(f"{i} {j}: {y2[0][0]}")
```

```
main()
```




```
             Weights

First layer:
[[5.75737991 3.03760004]
 [5.44424683 2.9882537 ]]
Second layer:
[[ 6.41527305]
 [-6.93803524]]

        Bias

First layer: [[-2.12602532 -4.55107464]]
Second layer: [[-2.81199417]]

        Predictions

0 0: 0.09969272428612788
0 1: 0.8981737598017909
1 0: 0.8989845083437787
```