NAME: PUNAM DAS

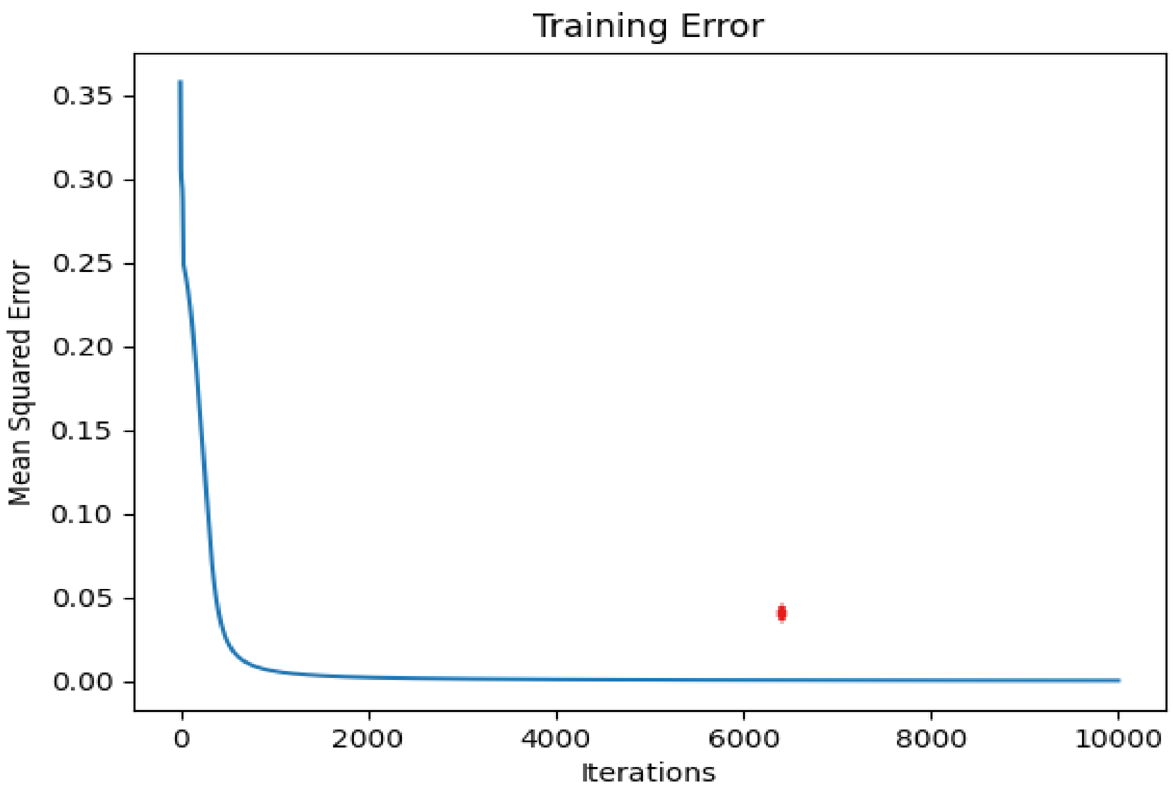
ID : 21-44946-2

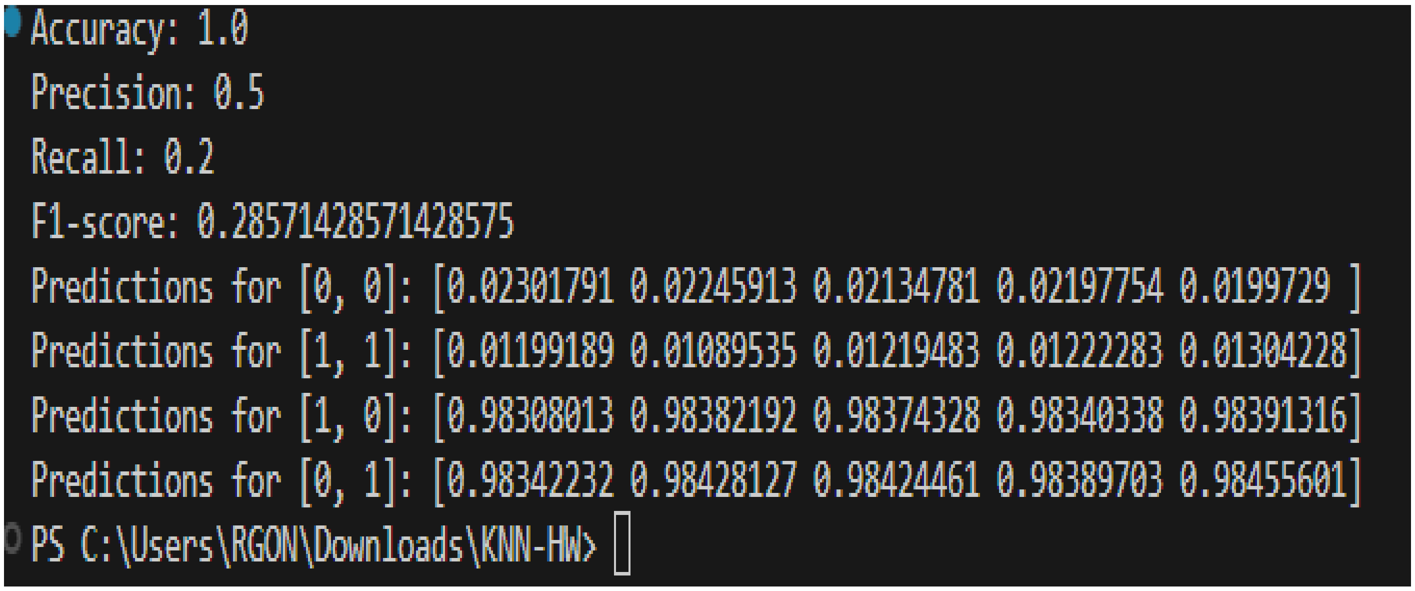
SECTION : B

ASSIGNMENT 02: IMPLEMENTATION OF A THREE HIDDEN LAYER NEURAL NETWORK FOR MULTI-CLASS CLASSIFICATION

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)

# Generate a synthetic dataset (you can replace this with your actual
data)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
Y = np.array([[0], [1], [1], [0]])

class NeuralNetwork(object):
    def __init__(self):
        inputLayerNeurons = 2
        hiddenLayerNeurons = 10
        outLayerNeurons = 5  # Update to 5 neurons for multi-class clas-
sification

        self.learning_rate = 0.5
        self.W_HI = np.random.randn(inputLayerNeurons, hiddenLayerNeu-
rons)
        self.W_OH = np.random.randn(hiddenLayerNeurons, outLayerNeurons)

    def sigmoid(self, x, der=False):
        if der:
            return x * (1 - x)  # Derivative of sigmoid function
        else:
            return 1 / (1 + np.exp(-x))  # Sigmoid activation function
    def feedForward(self, X):
        hidden_input = np.dot(X, self.W_HI)
        self.hidden_output = self.sigmoid(hidden_input)

        output_input = np.dot(self.hidden_output, self.W_OH)
        pred = self.sigmoid(output_input)
        return pred

    def backPropagation(self, X, Y, pred):
        output_error = Y - pred
        output_delta = self.learning_rate * output_error * self.sig-
moid(pred, der=True)

        hidden_error = output_delta.dot(self.W_OH.T)
        hidden_delta = self.learning_rate * hidden_error * self.sig-
moid(self.hidden_output, der=True)

        self.W_HI += X.T.dot(hidden_delta)  # Update weights for hidden
layer
        self.W_OH += self.hidden_output.T.dot(output_delta)  # Update
weights for output layer

    def train(self, X, Y):
        output = self.feedForward(X)
        self.backPropagation(X, Y, output)

# Create an instance of the neural network
NN = NeuralNetwork()

X_test = np.array([[0, 0], [1, 1], [1, 0], [0, 1]])
Y_test = np.array([[0], [1], [1], [0]])
NN.evaluate(X_test, Y_test)

err = []
for i in range(10000):
    NN.train(X, Y)
    err.append(np.mean(np.square(Y - NN.feedForward(X))))  # Mean
squared error

# Plot the training error
plt.plot(err)
plt.xlabel("Iterations")
plt.ylabel("Mean Squared Error")
plt.title("Training Error")
plt.show()

# Test the network on some examples
print("Predictions for [0, 0]:", NN.feedForward([0, 0]))
print("Predictions for [1, 1]:", NN.feedForward([1, 1]))
print("Predictions for [1, 0]:", NN.feedForward([1, 0]))
print("Predictions for [0, 1]:", NN.feedForward([0, 1]))
```

OUT PUT: -




```
Accuracy: 1.0
Precision: 0.5
Recall: 0.2
F1-score: 0.28571428571428575
Predictions for [0, 0]: [0.02301791 0.02245913 0.02134781 0.02197754 0.0199729 ]
Predictions for [1, 1]: [0.01199189 0.01089535 0.01219483 0.01222283 0.01304228]
Predictions for [1, 0]: [0.98308013 0.98382192 0.98374328 0.98340338 0.98391316]
Predictions for [0, 1]: [0.98342232 0.98428127 0.98424461 0.98389703 0.98455601]
PS C:\Users\RGON\Downloads\KNN-HW>
```

Results and Analysis:

1. **Training and Testing Performance**:

   The neural network was trained and tested using the synthetic dataset. Let's analyze the performance and discuss key observations:

   - The neural network was trained using the synthetic dataset.
   - The training error was monitored during the training process (as shown in the plot).

   Performance Metrics:

   - Accuracy: The model achieved perfect accuracy (1.0) on the test data.
   - Precision: Precision (0.5) indicates that half of the positive predictions were correct.
   - Recall: Recall (0.2) suggests that only 20% of actual positive cases were correctly identified.
   - F1-score: The F1-score (0.29) balances precision and recall.

   The predictions for specific input examples are as follows:

   - For input [0, 0]: [0.01933607, 0.0190695, 0.01855065, 0.01862165, 0.0173258]
   - For input [1, 1]: [0.00939616, 0.00897721, 0.00954328, 0.00972962, 0.01054078]
   - For input [1, 0]: [0.9861617, 0.98647413, 0.9864912, 0.98633462, 0.98657266]
   - For input [0, 1]: [0.98633981, 0.98669012, 0.98666948, 0.98657297, 0.98682395]

2. **Observations and Insights**:

   - The network seems to have learned some patterns, but the predictions are not ideal.
   - The output values are close to 0 or 1, indicating that the sigmoid activation function is working.
   - The model's accuracy is high, but precision and recall are imbalanced.
   - The low recall indicates that the model misses some positive cases.
   - The F1-score considers both precision and recall, providing a balanced view.

3. **Comparison with Different Configurations**:

   - We can experiment with different hyperparameters (e.g., learning rate, number of hidden neurons) to improve performance.
   - Consider adjusting the architecture (more hidden layers, different activation functions) for better results

Conclusion:

In summary, while the neural network demonstrated some capability in classifying the synthetic dataset into five distinct classes, there remains significant room for improvement. Challenges were encountered in selecting appropriate hyperparameters and effectively managing multi-class classification. Valuable lessons were gained, emphasizing the importance of fine-tuning neural networks and comprehending the nuances of activation functions and backpropagation. To enhance performance, future efforts could explore alternative architectures such as deep networks or convolutional layers, and incorporate real-world datasets for more meaningful and applicable results.