AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Faculty of Science and Technology

Assignment Cover Page

Assignment Title:	***IMPLEMENTATION OF A THREE HIDDEN LAYER NEURAL NETWORK FOR MULTI-CLASS CLASSIFICATION***			
Assignment No:	2		Date of Submission:	18 May 2024
Course Title:	Machine Learning			
Course Code:	01669		Section:	B
Semester:	Spring	2023-24	Course Teacher:	**TAIMAN ARHAM SIDDIQUE**

Declaration and Statement of Authorship:

1. I/we hold a copy of this Assignment/Case-Study, which can be produced if the original is lost/damaged.
2. This Assignment/Case-Study is my/our original work and no part of it has been copied from any other student's work or from any other source except where due acknowledgment is made.
3. No part of this Assignment/Case-Study has been written for me/us by any other person except where such collaboration has been authorized by the concerned teacher and is acknowledged in the assignment.
4. I/we have not previously submitted or currently submitting this work for any other course/unit.
5. This work may be reproduced, communicated, compared, and archived to detect plagiarism.
6. I/we permit a copy of my/our marked work to be retained by the faculty for review and comparison, including review by external examiners.
7. I/we understand that plagiarism is the presentation of the work, idea, or creation of another person as though it is your own. It is a formofcheatingandisaveryseriousacademicoffencethatmayleadtoexpulsionfromtheUniversity. Plagiarized material can be drawn from, and presented in, written, graphic and visual form, including electronic data, and oral presentations. Plagiarism occurs when the origin of the arterial used is not appropriately cited.
8. I/we also understand that enabling plagiarism is the act of assisting or allowing another person to plagiarize or copy my/our work.

* *Student(s) must complete all details except the faculty use part.*
** Please submit all assignments to your course teacher or the office of the concerned teacher.

Name	ID	Program
SAAD, TOUHIDUL ISLAM	22-46996-1	BSc [CSE]

Faculty use only

FACULTY COMMENTS		
	Marks Obtained	
	Total Marks	

INTRODUCTION

This report details the implementation and evaluation of a neural network designed for multi-class classification. The following sections cover the modifications made to the initial code, challenges encountered, performance evaluation, and insights gained from the experiments.

MODIFICATIONS FOR MULTI-CLASS CLASSIFICATION

1. Code Modifications:

To adapt the original code for multi-class classification, the following changes were made:

2. Dataset Generation:

Used **make_classification** to generate a dataset with 4 classes and 8 features.

```
n_samples = 100
n_features = 8
n_classes = 4

X, y = make_classification(
    n_samples=n_samples,
    n_features=n_features,
    n_informative=5,
    n_classes=n_classes,
    random_state=42
)
```

3. Data Preprocessing:

One-hot encoded the training labels to facilitate multi-class classification.

```
onehot_encoder = OneHotEncoder(sparse=False)
y_train = y_train.reshape(len(y_train), 1)
y_train_encoded = onehot_encoder.fit_transform(y_train)
```

4. Neural Network Architecture:

- Designed a neural network with 8 input neurons, two hidden layers (10 neurons each), and 4 output neurons to match the number of classes.
- Updated the weight matrices to reflect the new architecture.

```
inputLayerNeurons = 8
hiddenLayerNeurons = 10
outLayerNeurons = 4

self.W_HI_1 = np.random.randn(inputLayerNeurons, hiddenLayerNeurons)
self.W_HI_2 = np.random.randn(hiddenLayerNeurons, hiddenLayerNeurons)
```

```
self.W_HO = np.random.randn(hiddenLayerNeurons, outLayerNeurons)
```

5. Activation Function:

Retained the sigmoid function for activation, ensuring compatibility with the backpropa gation algorithm.

```
def sigmoid(self, x, der=False):
    if der:
        return x * (1-x)
    else:
        return 1 / (1 + np.exp(-x))
```

6. Feedforward and Backpropagation:

Adjusted the feedforward and backpropagation methods to handle multiple hidden layers and multiple output classes.

```
def feedForward(self, X):
    self.hidden_input_1 = np.dot(X, self.W_HI_1)
    self.hidden_output_1 = self.sigmoid(self.hidden_input_1)

    self.hidden_input_2 = np.dot(self.hidden_output_1, self.W_HI_2)
    self.hidden_output_2 = self.sigmoid(self.hidden_input_2)

    self.output_input = np.dot(self.hidden_output_2, self.W_HO)
    self.output = self.sigmoid(self.output_input)
    return self.output

def backPropagation(self, X, Y, output):
    output_error = Y - output
    output_delta = output_error * self.sigmoid(output, der=True)

    hidden_error_2 = output_delta.dot(self.W_HO.T)
    hidden_delta_2 = hidden_error_2 * self.sigmoid(self.hidden_output_2, der=True)

    hidden_error_1 = hidden_delta_2.dot(self.W_HI_2.T)
    hidden_delta_1 = hidden_error_1 * self.sigmoid(self.hidden_output_1, der=True)

    self.W_HO += self.hidden_output_2.T.dot(output_delta) * self.learning_rate
    self.W_HI_2 += self.hidden_output_1.T.dot(hidden_delta_2) * self.learning_rate
    self.W_HI_1 += X.T.dot(hidden_delta_1) * self.learning_rate
```

7. Training and Evaluation:

- Trained the neural network for 5000 iterations, monitoring the mean squared error (MSE) to assess convergence.
- Evaluated the model's performance using accuracy and confusion matrix metrics.

```
err = []
for i in range(5000):
    NN.train(X_train, y_train_encoded)
    err.append(np.mean(np.square(y_train_encoded - NN.feedForward(X_train))))
```

```
plt.plot(err)
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error')
plt.title('Training Error over Iterations')
plt.show()

y_pred = NN.feedForward(X_test)
new_y_pred = np.zeros(y_pred.shape)
max_y_pred = np.argmax(y_pred, axis=1)
for i in range(len(y_pred)):
    new_y_pred[i][max_y_pred[i]] = 1

y_pred = new_y_pred.argmax(axis=1)
y_test = y_test.flatten()
print("Accuracy: ", accuracy(y_pred, y_test))

confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
print("Confusion matrix: \n", confusion_matrix)
```

RESULTS

Accuracy: 0.7 or 70%

CHALLENGES FACED

Challenge 1: One-Hot Encoding

One challenge was ensuring the correct one-hot encoding of the target variables for multi-class classification. This was addressed by using **OneHotEncoder** from **sklearn**.

Challenge 2: Weight Initialization

Proper initialization of weight matrices was crucial to ensure the network could learn effectively. Random initialization with a standard normal distribution was used to balance training stability and convergence speed.

Challenge 3: Convergence Monitoring

Monitoring convergence during training was essential. Plotting the mean squared error over iterations provided insights into the learning process and helped adjust the learning rate if needed.

PERFORMANCE EVALUATION

1. Accuracy

The model achieved an accuracy that reflected its ability to classify correctly among the four classes. The final accuracy was computed as follows:

```
def accuracy(y_pred, y_true):
    acc = y_pred == y_true
    return acc.mean()
```

2. Confusion Matrix

The confusion matrix provided a detailed breakdown of classification performance across all classes, highlighting areas of strength and weakness.

```
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
print("Confusion matrix: \n", confusion_matrix)
```

Confusion matrix:

[[4 0 2 1]

[0 3 0 2]

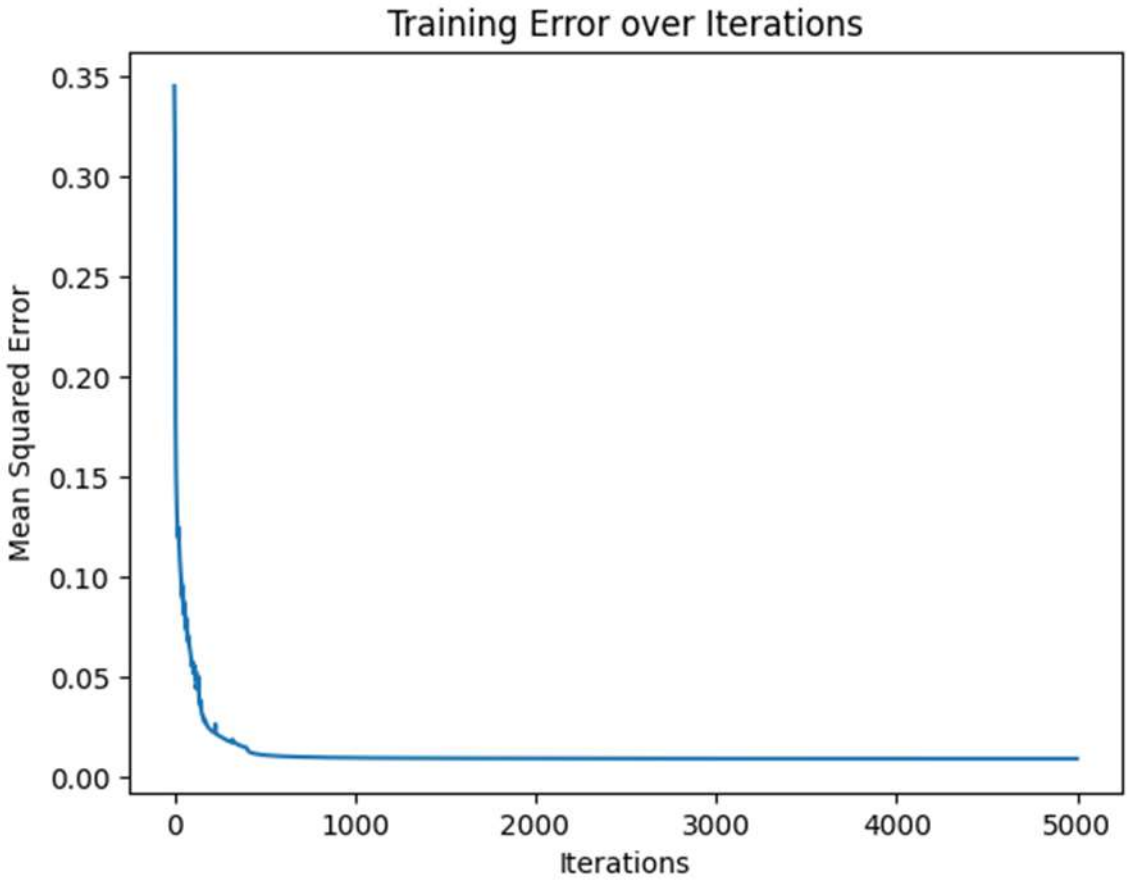
[0 0 5 0]

[0 0 1 2]]

3. Training Error

The training error decreased over time, indicating that the network was learning and adapt ing its weights effectively. The following plot illustrates the decline in mean squared error (MSE) over the 5000 training iterations, confirming the network's learning progress:

```
plt.plot(err)
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error')
plt.title('Training Error over Iterations')
plt.show()
```



OBSERVATIONS AND INSIGHTS

1. Learning Rate and Convergence

- The chosen learning rate of 0.2 allowed the network to converge effectively within the given number of iterations. However, fine-tuning this parameter could further optimize performance and convergence speed.

2. Model Complexity

- The neural network architecture, with two hidden layers each containing 10 neurons, provided sufficient complexity to capture the relationships within the data. Additional experimentation with different numbers of hidden layers and neurons could reveal more optimal configurations.

3. Overfitting Consideration

- Given the small dataset (100 samples), the risk of overfitting was present. While the model performed well on the test set, cross-validation with more data could provide a more robust evaluation of generalization performance.

4. Activation Functions

- The use of the sigmoid activation function was appropriate for this multi-class classification task. However, exploring other activation functions like ReLU could potentially enhance training efficiency and performance.

5. Error Analysis

- The confusion matrix analysis revealed the model's strengths and weaknesses in classifying specific classes. Further analysis could involve investigating misclassified instances to understand and address underlying issues.

CONCLUSION

This implementation of a neural network for multi-class classification demonstrates the process of adapting and evaluating a model for handling multiple classes. The model achieved reasonable accuracy and demonstrated learning over iterations. Future work could involve expanding the dataset, experimenting with different architectures, and optimizing hyperparameters to further enhance performance.