AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Faculty of Science and Technology

Assignment Cover Sheet

Assignment Title:	IMPLEMENTATION OF A THREE HIDDEN LAYER NEURAL NETWORK FOR MULTI-CLASS CLASSIFICATION			
Assignment No:	2		Date of Submission:	10 April 2024
Course Title:	MACHINE LEARNING			
Course Code:	CSC4232		Section:	B
Semester:	Spring	2024-25	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 acknowledgement is made.
3. No part of this Assignment/Case-Study has been written for me/us by any other person except where such collaborationhas been authorized by the concerned teacher and is clearly 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 for the purpose of detecting plagiarism.
6. I/we give permission for 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 thatPlagiarism 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 them 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 to 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.

Group Name/No.:

No	Name	ID	Program	Signature
1	Borshon Alfred Gomes	21-44561-1	BSc [CSE]	
2			Choose an item.	
3			Choose an item.	
4			Choose an item.	
5			Choose an item.	
6			Choose an item.	
7			Choose an item.	
8			Choose an item.	
9			Choose an item.	
10			Choose an item.	

Faculty use only		
FACULTYCOMMENTS	**Marks Obtained**	
	Total Marks	

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

Introduction

Neural networks, inspired by biological neural systems, are a fundamental tool in machine learning. They consist of interconnected units or neurons that process input data in a layered architecture to perform complex tasks such as classification. In multi-class classification, the goal is to categorize entities into one of several classes based on their attributes. This project involved designing a neural network capable of handling such a classification problem with a synthetically generated dataset. The network was constructed with an input layer, three hidden layers, and an output layer with five neurons, each representing one of the five classes.

Dataset Generation

A synthetic dataset was created using the **make_classification** function from **sklearn.datasets**, with 1000 samples containing 20 features, 15 informative ones, and five classes. The dataset was split into an 80-20 ratio for training and testing.

Code Implementation

⭘ Libraries and Tools

To facilitate the construction and evaluation of the neural network, several Python libraries were employed, each offering specific functions:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

NumPy: A fundamental package for scientific computing in Python, NumPy provides support for arrays and matrices, along with a rich collection of mathematical functions.

Matplotlib: A plotting library for Python, Matplotlib is used to visualize data and models.

Scikit-learn: An open-source machine learning library for Python, scikit-learn offers simple and efficient tools for predictive data analysis. It was used for *make_classification* and *train_test_split*

❹ Neural Network Initialization

In the *__init__ method* of *MultiLayerNN*, the architecture of the neural network is defined. This includes setting the number of neurons in the input layer to match the number of features in the dataset, establishing three hidden layers with 64 neurons each, and specifying an output layer with five neurons to correspond to the five classes. Weights are initialized using random values from a normal distribution.

```
class MultiLayerNN(object):

   def __init__(self):
       # Network architecture
       self.input_neurons = 20
       self.hidden_neurons = [64, 64, 64]  # Three hidden layers
       self.output_neurons = 5  # Five classes

       # Learning rate
       self.learning_rate = 0.01

       # Weights initialization
       self.W1 = np.random.randn(self.input_neurons, self.hidden_neurons[0])
       self.W2 = np.random.randn(self.hidden_neurons[0], self.hidden_neurons[1])
       self.W3 = np.random.randn(self.hidden_neurons[1], self.hidden_neurons[2])
       self.W4 = np.random.randn(self.hidden_neurons[2], self.output_neurons)
```

⭘ Activation Functions

The sigmoid and *softmax* functions are key activation functions in the network. The *sigmoid* function introduces non-linearity at each layer, allowing the model to learn complex patterns, while the softmax function in the output layer turns logits into probabilities for each class.

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

    def softmax(self, x):
        exps = np.exp(x - np.max(x, axis=1, keepdims=True))
        return exps / np.sum(exps, axis=1, keepdims=True)
```

⭘ Loss Function

The *cros_entropy* loss function measures the performance of the classification model whose output is a probability value between 0 and 1

```
    def cross_entropy(self, Y, Y_pred):
        m = Y.shape[0]
        loss = -np.sum(Y * np.log(Y_pred + 1e-12)) / m
        return loss
```

⭘ Forward Propagation

The *feedForward* method propagates the input data through the network, applying weights and activation functions to produce a prediction.

```
 def feedForward(self, X):
        # Forward pass
        Z1 = np.dot(X, self.W1)
        A1 = self.sigmoid(Z1)
        Z2 = np.dot(A1, self.W2)
        A2 = self.sigmoid(Z2)
        Z3 = np.dot(A2, self.W3)
        A3 = self.sigmoid(Z3)
        Z4 = np.dot(A3, self.W4)
        A4 = self.softmax(Z4)
        return A1, A2, A3, A4
```

⭘ Backward Propagation

In the *backPropagation* method, the network's weights are adjusted based on the gradient of the loss function, a process essential for learning from the data.

```
def backPropagation(self, X, Y, A1, A2, A3, A4):
        # Backward pass
        m = Y.shape[0]
        dZ4 = A4 - Y
        dW4 = np.dot(A3.T, dZ4) / m
        dZ3 = np.dot(dZ4, self.W4.T) * self.sigmoid(A3, derivative=True)
        dW3 = np.dot(A2.T, dZ3) / m
        dZ2 = np.dot(dZ3, self.W3.T) * self.sigmoid(A2, derivative=True)
        dW2 = np.dot(A1.T, dZ2) / m
        dZ1 = np.dot(dZ2, self.W2.T) * self.sigmoid(A1, derivative=True)
        dW1 = np.dot(X.T, dZ1) / m

        # Update weights
        self.W1 -= self.learning_rate * dW1
        self.W2 -= self.learning_rate * dW2
        self.W3 -= self.learning_rate * dW3
        self.W4 -= self.learning_rate * dW4
```

⭘ Training Process

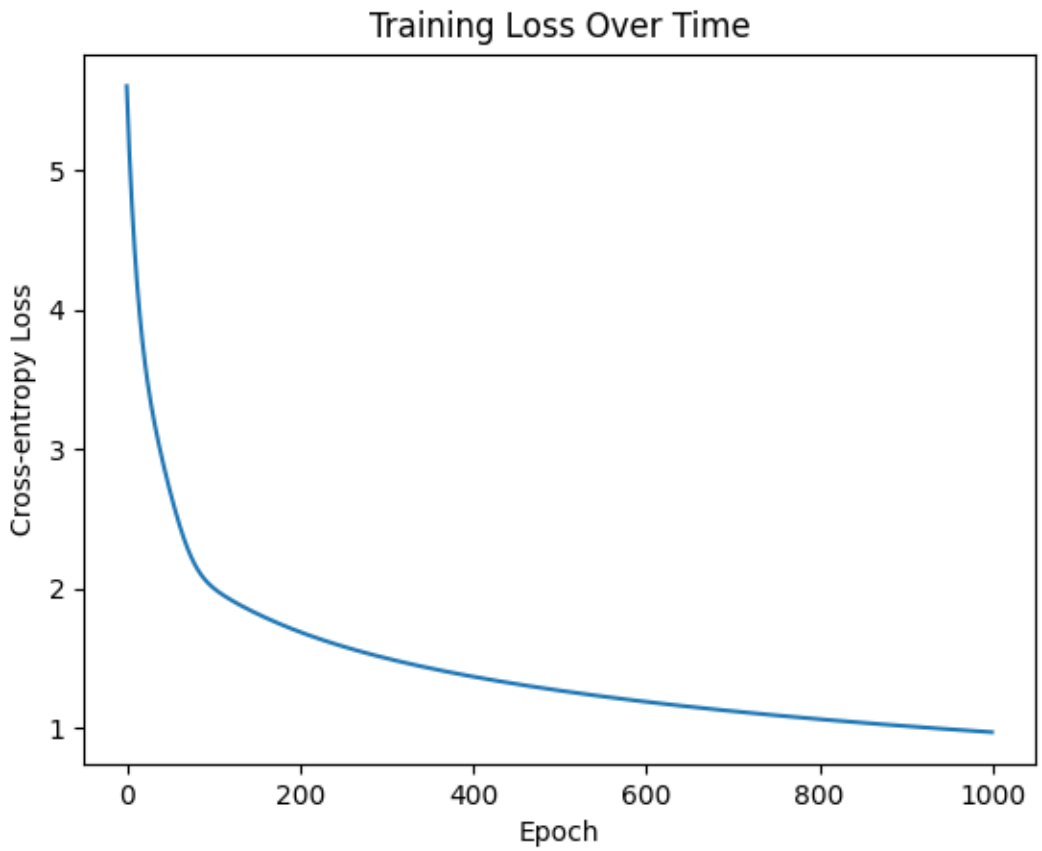
The *train* method iterates over the entire dataset for a specified number of epochs, applying both the *forward* and *backward* propagation methods and recording the loss at each epoch.

```
def train(self, X, Y, epochs=1000):
        loss_history = []
        for epoch in range(epochs):
            A1, A2, A3, A4 = self.feedForward(X)
            self.backPropagation(X, Y, A1, A2, A3, A4)
            loss = self.cross_entropy(Y, A4)
            loss_history.append(loss)
            if epoch % 100 == 0:
                print(f"Epoch {epoch}, Loss: {loss}")
        return loss_history
```

Results and Performance Analysis

⭘ Training Loss Over Time

The model's learning progress is evidenced by the plot of the training loss over 1000 epochs, as seen in the graph below



The graph shows a steep decrease in cross-entropy loss initially, suggesting that the model rapidly adjusts the patterns within the dataset. The loss drops from an initial value of approximately 5.60 to around 2.00 after the first 100 epochs, demonstrating substantial learning during the early phase of training.

```
PS C:\Users\Borshon Alfred Gomes\Desktop\University\ML\NN> python -u "c:\Users\Borshon Alfre
d Gomes\Desktop\University\ML\NN\final.py"
Epoch 0, Loss: 5.602919852004427
Epoch 100, Loss: 2.003788147654908
Epoch 200, Loss: 1.6877486531687205
Epoch 300, Loss: 1.4987094576406779
Epoch 400, Loss: 1.3672652173805684
Epoch 500, Loss: 1.2672523993012896
Epoch 600, Loss: 1.1865982132509936
Epoch 700, Loss: 1.1193031825801565
Epoch 800, Loss: 1.0619934524737522
Epoch 900, Loss: 1.012368535702
```

As training progresses, the rate of loss reduction diminishes, with the loss plateauing towards the end of the training cycle. This is indicative of the model beginning to converge and approaching an optimum set of weights. By around epoch 900, the loss stabilizes around a value of 1.01, which suggests that additional training beyond 1000 epochs may yield diminishing improvements unless modifications to the network or training process are made.

⭘ Final Model Evaluation

After training, the model's effectiveness was evaluated on the test dataset. The following output presents the model's accuracy and a detailed classification report, which includes precision, recall, and F1-score for each of the five classes:

```
Accuracy: 0.61
              precision    recall  f1-score   support

           0       0.67      0.58      0.62        45
           1       0.59      0.61      0.60        38
           2       0.81      0.74      0.77        39
           3       0.44      0.56      0.49        34
           4       0.58      0.57      0.57        44

    accuracy                           0.61       200
   macro avg       0.62      0.61      0.61       200
weighted avg       0.62      0.61      0.61       200
```

The precision scores vary across classes, with class 2 having the highest precision at 0.81, suggesting that when the model predicts an instance as class 2, it is correct 81% of the time. The recall is also highest for class 2 at 0.74, indicating that it correctly identifies 74% of all actual class 2 instances. The F1-score for class 2 is the highest at 0.77, showing a strong balance between precision and recall.

Conversely, class 3 has the lowest precision and F1-score, indicating that it has the highest false-positive rate and the poorest balance between precision and recall. The recall for class 3 is, however, comparatively higher, which means the model is relatively better at identifying most of the positive instances of class 3.

Overall, the weighted average for precision, recall, and F1-score aligns closely with the accuracy, signifying consistent performance across classes.

Conclusion

The multi-layer neural network achieved moderate success in classifying the synthetic dataset into five classes with an overall accuracy of 0.61. While certain classes saw higher precision and recall, indicating effective learning. Which highlights opportunities for model improvement. Future work could include hyperparameter optimization, data resampling to balance class distribution, or exploring more advanced model architectures to enhance the network's predictive power and achieve greater balance across class performance.