CS7267 Programming Assignment #5 (Machine Learning: Fall 2019)
By Neeraj Sharma

Overview

This document is to describe implementation of multilayer feed forward back-propagation algorithm for following tasks

1. XOR Function
2. Perform the classification for Iris dataset

1. **XOR Boolean Function:**

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

2. **IRIS dataset**
   150 observations, 4 features with 3 types of classification, Sentosa, Versicolor, Virginica

Program Run Results

XOR Function:

```
***XOR ANN***

Epochs: 10000 Learning Rate: 0.1

The accuracy of the test dataset is 1.0

Expected Output: [0] vs Actual Output: [0.0282492]

Expected Output: [1] vs Actual Output: [0.9820076]

Expected Output: [1] vs Actual Output: [0.97814312]

Expected Output: [0] vs Actual Output: [0.02347215]
```

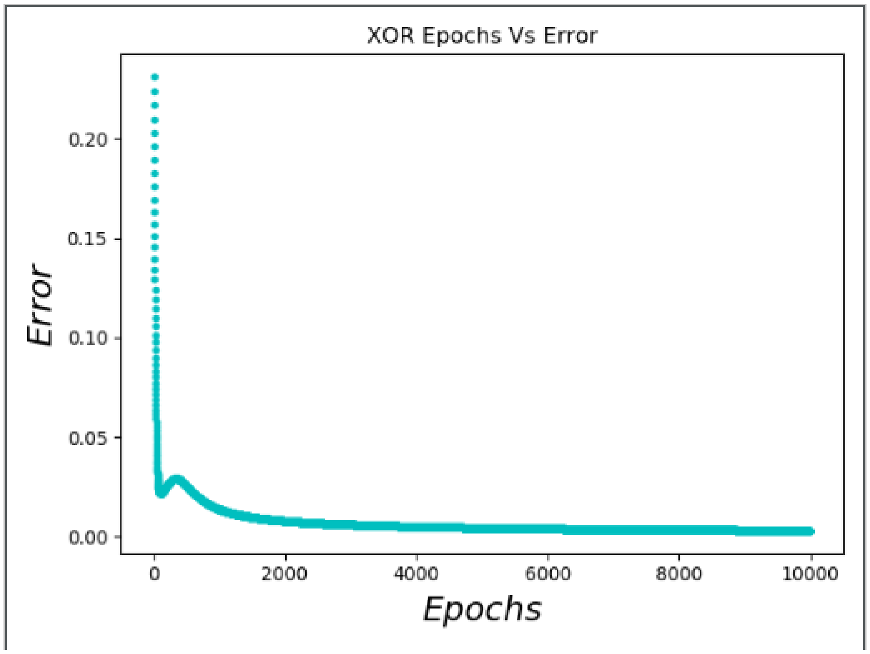
Explanation: This program was run for 10000 epochs with learning rate 0.1.

We got 100% accuracy as we can see expected output and actual output is same.

1st and 4th row actual outputs are very close to zero which is equal to Expected output.

2nd and 3rd row actual outputs are near 1 which is equal to Expected output.

Loss graph:



Loss graph clearly shows that with the epochs increasing first error dropped drastically then error was almost near 0 in with 10000 Epochs.

IRIS Data Set:

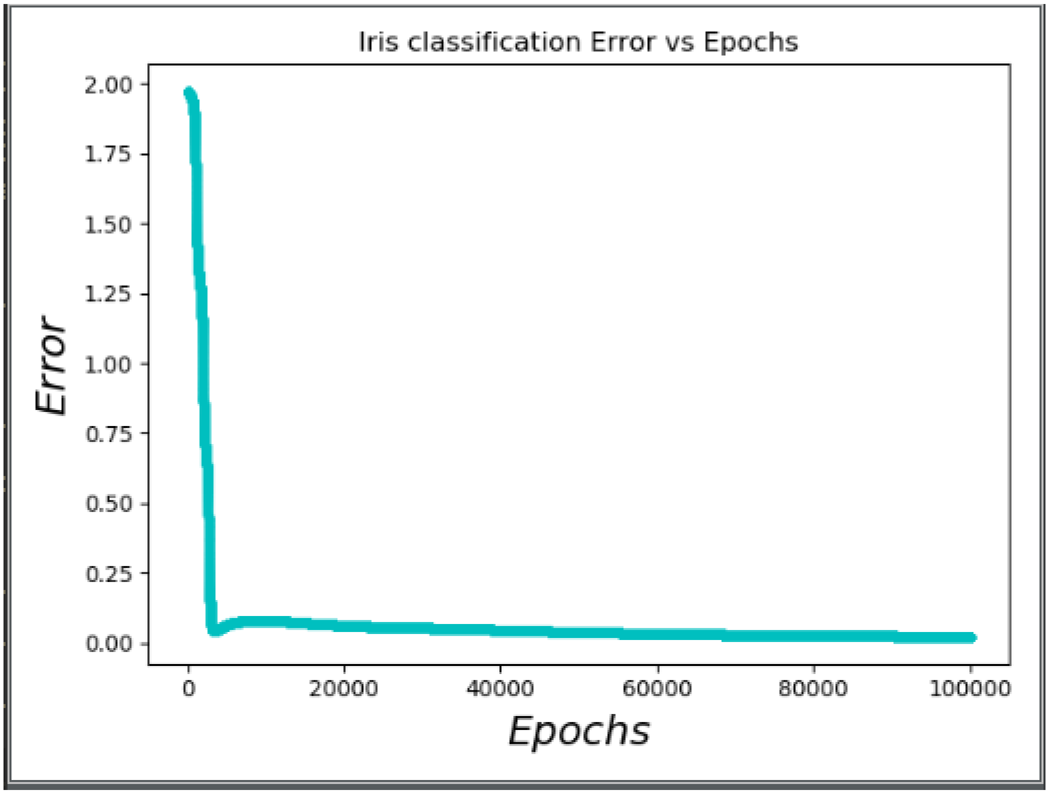
```
***Iris Classification ANN***

Epochs: 100000 Learning Rate: 0.0001

The accuracy of the test dataset is 0.9666666666666667

Expected Output: [0. 1. 0.] Actual output: [0.02640117 0.36030051 0.62789838]

Expected Output: [0. 1. 0.] Actual output: [0.04304946 0.8144195  0.14870474]

Expected Output: [1. 0. 0.] Actual output: [0.94331215 0.07408776 0.00523295]

Expected Output: [1. 0. 0.] Actual output: [0.94520363 0.07186581 0.00514937]

Expected Output: [0. 1. 0.] Actual output: [0.04788417 0.86731453 0.09980411]

Expected Output: [0. 1. 0.] Actual output: [0.05488014 0.917079   0.0575813 ]

Expected Output: [0. 0. 1.] Actual output: [0.01848204 0.11190256 0.89767354]

Expected Output: [1. 0. 0.] Actual output: [0.93858408 0.07963155 0.00545769]
Expected Output: [0. 1. 0.] Actual output: [0.04842081 0.87803847 0.09170476]
Expected Output: [0. 0. 1.] Actual output: [0.01681991 0.0783498  0.9311699 ]
Expected Output: [0. 0. 1.] Actual output: [0.01674904 0.07690752 0.93254465]
Expected Output: [1. 0. 0.] Actual output: [0.94413068 0.07314279 0.00520429]
Expected Output: [1. 0. 0.] Actual output: [0.93982333 0.07811175 0.00535656]
Expected Output: [1. 0. 0.] Actual output: [0.9449114  0.07220216 0.00516698]
Expected Output: [0. 1. 0.] Actual output: [0.05597426 0.92458444 0.05184403]
Expected Output: [1. 0. 0.] Actual output: [0.9452979  0.0717827  0.00515164]
Expected Output: [0. 1. 0.] Actual output: [0.05677644 0.920374   0.05346562]
Expected Output: [1. 0. 0.] Actual output: [0.93984986 0.0782089  0.00543296]
Expected Output: [0. 1. 0.] Actual output: [0.03784732 0.71413168 0.24459305]
Expected Output: [0. 0. 1.] Actual output: [0.01962083 0.13909398 0.86959764]
Expected Output: [0. 1. 0.] Actual output: [0.06162613 0.91501445 0.05155055]
Expected Output: [0. 0. 1.] Actual output: [0.01618062 0.06742714 0.9417689 ]
Expected Output: [0. 0. 1.] Actual output: [0.0159458  0.06367738 0.94534591]
Expected Output: [0. 0. 1.] Actual output: [0.01609693 0.06606921 0.943064  ]
Expected Output: [1. 0. 0.] Actual output: [0.94186827 0.07578932 0.00531558]
Expected Output: [0. 1. 0.] Actual output: [0.05475338 0.91066765 0.0613967 ]
Expected Output: [1. 0. 0.] Actual output: [0.94351984 0.07389309 0.00524334]
Expected Output: [1. 0. 0.] Actual output: [0.94348387 0.07398537 0.00526673]
Expected Output: [1. 0. 0.] Actual output: [0.93749736 0.08090943 0.00548888]
Expected Output: [0. 0. 1.] Actual output: [0.01733488 0.08787007 0.92179357]
```

Explanation: With the 100000 epochs and 0.0001 learning rate accuracy achieved is almost 97%.

I have taken first three outputs for the explanation: Following results shows that first output was wrong and second and third output is correct as the result is 0.8144195, 0.94331215 which is the highest among three classification.

```
Expected Output: [0. 1. 0.] Actual output: [0.02640117 0.36030051 0.62789838]
Expected Output: [0. 1. 0.] Actual output: [0.04304946 0.8144195  0.14870474]
Expected Output: [1. 0. 0.] Actual output: [0.94331215 0.07408776 0.00523295]
```

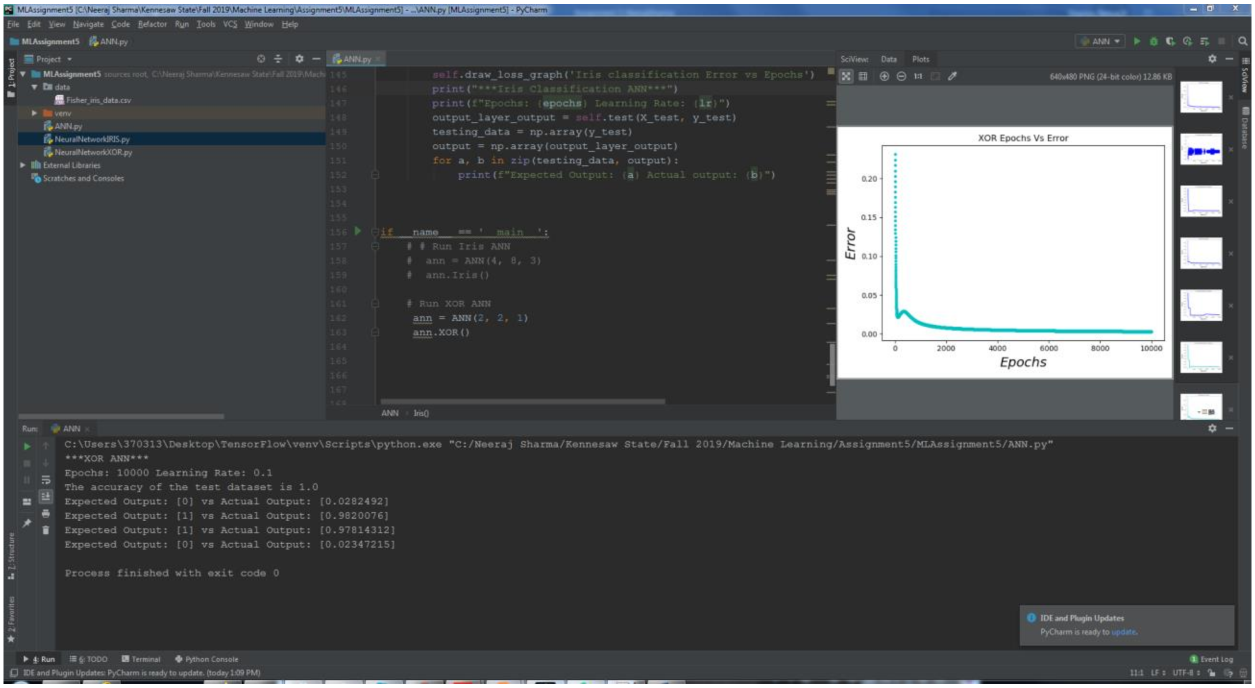
Loss Graph:



Loss graph clearly shows that with the epochs increasing first error dropped drastically then error was almost near 0 in with 100000 Epochs.

Screen shots of the program run in the editor

XOR Function:

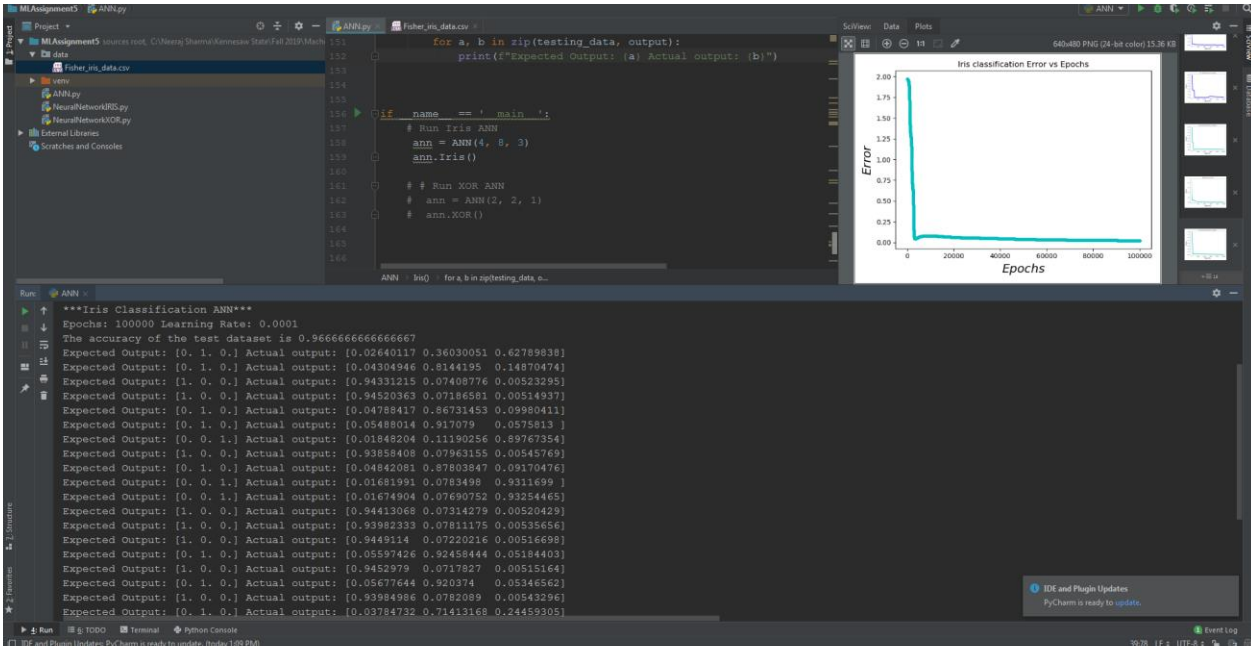


```python
        self.draw_loss_graph('Iris classification Error vs Epochs')
        print("***Iris Classification ANN***")
        print(f"Epochs: {epochs} Learning Rate: {lr}")
        output_layer_output = self.test(X_test, y_test)
        testing_data = np.array(y_test)
        output = np.array(output_layer_output)
        for a, b in zip(testing_data, output):
            print(f"Expected Output: {a} Actual output: {b}")


if __name__ == '__main__':
    # # Run Iris ANN
    #  ann = ANN(4, 8, 3)
    #  ann.Iris()

    # Run XOR ANN
     ann = ANN(2, 2, 1)
     ann.XOR()
```

```
C:\Users\370313\Desktop\TensorFlow\venv\Scripts\python.exe "C:/Neeraj Sharma/Kennesaw State/Fall 2019/Machine Learning/Assignment5/MLAssignment5/ANN.py"
***XOR ANN***
Epochs: 10000 Learning Rate: 0.1
The accuracy of the test dataset is 1.0
Expected Output: [0] vs Actual Output: [0.0282492]
Expected Output: [1] vs Actual Output: [0.9820076]
Expected Output: [1] vs Actual Output: [0.97814312]
Expected Output: [0] vs Actual Output: [0.02347215]

Process finished with exit code 0
```

MLAssignment5 [C:\Neeraj Sharma\Kennesaw State\Fall 2019\Machine Learning\Assignment5\MLAssignment5] - ...\ANN.py [MLAssignment5] - PyCharm
XOR Epochs Vs Error
Error
Epochs
0.20
0.15
0.10
0.05
0.00
0
2000
4000
6000
8000
10000
IDE and Plugin Updates
PyCharm is ready to update.

Classification for Iris dataset:



```python
        for a, b in zip(testing_data, output):
            print(f"Expected Output: {a} Actual output: {b}")


if __name__ == '__main__':
    # Run Iris ANN
     ann = ANN(4, 8, 3)
     ann.Iris()

    # # Run XOR ANN
    #  ann = ANN(2, 2, 1)
    #  ann.XOR()
```

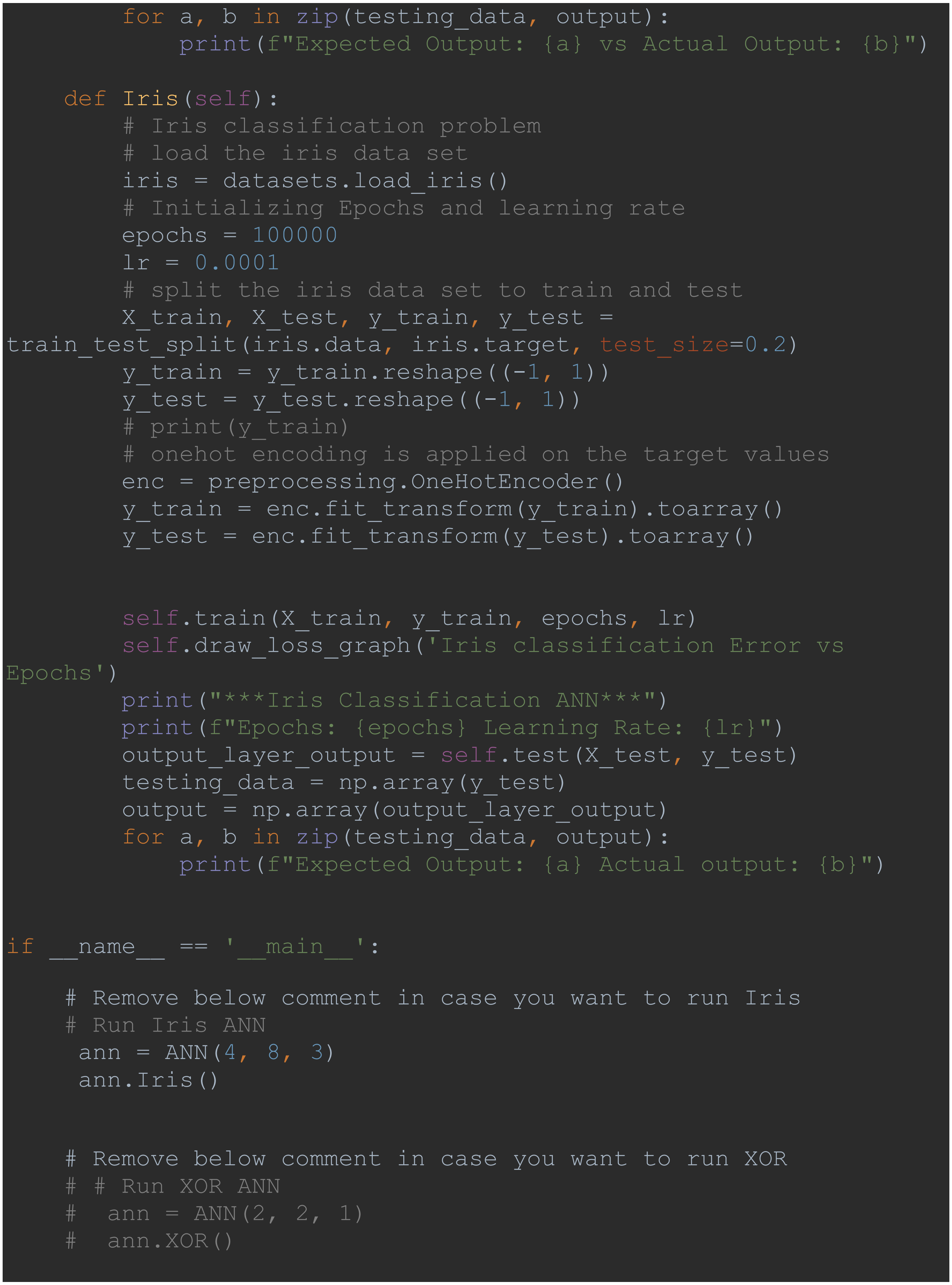
```
***Iris Classification ANN***
Epochs: 100000 Learning Rate: 0.0001
The accuracy of the test dataset is 0.9666666666666667
Expected Output: [0. 1. 0.] Actual output: [0.02640117 0.36030051 0.62789838]
Expected Output: [0. 1. 0.] Actual output: [0.04304946 0.8144195  0.14870474]
Expected Output: [1. 0. 0.] Actual output: [0.94331215 0.07408776 0.00523295]
Expected Output: [1. 0. 0.] Actual output: [0.94520363 0.07186581 0.00514937]
Expected Output: [0. 1. 0.] Actual output: [0.04788417 0.86731453 0.09980411]
Expected Output: [0. 1. 0.] Actual output: [0.05488014 0.917079   0.0575813 ]
Expected Output: [0. 0. 1.] Actual output: [0.01848204 0.11190256 0.89767354]
Expected Output: [1. 0. 0.] Actual output: [0.93858408 0.07963155 0.00545769]
Expected Output: [0. 1. 0.] Actual output: [0.04842081 0.87803847 0.09170476]
Expected Output: [0. 0. 1.] Actual output: [0.01681991 0.0783498  0.9311699 ]
Expected Output: [0. 0. 1.] Actual output: [0.01674904 0.07690752 0.93254465]
Expected Output: [1. 0. 0.] Actual output: [0.94413068 0.07314279 0.00520429]
Expected Output: [1. 0. 0.] Actual output: [0.93982333 0.07811175 0.00535656]
Expected Output: [1. 0. 0.] Actual output: [0.9449114  0.07220216 0.00516698]
Expected Output: [0. 1. 0.] Actual output: [0.05597426 0.92458444 0.05184403]
Expected Output: [1. 0. 0.] Actual output: [0.9452979  0.0717827  0.00515164]
Expected Output: [0. 1. 0.] Actual output: [0.05677644 0.920374   0.05346562]
Expected Output: [1. 0. 0.] Actual output: [0.93984986 0.0782089  0.00543296]
Expected Output: [0. 1. 0.] Actual output: [0.03784732 0.71413168 0.24459305]
```

Program Code:

Single program below covers both problems.

```
# Machine Learning
# Assignment 5 Submission
# Neeraj Sharma


import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

# Class ANN
class ANN:
    # initialize all class veriable through constructor
    def __init__(self, inputLayer, hiddenLayer, outputLayer):
        random.seed(0)
        self.hidden_weights =
np.random.uniform(size=(inputLayer, hiddenLayer))
        self.hidden_bias = np.random.uniform(size=(1,
hiddenLayer))

        self.output_weights =
np.random.uniform(size=(hiddenLayer, outputLayer))
        self.output_bias = np.random.uniform(size=(1,
outputLayer))

        self.epochs = None
        self.lr = None
        self.inputs = None
        self.target = None
        self.error_history=[]

    # train the model for the epochs and learning rate given by
each ANN
    def train(self, train_x, train_y, epochs, lr):
        self.epochs = epochs
        self.lr = lr
        self.inputs = train_x
        self.target = train_y
        for _ in range(epochs):
            self.forward()
            loss = self.loss()
            self.backpropagation(loss)
            self.update_weights()

    # activation sigmoid function
    def sigmoid(self, activation):
        return 1/(1+np.exp(-activation))

    # derivative of the sigmoid function
    def sigmoid_derivative(self, x):
        return x * (1 - x)

    # forward calculation
    def forward(self):
        self.hiddenLayer_activation = np.dot(self.inputs,
self.hidden_weights)
        self.hiddenLayer_activation += self.hidden_bias
        self.hiddenLayer_output =
self.sigmoid(self.hiddenLayer_activation)

        self.outputLayer_activation =
np.dot(self.hiddenLayer_output, self.output_weights)
        self.outputLayer_activation += self.output_bias
        self.outputLayer_output =
self.sigmoid(self.outputLayer_activation)

    # backpropagation
    def backpropagation(self, error):
        self.d_output = error *
(self.sigmoid_derivative(self.outputLayer_output))
        self.error_hidden_layer =
self.d_output.dot(self.output_weights.T)
        self.d_hidden_layer = self.error_hidden_layer *
self.sigmoid_derivative(self.hiddenLayer_output)

    # update the weights and biases
    def update_weights(self):
        self.output_weights +=
self.hiddenLayer_output.T.dot(self.d_output) * self.lr
        self.output_bias += np.sum(self.d_output, axis=0,
keepdims=True) * self.lr

        self.hidden_weights +=
self.inputs.T.dot(self.d_hidden_layer) * self.lr
        self.hidden_bias += np.sum(self.d_hidden_layer, axis=0,
keepdims=True) * self.lr

    # calculate the loss
    def loss(self):
        error = self.target - self.outputLayer_output
        self.error_history.append(-
(np.sum(error))/len(self.target))
```

```
        return error

    # test the model
    def test(self, X, y):
        # forward calculation
        hidden_layer = np.dot(X, self.hidden_weights)
        hidden_layer += self.hidden_bias
        hidden_layer_output = self.sigmoid(hidden_layer)

        output_layer = np.dot(hidden_layer_output,
self.output_weights)
        output_layer += self.output_bias
        output_layer_output = self.sigmoid(output_layer)

        # calculate the accuracy of the model
        accuracy = 0
        for prediction, target in zip(output_layer_output, y):
            if np.argmax(prediction) == np.argmax(target):
                accuracy += 1
        print('The accuracy of the test dataset is ' +
str(accuracy/len(output_layer_output)))
        return output_layer_output


    # draw the loss history our training
    def draw_loss_graph(self, title):
        plt.plot(range(self.epochs), self.error_history, 'c.')
        plt.title(title)
        plt.xlabel('$Epochs$', fontsize=18)
        plt.ylabel("$Error$", rotation=90, fontsize=18)
        plt.show()

    def XOR(self):
            # XOR
        inputs = np.array([[0,0], [0, 1], [1, 0], [1, 1]])
        targets = np.array([[0], [1], [1], [0]])

        # Initializing Epochs and learning rate
        epochs = 10000
        lr = 0.1
        self.train(inputs, targets, epochs, lr)
        self.draw_loss_graph('XOR Epochs Vs Error')
        print("***XOR ANN***")
        print(f"Epochs: {epochs} Learning Rate: {lr}")
        output_layer_output = self.test(inputs, targets)
        testing_data = np.array(targets)
        output = np.array(output_layer_output)
```



```
        for a, b in zip(testing_data, output):
            print(f"Expected Output: {a} vs Actual Output: {b}")

    def Iris(self):
        # Iris classification problem
        # load the iris data set
        iris = datasets.load_iris()
        # Initializing Epochs and learning rate
        epochs = 100000
        lr = 0.0001
        # split the iris data set to train and test
        X_train, X_test, y_train, y_test =
train_test_split(iris.data, iris.target, test_size=0.2)
        y_train = y_train.reshape((-1, 1))
        y_test = y_test.reshape((-1, 1))
        # print(y_train)
        # onehot encoding is applied on the target values
        enc = preprocessing.OneHotEncoder()
        y_train = enc.fit_transform(y_train).toarray()
        y_test = enc.fit_transform(y_test).toarray()


        self.train(X_train, y_train, epochs, lr)
        self.draw_loss_graph('Iris classification Error vs
Epochs')
        print("***Iris Classification ANN***")
        print(f"Epochs: {epochs} Learning Rate: {lr}")
        output_layer_output = self.test(X_test, y_test)
        testing_data = np.array(y_test)
        output = np.array(output_layer_output)
        for a, b in zip(testing_data, output):
            print(f"Expected Output: {a} Actual output: {b}")


if __name__ == '__main__':

    # Remove below comment in case you want to run Iris
    # Run Iris ANN
     ann = ANN(4, 8, 3)
     ann.Iris()


    # Remove below comment in case you want to run XOR
    # # Run XOR ANN
    #  ann = ANN(2, 2, 1)
    #  ann.XOR()
```