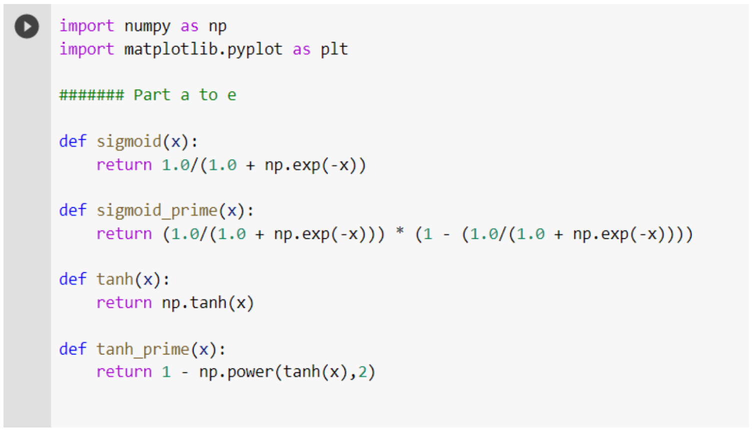
In the name of God

Computational Intelligence Ex3 Report

Ahmadreza Tavana

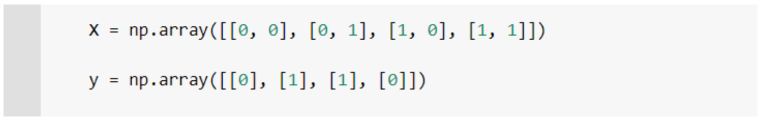
Student Number: 98104852

Part a)



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

####### Part a to e

def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))

def sigmoid_prime(x):
    return (1.0/(1.0 + np.exp(-x))) * (1 - (1.0/(1.0 + np.exp(-x))))

def tanh(x):
    return np.tanh(x)

def tanh_prime(x):
    return 1 - np.power(tanh(x),2)
```

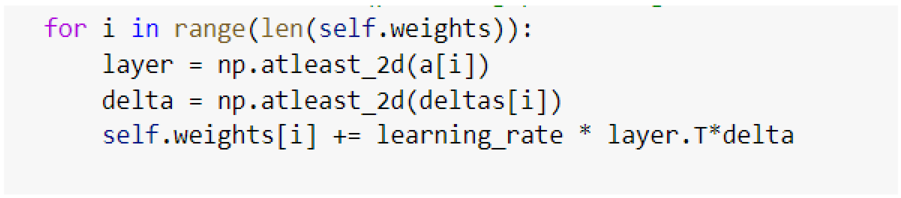
Part b)



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

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

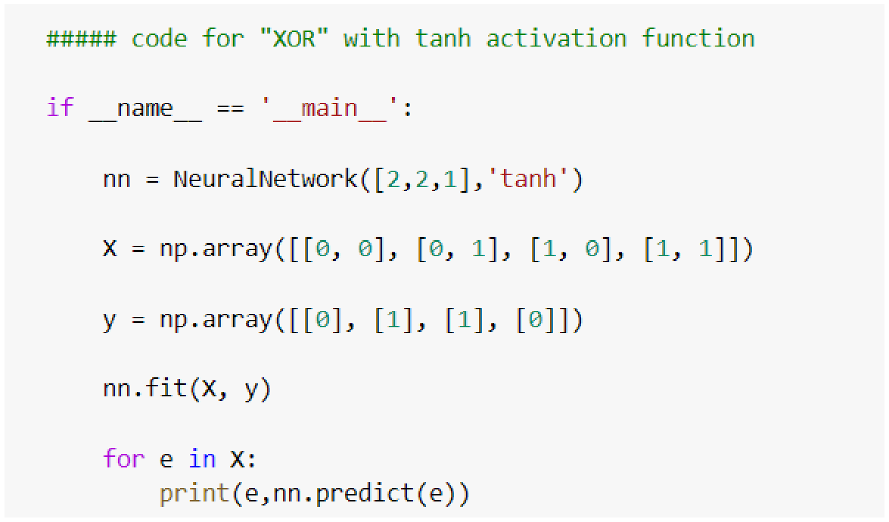
Part c)

At first we check the function **np.atleast_2d .** The numpy.atleast_2d is used to view given inputs as arrays with at least two dimensions. Layer in this code shows the number of layers of our NN. Delta shows the changes in errors use np.atleast_2d function to make it as a (1,1) matrix and then write self.weights as below:



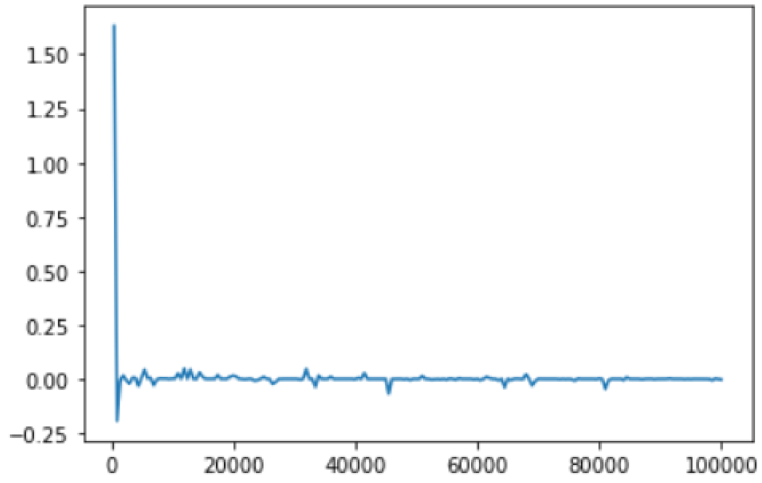
```
for i in range(len(self.weights)):
    layer = np.atleast_2d(a[i])
    delta = np.atleast_2d(deltas[i])
    self.weights[i] += learning_rate * layer.T*delta
```

Part d and e)



```
##### code for "XOR" with tanh activation function

if __name__ == '__main__':

    nn = NeuralNetwork([2,2,1],'tanh')

    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

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

    nn.fit(X, y)

    for e in X:
        print(e,nn.predict(e))
```

```
[0 0] [-0.0006737]
[0 1] [0.99985993]
[1 0] [0.99990347]
[1 1] [0.01110017]
```

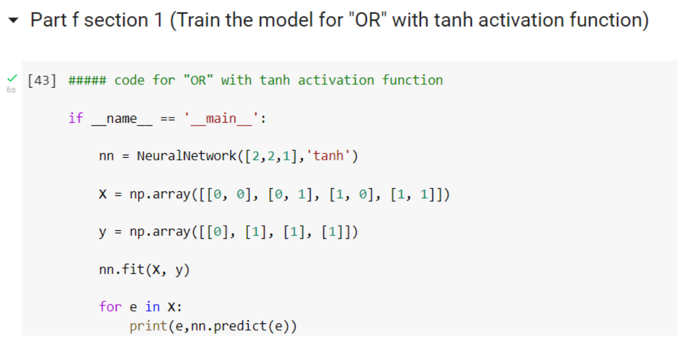


As we can see in the result above, the error converges to 0 and as we can see the results of the outputs for XOR is too close to the real answers (0,1,1,0). It shows that the back propagation algorithm is working well for this part because the NN is learned almost well and error converges to 0. Also I printed the errors in output in each 500 epochs and as you can see in above I put the plot of errors.

Part f)

Section 1 (NN for “OR” function with “tanh” activation function)

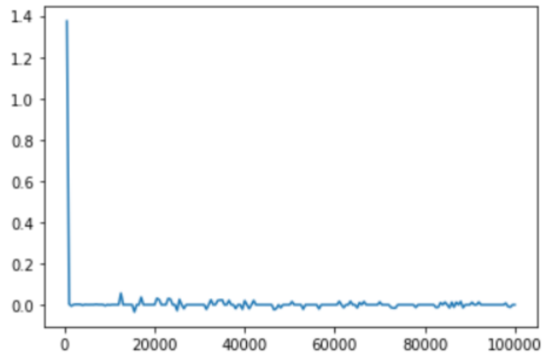
I put results of some different runs of my code in below for different functions. I check “OR and “AND” functions and you can see the results of “OR” function:



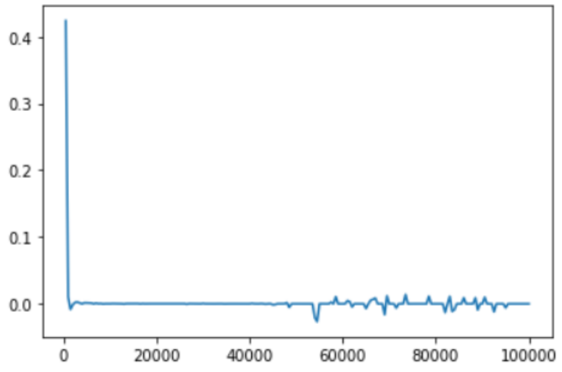
Part f section 1 (Train the model for "OR" with tanh activation function)

```
[43] ##### code for "OR" with tanh activation function

    if __name__ == '__main__':

        nn = NeuralNetwork([2,2,1],'tanh')

        X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

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

        nn.fit(X, y)

        for e in X:
            print(e,nn.predict(e))
```

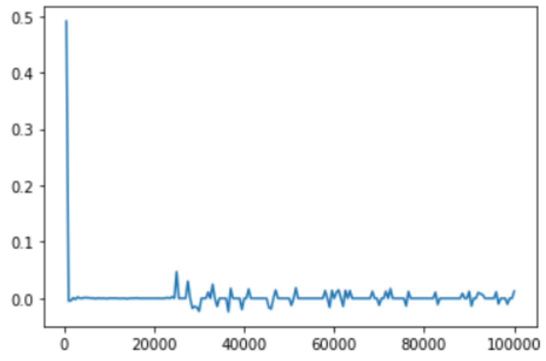
```
[0 0] [-0.01500554]
[0 1] [0.99994698]
[1 0] [0.99994531]
[1 1] [0.9999752]
```



```
[0 0] [0.00695556]
[0 1] [0.99996417]
[1 0] [0.99994489]
[1 1] [0.99998605]
```



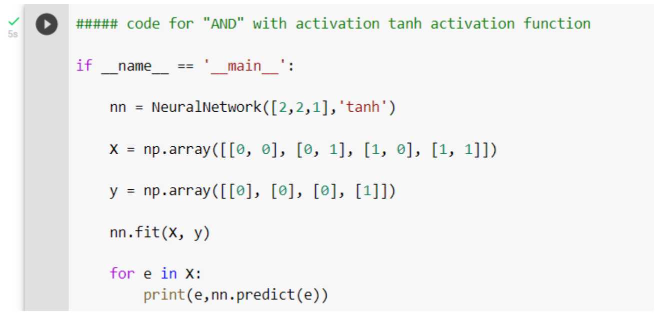
```
[0 0] [-0.01175654]
[0 1] [0.99994883]
[1 0] [0.99995493]
[1 1] [0.99997839]
```



As we can see the results are different but all of them are almost the same and their error converg to 0 and output are close to real answer (0,1,1,1). This difference shows that the way that the network is trained is randomly but as I said the answers are almost the same.

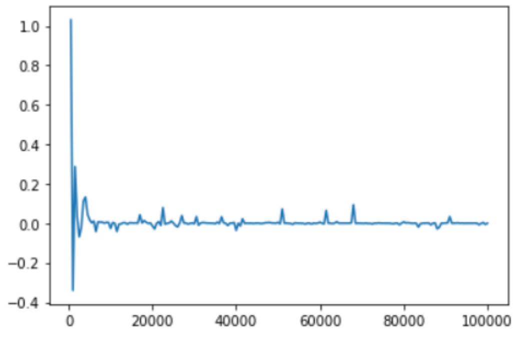
Section 2 (NN for "AND" function with "tanh" activation function)

I put results of some different runs of my code in below. I check "AND" and you can see the results of this function:

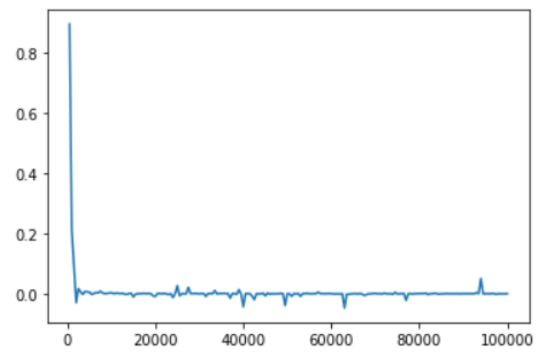


```
##### code for "AND" with activation tanh activation function

if __name__ == '__main__':

    nn = NeuralNetwork([2,2,1],'tanh')

    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

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

    nn.fit(X, y)

    for e in X:
        print(e,nn.predict(e))
```

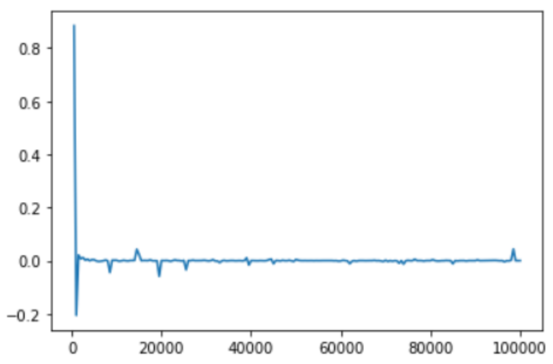
```
[0 0] [8.61757869e-05]
[0 1] [0.0001928]
[1 0] [0.00249018]
[1 1] [0.99978722]
```



```
[0 0] [0.00010956]
[0 1] [3.80579762e-05]
[1 0] [7.39425185e-06]
[1 1] [0.99982455]
```



```
[0 0] [-6.13720178e-05]
[0 1] [0.0013797]
[1 0] [-0.00081136]
[1 1] [0.99983439]
```



As we can see the results are different but all of them are almost the same and their error converge to 0 and output are close to real answer (0,0,0,1). This difference shows that the way that the network is trained is randomly but as I said the answers are almost the same.

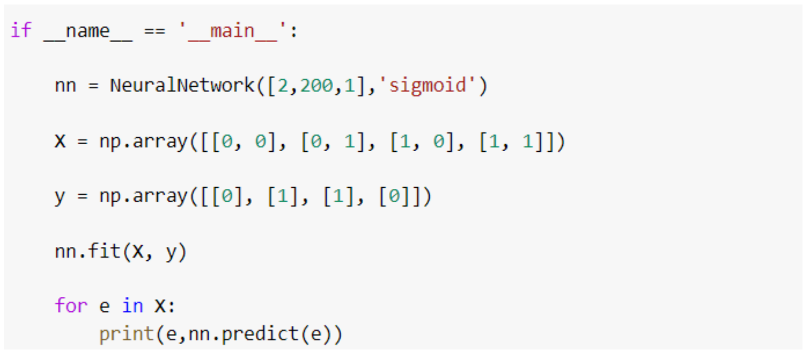
Part g)

For this part, with default parameters, the error doesn’t converge it means that the neural network can’t learn appropriately. So I tried many things to see what results they may have and I came to realize that if I increase the number of epochs and also increase the number of layers and decrease the amount of learning rate, the error converges when epochs increase.

I increase the number of epochs to 1 million and increase the layers of NN to 200 hidden layers and also decrease the learning rate to 0.02. There are the results that I had:

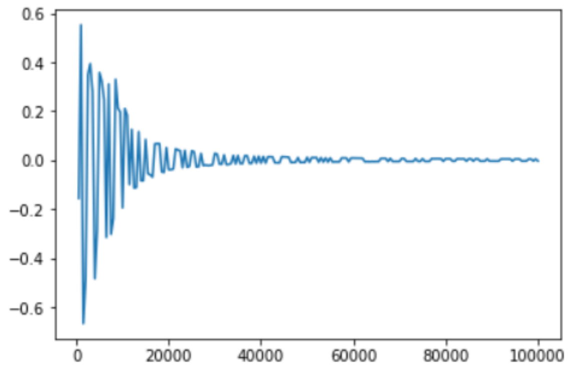
Section 1 (NN for “XOR” function with “logistic” activation function)

In this part I put my different output of my code with changing the default values for “XOR” in below:

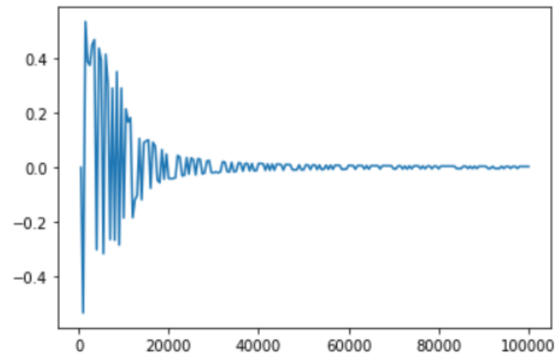


```
if __name__ == '__main__':

    nn = NeuralNetwork([2,200,1],'sigmoid')

    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

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

    nn.fit(X, y)

    for e in X:
        print(e,nn.predict(e))
```

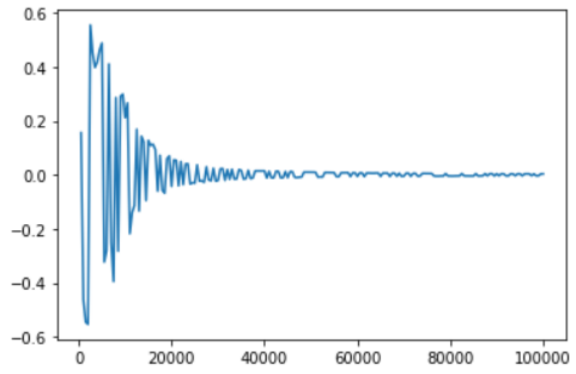
```
[0 0] [0.00331017]
[0 1] [0.99598761]
[1 0] [0.99611492]
[1 1] [0.00348913]
```



```
[0 0] [0.00283592]
[0 1] [0.99629338]
[1 0] [0.99636689]
[1 1] [0.00306279]
```



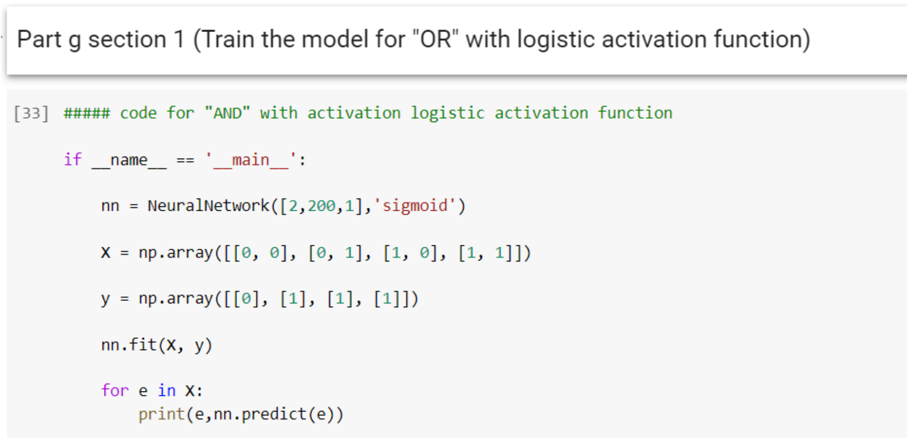
```
[0 0] [0.00281909]
[0 1] [0.99652224]
[1 0] [0.99651358]
[1 1] [0.00308916]
```



As we can see the results are different but all of them are almost the same and their error converge to 0 and output are close to real answer (0,1,1,0). This difference shows that the way that the network is trained is randomly but as I said the answers are almost the same.

Section 2 (NN for “OR” function with “logistic” activation function)

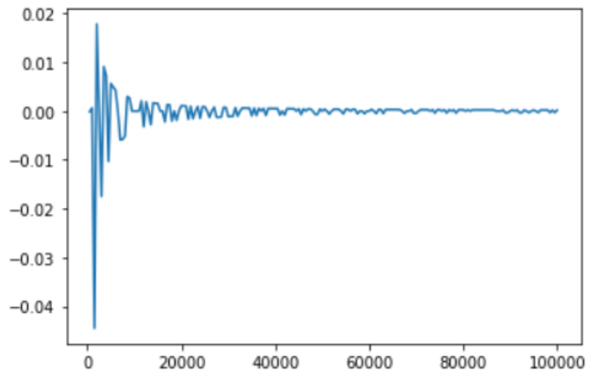
In this part I put my different output of my code with changing the default values for “OR” in below.



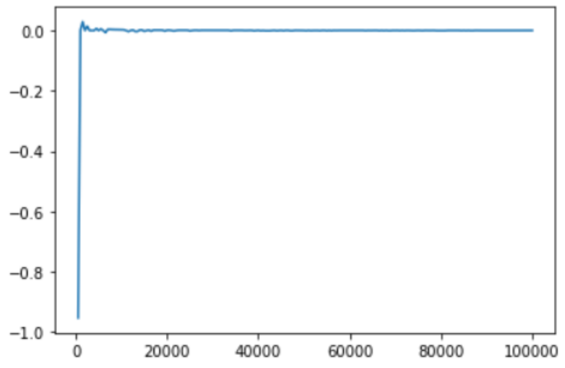
Part g section 1 (Train the model for "OR" with logistic activation function)

```
[33] ##### code for "AND" with activation logistic activation function

     if __name__ == '__main__':

         nn = NeuralNetwork([2,200,1],'sigmoid')

         X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

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

         nn.fit(X, y)

         for e in X:
             print(e,nn.predict(e))
```

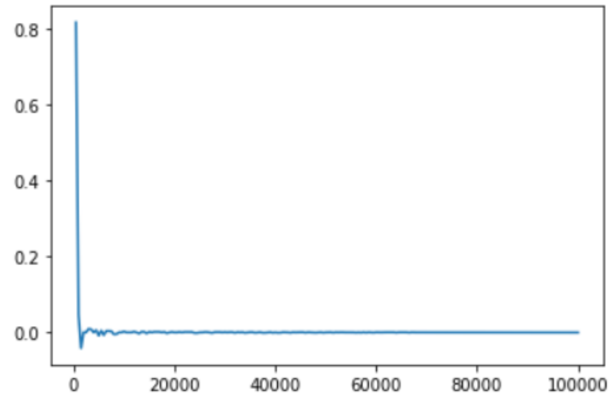
```
[0 0] [0.00029851]
[0 1] [0.9998173]
[1 0] [0.99981879]
[1 1] [1.]
```



```
[0 0] [0.00030817]
[0 1] [0.99981609]
[1 0] [0.99980983]
[1 1] [1.]
```



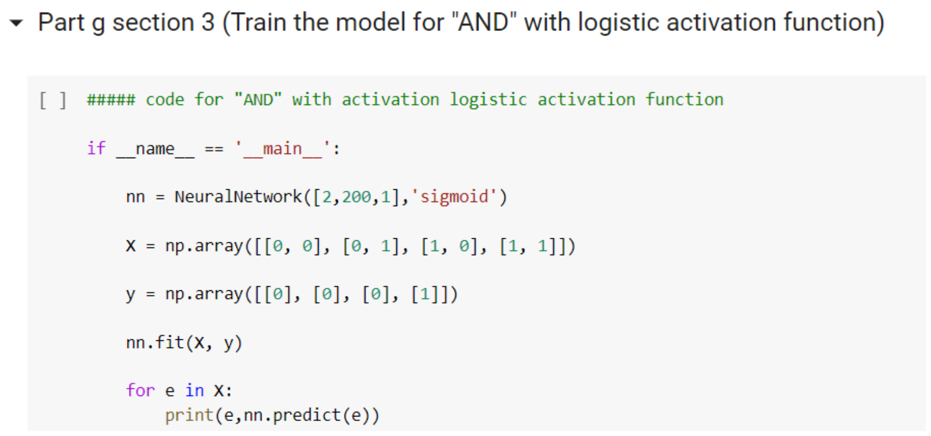
```
[0 0] [0.00029126]
[0 1] [0.99982132]
[1 0] [0.99982582]
[1 1] [1.]
```



As we can see the results are different but all of them are almost the same and their error converge to 0 and output are close to real answer (0,1,1,1). This difference shows that the way that the network is trained is randomly but as I said the answers are almost the same.

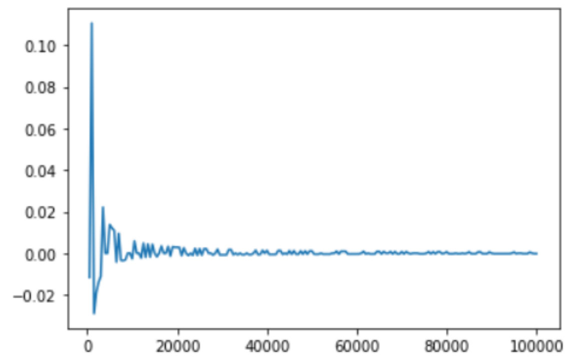
Section 3 (NN for "AND" function with "logistic" activation function)

In this part I put my different output of my code with changing the default values for "AND" in below.

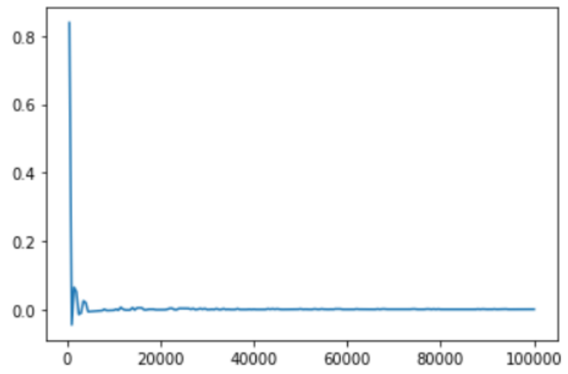

▾ Part g section 3 (Train the model for "AND" with logistic activation function)

```
[ ] ##### code for "AND" with activation logistic activation function

    if __name__ == '__main__':

        nn = NeuralNetwork([2,200,1],'sigmoid')

        X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

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

        nn.fit(X, y)

        for e in X:
            print(e,nn.predict(e))
```

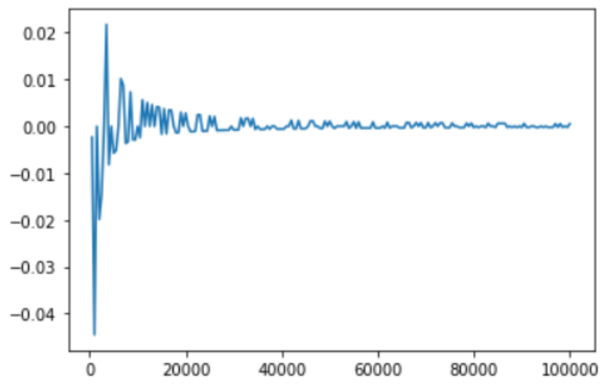
```
[0 0] [1.38607008e-12]
[0 1] [0.00021867]
[1 0] [0.00021906]
[1 1] [0.99949957]
```



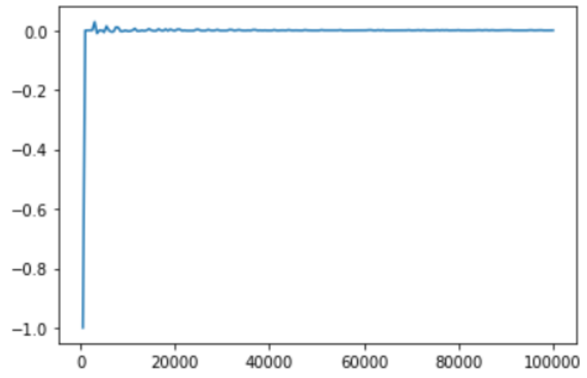
```
[0 0] [6.37476043e-13]
[0 1] [0.00023167]
[1 0] [0.00023644]
[1 1] [0.99946604]
```



```
[0 0] [7.39152457e-13]
[0 1] [0.00021716]
[1 0] [0.00021332]
[1 1] [0.99950603]
```



```
[0 0] [5.33682868e-13]
[0 1] [0.00024263]
[1 0] [0.00024267]
[1 1] [0.9994515]
```



As we can see the results are different but all of them are almost the same and their error converge to 0 and output are close to real answer (0,0,0,1). This difference shows that the way that the network is trained is randomly but as I said the answers are almost the same.