~/week4/MLP_single_hidden.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

MLP_single_hidden.py | or.py | and.py | xor.py

```python
import numpy as np      # for math functions
import pandas as pd     # for importing and using datasets
import random           # for generating random weights

class MLP_single_hidden():

    def __init__(self, numI, numH, numO, x_data, y_data):
        self.numI = numI
        self.numH = numH
        self.numO = numO
        self.x = x_data
        self.y = y_data
        self.epochs = 10000
        self.lr = 0.1
        self.cost_array = []
        self.initialize_weights()
        self.initialize_bias()

    def activation_func(self,x):
        return sigmoid(x)


    def sigmoid(self,x):
        # print("Exp: ", np.exp(-x))
        return 1/(1 + np.exp(-x))

    def sign(self,x):
        res = 0 if x <= 0 else 1
        # print(res)
        return res

    def sigmoid_deriv(self,x):
        return x * ( 1 - x )

    def initialize_weights(self):
        self.hiddenW = np.random.uniform(size=(self.numI, self.numH))
        self.outputW = np.random.uniform(size=(self.numH, self.numO))

    def initialize_bias(self):
        self.hiddenB = np.random.uniform(size=(1,self.numH))
        self.outputB = np.random.uniform(size=(1,self.numO))

```

SRIHARI.S - 2018103601

ML LAB EX-4 - IMPLEMENTATION OF MLP

~/week4/MLP_single_hidden.py - Sublime Text (UNREGISTERED)

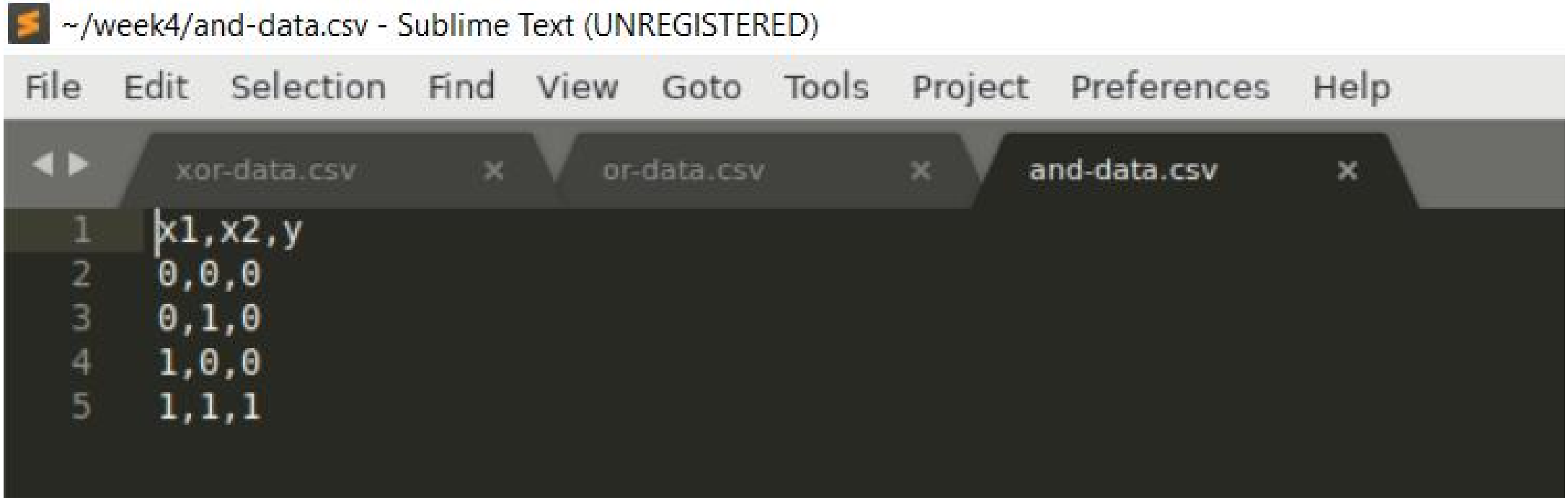
File Edit Selection Find View Goto Tools Project Preferences Help

MLP_single_hidden.py | or.py | and.py | xor.py

```
43      def forward_propagation(self):
44          self.hidden_layer_activation = np.dot(self.x, self.hiddenW)
45          self.hidden_layer_activation += self.hiddenB #considering bias as +1
46          self.hidden_layer_output = self.sigmoid(self.hidden_layer_activation)
47
48          self.output_layer_activation = np.dot(self.hidden_layer_output, self.outputW)
49          self.output_layer_activation += self.outputB
50          self.y_predict = self.sigmoid(self.output_layer_activation)
51
52      def backward_propagation(self):
53          self.error = ((self.y - self.y_predict)) #**2).mean()
54          self.d_y_predict = self.error * self.sigmoid_deriv(self.y_predict) #derivative of sigmoid. After computing we're storing in delta y
55
56          self.error_hidden_layer = self.d_y_predict.dot(self.outputW.T)
57          self.d_hidden_layer = self.error_hidden_layer * self.sigmoid_deriv(self.hidden_layer_output)
58
59      def update_weights(self):
60          self.outputW += self.hidden_layer_output.T.dot(self.d_y_predict) * self.lr
61          self.hiddenW += self.x.T.dot(self.d_hidden_layer) * self.lr
62
63      def update_bias(self):
64          self.outputB += np.sum(self.d_y_predict, axis=0, keepdims=True) * self.lr
65          self.hiddenB += np.sum(self.d_hidden_layer, axis=0, keepdims=True) * self.lr
66
67      def train(self):
68          temp_error = 100000
69          for i in range(self.epochs):
70              self.forward_propagation()
71              self.backward_propagation()
72              self.update_weights()
73              self.update_bias()
74              self.cost_array.append(self.error)
75              self.y_predict1 = self.y_predict
76              self.apply_threshold()
77              if (self.y - self.y_predict).any() == 0:
78                  print(self.y)
79                  print(self.y_predict)
80                  print(self.y - self.y_predict)
81                  print(i)
82                  break
83              self.y_predict = self.y_predict1
84
```

```
85
86      def print_weights(self):
87          print("HiddenW: ",end="")
88          print(*self.hiddenW)
89          print("OutputW: ",end="")
90          print(*self.outputW)
91
92      def print_bias(self):
93          print("HiddenB: ",end="")
94          print(*self.hiddenB)
95          print("OutputB: ",end="")
96          print(*self.outputB)
97
98      def print_y_predict(self):
99          print("\n\nPredicted Value of Y: ", end="")
100         print(*self.y_predict)
101         print("Expected Value of Y:  ", end="")
102         print(*self.y)
103
104     def apply_threshold(self):
105         for i in range(self.y_predict.size):
106             if self.y_predict[i][0]  < 0.5:
107                 self.y_predict[i][0] = 0
108             else:
109                 self.y_predict[i][0] = 1
110
```

IMPLEMENTING AND GATE



~/week4/and.py - Sublime Text (UNREGISTERED)

IMPLEMENTING AND GATE - CODE

File Edit Selection Find View Goto Tools Project Preferences Help

and.py | xor.py

```
from MLP_single_hidden import *
import numpy as np      # for math functions
import pandas as pd     # for importing and using datasets
import random           # for generating random weights

def load_data(filename, target):
    dataset = pd.read_csv(filename, sep=",")  # read .csv file into dataset variable

    print(dataset,"\n")

    x = np.array(dataset.drop([target],1)) # x contains all the features. Does not include target
    y = np.array(dataset[target]) # y contains the target class

    print("\nX = \n",x)
    print("\nY = \n",y)

    print("\n\n")

    return (dataset,x,y,target)

inp = load_data("and-data.csv","y")
x_data = np.array(inp[1])
y_data = np.array([inp[2]])
y_data = np.transpose(y_data)

print(x_data,"\n")
print(y_data)

XOR = MLP_single_hidden(2,2,1,x_data,y_data)

print("Initial: ")
XOR.print_weights()
XOR.print_bias()

XOR.train()

print("\nFinal: ")
XOR.print_weights()
XOR.print_bias()
XOR.apply_threshold()
XOR.print_y_predict()
```

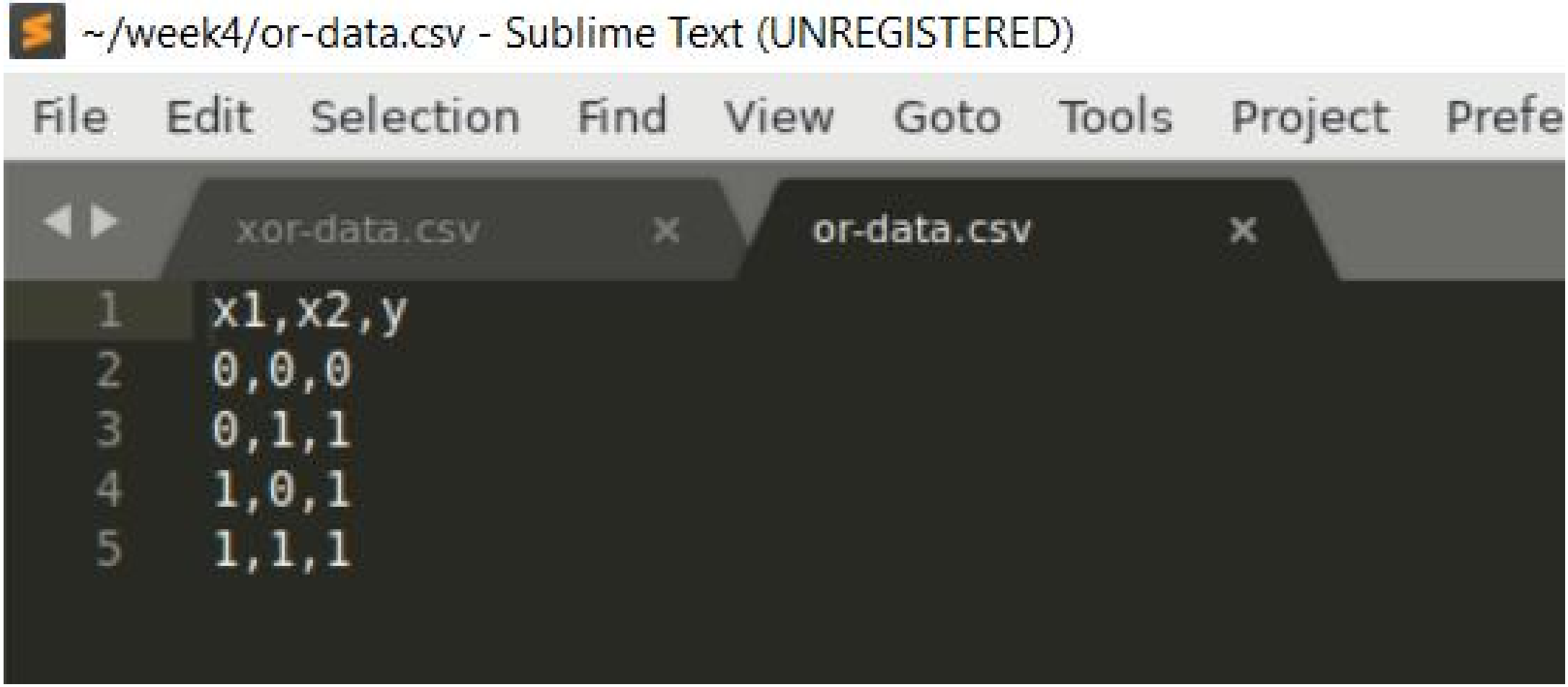
```
Initial:
HiddenW: [0.47729521 0.21235243] [0.72644449 0.78920144]
OutputW: [0.33495144] [0.09471496]
HiddenB: [0.24058661 0.34075707]
OutputB: [0.86031264]
[[0]
 [0]
 [0]
 [1]]
[[0.]
 [0.]
 [0.]
 [1.]]
[[0.]
 [0.]
 [0.]
 [0.]]
1346

Final:
HiddenW: [ 1.07475289 -1.16566318] [ 1.41719835 -0.70536799]
OutputW: [2.33581861] [-1.62081533]
HiddenB: [-0.99022406  0.79078814]
OutputB: [-1.49548224]


Predicted Value of Y: [0.] [0.] [0.] [1.]
Expected Value of Y:  [0] [0] [0] [1]
[s2018103601@centos8-linux Tue Mar 09 04:53 PM week4]$
```

AND GATE - OUTPUT

IMPLEMENTING OR GATE



~/week4/or.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

or.py | and.py | xor.py

IMPLEMENTING OR GATE - CODE

```
from MLP_single_hidden import *
import numpy as np      # for math functions
import pandas as pd     # for importing and using datasets
import random           # for generating random weights

def load_data(filename, target):
    dataset = pd.read_csv(filename, sep=",")  # read .csv file into dataset variable

    print(dataset,"\n")

    x = np.array(dataset.drop([target],1)) # x contains all the features. Does not include target
    y = np.array(dataset[target]) # y contains the target class

    print("\nX = \n",x)
    print("\nY = \n",y)

    print("\n\n")

    return (dataset,x,y,target)

inp = load_data("or-data.csv","y")
x_data = np.array(inp[1])
y_data = np.array([inp[2]])
y_data = np.transpose(y_data)

print(x_data,"\n")
print(y_data)

XOR = MLP_single_hidden(2,2,1,x_data,y_data)

print("Initial: ")
XOR.print_weights()
XOR.print_bias()

XOR.train()

print("\nFinal: ")
XOR.print_weights()
XOR.print_bias()
XOR.apply_threshold()
XOR.print_y_predict()
```

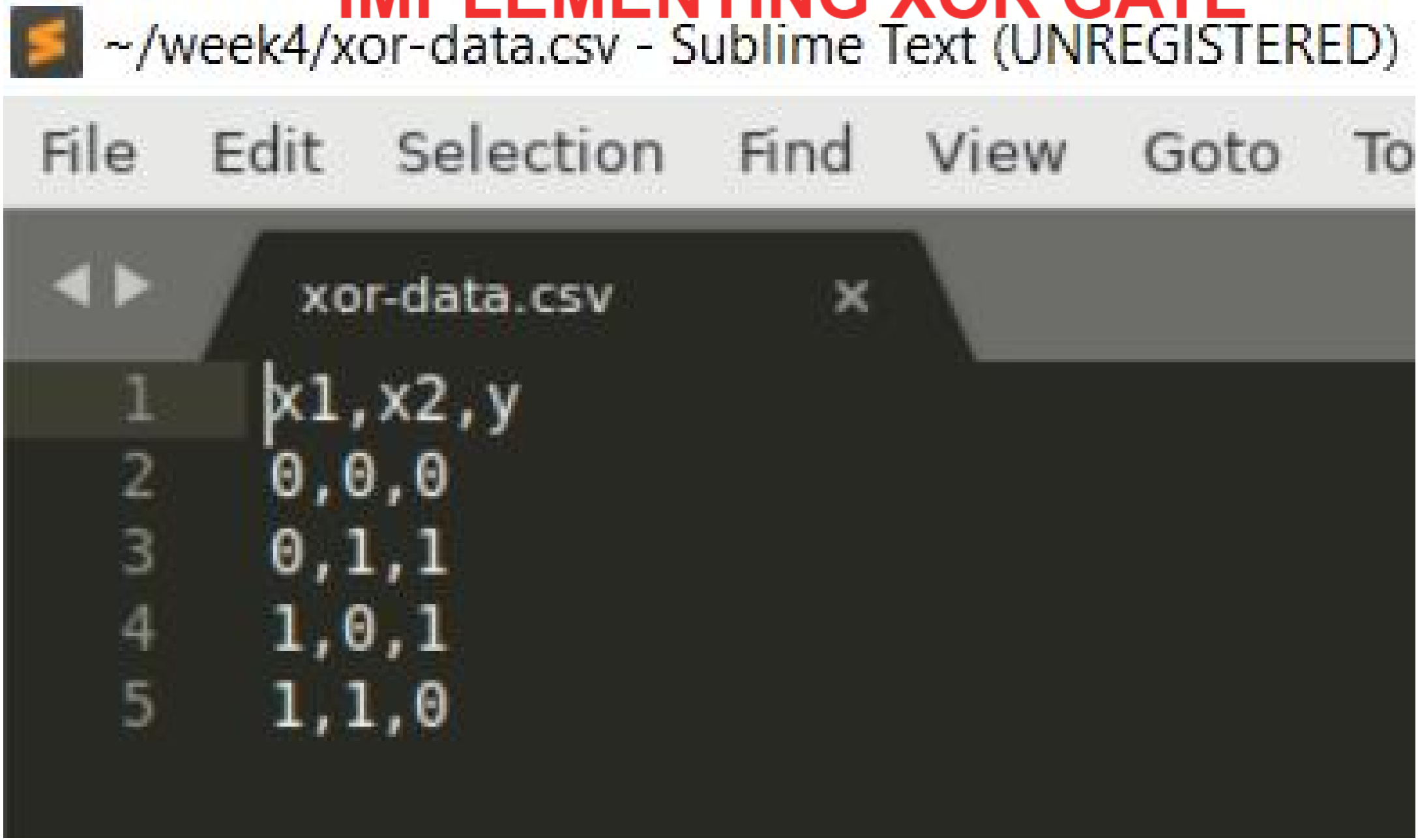
```
Initial:
HiddenW: [0.94745617 0.65488057] [0.2881487  0.38560581]
OutputW: [0.39380629] [0.91564512]
HiddenB: [0.2264022  0.34480522]
OutputB: [0.76159983]
[[0]
 [1]
 [1]
 [1]]
[[0.]
 [1.]
 [1.]
 [1.]]
[[0.]
 [0.]
 [0.]
 [0.]]
693

Final:
HiddenW: [1.46514875 1.66640919] [0.98484523 1.60113446]
OutputW: [1.25980501] [2.171095]
HiddenB: [-0.41077381 -0.79102435]
OutputB: [-1.1841132]


Predicted Value of Y: [0.] [1.] [1.] [1.]
Expected Value of Y:  [0] [1] [1] [1]
[s2018103601@centos8-linux Tue Mar 09 04:53 PM week4]$
```

OR GATE - OUTPUT

IMPLEMENTING XOR GATE



~/week4/xor.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

xor.py

IMPLEMENTING XOR GATE - CODE

```python
from MLP_single_hidden import *
import numpy as np      # for math functions
import pandas as pd     # for importing and using dataset
import random           # for generating random weights

def load_data(filename, target):
    dataset = pd.read_csv(filename, sep=",")  # read .csv file into dataset variable

    print(dataset,"\n")

    x = np.array(dataset.drop([target],1)) # x contains all the features. Does not include target
    y = np.array(dataset[target]) # y contains the target class

    print("\nX = \n",x)
    print("\nY = \n",y)

    print("\n\n")

    return (dataset,x,y,target)

inp = load_data("xor-data.csv","y")
x_data = np.array(inp[1])
y_data = np.array([inp[2]])
y_data = np.transpose(y_data)

print(x_data,"\n")
print(y_data)

XOR = MLP_single_hidden(2,2,1,x_data,y_data)

print("Initial: ")
XOR.print_weights()
XOR.print_bias()

XOR.train()

print("\nFinal: ")
XOR.print_weights()
XOR.print_bias()
XOR.apply_threshold()
XOR.print_y_predict()
```

```
Initial:
HiddenW: [0.39204322 0.74953303] [0.63006061 0.60446034]
OutputW: [0.24259093] [0.56427964]
HiddenB: [0.16783529 0.24310071]
OutputB: [0.97731917]
[[0]
 [1]
 [1]
 [0]]
[[0.]
 [1.]
 [1.]
 [0.]]
[[0.]
 [0.]
 [0.]
 [0.]]
3854

Final:
HiddenW: [1.0911265  4.052881137] [1.27010925 4.17600491]
OutputW: [-2.95190727] [3.96489288]
HiddenB: [-1.25563874 -0.91498964]
OutputB: [-1.74616248]


Predicted Value of Y: [0.] [1.] [1.] [0.]
Expected Value of Y:  [0] [1] [1] [0]
[s2018103601@centos8-linux Tue Mar 09 04:52 PM week4]$
```

XOR GATE - OUTPUT