Feedforward_N-Cases-Solution

April 20, 2021

Nicholas Paisley

1. Your commented, working code.Comments should explain what is going on in the algorithm.
2. Demonstrations of your code results including:

a. An evaluation of your NN accuracy; how well does it predict the observed targets based on the features as inputs.
b. A demonstration of the how the loss function decreases with each iteration (epoch). Plot a graph of loss function versus iteration. Do this for three different learning rates to show how convergence depends on learning rate.

Answers:

1. Code commented

2. a. I noticed that the initial predicted_targets are bad (before training). After training, I noticed that the higher the epoch is, the more consistent the targets predicted get to the targets observed. The lower the epoch, the worse the targets predicted compare to the targets observed.

2. b. After the 3 tests at 100, 500 and 1000 epoch I can see that the convergence of the loss function gets quicker and smooths out faster. The 3 graphs are at the end of each section of code.

EPOCH 100

[1]:
```
#importing libraries
import autograd.numpy as np
from autograd import grad
import matplotlib.pyplot as plt

def sigmoid(x): #Creating a definition for the sigmoid function
    return 1.0/(1+np.exp(-1.0*x)) #This is the sigmoid function. We are using␣
 ↪this because we are looking at a logistic model of an NN (only having 2␣
 ↪solutions).

def feed_forward(features,w1,b1,w2,b2,w3,b3): #Creating a definition for␣
 ↪"feed_foward". Basically the features,weights and biases going through the␣
 ↪NN starting from the first 5 features going to the final 2.
    #Calculating Hidden Layer 1
```

```python
    HL1 = np.matmul(w1,features) #multiplying our weights and features. Where the weights are a [4 x 5] and the the features are a [5 x 22] creating a [4 x 22] new matrix.
    HL1_with_bias = np.add(HL1,b1) #Adding the HL1 matrix (weights * features) which is a [4 x 22] to the biases which are also a [4 x 22].

    # Implement RELU activation (max(0,x))
    HL1_with_bias_and_activation = np.maximum(np.zeros((4,1)),HL1_with_bias) #We need to find the RELU to squish the numbers down to keep the numbers managable.
    HL2 = np.matmul(w2,HL1_with_bias_and_activation) #multiplying our weights and features. Where the weights are a [3 x 4] and the the features are a [4 x 22] creating a [3 x 22] new matrix.
    HL2_with_bias = np.add(HL2,b2) #Adding the HL1 matrix (weights * features) which is a [3 x 22] to the biases which are also a [3 x 22].

    # Implement RELU activation
    HL2_with_bias_and_activation = np.maximum(HL2_with_bias,np.zeros((3,1))) #We need to find the RELU to squish the numbers down to keep the numbers managable.
    targets_predicted = np.matmul(w3,HL2_with_bias_and_activation) #multiplying our weights and features. Where the weights are a [2 x 3] and the the features are a [3 x 22] creating a [2 x 22] new matrix.
    targets_predicted = np.add(targets_predicted,b3) #Adding the HL1 matrix (weights * features) which is a [2 x 22] to the biases which are also a [2 x 22].

    # Use sigmoid for the output activation
    targets_predicted = sigmoid(targets_predicted) #Calling the sigmoid on the targets predicted to get them to become "squished" in between 0 and 1.
    return targets_predicted #prints out the targets predicted.

def loss(features,w1,b1,w2,b2,w3,b3,targets_observed): #A method of evaluating how well your algorithm models your dataset. Greater is bad and lower is good.
    targets_predicted = feed_forward(features,w1,b1,w2,b2,w3,b3) #Getting the targets predicted to put into the loss function
    return np.sum((targets_predicted-targets_observed)**2) #Gives the loss function answer. The closer the sum is to 0, the closer the NN is to achieving the bests SSR.
#======================================================

print('Starting ...')

## Set up training data
## Each row is a case
```

```python
## Columns 0-4 are features
## Columns 5 & 6 are targets

#22 x 7 matrix
features_and_targets = np.array(
                                   [ [0, 0, 0, 0, 0, 0, 1],
                                     [0, 0, 0, 0, 1, 0, 1],
                                     [0, 0, 0, 1, 1, 0, 1],
                                     [0, 0, 1, 1, 1, 0, 1],
                                     [0, 1, 1, 1, 1, 0, 1],
                                     [1, 1, 1, 1, 0, 0, 1],
                                     [1, 1, 1, 0, 0, 0, 1],
                                     [1, 1, 0, 0, 0, 0, 1],
                                     [1, 0, 0, 0, 0, 0, 1],
                                     [1, 0, 0, 1, 0, 0, 1],
                                     [1, 0, 1, 1, 0, 0, 1],
                                     [1, 1, 0, 1, 0, 0, 1],
                                     [0, 1, 0, 1, 1, 0, 1],
                                     [0, 0, 1, 0, 1, 0, 1],
                                     [1, 0, 1, 1, 1, 1, 0],
                                     [1, 1, 0, 1, 1, 1, 0],
                                     [1, 0, 1, 0, 1, 1, 0],
                                     [1, 0, 0, 0, 1, 1, 0],
                                     [1, 1, 0, 0, 1, 1, 0],
                                     [1, 1, 1, 0, 1, 1, 0],
                                     [1, 1, 1, 1, 1, 1, 0],
                                     [1, 0, 0, 1, 1, 1, 0]  ]
                           , dtype=float)

# shuffle our cases (to create randomness)
np.random.shuffle(features_and_targets)

# Need to transpose to get them as 5 X N matrices
features = np.transpose(features_and_targets[:,0:5]) #Tranposes the matrix to 5␣
 ↪x 22. Picking up the 0-4 index.
print(features) #prints the features in the array

# Need to transpose to get the 2 x N matrices
targets_observed = np.transpose(features_and_targets[:,5:7]) #Transposing the␣
 ↪last 2 columns from N x 2 to a 2 x N. Picking up the 5 and 6 index
number_of_features,number_of_cases = features.shape #[5,22] (creating tuple␣
 ↪(unchangable))
print(number_of_features) #prints the number of features in the array
print(number_of_cases) #prints the number of cases in the array

#Set initial weights and biases
```

```
weights_1 = np.random.rand(4,5) #Setting the dimensions of the matrix for our␣
↪random weights (rows by columns)
biases_1 = np.random.rand(4,number_of_cases) #Setting the dimensions of the␣
↪matrix for our baises (rows by columns)

weights_2 = np.random.rand(3,4) #Setting the dimensions of the matrix for our␣
↪random weights (rows by columns)
biases_2 = np.random.rand(3,number_of_cases) #Setting the dimensions of the␣
↪matrix for our baises (rows by columns)

weights_3 = np.random.rand(2,3) #Setting the dimensions of the matrix for our␣
↪random weights (rows by columns)
biases_3 = np.random.rand(2,number_of_cases) #Setting the dimensions of the␣
↪matrix for our baises (rows by columns)

Targets_Predicted =␣
↪feed_forward(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3)␣
↪#Targets_Predicted before Epoch (before training is initialized)

#Printing the results to the following below:
print(' Features : ',features)
print(' Targets : ', targets_observed)
print(' Targets predicted : ', Targets_Predicted)

#Learning rate gives the rate of speed where the gradient moves during gradient␣
↪descent. Setting it too high would make your path instable, too low would␣
↪make convergence slow. Put it to zero means your model isn't learning␣
↪anything from the gradients.

learning_rate = 0.01 #The amount of change to the model during each iterations

# Find slope functions using autograd
d_by_w1 = grad(loss,1) #create the derivative/gradient --> stored d_by_w1,␣
↪(equation(loss), index=1 (weights_1))
d_by_b1 = grad(loss,2) #create the derivative/gradient --> stored d_by_b1,␣
↪(equation(loss), index=2 (biases_1))
d_by_w2 = grad(loss,3) #create the derivative/gradient --> stored d_by_w2,␣
↪(equation(loss), index=3 (weights_2))
d_by_b2 = grad(loss,4) #create the derivative/gradient --> stored d_by_b2,␣
↪(equation(loss), index=4 (biases_2))
d_by_w3 = grad(loss,5) #create the derivative/gradient --> stored d_by_w3,␣
↪(equation(loss), index=5 (weights_3))
d_by_b3 = grad(loss,6) #create the derivative/gradient --> stored d_by_b3,␣
↪(equation(loss), index=6 (biases_3))

epoch_list1 = [] #creating list for epoch iterations
```

```
lost1 = [] #creating list for loss fucntion results

#Setting the iteration (making higher helps making the loss function look␣
↪better)
for epoch in range(100):

    #At each iteration update weights and biases by subtracting
    #learning_rate times slope
    #Changing the random weights and random biases updating each of them after␣
↪each iteration according to range(X)
    weights_1 -=␣
↪learning_rate*d_by_w1(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    biases_1 -=␣
↪learning_rate*d_by_b1(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    weights_2 -=␣
↪learning_rate*d_by_w2(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    biases_2 -=␣
↪learning_rate*d_by_b2(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    weights_3 -=␣
↪learning_rate*d_by_w3(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    biases_3 -=␣
↪learning_rate*d_by_b3(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar

    #Print out the latest value of the loss
    #We would expect this to go down with each iteration
    #This shows how good of a test it was to get to the desired the results
    #The closer to 0, the better the NN is learning <- This is what the loss␣
↪function is doing in the equation.
   ␣
↪print(epoch,loss(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,targets_

    epoch_list1.append(epoch) #epoch list adding the iterations
    lost1.
↪append(loss(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,targets_obser
↪#loss function list adding for each iteration

Targets_Predicted =␣
↪feed_forward(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3)␣
↪#Targets_Predicted after Epoch (After training is complete)

#Printing the results to the following below:
print(' Features : ',features)
print(' Targets : ', targets_observed)
print(' Targets predicted : ', Targets_Predicted)

N = 22
```

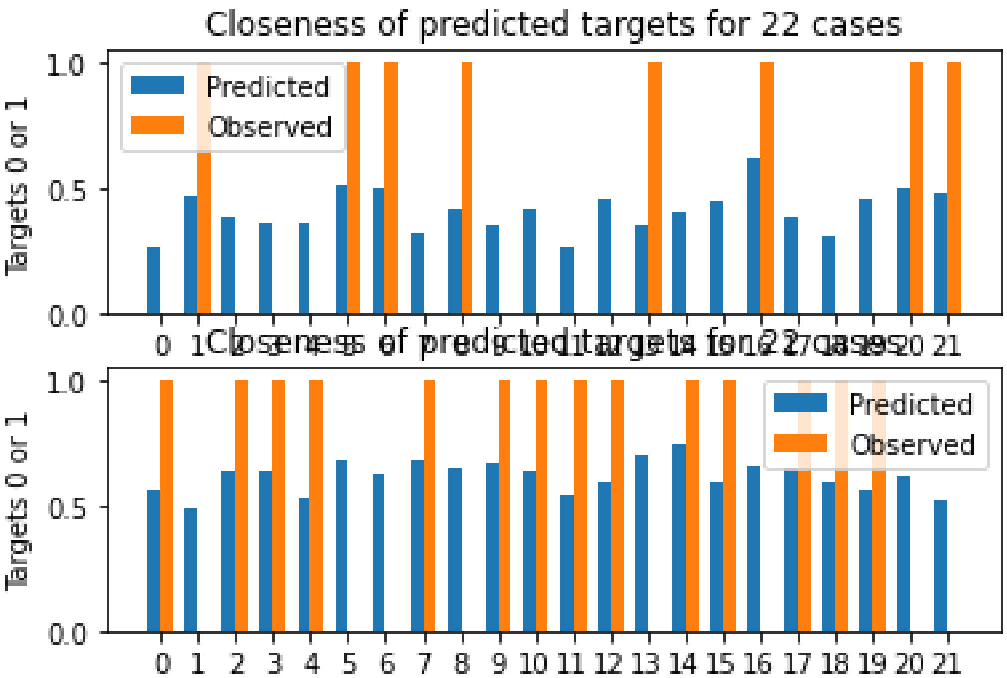
```python
target1_predicted = Targets_Predicted[0,:] #The target we are trying to achieve
target2_predicted = Targets_Predicted[1,:] #The target we are trying to achieve
target1_observed = targets_observed[0,:] #The target we want to achieve (what␣
 ↪the targets should be)
target2_observed = targets_observed[1,:] #The target we want to achieve (what␣
 ↪the targets should be)

ind = np.arange(N) #Return evenly spaced values within a given interval (N=22)
width = 0.35 #creating width to add distance between the 2 different bar graphs
plt.subplot(2,1,1) #creating first graph
plt.bar(ind, target1_predicted, width, label='Predicted') #Plotting the bar␣
 ↪graph of the predicted answers (First set)
plt.bar(ind + width, target1_observed, width,label='Observed') #Plotting the␣
 ↪bar graph of the observed answers (First set)
plt.ylabel('Targets 0 or 1') #y label
plt.title('Closeness of predicted targets for 22 cases') #title
plt.xticks(ind + width / 2, ind) #tick mark indicator
plt.legend(loc='best') #legend (automatically put in best location)
plt.subplot(2,1,2) #creating first graph
plt.bar(ind, target2_predicted, width, label='Predicted') #Plotting the bar␣
 ↪graph of the predicted answers (Second set)
plt.bar(ind + width, target2_observed, width,label='Observed') #Plotting the␣
 ↪bar graph of the observed answers (Second set)
plt.ylabel('Targets 0 or 1') #y label
plt.title('Closeness of predicted targets for 22 cases') #title
plt.xticks(ind + width / 2, ind) #tick mark indicator
plt.legend(loc='best') #legend (automatically put in best location)

plt.show() #display graphs
```

```
Starting …
[[0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1.]
 [1. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1.]
 [1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1.]
 [1. 1. 0. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 1.]
 [1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1. 1.]]
5
22
 Features :  [[0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1.
1.]
 [1. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1.]
 [1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1.]
 [1. 1. 0. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 1.]
 [1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1. 1.]]
 Targets :  [[0. 1. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1.]
 [1. 0. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0.]]
 Targets predicted :  [[0.99974552 0.99988233 0.99749198 0.98078918 0.99953068
```

```
0.99757348
  0.99961542 0.99924565 0.99983496 0.91088518 0.99835383 0.99907217
  0.99870431 0.99923959 0.99621789 0.99383098 0.998461   0.99922085
  0.99812055 0.97990719 0.99667477 0.99990526]
 [0.99953253 0.99954935 0.99591068 0.98030925 0.99850222 0.99760332
  0.99950385 0.99907038 0.999784   0.94481755 0.99732611 0.99889225
  0.9969958  0.99957247 0.99776471 0.988292   0.99803953 0.9988391
  0.99729983 0.96122579 0.99708802 0.99975973]]
0 21.68207917476477
1 21.66442147177232
2 21.644685013499593
3 21.62246341448813
4 21.597233560792183
5 21.568309736850857
6 21.534774122869976
7 21.49536812135138
8 21.448316124899858
9 21.391027262635838
10 21.319564451979215
11 21.22763998836021
12 21.104569197198632
13 20.930699448552
14 20.665946676258045
15 20.21655667159152
16 19.321640093649727
17 17.17097293463046
18 13.68018417526595
19 13.519778701641883
20 13.470551969176135
21 13.446892545263719
22 13.423704928576155
23 13.400974040098554
24 13.378442947869686
25 13.356144805498978
26 13.334051211169578
27 13.312152736717321
28 13.290434841955445
29 13.268884584144146
30 13.247488597358256
31 13.226233603663621
32 13.205106176667003
33 13.184092717818707
34 13.16317935900798
35 13.142351874900955
36 13.121595577210634
37 13.10089519610039
38 13.080234743147265
39 13.059597352656613
```

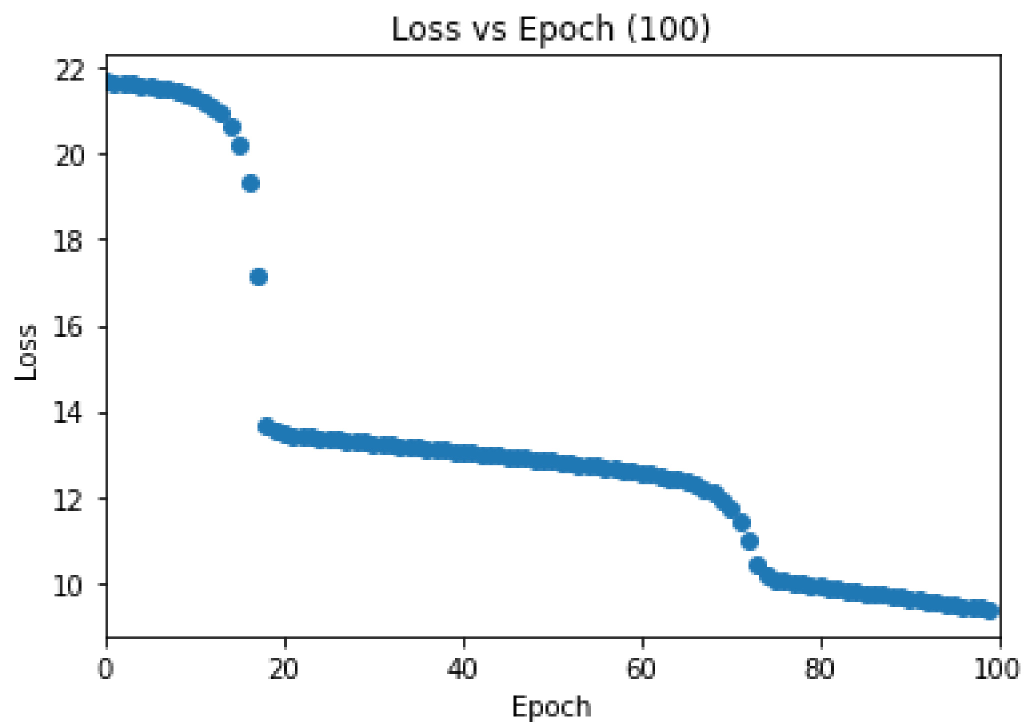
40 13.038965096198616
41 13.018318764298801
42 12.997637607522869
43 12.976899027139321
44 12.956078202794114
45 12.935147641004399
46 12.914076623427412
47 12.892830527320147
48 12.871369981699164
49 12.849649810454062
50 12.827617696615434
51 12.805212477966226
52 12.782361949940599
53 12.758980002243632
54 12.73496284298257
55 12.710183955819208
56 12.684487271470108
57 12.657677781307411
58 12.629508421297311
59 12.599661411354864
60 12.56772117542871
61 12.533134177064625
62 12.49514789760469
63 12.452715635823601
64 12.404343609079255
65 12.347837554670845
66 12.279868730906134
67 12.19520653256608
68 12.085328689025632
69 11.935913410847334
70 11.722748080845694
71 11.408323688756097
72 10.95937461419137
73 10.45291479946377
74 10.157049387404655
75 10.085060509528743
76 10.051161178611746
77 10.018944613779938
78 9.987144711106204
79 9.95570413018283
80 9.9246008033664
81 9.893815796778224
82 9.863331202063636
83 9.833129916352155
84 9.803195578001604
85 9.773512517518608
86 9.744065712089732
87 9.714840742652287

```
88 9.6858237532715
89 9.65700141266436
90 9.628360877725779
91 9.599889758922535
92 9.571576087429799
93 9.543408283893337
94 9.515375128708646
95 9.487465733715279
96 9.4596695152118
97 9.431976168202944
98 9.404375641796452
99 9.376858115672626
 Features :  [[0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1.
1.]
 [1. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1.]
 [1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1.]
 [1. 1. 0. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 1.]
 [1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1. 1.]]
 Targets :  [[0. 1. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1.]
 [1. 0. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0.]]
 Targets predicted :  [[0.26917312 0.47480888 0.38383005 0.36702548 0.36847297
0.51495017
  0.50149954 0.32170939 0.41312076 0.34995368 0.42156213 0.26369194
  0.46186874 0.35517612 0.40653479 0.44888257 0.61773094 0.38692549
  0.31206891 0.46131229 0.50265826 0.48393133]
 [0.55865654 0.48808607 0.63371977 0.63643014 0.53366288 0.6752366
  0.62461212 0.67735273 0.6462287  0.66861434 0.63973254 0.53953301
  0.59130755 0.69949288 0.74351365 0.59103464 0.6579599  0.64864999
  0.59302691 0.56007428 0.61609115 0.52254045]]
```



[2]:
```
plt.scatter(x=epoch_list1,y=lost1)
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.xlim(0,100)
plt.title('Loss vs Epoch (100)')
```

[2]:
```
Text(0.5, 1.0, 'Loss vs Epoch (100)')
```



EPOCH 500

[3]:
```
def sigmoid(x): #Creating a definition for the sigmoid function
    return 1.0/(1+np.exp(-1.0*x)) #This is the sigmoid function. We are using
 ↪this because we are looking at a logistic model of an NN (only having 2
 ↪solutions).

def feed_forward(features,w1,b1,w2,b2,w3,b3): #Creating a definition for
 ↪"feed_foward". Basically the features,weights and biases going through the
 ↪NN starting from the first 5 features going to the final 2.
    #Calculating Hidden Layer 1
    HL1 = np.matmul(w1,features) #multiplying our weights and features. Where
 ↪the weights are a [4 x 5] and the the features are a [5 x 22] creating a [4
 ↪x 22] new matrix.
    HL1_with_bias = np.add(HL1,b1) #Adding the HL1 matrix (weights * features)
 ↪which is a [4 x 22] to the biases which are also a [4 x 22].

    # Implement RELU activation (max(0,x))
    HL1_with_bias_and_activation = np.maximum(np.zeros((4,1)),HL1_with_bias)
 ↪#We need to find the RELU to squish the numbers down to keep the numbers
 ↪managable.
```

```
    HL2 = np.matmul(w2,HL1_with_bias_and_activation) #multiplying our weights␣
↪and features. Where the weights are a [3 x 4] and the the features are a [4␣
↪x 22] creating a [3 x 22] new matrix.
    HL2_with_bias = np.add(HL2,b2) #Adding the HL1 matrix (weights * features)␣
↪which is a [3 x 22] to the biases which are also a [3 x 22].

    # Implement RELU activation
    HL2_with_bias_and_activation = np.maximum(HL2_with_bias,np.zeros((3,1)))␣
↪#We need to find the RELU to squish the numbers down to keep the numbers␣
↪managable.
    targets_predicted = np.matmul(w3,HL2_with_bias_and_activation) #multiplying␣
↪our weights and features. Where the weights are a [2 x 3] and the the␣
↪features are a [3 x 22] creating a [2 x 22] new matrix.
    targets_predicted = np.add(targets_predicted,b3) #Adding the HL1 matrix␣
↪(weights * features) which is a [2 x 22] to the biases which are also a [2 x␣
↪22].

    # Use sigmoid for the output activation
    targets_predicted = sigmoid(targets_predicted) #Calling the sigmoid on the␣
↪targets predicted to get them to become "squished" in between 0 and 1.
    return targets_predicted #prints out the targets predicted.

def loss(features,w1,b1,w2,b2,w3,b3,targets_observed): #A method of evaluating␣
↪how well your algorithm models your dataset. Greater is bad and lower is␣
↪good.
    targets_predicted = feed_forward(features,w1,b1,w2,b2,w3,b3) #Getting the␣
↪targets predicted to put into the loss function
    return np.sum((targets_predicted-targets_observed)**2) #Gives the loss␣
↪function answer. The closer the sum is to 0, the closer the NN is to␣
↪achieving the bests SSR.
#======================================================
print('Starting ...')

## Set up training data
## Each row is a case
## Columns 0-4 are features
## Columns 5 & 6 are targets

#22 x 7 matrix
features_and_targets = np.array(
                                   [ [0, 0, 0, 0, 0, 0, 1],
                                     [0, 0, 0, 0, 1, 0, 1],
                                     [0, 0, 0, 1, 1, 0, 1],
                                     [0, 0, 1, 1, 1, 0, 1],
                                     [0, 1, 1, 1, 1, 0, 1],
                                     [1, 1, 1, 1, 0, 0, 1],
```

```
                                    [1, 1, 1, 0, 0, 0, 1],
                                    [1, 1, 0, 0, 0, 0, 1],
                                    [1, 0, 0, 0, 0, 0, 1],
                                    [1, 0, 0, 1, 0, 0, 1],
                                    [1, 0, 1, 1, 0, 0, 1],
                                    [1, 1, 0, 1, 0, 0, 1],
                                    [0, 1, 0, 1, 1, 0, 1],
                                    [0, 0, 1, 0, 1, 0, 1],
                                    [1, 0, 1, 1, 1, 1, 0],
                                    [1, 1, 0, 1, 1, 1, 0],
                                    [1, 0, 1, 0, 1, 1, 0],
                                    [1, 0, 0, 0, 1, 1, 0],
                                    [1, 1, 0, 0, 1, 1, 0],
                                    [1, 1, 1, 0, 1, 1, 0],
                                    [1, 1, 1, 1, 1, 1, 0],
                                    [1, 0, 0, 1, 1, 1, 0]  ]
                          , dtype=float)

# shuffle our cases (to create randomness)
np.random.shuffle(features_and_targets)

# Need to transpose to get them as 5 X N matrices
features = np.transpose(features_and_targets[:,0:5]) #Tranposes the matrix to 5␣
 ↪x 22. Picking up the 0-4 index.
print(features) #prints the features in the array

# Need to transpose to get the 2 x N matrices
targets_observed = np.transpose(features_and_targets[:,5:7]) #Transposing the␣
 ↪last 2 columns from N x 2 to a 2 x N. Picking up the 5 and 6 index
number_of_features,number_of_cases = features.shape #[5,22] (creating tuple␣
 ↪(unchangable))
print(number_of_features) #prints the number of features in the array
print(number_of_cases) #prints the number of cases in the array

#Set initial weights and biases

weights_1 = np.random.rand(4,5) #Setting the dimensions of the matrix for our␣
 ↪random weights (rows by columns)
biases_1 = np.random.rand(4,number_of_cases) #Setting the dimensions of the␣
 ↪matrix for our baises (rows by columns)

weights_2 = np.random.rand(3,4) #Setting the dimensions of the matrix for our␣
 ↪random weights (rows by columns)
biases_2 = np.random.rand(3,number_of_cases) #Setting the dimensions of the␣
 ↪matrix for our baises (rows by columns)
```

```python
weights_3 = np.random.rand(2,3) #Setting the dimensions of the matrix for our random weights (rows by columns)
biases_3 = np.random.rand(2,number_of_cases) #Setting the dimensions of the matrix for our baises (rows by columns)

Targets_Predicted = feed_forward(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3) #Targets_Predicted before Epoch (before training is initialized)

#Printing the results to the following below:
print(' Features : ',features)
print(' Targets : ', targets_observed)
print(' Targets predicted : ', Targets_Predicted)

#Learning rate gives the rate of speed where the gradient moves during gradient descent. Setting it too high would make your path instable, too low would make convergence slow. Put it to zero means your model isn't learning anything from the gradients.

learning_rate = 0.01 #The amount of change to the model during each iterations

# Find slope functions using autograd
d_by_w1 = grad(loss,1) #create the derivative/gradient --> stored d_by_w1, (equation(loss), index=1 (weights_1))
d_by_b1 = grad(loss,2) #create the derivative/gradient --> stored d_by_b1, (equation(loss), index=2 (biases_1))
d_by_w2 = grad(loss,3) #create the derivative/gradient --> stored d_by_w2, (equation(loss), index=3 (weights_2))
d_by_b2 = grad(loss,4) #create the derivative/gradient --> stored d_by_b2, (equation(loss), index=4 (biases_2))
d_by_w3 = grad(loss,5) #create the derivative/gradient --> stored d_by_w3, (equation(loss), index=5 (weights_3))
d_by_b3 = grad(loss,6) #create the derivative/gradient --> stored d_by_b3, (equation(loss), index=6 (biases_3))

epoch_list2 = [] #creating list for epoch iterations
lost2 = [] #creating list for loss fucntion results

#Setting the iteration (making higher helps making the loss function look better)
for epoch in range(500):

    #At each iteration update weights and biases by subtracting
    #learning_rate times slope
    #Changing the random weights and random biases updating each of them after each iteration according to range(X)
```

```
    weights_1 -=
↪learning_rate*d_by_w1(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    biases_1 -=
↪learning_rate*d_by_b1(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    weights_2 -=
↪learning_rate*d_by_w2(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    biases_2 -=
↪learning_rate*d_by_b2(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    weights_3 -=
↪learning_rate*d_by_w3(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    biases_3 -=
↪learning_rate*d_by_b3(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar

    #Print out the latest value of the loss
    #We would expect this to go down with each iteration
    #This shows how good of a test it was to get to the desired the results
    #The closer to 0, the better the NN is learning <- This is what the loss
↪function is doing in the equation.

↪print(epoch,loss(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,targets_

    epoch_list2.append(epoch) #epoch list adding the iterations
    lost2.
↪append(loss(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,targets_obser
↪#loss function list adding for each iteration

Targets_Predicted =
↪feed_forward(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3)
↪#Targets_Predicted after Epoch (After training is complete)

#Printing the results to the following below:
print(' Features : ',features)
print(' Targets : ', targets_observed)
print(' Targets predicted : ', Targets_Predicted)

N = 22
target1_predicted = Targets_Predicted[0,:] #The target we are trying to achieve
target2_predicted = Targets_Predicted[1,:] #The target we are trying to achieve
target1_observed = targets_observed[0,:] #The target we want to achieve (what
↪the targets should be)
target2_observed = targets_observed[1,:] #The target we want to achieve (what
↪the targets should be)

ind = np.arange(N) #Return evenly spaced values within a given interval (N=22)
width = 0.35 #creating width to add distance between the 2 different bar graphs
plt.subplot(2,1,1) #creating first graph
```

```
plt.bar(ind, target1_predicted, width, label='Predicted') #Plotting the bar␣
 ↪graph of the predicted answers (First set)
plt.bar(ind + width, target1_observed, width,label='Observed') #Plotting the␣
 ↪bar graph of the observed answers (First set)
plt.ylabel('Targets 0 or 1') #y label
plt.title('Closeness of predicted targets for 22 cases') #title
plt.xticks(ind + width / 2, ind) #tick mark indicator
plt.legend(loc='best') #legend (automatically put in best location)
plt.subplot(2,1,2) #creating first graph
plt.bar(ind, target2_predicted, width, label='Predicted') #Plotting the bar␣
 ↪graph of the predicted answers (Second set)
plt.bar(ind + width, target2_observed, width,label='Observed') #Plotting the␣
 ↪bar graph of the observed answers (Second set)
plt.ylabel('Targets 0 or 1') #y label
plt.title('Closeness of predicted targets for 22 cases') #title
plt.xticks(ind + width / 2, ind) #tick mark indicator
plt.legend(loc='best') #legend (automatically put in best location)

plt.show() #display graphs
```

```
Starting …
[[1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1.]
 [0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0.]
 [0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1.]
 [0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1.]
 [1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1.]]
5
22
 Features :  [[1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1.
1.]
 [0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0.]
 [0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1.]
 [0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1.]
 [1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1.]]
 Targets :  [[1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1.]
 [0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0.]]
 Targets predicted :  [[0.99971564 0.99984954 0.99715073 0.99970074 0.99987584
0.9996942
  0.99912618 0.99696804 0.99258852 0.99781382 0.99834036 0.99839137
  0.99997358 0.99984343 0.9995745  0.99875532 0.99970555 0.99962038
  0.96678248 0.99982654 0.99998842 0.99996984]
 [0.99999461 0.99999822 0.99968697 0.99999067 0.99999152 0.99999628
  0.99998298 0.999247   0.99812369 0.9999094  0.99995738 0.99992678
  0.99999955 0.99999503 0.99999462 0.99992936 0.99999036 0.99998968
  0.98086104 0.99999483 0.99999996 0.99999917]]
0 21.88803881808903
1 21.885384947752698
```

```
2 21.882597346286598
3 21.87966520479288
4 21.876576489991958
5 21.87331776382429
6 21.869873970015814
7 21.86622818030975
8 21.86236129116416
9 21.85825165923373
10 21.8538746606951
11 21.8492021551564
12 21.844201829117186
13 21.8388363861482
14 21.83306254032561
15 21.826829754784168
16 21.820078646779223
17 21.812738951701554
18 21.804726897002464
19 21.79594177663244
20 21.786261427366338
21 21.77553617412372
22 21.763580605447135
23 21.75016221767121
24 21.73498544900599
25 21.71766877367062
26 21.697711085279252
27 21.674441079542486
28 21.646938779973848
29 21.613909725902825
30 21.573475269545433
31 21.522806748150533
32 21.457451859125165
33 21.370011052010884
34 21.24732271793102
35 21.063864469405395
36 20.76431118241779
37 20.210520527404412
38 19.00412726656551
39 16.242164447298677
40 14.440262309328011
41 14.400511432106494
42 14.367189441463577
43 14.334453473169331
44 14.302335281705423
45 14.27080427507647
46 14.239837819621485
47 14.209413523153401
48 14.179509953923217
49 14.150106511419454
```

50	14.121183394541932
51	14.09272156056997
52	14.0647026877173
53	14.037109139454476
54	14.009923930711084
55	13.983130695809344
56	13.9567136580285
57	13.930657600700505
58	13.904947839745196
59	13.879570197559215
60	13.854510978178617
61	13.82975694364046
62	13.805295291473534
63	13.781113633252888
64	13.757199974157105
65	13.733542693471163
66	13.710130525981263
67	13.686952544211547
68	13.663998141455705
69	13.641257015559374
70	13.618719153412076
71	13.596374816109984
72	13.574214524753156
73	13.552229046843232
74	13.53040938324966
75	13.508746755714508
76	13.487232594867809
77	13.465858528727168
78	13.444616371656993
79	13.423498113764376
80	13.402495910710003
81	13.381602073914083
82	13.360809061138452
83	13.34010946742749
84	13.31949601639149
85	13.298961551817552
86	13.27849902959403
87	13.25810150993571
88	13.23776214989799
89	13.217474196169286
90	13.197230978131966
91	13.177025901183013
92	13.156852440306604
93	13.13670413389173
94	13.116574577788903
95	13.096457419600885
96	13.07634635320328
97	13.05623511349182

98 13.036117471353851
99 13.015987228862759
100 12.995838214694636
101 12.975664279767674
102 12.95545929310548
103 12.935217137926669
104 12.91493170796378
105 12.894596904015717
106 12.874206630738858
107 12.853754793682823
108 12.833235296578167
109 12.812642038884038
110 12.791968913605
111 12.771209805387368
112 12.750358588906293
113 12.729409127556103
114 12.708355272457437
115 12.687190861795802
116 12.66590972050743
117 12.64450566032925
118 12.62297248023109
119 12.601303967249219
120 12.579493897741466
121 12.557536039085216
122 12.53542415184053
123 12.513151992401724
124 12.49071331616149
125 12.468101881212533
126 12.445311452612371
127 12.422335807237518
128 12.399168739253692
129 12.37580406622895
130 12.3522356359167
131 12.328457333735345
132 12.304463090970929
133 12.280246893728302
134 12.25580279265542
135 12.231124913463749
136 12.206207468266076
137 12.181044767750556
138 12.155631234207172
139 12.12996141541928
140 12.104029999429079
141 12.077831830181314
142 12.0513619240442
143 12.024615487200693
144 11.997587933896664
145 11.970274905524956

146	11.942672290516308
147	11.914776244999059
148	11.886583214179826
149	11.85808995438693
150	11.82929355570699
151	11.800191465133341
152	11.770781510132283
153	11.74106192252027
154	11.711031362531594
155	11.680688942942545
156	11.650034253104117
157	11.619067382721806
158	11.587788945207521
159	11.55620010041607
160	11.524302576566496
161	11.49209869113812
162	11.459591370521526
163	11.426784168197564
164	11.393681281211938
165	11.360287564710287
166	11.326608544298665
167	11.292650425997648
168	11.25842010356498
169	11.223925162972163
170	11.189173883834851
171	11.15417523761542
172	11.118938882438842
173	11.083475154389818
174	11.047795055189994
175	11.011910236189015
176	10.975832978641046
177	10.939576170279988
178	10.903153278249972
179	10.86657831849348
180	10.829865821745615
181	10.793030796329791
182	10.75608868799563
183	10.719055337083878
184	10.681946933343895
185	10.644779968766512
186	10.607571188827073
187	10.570337542559862
188	10.533096131904914
189	10.495864160780489
190	10.45865888433931
191	10.421497558862919
192	10.38439739273694
193	10.347375498929933

194 10.310448849370895
195 10.273634231585175
196 10.236948207907153
197 10.200407077540566
198 10.164026841685635
199 10.127823171896992
200 10.091811381779184
201 10.056006402068803
202 10.020422759095075
203 9.985074556555691
204 9.949975460492887
205 9.915138687307373
206 9.88057699460583
207 9.846302674642107
208 9.812327550083527
209 9.77866297181259
210 9.745319818460727
211 9.71230849736501
212 9.679638946640589
213 9.647320638070658
214 9.615362580531675
215 9.58377332369373
216 9.552560961763094
217 9.521733137066162
218 9.491297043309066
219 9.461259428385246
220 9.431626596642289
221 9.402404410558901
222 9.373598291821596
223 9.345368789129616
224 9.318139012869153
225 9.2913022628504
226 9.26486465885241
227 9.238830234790907
228 9.213202579917713
229 9.18798487086481
230 9.163179866312088
231 9.13878990159002
232 9.114816884391994
233 9.091262291768428
234 9.068127168525942
235 9.04541212713277
236 9.023117349208505
237 9.001242588651523
238 8.979787176432017
239 8.958750027052952
240 8.938129646656064
241 8.917924142726072

242	8.898131235323818
243	8.878748269758864
244	8.859772230594183
245	8.841199756860613
246	8.823027158346623
247	8.805250432819756
248	8.78786528403019
249	8.770867140343466
250	8.75425117384922
251	8.738012319794683
252	8.722145296196302
253	8.70664462348905
254	8.691504644081112
255	8.676719541691103
256	8.662283360355248
257	8.648190023003277
258	8.634433349513248
259	8.621007074167428
260	8.607904862442917
261	8.59512032708231
262	8.582647043400634
263	8.570478563795351
264	8.558608431435887
265	8.54703019311837
266	8.535737411279365
267	8.524723675169893
268	8.513982611197559
269	8.503507892450445
270	8.493293247421242
271	8.483332467954432
272	8.473619416442672
273	8.464148032301372
274	8.45491233775262
275	8.44590644295117
276	8.437124550486336
277	8.42856095929432
278	8.420210068015605
279	8.41206637783212
280	8.404124494818404
281	8.396379131840359
282	8.388825110034357
283	8.381457359898498
284	8.374270922026584
285	8.367260947514252
286	8.360422698065253
287	8.35375154582467
288	8.347242972964283
289	8.340892571044094

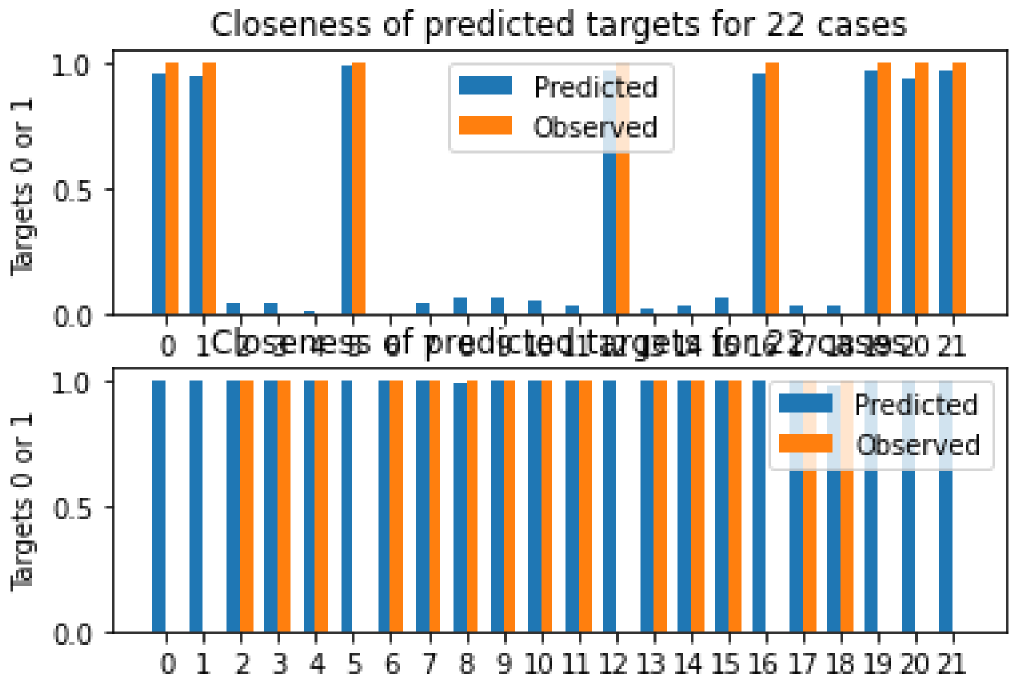
290 8.33469604017241
291 8.328649187985672
292 8.322747928467772
293 8.316988280627278
294 8.311366367049756
295 8.305878412341057
296 8.300520741476392
297 8.29528977806871
298 8.290182042568972
299 8.285194150409827
300 8.2803228101032
301 8.2755648213015
302 8.270917072831189
303 8.2663765407067
304 8.26194028613199
305 8.257605453496234
306 8.253369268369568
307 8.249229035504197
308 8.245182136845555
309 8.24122602955781
310 8.237358244067366
311 8.233576382127731
312 8.229878114908642
313 8.22626118111192
314 8.222723385116284
315 8.219262595153008
316 8.21587674151394
317 8.212563814793281
318 8.20932186416417
319 8.206148995690974
320 8.203043370677943
321 8.200003204054743
322 8.197026762799224
323 8.194112364397611
324 8.191258375342201
325 8.188463209666525
326 8.18572532751784
327 8.18304323376672
328 8.180415476653398
329 8.177840646470573
330 8.17531737428212
331 8.172844330677286
332 8.170420224559818
333 8.168043801971411
334 8.165713844948886
335 8.163429170414458
336 8.161188629098403
337 8.158991104493447

338	8.156835511840212
339	8.154720797142923
340	8.152645936214784
341	8.15060993375215
342	8.148611822436909
343	8.14665066206623
344	8.144725538709073
345	8.142835563888603
346	8.140979873789934
347	8.139157628492393
348	8.137368011225652
349	8.135610227649043
350	8.133883505153362
351	8.132187092184473
352	8.130520257588145
353	8.128882289975332
354	8.127272497107432
355	8.12569020530077
356	8.124134758849788
357	8.122605519468301
358	8.121101865748276
359	8.119623192635553
360	8.118168910921966
361	8.116738446753315
362	8.115331241152697
363	8.113946749558648
364	8.11258444137765
365	8.111243799550452
366	8.109924320131825
367	8.108625511883218
368	8.10734689587791
369	8.106088005118226
370	8.104848384164384
371	8.103627588774568
372	8.102425185555854
373	8.101240751625571
374	8.10007387428275
375	8.098924150689262
376	8.097791187560373
377	8.096674600864233
378	8.095574015530127
379	8.094489065165027
380	8.093419391778223
381	8.092374187239725
382	8.091347104376927
383	8.090334158129906
384	8.089335046997743
385	8.088349478130366

386	8.087377162222015
387	8.086417812745717
388	8.08547114642774
389	8.084536883899556
390	8.083614750216686
391	8.082704475193896
392	8.081805793584
393	8.080918445141661
394	8.080042174608371
395	8.079176731645566
396	8.078321870734706
397	8.077477351056919
398	8.076642936360512
399	8.075818394821738
400	8.075003498902241
401	8.07419802520535
402	8.07340175433258
403	8.072614470741168
404	8.071835962603101
405	8.071066021665915
406	8.070304443115315
407	8.069551025439734
408	8.068805570296673
409	8.0680678823808
410	8.067337769293703
411	8.0666150414151
412	8.065899511775447
413	8.065190995929733
414	8.064489311832325
415	8.063794279712699
416	8.063105721951892
417	8.06242346295949
418	8.061747329050977
419	8.061077148325287
420	8.060412750542334
421	8.059753967000358
422	8.059100630412876
423	8.058452574785022
424	8.057809635289097
425	8.05717164813907
426	8.05653845046382
427	8.055909880178893
428	8.055285775856495
429	8.054665976593478
430	8.054050321877034
431	8.053438651447836
432	8.05283080516029
433	8.052226622839571

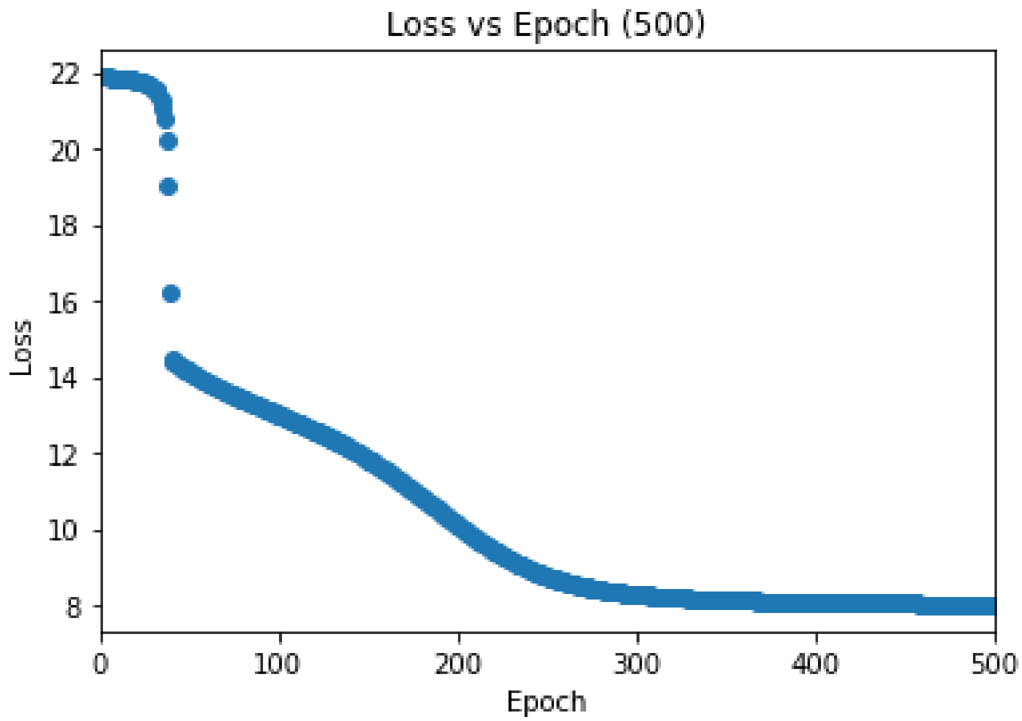
434	8.05162594413517
435	8.051028608370501
436	8.050434454388258
437	8.049843320391094
438	8.049255043777185
439	8.048669460970228
440	8.0480864072434
441	8.047505716536717
442	8.046927221267284
443	8.046350752131787
444	8.045776137900628
445	8.045203205202984
446	8.044631778302065
447	8.044061678859746
448	8.043492725689763
449	8.042924734498474
450	8.042357517612242
451	8.041790883690314
452	8.041224637422037
453	8.040658579207127
454	8.040092504817615
455	8.039526205039948
456	8.038959465295633
457	8.038392065238623
458	8.037823778327505
459	8.037254371370373
460	8.03668360404009
461	8.036111228357374
462	8.035536988138936
463	8.03496061840769
464	8.034381844761613
465	8.033800382697674
466	8.033215936886737
467	8.03262820039506
468	8.032036853847439
469	8.031441564526668
470	8.030841985403272
471	8.03023775408896
472	8.029628491706461
473	8.02901380166762
474	8.02839326835071
475	8.02776645566695
476	8.027132905504985
477	8.026492136040886
478	8.025843639899648
479	8.025186882152573
480	8.024521298133001
481	8.023846291050686

```
482 8.023161229382652
483 8.02246544401553
484 8.02175822511122
485 8.02103881866389
486 8.020306422712215
487 8.019560183165787
488 8.018799189198868
489 8.01802246815834
490 8.01722897992484
491 8.01641761065737
492 8.015587165841369
493 8.014736362547993
494 8.013863820798491
495 8.012968053910635
496 8.012047457684773
497 8.011100298263777
498 8.010124698473682
499 8.009118622419036
 Features :  [[1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1.
1.]
 [0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0.]
 [0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1.]
 [0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1.]
 [1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1.]]
 Targets :  [[1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1.]
 [0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0.]]
 Targets predicted :  [[0.95397716 0.94232141 0.04547998 0.04365808 0.01521292
0.98717156
  0.00620891 0.04397697 0.07064264 0.06299677 0.05774408 0.02870492
  0.96911034 0.02569967 0.03133324 0.0652091  0.95814137 0.02997042
  0.03658198 0.96752578 0.93976454 0.96483633]
 [0.99795514 0.99824149 0.99501073 0.99954984 0.9994411  0.99759924
  0.99961974 0.99193798 0.9900403  0.99758663 0.99869867 0.99892031
  0.99931879 0.99958783 0.99968673 0.99832409 0.99499548 0.99925003
  0.97017541 0.99614899 0.99986195 0.99903435]]
```



[4]:
```
plt.scatter(x=epoch_list2,y=lost2)
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.xlim(0,500)
plt.title('Loss vs Epoch (500)')
```

[4]: Text(0.5, 1.0, 'Loss vs Epoch (500)')



EPOCH 1000

```
[5]: def sigmoid(x): #Creating a definition for the sigmoid function
         return 1.0/(1+np.exp(-1.0*x)) #This is the sigmoid function. We are using␣
      ↪this because we are looking at a logistic model of an NN (only having 2␣
      ↪solutions).

     def feed_forward(features,w1,b1,w2,b2,w3,b3): #Creating a definition for␣
      ↪"feed_foward". Basically the features,weights and biases going through the␣
      ↪NN starting from the first 5 features going to the final 2.
         #Calculating Hidden Layer 1
         HL1 = np.matmul(w1,features) #multiplying our weights and features. Where␣
      ↪the weights are a [4 x 5] and the the features are a [5 x 22] creating a [4␣
      ↪x 22] new matrix.
         HL1_with_bias = np.add(HL1,b1) #Adding the HL1 matrix (weights * features)␣
      ↪which is a [4 x 22] to the biases which are also a [4 x 22].

         # Implement RELU activation (max(0,x))
         HL1_with_bias_and_activation = np.maximum(np.zeros((4,1)),HL1_with_bias)␣
      ↪#We need to find the RELU to squish the numbers down to keep the numbers␣
      ↪managable.
```

```python
    HL2 = np.matmul(w2,HL1_with_bias_and_activation) #multiplying our weights␣
↪and features. Where the weights are a [3 x 4] and the the features are a [4␣
↪x 22] creating a [3 x 22] new matrix.
    HL2_with_bias = np.add(HL2,b2) #Adding the HL1 matrix (weights * features)␣
↪which is a [3 x 22] to the biases which are also a [3 x 22].

    # Implement RELU activation
    HL2_with_bias_and_activation = np.maximum(HL2_with_bias,np.zeros((3,1)))␣
↪#We need to find the RELU to squish the numbers down to keep the numbers␣
↪managable.
    targets_predicted = np.matmul(w3,HL2_with_bias_and_activation) #multiplying␣
↪our weights and features. Where the weights are a [2 x 3] and the the␣
↪features are a [3 x 22] creating a [2 x 22] new matrix.
    targets_predicted = np.add(targets_predicted,b3) #Adding the HL1 matrix␣
↪(weights * features) which is a [2 x 22] to the biases which are also a [2 x␣
↪22].

    # Use sigmoid for the output activation
    targets_predicted = sigmoid(targets_predicted) #Calling the sigmoid on the␣
↪targets predicted to get them to become "squished" in between 0 and 1.
    return targets_predicted #prints out the targets predicted.

def loss(features,w1,b1,w2,b2,w3,b3,targets_observed): #A method of evaluating␣
↪how well your algorithm models your dataset. Greater is bad and lower is␣
↪good.
    targets_predicted = feed_forward(features,w1,b1,w2,b2,w3,b3) #Getting the␣
↪targets predicted to put into the loss function
    return np.sum((targets_predicted-targets_observed)**2) #Gives the loss␣
↪function answer. The closer the sum is to 0, the closer the NN is to␣
↪achieving the bests SSR.
#=======================================================
print('Starting ...')

## Set up training data
## Each row is a case
## Columns 0-4 are features
## Columns 5 & 6 are targets

#22 x 7 matrix
features_and_targets = np.array(
                                    [ [0, 0, 0, 0, 0, 0, 1],
                                      [0, 0, 0, 0, 1, 0, 1],
                                      [0, 0, 0, 1, 1, 0, 1],
                                      [0, 0, 1, 1, 1, 0, 1],
                                      [0, 1, 1, 1, 1, 0, 1],
                                      [1, 1, 1, 1, 0, 0, 1],
```

```
                                    [1, 1, 1, 0, 0, 0, 1],
                                    [1, 1, 0, 0, 0, 0, 1],
                                    [1, 0, 0, 0, 0, 0, 1],
                                    [1, 0, 0, 1, 0, 0, 1],
                                    [1, 0, 1, 1, 0, 0, 1],
                                    [1, 1, 0, 1, 0, 0, 1],
                                    [0, 1, 0, 1, 1, 0, 1],
                                    [0, 0, 1, 0, 1, 0, 1],
                                    [1, 0, 1, 1, 1, 1, 0],
                                    [1, 1, 0, 1, 1, 1, 0],
                                    [1, 0, 1, 0, 1, 1, 0],
                                    [1, 0, 0, 0, 1, 1, 0],
                                    [1, 1, 0, 0, 1, 1, 0],
                                    [1, 1, 1, 0, 1, 1, 0],
                                    [1, 1, 1, 1, 1, 1, 0],
                                    [1, 0, 0, 1, 1, 1, 0]  ]
                          , dtype=float)

# shuffle our cases (to create randomness)
np.random.shuffle(features_and_targets)

# Need to transpose to get them as 5 X N matrices
features = np.transpose(features_and_targets[:,0:5]) #Tranposes the matrix to 5 x 22. Picking up the 0-4 index.
print(features) #prints the features in the array

# Need to transpose to get the 2 x N matrices
targets_observed = np.transpose(features_and_targets[:,5:7]) #Transposing the last 2 columns from N x 2 to a 2 x N. Picking up the 5 and 6 index
number_of_features,number_of_cases = features.shape #[5,22] (creating tuple (unchangable))
print(number_of_features) #prints the number of features in the array
print(number_of_cases) #prints the number of cases in the array

#Set initial weights and biases

weights_1 = np.random.rand(4,5) #Setting the dimensions of the matrix for our random weights (rows by columns)
biases_1 = np.random.rand(4,number_of_cases) #Setting the dimensions of the matrix for our baises (rows by columns)

weights_2 = np.random.rand(3,4) #Setting the dimensions of the matrix for our random weights (rows by columns)
biases_2 = np.random.rand(3,number_of_cases) #Setting the dimensions of the matrix for our baises (rows by columns)
```

```
weights_3 = np.random.rand(2,3) #Setting the dimensions of the matrix for our␣
 ↪random weights (rows by columns)
biases_3 = np.random.rand(2,number_of_cases) #Setting the dimensions of the␣
 ↪matrix for our baises (rows by columns)

Targets_Predicted =␣
 ↪feed_forward(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3)␣
 ↪#Targets_Predicted before Epoch (before training is initialized)

#Printing the results to the following below:
print(' Features : ',features)
print(' Targets : ', targets_observed)
print(' Targets predicted : ', Targets_Predicted)

#Learning rate gives the rate of speed where the gradient moves during gradient␣
 ↪descent. Setting it too high would make your path instable, too low would␣
 ↪make convergence slow. Put it to zero means your model isn't learning␣
 ↪anything from the gradients.

learning_rate = 0.01 #The amount of change to the model during each iterations

# Find slope functions using autograd
d_by_w1 = grad(loss,1) #create the derivative/gradient --> stored d_by_w1,␣
 ↪(equation(loss), index=1 (weights_1))
d_by_b1 = grad(loss,2) #create the derivative/gradient --> stored d_by_b1,␣
 ↪(equation(loss), index=2 (biases_1))
d_by_w2 = grad(loss,3) #create the derivative/gradient --> stored d_by_w2,␣
 ↪(equation(loss), index=3 (weights_2))
d_by_b2 = grad(loss,4) #create the derivative/gradient --> stored d_by_b2,␣
 ↪(equation(loss), index=4 (biases_2))
d_by_w3 = grad(loss,5) #create the derivative/gradient --> stored d_by_w3,␣
 ↪(equation(loss), index=5 (weights_3))
d_by_b3 = grad(loss,6) #create the derivative/gradient --> stored d_by_b3,␣
 ↪(equation(loss), index=6 (biases_3))

epoch_list3 = [] #creating list for epoch iterations
lost3 = [] #creating list for loss fucntion results

#Setting the iteration (making higher helps making the loss function look␣
 ↪better)
for epoch in range(1000):

    #At each iteration update weights and biases by subtracting
    #learning_rate times slope
    #Changing the random weights and random biases updating each of them after␣
 ↪each iteration according to range(X)
```

```
    weights_1 -=␣
↪learning_rate*d_by_w1(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    biases_1 -=␣
↪learning_rate*d_by_b1(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    weights_2 -=␣
↪learning_rate*d_by_w2(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    biases_2 -=␣
↪learning_rate*d_by_b2(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    weights_3 -=␣
↪learning_rate*d_by_w3(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar
    biases_3 -=␣
↪learning_rate*d_by_b3(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,tar

    #Print out the latest value of the loss
    #We would expect this to go down with each iteration
    #This shows how good of a test it was to get to the desired the results
    #The closer to 0, the better the NN is learning <- This is what the loss␣
↪function is doing in the equation.
  ␣
↪print(epoch,loss(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,targets_

    epoch_list3.append(epoch) #epoch list adding the iterations
    lost3.
↪append(loss(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3,targets_obser
↪#loss function list adding for each iteration

Targets_Predicted =␣
↪feed_forward(features,weights_1,biases_1,weights_2,biases_2,weights_3,biases_3)␣
↪#Targets_Predicted after Epoch (After training is complete)

#Printing the results to the following below:
print(' Features : ',features)
print(' Targets : ', targets_observed)
print(' Targets predicted : ', Targets_Predicted)

N = 22
target1_predicted = Targets_Predicted[0,:] #The target we are trying to achieve
target2_predicted = Targets_Predicted[1,:] #The target we are trying to achieve
target1_observed = targets_observed[0,:] #The target we want to achieve (what␣
↪the targets should be)
target2_observed = targets_observed[1,:] #The target we want to achieve (what␣
↪the targets should be)

ind = np.arange(N) #Return evenly spaced values within a given interval (N=22)
width = 0.35 #creating width to add distance between the 2 different bar graphs
plt.subplot(2,1,1) #creating first graph
```

```
plt.bar(ind, target1_predicted, width, label='Predicted') #Plotting the bar␣
↪graph of the predicted answers (First set)
plt.bar(ind + width, target1_observed, width,label='Observed') #Plotting the␣
↪bar graph of the observed answers (First set)
plt.ylabel('Targets 0 or 1') #y label
plt.title('Closeness of predicted targets for 22 cases') #title
plt.xticks(ind + width / 2, ind) #tick mark indicator
plt.legend(loc='best') #legend (automatically put in best location)
plt.subplot(2,1,2) #creating first graph
plt.bar(ind, target2_predicted, width, label='Predicted') #Plotting the bar␣
↪graph of the predicted answers (Second set)
plt.bar(ind + width, target2_observed, width,label='Observed') #Plotting the␣
↪bar graph of the observed answers (Second set)
plt.ylabel('Targets 0 or 1') #y label
plt.title('Closeness of predicted targets for 22 cases') #title
plt.xticks(ind + width / 2, ind) #tick mark indicator
plt.legend(loc='best') #legend (automatically put in best location)

plt.show() #display graphs
```

```
Starting …
[[0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 0.]
 [0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1.]
 [0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0.]
 [1. 1. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 0. 1. 1. 1.]
 [1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 0. 1.]]
5
22
 Features :  [[0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1.
0.]
 [0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1.]
 [0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0.]
 [1. 1. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 0. 1. 1. 1.]
 [1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 0. 1.]]
 Targets :  [[0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0.]
 [1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1.]]
 Targets predicted :  [[0.98363339 0.99553079 0.99934737 0.95257727 0.99826303
0.99871201
  0.98919614 0.99883202 0.99019502 0.99429307 0.99930612 0.99882063
  0.98690107 0.99763554 0.91307719 0.99684356 0.99773331 0.9872429
  0.95902614 0.99951054 0.99423249 0.99719311]
 [0.98294772 0.99131575 0.99972275 0.97727298 0.99827293 0.99725721
  0.98934977 0.99933964 0.99375122 0.9937413  0.99947837 0.99917346
  0.98143573 0.99738043 0.89345232 0.99566856 0.99757773 0.98383728
  0.9305845  0.9994398  0.9963027  0.99625893]]
0 21.427371018966134
1 21.3474094713935
```

2 21.23944055301944
3 21.085622384920093
4 20.849554186748573
5 20.445855189275537
6 19.632560816547052
7 17.584083618323618
8 13.971302786030119
9 13.898346346726067
10 13.854165176504155
11 13.828339020977296
12 13.801772625452505
13 13.775879938524328
14 13.74926387334698
15 13.7222930281838
16 13.694676534891187
17 13.666406338760885
18 13.637326742309508
19 13.607310670283926
20 13.576169411471536
21 13.543674699032449
22 13.509523104533184
23 13.473313363115718
24 13.434497390747548
25 13.392308787266195
26 13.345642055051163
27 13.292846744125653
28 13.231356756072053
29 13.15698740320604
30 13.06251650454815
31 12.934614772764126
32 12.746718008635485
33 12.441747708599728
34 11.895722213774802
35 10.944049266957688
36 10.121822219055709
37 10.038709993044993
38 9.977475706395708
39 9.9147218215284
40 9.851744306131895
41 9.786939306395862
42 9.72064942688359
43 9.652630665187061
44 9.582877732213559
45 9.511263305979222
46 9.437729113281355
47 9.36219500860357
48 9.284590239822666
49 9.204840072129429

50 9.122872295146935
51 9.038614905110272
52 8.951997877563326
53 8.862953162157748
54 8.771415600020898
55 8.677323517376992
56 8.580619581647262
57 8.481251646215933
58 8.379173713936996
59 8.274346954509156
60 8.16674080089883
61 8.056334102232537
62 7.943116328943379
63 7.8270888111315635
64 7.7082659920336525
65 7.586676670889057
66 7.462365205859085
67 7.3353926418266795
68 7.205837723551595
69 7.073797750317664
70 6.939389225347856
71 6.802748251790676
72 6.664030627866606
73 6.523411597103519
74 6.381085216044521
75 6.237263311588089
76 6.092174013336031
77 5.946059862723004
78 5.799175519727154
79 5.658527851840609
80 5.523576500902635
81 5.388274709400487
82 5.252954832738647
83 5.1177884220953045
84 4.982958275072221
85 4.848640205429011
86 4.715009359256866
87 4.5822355763763944
88 4.450484059784387
89 4.3199138926903
90 4.190677750582578
91 4.062921360451568
92 3.93678329971714
93 3.8123948697722634
94 3.689880108909555
95 3.5693558746632403
96 3.4509319696917826
97 3.334711269063846

98 3.220789815394682
99 3.109256852758329
100 3.000194779332662
101 2.893679008840442
102 2.7897777418703766
103 2.6885516588967358
104 2.5900535563175473
105 2.4943279542232704
106 2.401410709244293
107 2.3113286673516233
108 2.224099389876106
109 2.13973098156697
110 2.0582220428077362
111 1.9795617599177575
112 1.903730138653046
113 1.8306983774232892
114 1.760429369091121
115 1.6928783140571768
116 1.6279934229807698
117 1.5657166850380841
118 1.505984676978471
119 1.4487293891563147
120 1.393879046850719
121 1.3413589081597004
122 1.2910920232019054
123 1.2429999429527074
124 1.1970033695230347
125 1.1530227428705087
126 1.1109787616950293
127 1.070792838557358
128 1.0323874910603463
129 0.9956866722739791
130 0.96061604451572
131 0.9271032011768465
132 0.895077841577649
133 0.8644719039013138
134 0.835219661154429
135 0.8072577848798886
136 0.7805253810458955
137 0.7549640021850723
138 0.7305176394851718
139 0.7071326981560847
140 0.6847579590298415
141 0.6633445290001136
142 0.6428457825807589
143 0.6232172965624091
144 0.6044167794731529
145 0.5864039973039322

146 0.5691406967404732
147 0.5525905269498554
148 0.5367189607994234
149 0.521493216236634
150 0.5068821784286731
151 0.4928563231482377
152 0.4793876417949223
153 0.4664495683584293
154 0.45401690855871196
155 0.4420657713376788
156 0.43057350282596396
157 0.4195186228652232
158 0.4088807641304901
159 0.3986406138672777
160 0.3887798582335861
161 0.3792811292169865
162 0.3701279540808705
163 0.3613047072812241
164 0.35279656478538646
165 0.34458946071677776
166 0.3366700462441325
167 0.3290256506300495
168 0.32164424435136757
169 0.3145144042027534
170 0.30762528029475406
171 0.3009665648582252
172 0.29452846276833394
173 0.2883016637031569
174 0.28227731585409405
175 0.27644700110783876
176 0.270802711622377
177 0.2653368277223929
178 0.26004209704245224
179 0.2549116148493912
180 0.24993880547842146
181 0.2451174048205105
182 0.2404414438016335
183 0.2359052327974514
184 0.23150334692986832
185 0.2272306121947275
186 0.22308209237261922
187 0.21905307667738813
188 0.21513906809943992
189 0.21133577240333984
190 0.20763908774150028
191 0.20404509484793254
192 0.20055004777812074
193 0.1971503651630463

194 0.1938426219472677
195 0.19062354158272637
196 0.1874899886516304
197 0.18443896189334935
198 0.18146758761174542
199 0.17857311344078056
200 0.17575290244756273
201 0.17300442755324827
202 0.17032526625339225
203 0.16771309562044867
204 0.16516568757216263
205 0.16268090439057498
206 0.16025669447728086
207 0.1578910883314469
208 0.1555821947379039
209 0.15332819715339194
210 0.1511273502797534
211 0.14897797681353675
212 0.1468784643621062
213 0.1448272625169437
214 0.14282288007538294
215 0.14086388240253778
216 0.13894888892567236
217 0.13707657075372268
218 0.1352456484151067
219 0.13345488970736452
220 0.13170310765254725
221 0.1299891585526323
222 0.1283119401395712
223 0.1266703898148947
224 0.1250634829740887
225 0.1234902314112331
226 0.12194968179965697
227 0.12044091424460088
228 0.11896304090411028
229 0.1175152046746
230 0.11609657793772478
231 0.11470636136538877
232 0.113343782779898
233 0.1120080960664321
234 0.11069858013516776
235 0.10941453793053203
236 0.1081552954852065
237 0.1069202010166335
238 0.10570862406389754
239 0.10451995466297039
240 0.10335360255842305
241 0.1022089964498029

242 0.10108558327097784
243 0.0999828275008366
244 0.0989002105038224
245 0.09783722989885732
246 0.09679339895529007
247 0.09576824601457698
248 0.09476131393646511
249 0.0937721595685205
250 0.09280035323789605
251 0.09184547826429802
252 0.0909071304931586
253 0.08998491784807598
254 0.0890784599016302
255 0.08818738746372852
256 0.08731134218667728
257 0.08644997618621744
258 0.08560295167779948
259 0.08476994062740968
260 0.08395062441629293
261 0.08314469351895198
262 0.08235184719383125
263 0.0815717931861237
264 0.08080424744216617
265 0.08004893383491553
266 0.07930558390002028
267 0.07857393658203005
268 0.07785373799030088
269 0.07714474116418296
270 0.0764467058470901
271 0.07575939826907375
272 0.07508259093754173
273 0.07441606243577428
274 0.07375959722891513
275 0.07311298547711946
276 0.07247602285556431
277 0.07184851038103596
278 0.07123025424482296
279 0.07062106565165582
280 0.07002076066444768
281 0.06942916005459683
282 0.06884608915763063
283 0.06827137773397048
284 0.06770485983461688
285 0.0671463736715548
286 0.06659576149269399
287 0.0660528694611641
288 0.06551754753879172
289 0.06498964937359834

290	0.0644690321911593
291	0.06395555668967613
292	0.0634490869386164
293	0.06294949028078634
294	0.06245663723770186
295	0.06197040141813443
296	0.06149065942971011
297	0.06101729079344585
298	0.060550177861113844
299	0.060089205735326584
300	0.05963426219224195
301	0.05918523760679019
302	0.05874202488033041
303	0.05830451937064635
304	0.05787261882419613
305	0.05744622331053313
306	0.05702523515881957
307	0.05660955889635681
308	0.056199101189060276
309	0.05579377078380745
310	0.05539347845259411
311	0.05499813693843351
312	0.05460766090293638
313	0.05422196687551331
314	0.053840973204140996
315	0.05346460000764083
316	0.053092769129412125
317	0.05272540409257478
318	0.05236243005646917
319	0.05200377377446705
320	0.05164936355305127
321	0.05129912921211616
322	0.05095300204645195
323	0.05061091478836986
324	0.05027280157143133
325	0.049938597895242666
326	0.0496082405912813
327	0.049281667789717834
328	0.04895881888720147
329	0.04863963451557716
330	0.0483240565115032
331	0.04801202788694056
332	0.04770349280048497
333	0.047398396529514275
334	0.04709668544312497
335	0.046798306975832836
336	0.04650320960201178
337	0.046211342811049015

338	0.04592265708319216
339	0.04563710386606784
340	0.04535463555184844
341	0.045075205455049346
342	0.04479876779093434
343	0.04452527765451173
344	0.0442546910001019
345	0.04398696462145949
346	0.04372205613243204
347	0.043459923948139335
348	0.04320052726665702
349	0.042943826051189474
350	0.04268978101271682
351	0.042438353593101476
352	0.042189505948640856
353	0.041943200934052904
354	0.041699402086879805
355	0.04145807361230056
356	0.041219180368336834
357	0.04098268785144287
358	0.04074856218246677
359	0.04051677009297298
360	0.04028727891191477
361	0.04006005655264721
362	0.03983507150026966
363	0.03961229279928963
364	0.03939169004159742
365	0.039173233354742994
366	0.03895689339050711
367	0.03874264131375724
368	0.03853044879158107
369	0.03832028798268845
370	0.03811213152707595
371	0.03790595253594549
372	0.03770172458186926
373	0.03749942168919633
374	0.03729901832469122
375	0.03710048938840075
376	0.036903810204740824
377	0.036708956513798456
378	0.03651590446284177
379	0.036324630598033925
380	0.036135111856343875
381	0.035947325557649235
382	0.03576124939702637
383	0.03557686143722223
384	0.03539414010130253
385	0.03521306416547322

386	0.035033612752068295
387	0.034855765322701585
388	0.03467950167157703
389	0.03450480191895312
390	0.03433164650475839
391	0.03416001618235327
392	0.033989892012434235
393	0.03382125535707724
394	0.033654087873916404
395	0.033488371510454494
396	0.03332408849850155
397	0.03316122134873895
398	0.03299975284540525
399	0.032839666041100266
400	0.032680944251705674
401	0.03252357105141777
402	0.03236753026788985
403	0.03221280597748293
404	0.03205938250061936
405	0.031907244397238924
406	0.031756376462354526
407	0.03160676372170405
408	0.0314583914274964
409	0.03131124505425035
410	0.031165310294722075
411	0.03102057305592131
412	0.03087701945521189
413	0.030734635816495822
414	0.030593408666479038
415	0.0304533247310157
416	0.030314370931530298
417	0.030176534381515092
418	0.0300398023831008
419	0.02990416242369999
420	0.02976960217271956
421	0.029636109478342453
422	0.029503672364375774
423	0.02937227902716444
424	0.02924191783256805
425	0.029112577313000498
426	0.02898424616452999
427	0.028856913244038062
428	0.028730567566437267
429	0.028605198301944707
430	0.028480794773410895
431	0.028357346453702997
432	0.028234842963140352
433	0.028113274066981532

434 0.027992629672962
435 0.027872899828880235
436 0.0277540747202329
437 0.027636144667895997
438 0.027519100125852302
439 0.027402931678963648
440 0.02728763004078685
441 0.027173186051432516
442 0.027059590675465973
443 0.026946834999848394
444 0.026834910231918984
445 0.026723807697415083
446 0.026613518838531726
447 0.026504035212017555
448 0.026395348487307978
449 0.026287450444693855
450 0.026180332973525018
451 0.026073988070448414
452 0.02596840783767923
453 0.025863584481305033
454 0.025759510309622115
455 0.025656177731502583
456 0.025553579254792895
457 0.025451707484741494
458 0.025350555122456687
459 0.02525011496339261
460 0.025150379895863575
461 0.025051342899585785
462 0.024952997044246344
463 0.024855335488097974
464 0.024758351476580162
465 0.0246620383409652
466 0.0245663894970291
467 0.024471398443746663
468 0.024377058762010163
469 0.024283364113371585
470 0.024190308238806926
471 0.024097884957503705
472 0.02400608816566913
473 0.023914911835360625
474 0.02382435001333678
475 0.02373439681992862
476 0.023645046447931386
477 0.023556293161515623
478 0.02346813129515769
479 0.023380555252588902
480 0.02329355950576354
481 0.023207138593844615

482 0.023121287122207363
483 0.02303599976146064
484 0.022951271246484436
485 0.022867096375485143
486 0.022783470009066213
487 0.022700387069315632
488 0.02261784253890851
489 0.022535831460225576
490 0.02245434893448636
491 0.022373390120897563
492 0.02229295023581551
493 0.02221302455192336
494 0.022133608397421685
495 0.022054697155233362
496 0.021976286262221242
497 0.021898371208419542
498 0.021820947536277703
499 0.021744010839917303
500 0.021667556764400957
501 0.02159158100501373
502 0.02151607930655625
503 0.021441047462649895
504 0.021366481315052817
505 0.021292376752988135
506 0.02121872971248224
507 0.02114553617571465
508 0.02107279217037828
509 0.0210004937690501
510 0.020928637088571882
511 0.020857218289441568
512 0.020786233575213867
513 0.02071567919191092
514 0.020645551427442406
515 0.020575846611034725
516 0.02050656111266991
517 0.020437691342532743
518 0.020369233750467446
519 0.020301184825442676
520 0.020233541095024767
521 0.020166299124859927
522 0.0200994555181642
523 0.020033006915221457
524 0.019966949992889525
525 0.019901281464114084
526 0.019835998077449773
527 0.019771096616589167
528 0.01970657389989914
529 0.019642426779964237

530	0.019578652143136876
531	0.019515246909095327
532	0.019452208030407595
533	0.019389532492102735
534	0.019327217311248278
535	0.019265259536534313
536	0.019203656247864232
537	0.019142404555950968
538	0.019081501601920414
539	0.01902094455691963
540	0.018960730621732366
541	0.018900857026399175
542	0.018841321029844055
543	0.018782119919506297
544	0.018723251010978097
545	0.0186647116476474747
546	0.018606499200346585
547	0.018548611067005217
548	0.018491044672309515
549	0.018433797467365703
550	0.018376866929368896
551	0.018320250561276643
552	0.01826394589148762
553	0.018207950473524435
554	0.018152261885721846
555	0.01809687773091884
556	0.018041795636155978
557	0.01798701325237623
558	0.01793252825413098
559	0.01787833833928993
560	0.017824441228755185
561	0.01777083466617954
562	0.017717516417689004
563	0.0176644842716089
564	0.017611736038194194
565	0.017559269549363914
566	0.017507082658438724
567	0.0174551732398829
568	0.017403539189049477
569	0.01735217842192939
570	0.01730108887490388
571	0.017250268504500704
572	0.017199715287153482
573	0.017149427218964477
574	0.017099402315471208
575	0.017049638611415438
576	0.017000134160516104
577	0.016950887035245325

578 0.01690189532660716
579 0.016853157143919877
580 0.0168046706146009
581 0.016756433883955063
582 0.01670844511496544
583 0.016660702488087428
584 0.01661320420104518
585 0.016565948468631415
586 0.01651893352250951
587 0.016472157611018598
588 0.016425618998980936
589 0.016379315967512437
590 0.016333246813835352
591 0.016287409851093643
592 0.01624180340817081
593 0.016196425829510246
594 0.01615127547493805
595 0.016106350719487943
596 0.01606164995322899
597 0.016017171581095134
598 0.015972914022717433
599 0.01592887571225826
600 0.015885055098247884
601 0.015841450643422943
602 0.01579806082456761
603 0.015754884132356022
604 0.01571191907119773
605 0.015669164159084488
606 0.015626617927439242
607 0.015584278920967417
608 0.015542145697509611
609 0.01550021682789657
610 0.015458490895806087
611 0.015416966497621525
612 0.015375642242292379
613 0.01533451675119646
614 0.01529358865800416
615 0.0152528566085441
616 0.015212319260670851
617 0.015171975284134028
618 0.015131823360449388
619 0.015091862182771304
620 0.015052090455767112
621 0.015012506895492855
622 0.014973110229270671
623 0.0149338991955679
624 0.014894872543877467
625 0.014856029034599903

626 0.01481736743892692
627 0.014778886538726355
628 0.0147405851264286
629 0.014702462004914443
630 0.014664515987404375
631 0.01462674589734919
632 0.014589150568321971
633 0.014551728843911527
634 0.01451447957761704
635 0.014477401632744094
636 0.014440493882301814
637 0.014403755208901583
638 0.014367184504656894
639 0.014330780671084155
640 0.014294542619005113
641 0.014258469268450376
642 0.014222559548563794
643 0.014186812397508592
644 0.014151226762374146
645 0.01411580159908401
646 0.014080535872305318
647 0.014045428555359002
648 0.014010478630131114
649 0.013975685086985415
650 0.013941046924676874
651 0.01390656315026624
652 0.01387223277903555
653 0.013838054834404897
654 0.013804028347849905
655 0.013770152358820417
656 0.013736425914659997
657 0.01370284807052651
658 0.01366941788931356
659 0.013636134441572884
660 0.013602996805437773
661 0.013570004066547035
662 0.013537155317970598
663 0.013504449660135042
664 0.013471886200750662
665 0.013439464054739336
666 0.013407182344162896
667 0.013375040198152667
668 0.01334303675283969
669 0.013311171151285819
670 0.013279442543415438
671 0.013247850085948355
672 0.013216392942333063
673 0.013185070282680884

674 0.013153881283701199
675 0.01312282512863689
676 0.013091901007200877
677 0.013061108115513474
678 0.013030445656039995
679 0.01299991283752969
680 0.012969508874954802
681 0.01293923298945071
682 0.012909084408256586
683 0.012879062364656869
684 0.012849166097923095
685 0.01281939485325678
686 0.012789747881732752
687 0.01276022444024291
688 0.012730823791441161
689 0.012701545203688403
690 0.01267238795099855
691 0.012643351312984835
692 0.012614434574806958
693 0.012585637027118835
694 0.012556957966016613
695 0.012528396692987604
696 0.012499952514859681
697 0.012471624743751237
698 0.01244341269702153
699 0.012415315697221952
700 0.012387333072047448
701 0.012359464154288676
702 0.012331708281784757
703 0.012304064797376219
704 0.012276533048858881
705 0.012249112388937977
706 0.012221802175182886
707 0.0121946017699821
708 0.01216751054049922
709 0.01214052785862896
710 0.012113653100953643
711 0.0120868856487005
712 0.012060224887699195
713 0.012033670208339746
714 0.012007221005531053
715 0.011980876678659932
716 0.011954636631550301
717 0.011928500272423103
718 0.01190246701385647
719 0.011876536272746362
720 0.0118507074702678
721 0.011824980031836107

722	0.011799353387069061
723	0.011773826969749015
724	0.011748400217785678
725	0.011723072573179219
726	0.01169784348198379
727	0.011672712394271239
728	0.011647678764095618
729	0.011622742049457645
730	0.011597901712269756
731	0.011573157218321438
732	0.011548508037245013
733	0.01152395364248174
734	0.011499493511248213
735	0.011475127124503182
736	0.011450853966914715
737	0.011426673526827611
738	0.011402585296231298
739	0.011378588770727956
740	0.011354683449501007
741	0.011330868835283904
742	0.011307144434329248
743	0.011283509756378339
744	0.0112599643146307
745	0.011236507625714487
746	0.011213139209656536
747	0.011189858589853247
748	0.011166665293041535
749	0.011143558849269977
750	0.011120538791870575
751	0.011097604657430466
752	0.011074755985764118
753	0.011051992319885736
754	0.011029313205981961
755	0.011006718193384876
756	0.010984206834545254
757	0.010961778685006016
758	0.010939433303376067
759	0.010917170251304403
760	0.010894989093454294
761	0.010872889397477915
762	0.010850870733991186
763	0.010828932676548742
764	0.010807074801619443
765	0.010785296688561723
766	0.010763597919599471
767	0.01074197807979824
768	0.010720436757041166
769	0.010698973542005935

770 0.010677588028141168
771 0.0106562798116435
772 0.010635048491434915
773 0.010613893669139994
774 0.010592814949063667
775 0.010571811938169084
776 0.010550884246055619
777 0.010530031484937214
778 0.010509253269621006
779 0.01048854921748575
780 0.01046791894846109
781 0.01044736208500639
782 0.010426878252090294
783 0.010406467077170078
784 0.010386128190171573
785 0.010365861223468862
786 0.010345665811864574
787 0.010325541592570082
788 0.010305488205186161
789 0.010285505291683398
790 0.010265592496383322
791 0.010245749465939431
792 0.010225975849318171
793 0.010206271297780642
794 0.010186635464863996
795 0.010167068006363408
796 0.010147568580313767
797 0.010128136846971923
798 0.010108772468798932
799 0.010089475110442536
800 0.01007024443871967
801 0.010051080122599396
802 0.010031981833185662
803 0.01001294924370053
804 0.009993982029467387
805 0.00997507986789431
806 0.009956242438457689
807 0.00993746942268593
808 0.009918760504143332
809 0.009900115368414052
810 0.0098815337030864
811 0.009863015197736994
812 0.009844559543915343
813 0.009826166435128377
814 0.009807835566825306
815 0.009789566636382318
816 0.009771359343087815
817 0.009753213388127442

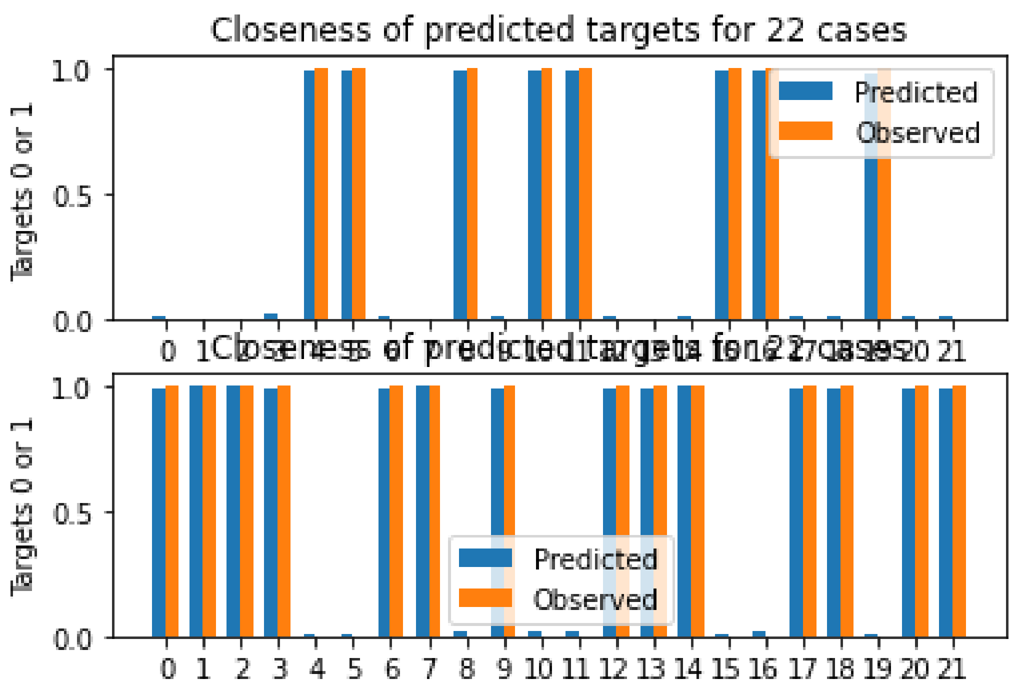
818 0.009735128474569373
819 0.0097171043073499
820 0.009699140593258828
821 0.009681237040925265
822 0.009663393360803465
823 0.009645609265158717
824 0.009627884468053542
825 0.00961021868533378
826 0.009592611634615073
827 0.00957506303526915
828 0.009557572608410626
829 0.009540140076883535
830 0.009522765165248226
831 0.009505447599768308
832 0.00948818710839777
833 0.009470983420768108
834 0.009453836268175473
835 0.00943674538356842
836 0.009419710501535146
837 0.00940273135829119
838 0.009385807691667301
839 0.009368939241097112
840 0.009352125747605194
841 0.009335366953795186
842 0.00931866260383782
843 0.009302012443459346
844 0.009285416219929789
845 0.009268873682051579
846 0.009252384580148005
847 0.009235948666051992
848 0.009219565693094815
849 0.009203235416095005
850 0.009186957591347515
851 0.00917073197661241
852 0.009154558331104467
853 0.009138436415482195
854 0.0091223659918372
855 0.009106346823683775
856 0.009090378675948403
857 0.009074461314959287
858 0.009058594508436223
859 0.009042778025480386
860 0.009027011636564154
861 0.00901129511352123
862 0.008995628229536643
863 0.008980010759136906
864 0.008964442478180311
865 0.008948923163847275

866	0.008933452594630685
867	0.008918030550326507
868	0.008902656812024306
869	0.008887331162097827
870	0.008872053384195942
871	0.008856823263233294
872	0.008841640585381234
873	0.008826505138058903
874	0.008811416709924136
875	0.008796375090864724
876	0.00878138007198952
877	0.008766431445619753
878	0.008751529005280423
879	0.008736672545691701
880	0.008721861862760381
881	0.008707096753571519
882	0.008692377016380051
883	0.008677702450602441
884	0.008663072856808682
885	0.008648488036713815
886	0.008633947793170112
887	0.00861945193015896
888	0.008605000252782883
889	0.008590592567257842
890	0.008576228680905084
891	0.008561908402143792
892	0.008547631540483007
893	0.008533397906514358
894	0.008519207311904288
895	0.008505059569386623
896	0.008490954492755061
897	0.008476891896855979
898	0.008462871597580985
899	0.008448893411859578
900	0.008434957157652216
901	0.008421062653942983
902	0.008407209720732532
903	0.00839339817903117
904	0.008379627850851735
905	0.008365898559202908
906	0.00835221012808217
907	0.00833856238246911
908	0.008324955148318715
909	0.008311388252554558
910	0.008297861523062347
911	0.0082843747886832
912	0.00827092787920722
913	0.008257520625367021

914 0.008244152858831228
915 0.008230824412198265
916 0.008217535118989852
917 0.00820428481364498
918 0.008191073331513409
919 0.008177900508849811
920 0.008164766182807431
921 0.008151670191432146
922 0.008138612373656406
923 0.00812559256929327
924 0.008112610619030498
925 0.008099666364424664
926 0.008086759647895386
927 0.00807389031271944
928 0.008061058203025167
929 0.008048263163786656
930 0.008035505040818186
931 0.00802278368076859
932 0.008010098931115752
933 0.007997450640160955
934 0.007984838657023626
935 0.007972262831635761
936 0.007959723014736624
937 0.007947219057867305
938 0.007934750813365568
939 0.007922318134360444
940 0.007909920874767221
941 0.007897558889281986
942 0.007885232033376786
943 0.007872940163294293
944 0.007860683136042958
945 0.007848460809391824
946 0.007836273041865661
947 0.007824119692739946
948 0.007812000622036117
949 0.007799915690516461
950 0.007787864759679554
951 0.007775847691755353
952 0.0077638643497003855
953 0.007751914597193145
954 0.007739998298629388
955 0.0077281153191174285
956 0.007716265524473693
957 0.007704448781217924
958 0.007692664956568844
959 0.007680913918439545
960 0.007669195535433086
961 0.007657509676838012

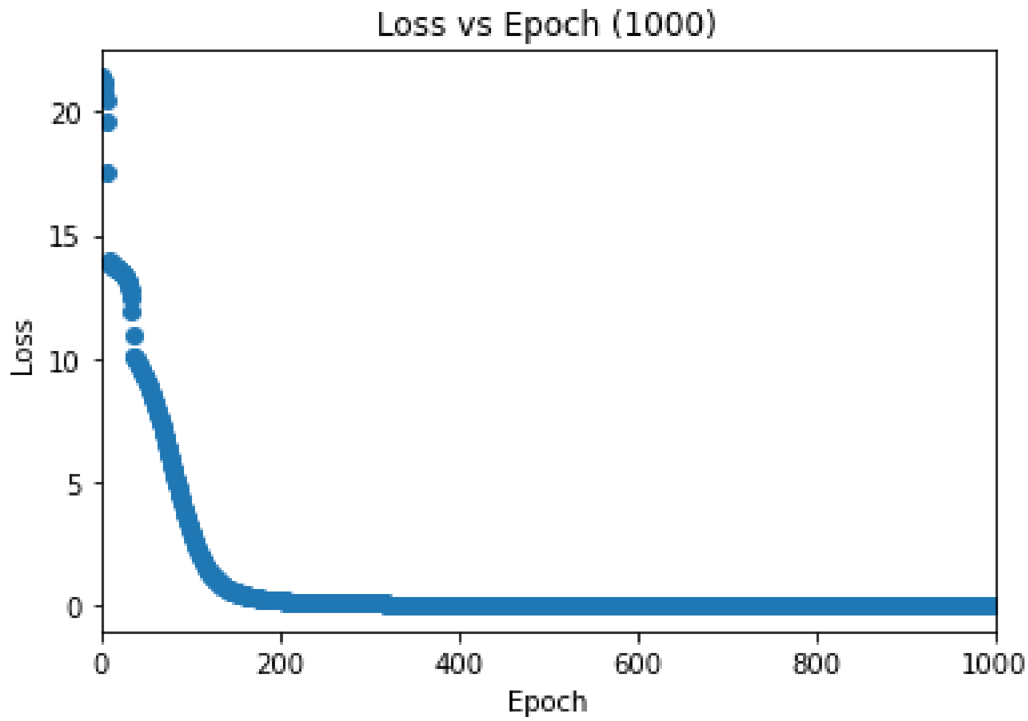
```
962 0.007645856212623963
963 0.00763423501343736
964 0.007622645950597027
965 0.0076110888960898805
966 0.007599563722566754
967 0.007588070303338003
968 0.0075766085123694605
969 0.007565178224278185
970 0.0075537793143283976
971 0.007542411658427209
972 0.007531075133120787
973 0.007519769615590076
974 0.0075084949836469895
975 0.007497251115730224
976 0.007486037890901416
977 0.007474855188841235
978 0.0074637028898454656
979 0.007452580874821113
980 0.007441489025282539
981 0.007430427223347808
982 0.007419395351734643
983 0.007408393293756922
984 0.007397420933320841
985 0.007386478154921195
986 0.007375564843637772
987 0.007364680885131625
988 0.007353826165641599
989 0.007343000571980567
990 0.007332203991531968
991 0.007321436312246295
992 0.007310697422637483
993 0.007299987211779513
994 0.007289305569302929
995 0.007278652385391356
996 0.007268027550778187
997 0.007257430956743062
998 0.007246862495108664
999 0.007236322058237272
 Features :  [[0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1.
0.]
 [0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1.]
 [0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0.]
 [1. 1. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 0. 1. 1. 1.]
 [1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 0. 1.]]
 Targets :  [[0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0.]
 [1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1.]]
 Targets predicted :  [[1.21138444e-02 6.28387974e-03 2.36457555e-04
2.28900877e-02
```

```
 9.92877773e-01 9.86762856e-01 9.54032646e-03 1.20569227e-03
 9.86564714e-01 1.36486824e-02 9.85924798e-01 9.91010673e-01
 1.69296566e-02 6.94858781e-03 1.06918635e-02 9.87097884e-01
 9.88519534e-01 8.04919728e-03 1.64420890e-02 9.82528031e-01
 1.34057238e-02 1.33780349e-02]
[9.82223875e-01 9.93716798e-01 9.99670259e-01 9.84999999e-01
 6.85775265e-03 1.16753981e-02 9.85568336e-01 9.98400300e-01
 2.10342920e-02 9.85004188e-01 1.51119780e-02 1.44702433e-02
 9.81214052e-01 9.89702164e-01 9.94275506e-01 9.25890113e-03
 1.70796335e-02 9.88290285e-01 9.83574417e-01 1.13674080e-02
 9.84372298e-01 9.90725146e-01]]
```




```
[6]: plt.scatter(x=epoch_list3,y=lost3)
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.xlim(0,1000)
plt.title('Loss vs Epoch (1000)')
```

```
[6]: Text(0.5, 1.0, 'Loss vs Epoch (1000)')
```


Loss vs Epoch (1000)
Loss
20
15
10
5
0
0
200
400
600
800
1000
Epoch