



In [89]:

```

1 import numpy as np
2 import matplotlib.pyplot as pyp
3 x = np.load("assignment8_X.npy")
4 y = np.load("assignment8_Y.npy")
5 x = x.T
6 y = y.T
7
8 alpha = 0.01
9 n = x.shape[0] # n = 10
10 t = x.shape[1] # t = 25
11 #w1 = np.ones((m,n))*0.01
12 #w2 = np.ones((n,m))*0.01
13
14 print(n,t)
15
16 def NN1_forward_pass(x,y,w1,w2,w3):
17     #print("before forward pass, x")
18     #print(x)
19     #print("w2")
20     #print(w2)
21     #print("w1")
22     #print(w1)
23     #print("y")
24     #print(y)
25     fwx = w3.dot(w2.dot(w1.dot(x)))
26     #print("fwx")
27     #print(fwx)
28     loss = fwx - y
29     #print("Loss")
30     #print(loss)
31     return loss
32
33 def NN1_backprop(x,w1,w2,w3,n,m):
34     dfdw3 = np.zeros((n,n,m)) # 10x10x25
35     dfdw2 = np.zeros((n,m,m)) # 10x25x25
36     dfdw1 = np.zeros((n,m,n)) # 10x25x10
37     dfdh1 = w3.dot(w2) # 10x25
38     dfdh2 = w3 # 10x25
39     dh1dw1 = np.zeros((m,m,n)) # 25x25x10
40     dh2dw2 = np.zeros((m,m,m)) # 25x25x25
41     h1 = w1.dot(x) # 25x1
42     h2 = w2.dot(h1) # 25x1
43     #print("In BP, h1", x.shape)
44     for i in range(n):
45         dfdw3[i][i] = h2
46     for i in range(m):
47         dh1dw1[i][i] = x
48         dh2dw2[i][i] = h1
49     for i in range(n):
50         for j in range(m):
51             dfdw2[i] += dh2dw2[j]*dfdh2[i][j]
52             dfdw1[i] += dh1dw1[j]*dfdh1[i][j]
53     return dfdw1,dfdw2,dfdw3
54
55 def VecXTen(vec,tensor):
56     result = np.zeros((tensor.shape[1],tensor.shape[2]))

```

```
57     for i in range(len(vec)):
58         result += vec[i]*tensor[i]
59     return result
```

10 25

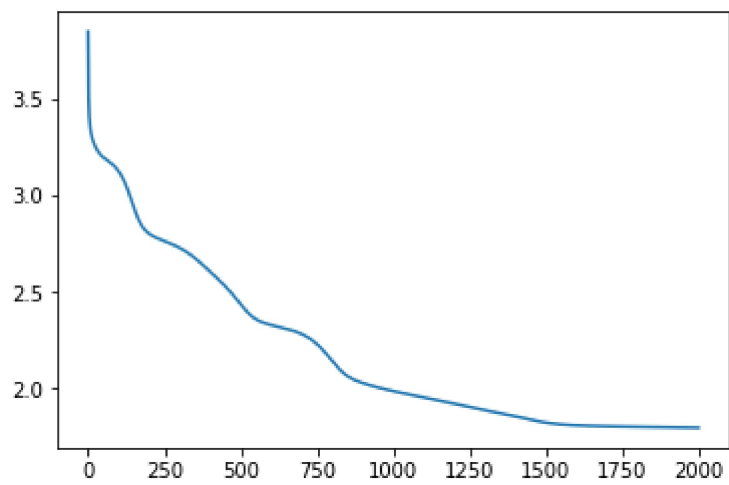
In [91]:

```

1  # NN1 with m = 10
2  m = 10
3  w1 = np.random.rand(m,n)*0.1
4  w2 = np.random.rand(m,m)*0.1
5  w3 = np.random.rand(n,m)*0.1
6  loss_list = []
7  for steps in range(2000):
8      Gradient_w1 = np.zeros(w1.shape)
9      Gradient_w2 = np.zeros(w2.shape)
10     Gradient_w3 = np.zeros(w3.shape)
11     total_loss = 0
12     for col in range(t):
13         loss = NN1_forward_pass(x[:,col],y[:,col],w1,w2,w3)
14         total_loss += np.linalg.norm(loss)**2
15         dfdw1,dfdw2,dfdw3 = NN1_backprop(x[:,col],w1,w2,w3,n,m)
16         Gradient_w1 += VecXTen(loss,dfdw1)
17         Gradient_w2 += VecXTen(loss,dfdw2)
18         Gradient_w3 += VecXTen(loss,dfdw3)
19     Gradient_w1 = Gradient_w1 * 2 / t
20     Gradient_w2 = Gradient_w2 * 2 / t
21     Gradient_w3 = Gradient_w3 * 2 / t
22     w1 = w1 - alpha * Gradient_w1
23     w2 = w2 - alpha * Gradient_w2
24     w3 = w3 - alpha * Gradient_w3
25     #print("at iteration ", steps, " loss is ", total_loss/t)
26     loss_list.append(total_loss/t)
27 print("Final Loss: ", total_loss/t)
28 #print("final W2 * W1: ")
29 #print(w2.dot(w1))
30 pyp.plot(loss_list[1:])
31 pyp.show()
32
33     #print("GradientW1: ")
34     #print(Gradient_w1)
35     #print("loss:")
36     #print(loss)
37     #print("dfdw1:")
38     #print(dfdw1)
39     #print("dfdw2:")
40     #print(dfdw2)
41

```

Final Loss: 1.7962366226647475



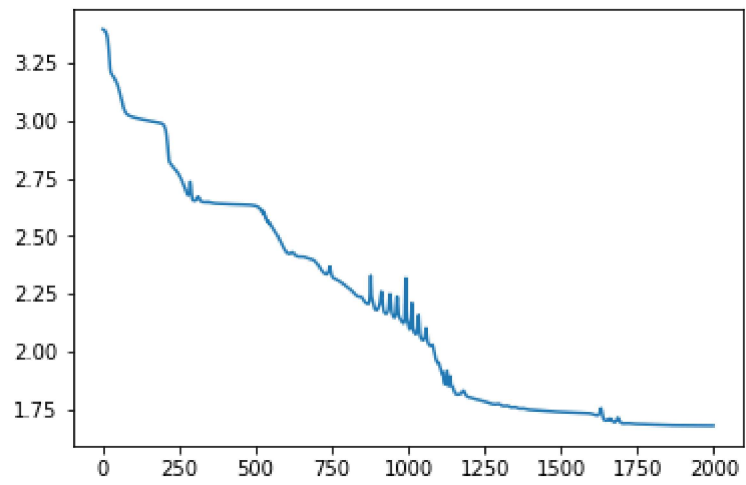
In [92]:

```

1  # NN1 with m = 25
2  m = 25
3  alpha = 0.05
4  w1 = np.random.rand(m,n)*0.01
5  w2 = np.random.rand(m,m)*0.01
6  w3 = np.random.rand(n,m)*0.01
7  loss_list = []
8  for steps in range(2000):
9      Gradient_w1 = np.zeros(w1.shape)
10     Gradient_w2 = np.zeros(w2.shape)
11     Gradient_w3 = np.zeros(w3.shape)
12     total_loss = 0
13     for col in range(t):
14         loss = NN1_forward_pass(x[:,col],y[:,col],w1,w2,w3)
15         total_loss += np.linalg.norm(loss)**2
16         dfdw1,dfdw2,dfdw3 = NN1_backprop(x[:,col],w1,w2,w3,n,m)
17         Gradient_w1 += VecXTen(loss,dfdw1)
18         Gradient_w2 += VecXTen(loss,dfdw2)
19         Gradient_w3 += VecXTen(loss,dfdw3)
20     Gradient_w1 = Gradient_w1 * 2 / t
21     Gradient_w2 = Gradient_w2 * 2 / t
22     Gradient_w3 = Gradient_w3 * 2 / t
23     w1 = w1 - alpha * Gradient_w1
24     w2 = w2 - alpha * Gradient_w2
25     w3 = w3 - alpha * Gradient_w3
26     #print("at iteration ", steps, " loss is ", total_loss/t)
27     loss_list.append(total_loss/t)
28 print("Final Loss: ", total_loss/t)
29 #print("final W2 * W1: ")
30 #print(w2.dot(w1))
31 pyp.plot(loss_list[1:])
32 pyp.show()
33
34     #print("GradientW1: ")
35     #print(Gradient_w1)
36     #print("loss:")
37     #print(loss)
38     #print("dfdw1:")
39     #print(dfdw1)
40     #print("dfdw2:")
41     #print(dfdw2)
42

```

Final Loss: 1.6788260271653126



In [95]:

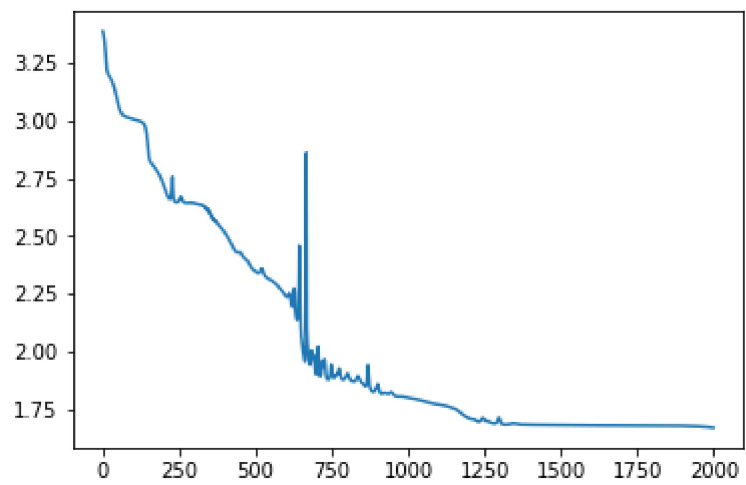
```

1  # NN1 with m = 50
2  m = 50
3  alpha = 0.05
4  w1 = np.random.rand(m,n)*0.01
5  w2 = np.random.rand(m,m)*0.01
6  w3 = np.random.rand(n,m)*0.01
7  loss_list = []
8  for steps in range(2000):
9      Gradient_w1 = np.zeros(w1.shape)
10     Gradient_w2 = np.zeros(w2.shape)
11     Gradient_w3 = np.zeros(w3.shape)
12     total_loss = 0
13     for col in range(t):
14         loss = NN1_forward_pass(x[:,col],y[:,col],w1,w2,w3)
15         total_loss += np.linalg.norm(loss)**2
16         dfdw1,dfdw2,dfdw3 = NN1_backprop(x[:,col],w1,w2,w3,n,m)
17         Gradient_w1 += VecXTen(loss,dfdw1)
18         Gradient_w2 += VecXTen(loss,dfdw2)
19         Gradient_w3 += VecXTen(loss,dfdw3)
20     Gradient_w1 = Gradient_w1 * 2 / t
21     Gradient_w2 = Gradient_w2 * 2 / t
22     Gradient_w3 = Gradient_w3 * 2 / t
23     w1 = w1 - alpha * Gradient_w1
24     w2 = w2 - alpha * Gradient_w2
25     w3 = w3 - alpha * Gradient_w3
26     #print("at iteration ", steps, " loss is ", total_loss/t)
27     loss_list.append(total_loss/t)
28 print("Final Loss: ", total_loss/t)
29 #print("final W2 * W1: ")
30 #print(w2.dot(w1))
31 pyp.plot(loss_list[1:])
32 pyp.show()
33
34     #print("GradientW1: ")
35     #print(Gradient_w1)
36     #print("loss:")
37     #print(loss)
38     #print("dfdw1:")
39     #print(dfdw1)
40     #print("dfdw2:")
41     #print(dfdw2)

```

Final Loss: 1.668421258126552





```

In [111]: 1 x = np.load("assignment8_X.npy")
2 y = np.load("assignment8_Y.npy")
3 x = x.T
4 y = y.T
5
6 alpha = 0.01
7 n = x.shape[0] # n = 10
8 t = x.shape[1] # t = 25
9 def vec_sigmod(x):
10     for i in range(len(x)):
11         x[i] = 1/(1+np.exp(-x[i]))
12     return x
13 def sigmod(x):
14     return 1/(1+np.exp(-x))
15 def Sigmod_forward_pass(x,y,w1,w2,w3):
16     #print("before forward pass, x")
17     #print(x)
18     #print("w2")
19     #print(w2)
20     #print("w1")
21     #print(w1)
22     #print("y")
23     #print(y)
24     fwx = w3.dot(vec_sigmod(w2.dot(vec_sigmod(w1.dot(x)))))
25     #print("fwx")
26     #print(fwx)
27     loss = fwx - y
28     #print("loss")
29     #print(loss)
30     return loss
31
32 def Sigmod_backprop(x,w1,w2,w3,n,m): # w1 25x10 w2 10x25
33     dfdw3 = np.zeros((n,n,m))
34     dfdw2 = np.zeros((n,m,m))
35     dfdw1 = np.zeros((n,m,n))
36
37     # Compute dv2dw2, dv1dw1
38     dv2dw2 = np.zeros((m,m,m))
39     dv1dw1 = np.zeros((m,m,n))
40     h1 = vec_sigmod(w1.dot(x))
41     h2 = vec_sigmod(w2.dot(h1))
42     for i in range(m):
43         dv2dw2[i][i] = h1.T
44         dv1dw1[i][i] = x.T
45
46     # Compute dfdv2, dfdv1
47     dh2dv2 = np.zeros((m,m)) # 25x25
48     dh1dv1 = np.zeros((m,m)) # 25x25
49     v1 = w1.dot(x) # 25x1
50     v2 = w2.dot(vec_sigmod(v1))
51     for i in range(m):
52         #dh2dv[i][i] = sigmod(v[i])(1-sigmod(v[i]))
53         dh2dv2[i][i] = h2[i]*(1-h2[i])
54         dh1dv1[i][i] = h1[i]*(1-h1[i])
55     dfdv2 = w3.dot(dh2dv2) # nxm
56     dfdv1 = w3.dot(dh2dv2.dot(w2.dot(dh1dv1))) # nxm

```

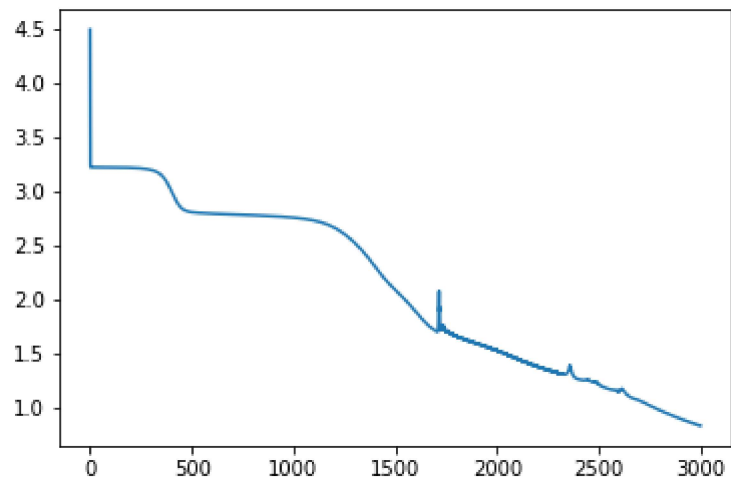
```
57
58     # Compute dfdw1, dfdw2, dfdw3
59     for i in range (n):
60         for j in range(m):
61             #print(dv2dw2.shape, dfdv2.shape)
62             dfdw2[i] += dv2dw2[j]*dfdv2[i][j]
63             dfdw1[i] += dv1dw1[j]*dfdv1[i][j]
64     for i in range(n):
65         dfdw3[i][i] = h2.T
66
67     return dfdw1, dfdw2, dfdw3
```

```

In [80]: 1 # NN2 with m = 10
2 alpha = 0.1
3 m = 10
4 w1 = np.random.rand(m,n)*0.1 # 10x10
5 w2 = np.random.rand(m,m)*0.1 # 10x10
6 w3 = np.random.rand(n,m)*0.1 # 10x10
7
8 loss_list = []
9 for steps in range(3000):
10     Gradient_w1 = np.zeros(w1.shape)
11     Gradient_w2 = np.zeros(w2.shape)
12     Gradient_w3 = np.zeros(w3.shape)
13     total_loss = 0
14     for col in range(t):
15         loss = Sigmod_forward_pass(x[:,col],y[:,col],w1,w2,w3)
16         total_loss += np.linalg.norm(loss)**2
17         dfdw1,dfdw2,dfdw3 = Sigmod_backprop(x[:,col],w1,w2,w3,n,m)
18         Gradient_w1 += VecXTen(loss,dfdw1)
19         Gradient_w2 += VecXTen(loss,dfdw2)
20         Gradient_w3 += VecXTen(loss,dfdw3)
21     Gradient_w1 = Gradient_w1 * 2 / t
22     Gradient_w2 = Gradient_w2 * 2 / t
23     Gradient_w3 = Gradient_w3 * 2 / t
24     w1 = w1 - alpha * Gradient_w1
25     w2 = w2 - alpha * Gradient_w2
26     w3 = w3 - alpha * Gradient_w3
27     #print("at iteration ", steps, " loss is ", total_loss/t)
28     loss_list.append(total_loss/t)
29 print("Final Loss: ", total_loss/t)
30 #print("final W2 * W1: ")
31 #print(w2.dot(w1))
32 pyp.plot(loss_list)
33 pyp.show()
34 #print("GradientW1: ")
35 #print(Gradient_w1)
36 #print("loss:")
37 #print(loss)
38 #print("dfdw1:")
39 #print(dfdw1)
40 #print("dfdw2:")
41 #print(dfdw2)
42

```

Final Loss: 0.8272811067483257

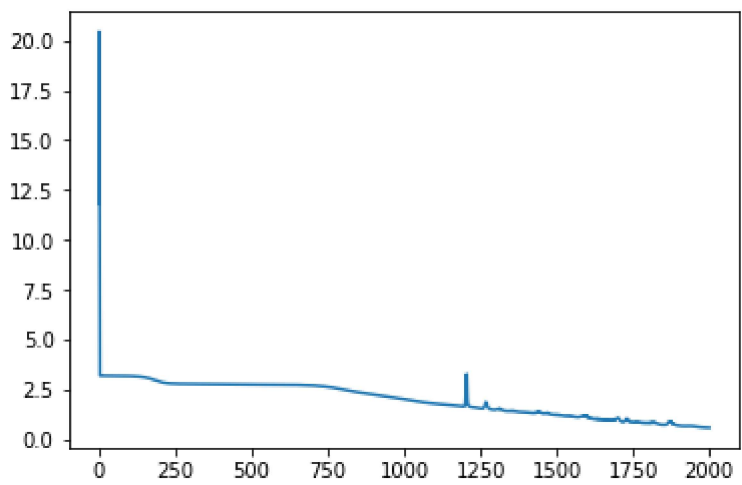


```

In [112]: 1 # NN2 with m = 25
2 alpha = 0.1
3 m = 25
4 w1 = np.random.rand(m,n)*0.1 # 10x10
5 w2 = np.random.rand(m,m)*0.1 # 10x10
6 w3 = np.random.rand(n,m)*0.1 # 10x10
7
8 loss_list = []
9 for steps in range(2000):
10     Gradient_w1 = np.zeros(w1.shape)
11     Gradient_w2 = np.zeros(w2.shape)
12     Gradient_w3 = np.zeros(w3.shape)
13     total_loss = 0
14     for col in range(t):
15         loss = Sigmod_forward_pass(x[:,col],y[:,col],w1,w2,w3)
16         total_loss += np.linalg.norm(loss)**2
17         dfdw1,dfdw2,dfdw3 = Sigmod_backprop(x[:,col],w1,w2,w3,n,m)
18         Gradient_w1 += VecXTen(loss,dfdw1)
19         Gradient_w2 += VecXTen(loss,dfdw2)
20         Gradient_w3 += VecXTen(loss,dfdw3)
21     Gradient_w1 = Gradient_w1 * 2 / t
22     Gradient_w2 = Gradient_w2 * 2 / t
23     Gradient_w3 = Gradient_w3 * 2 / t
24     w1 = w1 - alpha * Gradient_w1
25     w2 = w2 - alpha * Gradient_w2
26     w3 = w3 - alpha * Gradient_w3
27     #print("at iteration ", steps, " loss is ", total_loss/t)
28     loss_list.append(total_loss/t)
29 print("Final Loss: ", total_loss/t)
30 #print("final W2 * W1: ")
31 #print(w2.dot(w1))
32 pyp.plot(loss_list)
33 pyp.show()
34 #print("GradientW1: ")
35 #print(Gradient_w1)
36 #print("Loss:")
37 #print(loss)
38 #print("dfdw1:")
39 #print(dfdw1)
40 #print("dfdw2:")
41 #print(dfdw2)

```

Final Loss: 0.6054593372685161



```

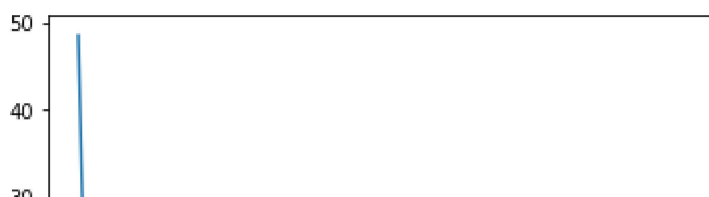
In [115]: 1 # NN2 with m = 50
2 alpha = 0.005
3 m = 50
4 w1 = np.random.rand(m,n)*0.1 # 10x10
5 w2 = np.random.rand(m,m)*0.1 # 10x10
6 w3 = np.random.rand(n,m)*0.1 # 10x10
7
8 loss_list = []
9 for steps in range(100):
10     Gradient_w1 = np.zeros(w1.shape)
11     Gradient_w2 = np.zeros(w2.shape)
12     Gradient_w3 = np.zeros(w3.shape)
13     total_loss = 0
14     for col in range(t):
15         loss = Sigmod_forward_pass(x[:,col],y[:,col],w1,w2,w3)
16         total_loss += np.linalg.norm(loss)**2
17         dfdw1,dfdw2,dfdw3 = Sigmod_backprop(x[:,col],w1,w2,w3,n,m)
18         Gradient_w1 += VecXTen(loss,dfdw1)
19         Gradient_w2 += VecXTen(loss,dfdw2)
20         Gradient_w3 += VecXTen(loss,dfdw3)
21     Gradient_w1 = Gradient_w1 * 2 / t
22     Gradient_w2 = Gradient_w2 * 2 / t
23     Gradient_w3 = Gradient_w3 * 2 / t
24     w1 = w1 - alpha * Gradient_w1
25     w2 = w2 - alpha * Gradient_w2
26     w3 = w3 - alpha * Gradient_w3
27     print("at iteration ", steps, " loss is ", total_loss/t)
28     loss_list.append(total_loss/t)
29 print("Final Loss: ", total_loss/t)
30 #print("final W2 * W1: ")
31 #print(w2.dot(w1))
32 pyp.plot(loss_list)
33 pyp.show()
34 #print("GradientW1: ")
35 #print(Gradient_w1)
36 #print("Loss:")
37 #print(loss)
38 #print("dfdw1:")
39 #print(dfdw1)
40 #print("dfdw2:")
41 #print(dfdw2)

```

```

at iteration 93 loss is 3.2200016407132317
at iteration 94 loss is 3.219999372886718
at iteration 95 loss is 3.21999710463573
at iteration 96 loss is 3.219994835959345
at iteration 97 loss is 3.219992566856646
at iteration 98 loss is 3.219990297326709
at iteration 99 loss is 3.219988027368617
Final Loss: 3.219988027368617

```







In [109]:

```

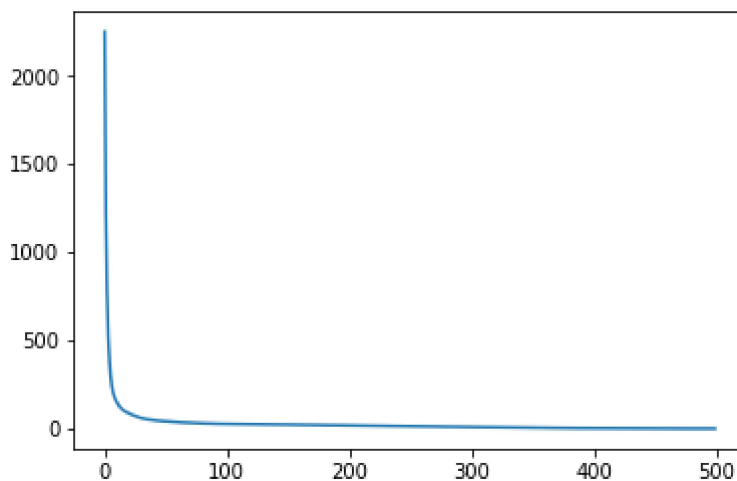
1  # Linear SVM NN
2  X = np.load("assignment9_X.npy")
3  Y = np.load("assignment9_Y.npy")
4  t = np.shape(X)[1]  # number of samples 50 (=np.shape(Y))
5  n = np.shape(X)[0]  # dimension of samples 10
6  k = 10              # number of classes
7  L = 0               # loss
8  total_loss = 0      # total loss
9  L_list = []
10 alpha = 0.0012      # learning rate
11 W = np.random.rand(k,n)*0.01 # weight matrix
12 fix_margin = 5
13 for steps in range(500):
14     dLdW = np.zeros((k,n))  # partial devirative of L with respect to W
15     for i in range(t):
16         classOfx = Y[i]
17         counter = 0
18         for j in range(k):
19             if j != classOfx:
20                 L += max(0, W[j].dot(X[:,i]) - W[classOfx].dot(X[:,i]) + fix_margin)
21                 #print(X[:,i].shape)
22                 if (W[j].dot(X[:,i]) - W[classOfx].dot(X[:,i]) + fix_margin > 0):
23                     counter += 1
24                 total_loss += L
25                 L = 0
26             dLdW[classOfx] += -1 * counter * X[:,i].T
27         W = W - alpha * dLdW
28         print("at iteration ", steps, " loss is ", total_loss)
29         L_list.append(total_loss)
30         total_loss = 0
31 pyp.plot(L_list)
32 pyp.show()

```

```

at iteration 496 loss is 0.2264006135887655
at iteration 497 loss is 0.25355378882571245
at iteration 498 loss is 0.2648470522566271
at iteration 499 loss is 0.2154416620994013

```



In [32]: 1 Y

Out[32]: array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4,  
4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 8, 8, 8, 8,  
8, 9, 9, 9, 9, 9, 9])

In [ ]: 1