In [89]:

```
import numpy as np
import matplotlib.pyplot as pyp
x = np.load("assignment8_X.npy")
y = np.load("assignment8_Y.npy")
x = x.T
y = y.T

alpha = 0.01
n = x.shape[0] # n = 10
t = x.shape[1] # t = 25
#w1 = np.ones((m,n))*0.01
#w2 = np.ones((n,m))*0.01

print(n,t)

def NN1_forward_pass(x,y,w1,w2,w3):
    #print("before forward pass, x")
    #print(x)
    #print("w2")
    #print(w2)
    #print("w1")
    #print(w1)
    #print("y")
    #print(y)
    fwx = w3.dot(w2.dot(w1.dot(x)))
    #print("fwx")
    #print(fwx)
    loss = fwx - y
    #print("loss")
    #print(loss)
    return loss

def NN1_backprop(x,w1,w2,w3,n,m):
    dfdw3 = np.zeros((n,n,m))  # 10x10x25
    dfdw2 = np.zeros((n,m,m))  # 10x25x25
    dfdw1 = np.zeros((n,m,n))  # 10x25x10
    dfdh1 = w3.dot(w2)         # 10x25
    dfdh2 = w3                 # 10x25
    dh1dw1 = np.zeros((m,m,n)) # 25x25x10
    dh2dw2 = np.zeros((m,m,m)) # 25x25x25
    h1 = w1.dot(x)             # 25x1
    h2 = w2.dot(h1)            # 25x1
    #print("In BP, h1", x.shape)
    for i in range(n):
        dfdw3[i][i] = h2
    for i in range(m):
        dh1dw1[i][i] = x
        dh2dw2[i][i] = h1
    for i in range(n):
            for j in range(m):
                dfdw2[i] += dh2dw2[j]*dfdh2[i][j]
                dfdw1[i] += dh1dw1[j]*dfdh1[i][j]
    return dfdw1,dfdw2,dfdw3

def VecXTen(vec,tensor):
    result = np.zeros((tensor.shape[1],tensor.shape[2]))
```

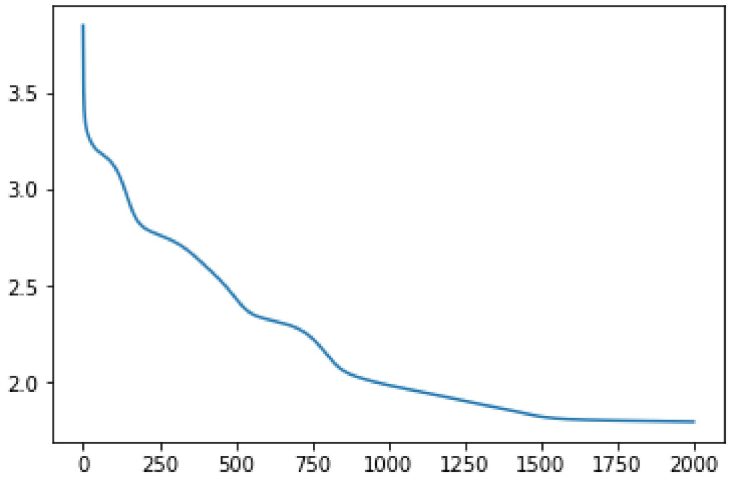
```
    for i in range(len(vec)):
        result += vec[i]*tensor[i]
    return result
```

```
10 25
```

In [91]:

```
# NN1 with m = 10
m = 10
w1 = np.random.rand(m,n)*0.1
w2 = np.random.rand(m,m)*0.1
w3 = np.random.rand(n,m)*0.1
loss_list = []
for steps in range(2000):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    Gradient_w3 = np.zeros(w3.shape)
    total_loss = 0
    for col in range(t):
        loss = NN1_forward_pass(x[:,col],y[:,col],w1,w2,w3)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2,dfdw3 = NN1_backprop(x[:,col],w1,w2,w3,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
        Gradient_w3 += VecXTen(loss,dfdw3)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    Gradient_w3 = Gradient_w3 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    w3 = w3 - alpha * Gradient_w3
    #print("at iteration ", steps, " loss is ", total_loss/t)
    loss_list.append(total_loss/t)
print("Final Loss: ", total_loss/t)
#print("final W2 * W1: ")
#print(w2.dot(w1))
pyp.plot(loss_list[1:])
pyp.show()

    #print("GradientW1: ")
    #print(Gradient_w1)
    #print("loss:")
    #print(loss)
    #print("dfdw1:")
    #print(dfdw1)
    #print("dfdw2:")
    #print(dfdw2)

```

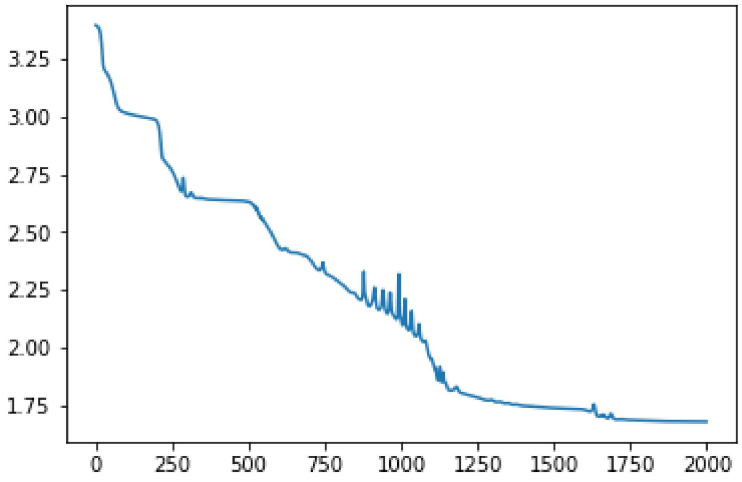
```
Final Loss:  1.7962366226647475
```


3.5
3.0
2.5
2.0
0
250
500
750
1000
1250
1500
1750
2000

In [92]:

```
# NN1 with m = 25
m = 25
alpha = 0.05
w1 = np.random.rand(m,n)*0.01
w2 = np.random.rand(m,m)*0.01
w3 = np.random.rand(n,m)*0.01
loss_list = []
for steps in range(2000):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    Gradient_w3 = np.zeros(w3.shape)
    total_loss = 0
    for col in range(t):
        loss = NN1_forward_pass(x[:,col],y[:,col],w1,w2,w3)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2,dfdw3 = NN1_backprop(x[:,col],w1,w2,w3,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
        Gradient_w3 += VecXTen(loss,dfdw3)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    Gradient_w3 = Gradient_w3 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    w3 = w3 - alpha * Gradient_w3
    #print("at iteration ", steps, " loss is ", total_loss/t)
    loss_list.append(total_loss/t)
print("Final Loss: ", total_loss/t)
#print("final W2 * W1: ")
#print(w2.dot(w1))
pyp.plot(loss_list[1:])
pyp.show()

    #print("GradientW1: ")
    #print(Gradient_w1)
    #print("loss:")
    #print(loss)
    #print("dfdw1:")
    #print(dfdw1)
    #print("dfdw2:")
    #print(dfdw2)

```

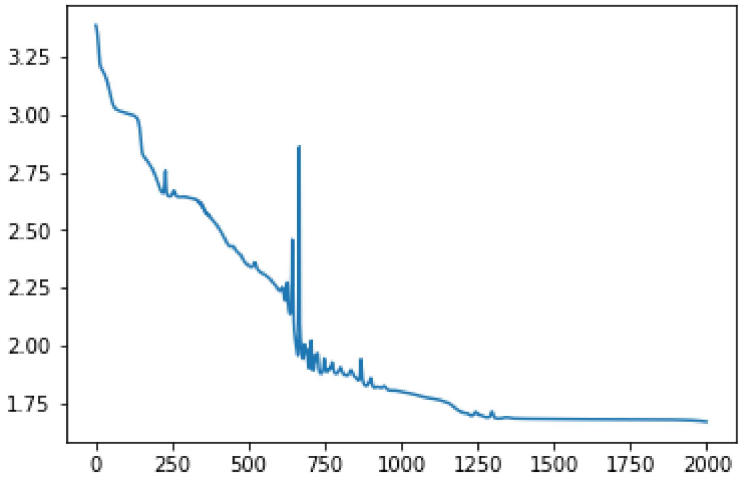
```
Final Loss:  1.6788260271653126
```


3.25
3.00
2.75
2.50
2.25
2.00
1.75
0
250
500
750
1000
1250
1500
1750
2000

In [95]:

```
# NN1 with m = 50
m = 50
alpha = 0.05
w1 = np.random.rand(m,n)*0.01
w2 = np.random.rand(m,m)*0.01
w3 = np.random.rand(n,m)*0.01
loss_list = []
for steps in range(2000):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    Gradient_w3 = np.zeros(w3.shape)
    total_loss = 0
    for col in range(t):
        loss = NN1_forward_pass(x[:,col],y[:,col],w1,w2,w3)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2,dfdw3 = NN1_backprop(x[:,col],w1,w2,w3,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
        Gradient_w3 += VecXTen(loss,dfdw3)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    Gradient_w3 = Gradient_w3 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    w3 = w3 - alpha * Gradient_w3
    #print("at iteration ", steps, " loss is ", total_loss/t)
    loss_list.append(total_loss/t)
print("Final Loss: ", total_loss/t)
#print("final W2 * W1: ")
#print(w2.dot(w1))
pyp.plot(loss_list[1:])
pyp.show()

    #print("GradientW1: ")
    #print(Gradient_w1)
    #print("loss:")
    #print(loss)
    #print("dfdw1:")
    #print(dfdw1)
    #print("dfdw2:")
    #print(dfdw2)
```

```
Final Loss:  1.668421258126552
```


3.25
3.00
2.75
2.50
2.25
2.00
1.75
0
250
500
750
1000
1250
1500
1750
2000

In [111]:

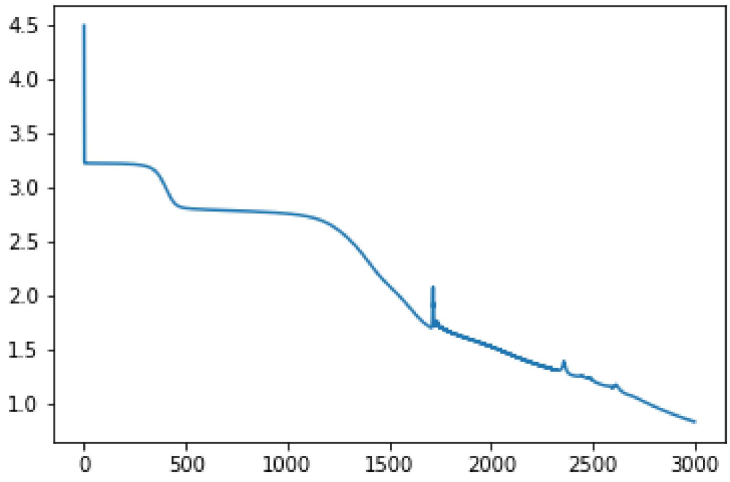
```
x = np.load("assignment8_X.npy")
y = np.load("assignment8_Y.npy")
x = x.T
y = y.T

alpha = 0.01
n = x.shape[0] # n = 10
t = x.shape[1] # t = 25
def vec_sigmod(x):
    for i in range(len(x)):
        x[i] = 1/(1+np.exp(-x[i]))
    return x
def sigmod(x):
    return 1/(1+np.exp(-x))
def Sigmod_forward_pass(x,y,w1,w2,w3):
    #print("before forward pass, x")
    #print(x)
    #print("w2")
    #print(w2)
    #print("w1")
    #print(w1)
    #print("y")
    #print(y)
    fwx = w3.dot(vec_sigmod(w2.dot(vec_sigmod(w1.dot(x)))))
    #print("fwx")
    #print(fwx)
    loss = fwx - y
    #print("loss")
    #print(loss)
    return loss

def Sigmod_backprop(x,w1,w2,w3,n,m): # w1 25x10 w2 10x25
    dfdw3 = np.zeros((n,n,m))
    dfdw2 = np.zeros((n,m,m))
    dfdw1 = np.zeros((n,m,n))

    # Compute dv2dw2, dv1dw1
    dv2dw2 = np.zeros((m,m,m))
    dv1dw1 = np.zeros((m,m,n))
    h1 = vec_sigmod(w1.dot(x))
    h2 = vec_sigmod(w2.dot(h1))
    for i in range(m):
        dv2dw2[i][i] = h1.T
        dv1dw1[i][i] = x.T

    # Compute dfdv2,dfdv1
    dh2dv2 = np.zeros((m,m))    # 25x25
    dh1dv1 = np.zeros((m,m))    # 25x25
    v1 = w1.dot(x) # 25x1
    v2 = w2.dot(vec_sigmod(v1))
    for i in range(m):
        #dh2dv[i][i] = sigmod(v[i])(1-sigmod(v[i]))
        dh2dv2[i][i] = h2[i]*(1-h2[i])
        dh1dv1[i][i] = h1[i]*(1-h1[i])
    dfdv2 = w3.dot(dh2dv2)     # nxm
    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]:

```
# NN2 with m = 10
alpha = 0.1
m = 10
w1 = np.random.rand(m,n)*0.1 # 10x10
w2 = np.random.rand(m,m)*0.1 # 10x10
w3 = np.random.rand(n,m)*0.1 # 10x10

loss_list = []
for steps in range(3000):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    Gradient_w3 = np.zeros(w3.shape)
    total_loss = 0
    for col in range(t):
        loss = Sigmod_forward_pass(x[:,col],y[:,col],w1,w2,w3)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2,dfdw3 = Sigmod_backprop(x[:,col],w1,w2,w3,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
        Gradient_w3 += VecXTen(loss,dfdw3)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    Gradient_w3 = Gradient_w3 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    w3 = w3 - alpha * Gradient_w3
    #print("at iteration ", steps, " loss is ", total_loss/t)
    loss_list.append(total_loss/t)
print("Final Loss: ", total_loss/t)
#print("final W2 * W1: ")
#print(w2.dot(w1))
pyp.plot(loss_list)
pyp.show()
    #print("GradientW1: ")
    #print(Gradient_w1)
    #print("loss:")
    #print(loss)
    #print("dfdw1:")
    #print(dfdw1)
    #print("dfdw2:")
    #print(dfdw2)

```

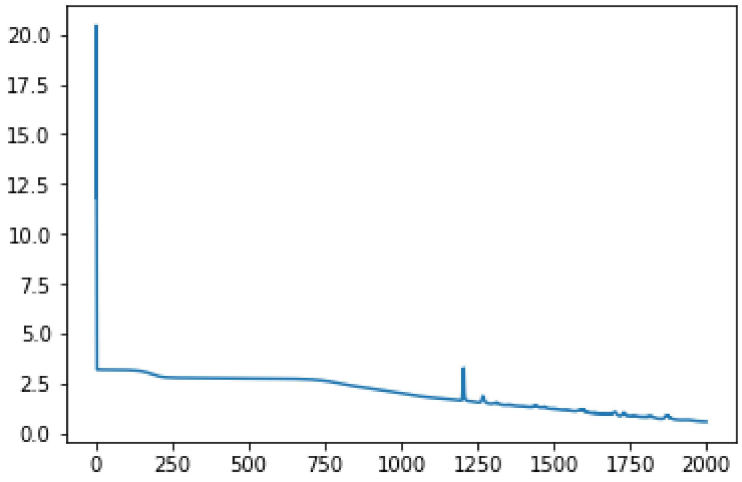
```
Final Loss:  0.8272811067483257
```


4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
500
1000
1500
2000
2500
3000

In [112]:

```
# NN2 with m = 25
alpha = 0.1
m = 25
w1 = np.random.rand(m,n)*0.1 # 10x10
w2 = np.random.rand(m,m)*0.1 # 10x10
w3 = np.random.rand(n,m)*0.1 # 10x10

loss_list = []
for steps in range(2000):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    Gradient_w3 = np.zeros(w3.shape)
    total_loss = 0
    for col in range(t):
        loss = Sigmod_forward_pass(x[:,col],y[:,col],w1,w2,w3)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2,dfdw3 = Sigmod_backprop(x[:,col],w1,w2,w3,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
        Gradient_w3 += VecXTen(loss,dfdw3)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    Gradient_w3 = Gradient_w3 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    w3 = w3 - alpha * Gradient_w3
    #print("at iteration ", steps, " loss is ", total_loss/t)
    loss_list.append(total_loss/t)
print("Final Loss: ", total_loss/t)
#print("final W2 * W1: ")
#print(w2.dot(w1))
pyp.plot(loss_list)
pyp.show()
    #print("GradientW1: ")
    #print(Gradient_w1)
    #print("loss:")
    #print(loss)
    #print("dfdw1:")
    #print(dfdw1)
    #print("dfdw2:")
    #print(dfdw2)
```

```
Final Loss:  0.6054593372685161
```


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
250
500
750
1000
1250
1500
1750
2000

In [115]:

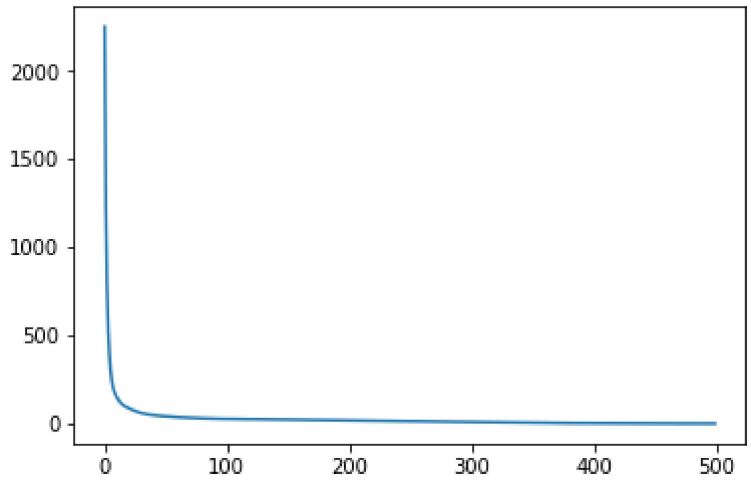
```
# NN2 with m = 50
alpha = 0.005
m = 50
w1 = np.random.rand(m,n)*0.1 # 10x10
w2 = np.random.rand(m,m)*0.1 # 10x10
w3 = np.random.rand(n,m)*0.1 # 10x10

loss_list = []
for steps in range(100):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    Gradient_w3 = np.zeros(w3.shape)
    total_loss = 0
    for col in range(t):
        loss = Sigmod_forward_pass(x[:,col],y[:,col],w1,w2,w3)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2,dfdw3 = Sigmod_backprop(x[:,col],w1,w2,w3,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
        Gradient_w3 += VecXTen(loss,dfdw3)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    Gradient_w3 = Gradient_w3 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    w3 = w3 - alpha * Gradient_w3
    print("at iteration ", steps, " loss is ", total_loss/t)
    loss_list.append(total_loss/t)
print("Final Loss: ", total_loss/t)
#print("final W2 * W1: ")
#print(w2.dot(w1))
pyp.plot(loss_list)
pyp.show()
    #print("GradientW1: ")
    #print(Gradient_w1)
    #print("loss:")
    #print(loss)
    #print("dfdw1:")
    #print(dfdw1)
    #print("dfdw2:")
    #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
```


20

In [109]:
```
# Linear SVM NN
X = np.load("assignment9_X.npy")
Y = np.load("assignment9_Y.npy")
t = np.shape(X)[1]   # number of sameples 50 (=np.shape(Y))
n = np.shape(X)[0]   # dimension of samples 10
k = 10               # number of classes
L = 0                # loss
total_loss = 0       # total loss
L_list = []
alpha = 0.0012         # learning rate
W = np.random.rand(k,n)*0.01 # weight matrix
fix_margin = 5
for steps in range(500):
    dLdW = np.zeros((k,n))    # partial devirative of L with respect to W
    for i in range(t):
        classOfx = Y[i]
        counter = 0
        for j in range(k):
            if j != classOfx:
                L += max(0,W[j].dot(X[:,i])-W[classOfx].dot(X[:,i])+fix_marg
                #print(X[:,i].shape)
                if(W[j].dot(X[:,i])-W[classOfx].dot(X[:,i])+fix_margin > 0):
                    counter += 1
                total_loss += L
                L = 0
        dLdW[classOfx] += -1 * counter * X[:,i].T
    W = W - alpha * dLdW
    print("at iteration ", steps, " loss is ", total_loss)
    L_list.append(total_loss)
    total_loss = 0
pyp.plot(L_list)
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])
```

In [ ]:
```
1
```