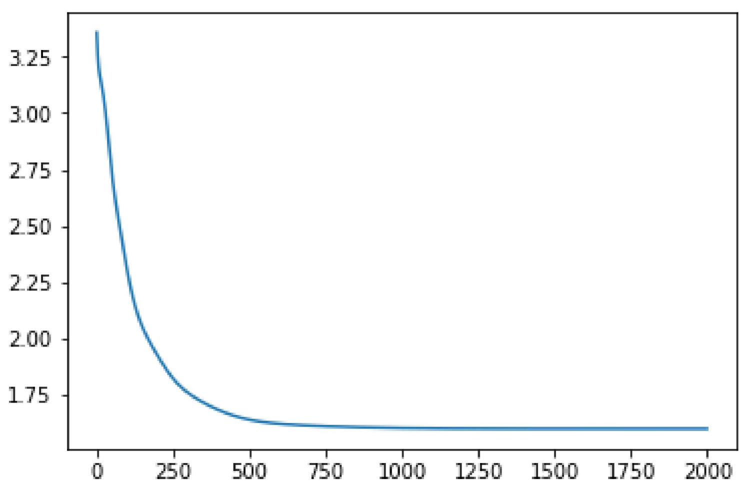
In [92]:
```
import numpy as np
import matplotlib.pyplot as pyp
x = np.load("assignment8_X.npy")
y = np.load("assignment8_Y.npy")
x = x.T
y = y.T

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

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

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
```

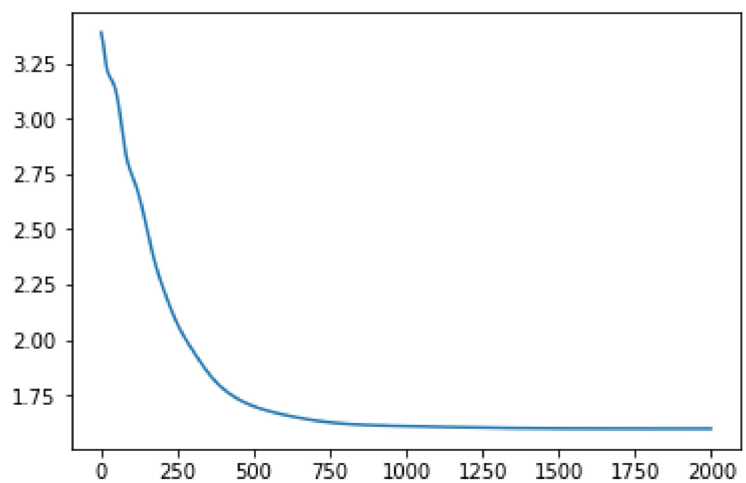
In [93]:
```
# NN1 with m = 10
m = 10
w1 = np.random.rand(m,n)*0.1
w2 = np.random.rand(n,m)*0.1
loss_list = []
for steps in range(2000):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    total_loss = 0
    for col in range(t):
        loss = NN1_forward_pass(x[:,col],y[:,col],w1,w2)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2 = NN1_backprop(x[:,col],w1,w2,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    #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:  1.5996682905619355
final W2 * W1:
[[ 0.04000553  0.04277312  0.00727076 -0.10074972 -0.04926801  0.01812062
   0.16405513 -0.07455311  0.09417713  0.00769529]
 [ 0.19056985 -0.00950993 -0.02441188 -0.03965862 -0.06275345 -0.07666929
  -0.07411455  0.04234344  0.06156518  0.04591418]
 [-0.07964575  0.11123149 -0.00404499 -0.0422513  -0.00223563  0.07931053
  -0.07794135  0.07692016  0.04460551 -0.00203585]
 [-0.08860784  0.07459369 -0.06969219 -0.0393823  -0.08297476  0.21171445
   0.06491159  0.02004106 -0.04257376  0.09869675]
 [-0.01775407  0.05293991  0.05674882 -0.16656887 -0.05560876  0.01569452
   0.06579749 -0.04484191  0.06334307  0.18209319]
 [ 0.10603431  0.07128906  0.05888189  0.01104256 -0.00742511  0.01194603
  -0.09377751 -0.14406504 -0.11928831  0.02950298]
 [ 0.04830487 -0.01969746  0.13453888 -0.02210835 -0.06462217  0.0530446
  -0.04083299  0.0088379  -0.07215325  0.02148751]
 [-0.10987982 -0.09039759  0.08056817  0.12137595  0.04381749 -0.20047274
   0.05487633  0.00693684 -0.03855584 -0.00739978]
 [-0.03042103 -0.07150182  0.09929712  0.14935695  0.10468111 -0.23262494
  -0.07545212 -0.03082965  0.04508484 -0.082177  ]
 [-0.09517868  0.046199    0.01650886 -0.08242874  0.15080898 -0.18602401
   0.0020888   0.10102352 -0.04069869  0.10602817]]
```



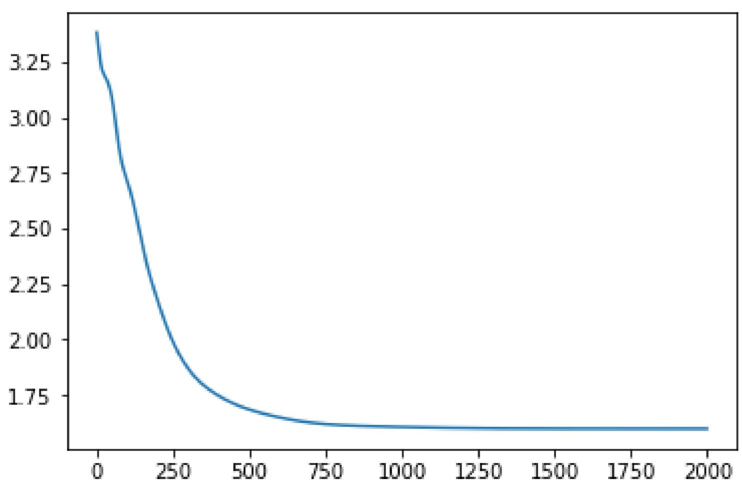
In [94]:
```
# NN1 with m = 25
m = 25
w1 = np.random.rand(m,n)*0.01 # 25x10
w2 = np.random.rand(n,m)*0.01 # 10x25

def MatrixDotTensor(M,T):
    result

loss_list = []
for steps in range(2000):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    total_loss = 0
    for col in range(t):
        loss = NN1_forward_pass(x[:,col],y[:,col],w1,w2)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2 = NN1_backprop(x[:,col],w1,w2,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    #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:  1.5996846794554287
final W2 * W1:
[[ 0.04028455  0.04255791  0.00744263 -0.10084159 -0.04894787  0.01791239
   0.16382118 -0.07443427  0.09407809  0.00736689]
 [ 0.19125012 -0.01002701 -0.02396359 -0.03983186 -0.06197547 -0.07723066
  -0.07469015  0.04259625  0.06132685  0.04509187]
 [-0.07922443  0.11090086 -0.00377334 -0.0423536  -0.00175354  0.07896071
  -0.07830031  0.0770699   0.04446933 -0.0025365 ]
 [-0.08767172  0.07389067 -0.06906594 -0.03961794 -0.08190418  0.21093379
   0.06412252  0.02039097 -0.04291098  0.09755439]
 [-0.01786574  0.05302554  0.05669008 -0.16651105 -0.05573742  0.01575498
   0.06588941 -0.04490497  0.06338684  0.18221814]
 [ 0.1063882   0.07101633  0.05911441  0.01095743 -0.00702049  0.0116498
  -0.0940781  -0.14393833 -0.11940747  0.02907738]
 [ 0.04793341 -0.01941147  0.13428241 -0.02204338 -0.06504642  0.05338191
  -0.04051522  0.00872299 -0.07203209  0.02194265]
 [-0.1091928  -0.09093534  0.08102204  0.12122655  0.04460402 -0.20106294
   0.05428981  0.00716798 -0.038777   -0.00822449]
 [-0.03006726 -0.07175514  0.09953746  0.14925728  0.10508437 -0.23291223
  -0.0757465  -0.03068511  0.04494376 -0.0826162 ]
 [-0.09534964  0.04633269  0.01638796 -0.08240817  0.15061395 -0.18585947
   0.00223626  0.10097826 -0.04064672  0.10623854]]
```



In [95]:
```
# NN1 with  = 50
m = 50
w1 = np.random.rand(m,n)*0.01 # 25x10
w2 = np.random.rand(n,m)*0.01 # 10x25

def MatrixDotTensor(M,T):
    result

loss_list = []
for steps in range(2000):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    total_loss = 0
    for col in range(25):
        loss = NN1_forward_pass(x[:,col],y[:,col],w1,w2)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2 = NN1_backprop(x[:,col],w1,w2,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    #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()
```

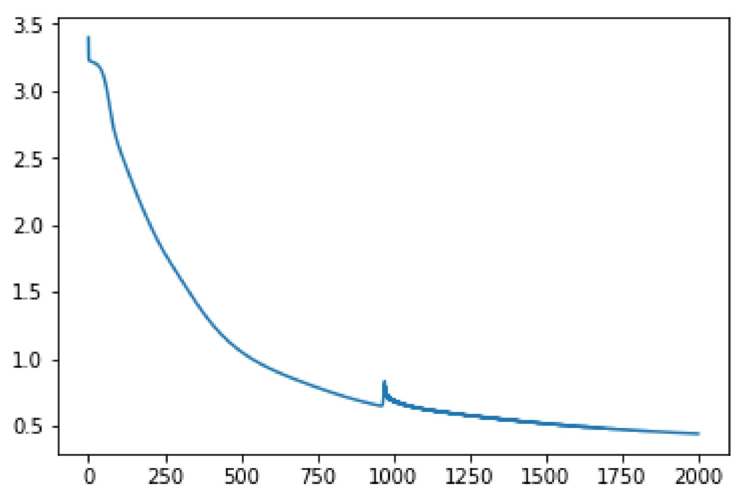
```
Final Loss:  1.5996719155717478
final W2 * W1:
[[ 0.04016382  0.04265043  0.00736788 -0.10080255 -0.04908586  0.01800366
   0.16392212 -0.07448566  0.09412081  0.00750937]
 [ 0.1908692  -0.00973697 -0.02421165 -0.03973425 -0.06240879 -0.07691716
  -0.07436921  0.04245291  0.06145857  0.04554893]
 [-0.07954292  0.11115414 -0.00397178 -0.04227031 -0.00211658  0.07921758
  -0.07803065  0.0769524   0.04456991 -0.00216511]
 [-0.08808195  0.07419232 -0.06934644 -0.03952153 -0.08237066  0.21128578
   0.06446716  0.02023868 -0.04276018  0.09806103]
 [-0.0178375   0.05300478  0.05670257 -0.16653162 -0.05570496  0.01574576
   0.06586687 -0.04488447  0.06337433  0.18218773]
 [ 0.10615344  0.07119925  0.05896461  0.01101717 -0.00728768  0.01184204
  -0.09387973 -0.14402502 -0.11933023  0.02935507]
 [ 0.04820693 -0.01962464  0.1344627  -0.02210143 -0.06473529  0.05314544
  -0.04074737  0.0088156  -0.07212034  0.02161529]
 [-0.10971566 -0.09052159  0.0806887   0.12135211  0.04400809 -0.20062814
   0.05473304  0.00698277 -0.0386116  -0.00760867]
 [-0.03011037 -0.07173883  0.09949644  0.14926711  0.10503657 -0.23286981
  -0.07571245 -0.03070648  0.04497351 -0.0825484 ]
 [-0.09519775  0.04621236  0.01648776 -0.08243797  0.15078676 -0.18599274
   0.00210689  0.10102722 -0.04069353  0.10605787]]
```



In [96]:
```
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):
    #print("before forward pass, x")
    #print(x)
    #print("w2")
    #print(w2)
    #print("w1")
    #print(w1)
    #print("y")
    #print(y)
    fwx = 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,n,m): # w1 25x10 w2 10x25
    # Compute dh2dv
    dh2dv = np.zeros((m,m))    # now it's the diagnal matrix 25x25
    v = w1.dot(x) # 25x1
    for i in range(m):
        #dh2dv[i][i] = sigmod(v[i])(1-sigmod(v[i]))
        dh2dv[i][i] = 1/(1+np.exp(-v[i]))*(1-(1/(1+np.exp(-v[i]))))
    dh2dv = w2.dot(dh2dv)      # now becomes 10x25

    # Compute dh2dw2
    dh2dw2 = np.zeros((n,n,m)) # 10x10x25
    h1 = vec_sigmod(v) # 25x1
    for i in range (n):
        dh2dw2[i][i] = h1.T

    # Compute dh2dw1
    dh2dw1 = np.zeros((n,m,n)) # 10x25x10
    dh2dh1 = w2 # 10x25
    dvdw1 = np.zeros((m,m,n)) # 25x25x10
    for i in range(n):
        dvdw1[i][i] = x.T
    for i in range(n):
        for j in range(m):
            dh2dw1[i] += dh2dv[i][j] * dvdw1[j]
    #h2 = w2.dot(h1)
    #print("In BP, h1", x.shape)
    return dh2dw1,dh2dw2
```

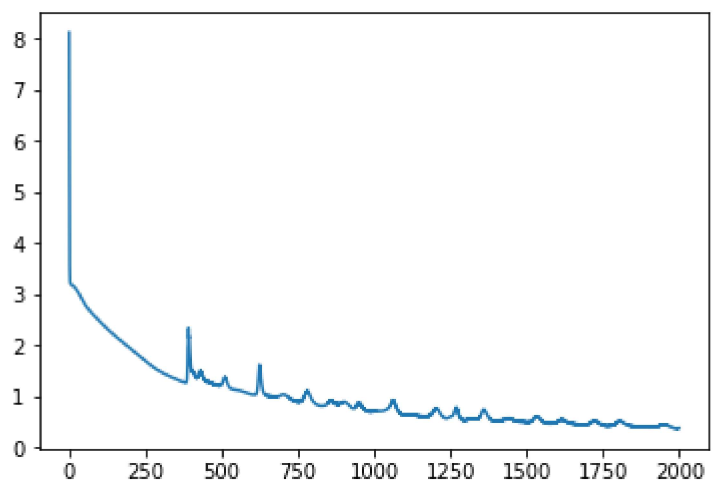
In [87]:
```
# NN2 with m = 10
alpha = 0.05
m = 10
w1 = np.random.rand(m,n)*0.1 # 10x10
w2 = np.random.rand(n,m)*0.1 # 10x10

def MatrixDotTensor(M,T):
    result

loss_list = []
for steps in range(2000):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    total_loss = 0
    for col in range(t):
        loss = Sigmod_forward_pass(x[:,col],y[:,col],w1,w2)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2 = Sigmod_backprop(x[:,col],w1,w2,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    #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:  0.4415079029923033
final W2 * W1:
[[-0.07792875  1.26047217  0.24288753 -1.04101342 -0.03236551  1.26265412
   1.73978784 -1.06861996  0.6968375  -0.10108926]
 [ 2.39096049 -0.69485861 -0.46023173 -0.41114585 -0.83896644 -2.0238753
  -1.1593185   0.91894712  0.57277413  0.89487216]
 [-0.340313    0.70868366  0.41592106 -0.61757281  0.26053334  0.91590736
  -1.43478285  0.63142013  1.67365308 -0.56879379]
 [ 0.14474475 -0.53820022 -1.54058198 -0.32504509 -0.33554511  1.64633654
   1.03731859  2.00499816 -2.32434775  0.11381506]
 [-0.75549634  1.76734983  1.71113761 -2.52907339 -0.35204513  0.32977498
   0.58814042 -1.36227611  1.66851674  1.90567084]
 [-0.20354824  2.0964318  -0.39562397  1.1328748  -0.1706735   1.88008042
  -0.15706535 -1.719455   -1.54668962  0.8012913 ]
 [ 0.83909138 -0.83872352  1.39393155 -0.46800796 -0.72893599  0.09636064
  -0.99307511  0.15321259  0.69068093  0.32015278]
 [-0.65167119 -0.74365659  0.79080324  0.60564337  0.50823699 -1.70559284
   0.28723696  0.37849398  0.19524159  0.20977487]
 [-0.50006357  0.77805798  0.60006463  1.73298767  0.65194041  0.3837412
  -0.71954299 -0.08778103  0.52653876 -0.82976834]
 [-0.19500948  0.37408055 -0.80647183 -2.35406319  2.31737755  0.48351757
  -0.36270931  3.59424657 -0.64124876  1.85121244]]
```



In [88]:
```
# NN2 with m = 25
m = 25
w1 = np.random.rand(m,n)*0.1 # 25x10
w2 = np.random.rand(n,m)*0.1 # 10x25

def MatrixDotTensor(M,T):
    result

loss_list = []
for steps in range(2000):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    total_loss = 0
    for col in range(t):
        loss = Sigmod_forward_pass(x[:,col],y[:,col],w1,w2)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2 = Sigmod_backprop(x[:,col],w1,w2,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    #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:  0.384848546520831
final W2 * W1:
[[ 1.47180089  0.10794128 -0.15319146 -1.62869891 -0.27252394  0.7218124
   0.66047549 -0.18687716  1.28958492  0.34814913]
 [ 2.97973782 -0.32251375  0.93710612 -0.30757407 -0.23875525 -3.3969485
  -2.07003161 -0.41991943  0.72906786 -0.25689576]
 [-0.86428934  0.95618765 -0.53644543 -0.40917083 -0.05667087  1.11011502
  -0.42041211  1.24824892  1.29390965 -0.28903272]
 [-1.32605553  0.30389395 -1.63188695 -1.09863537 -0.90934909  3.50259509
   1.25109027  0.98722907 -1.13999878  1.33746171]
 [-0.49923332  0.73990019  1.06936641 -2.33616208 -0.91581545  0.41486716
   0.67195345 -0.53691224  2.26120601  2.47516764]
 [ 0.38179611  1.87286024  0.13130928  0.27188112 -0.24111789  1.44087738
  -0.28592839 -0.90660852 -1.14418381  0.42471796]
 [ 0.0522099  -0.37859722  1.47363116 -0.03444832 -0.82227596  0.37031911
  -0.63406408  0.31711218 -0.57618017  0.21217306]
 [-1.02364498 -0.96950058  0.73912111  0.98826479  0.53290817 -1.95510143
   0.32700689  0.31535519 -0.12835918  0.50497389]
 [ 0.14694293 -0.81900828  0.15774315  1.16804755  0.97489469 -1.37041852
  -0.7014327   0.85005425  1.46275031 -0.78372666]
 [-1.74097876  0.30849883 -0.07796264 -0.57584104  2.3003075  -0.88860775
   0.08211258  0.56142316 -1.35122964  1.3729778 ]]
```



In [97]:
```
# NN2 with m = 50
m = 50
alpha = 0.01
w1 = np.random.rand(m,n)*0.1 # 25x10
w2 = np.random.rand(n,m)*0.1 # 10x25

loss_list = []
for steps in range(3000):
    Gradient_w1 = np.zeros(w1.shape)
    Gradient_w2 = np.zeros(w2.shape)
    total_loss = 0
    for col in range(t):
        loss = Sigmod_forward_pass(x[:,col],y[:,col],w1,w2)
        total_loss += np.linalg.norm(loss)**2
        dfdw1,dfdw2 = Sigmod_backprop(x[:,col],w1,w2,n,m)
        Gradient_w1 += VecXTen(loss,dfdw1)
        Gradient_w2 += VecXTen(loss,dfdw2)
    Gradient_w1 = Gradient_w1 * 2 / t
    Gradient_w2 = Gradient_w2 * 2 / t
    w1 = w1 - alpha * Gradient_w1
    w2 = w2 - alpha * Gradient_w2
    #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[2:])
pyp.show()
    #print("GradientW1: ")
    #print(Gradient_w1)
    #print("loss:")
    #print(loss)
    #print("dfdw1:")
    #print(dfdw1)
    #print("dfdw2:")
    #print(dfdw2)
```

```
Final Loss:  0.9863187925716322
final W2 * W1:
[[ 0.42889014  0.17546646 -0.19683495 -0.82671975 -0.55920549  0.33970443
   0.95401632 -0.75653598  0.81211361  0.21135297]
 [ 1.25459271  0.35661764  0.2997379  -0.13877681 -0.48652803 -0.67250609
  -0.91624242 -0.16100282 -0.03284307  0.38815829]
 [-0.73802589  0.32118981  0.23015527 -0.321542   -0.08149486  0.24039566
  -1.08818103  0.17276021  0.62167946 -0.1889723 ]
 [-0.56252897 -0.00743492 -0.30109083 -0.51002061  0.01422624  0.96927679
   0.72037447  0.12477671 -0.88722727  0.27716824]
 [ 0.27647215  0.13781209  0.61134736 -1.44788394 -0.65580238  0.24045538
   0.24538112 -0.51154041  1.1203239   0.9880599 ]
 [ 0.75108316  0.26856963 -0.14457476 -0.11189094  0.18886235 -0.07566383
  -0.09155604 -0.46038683 -0.3543803   0.18934409]
 [-0.16599852 -0.2936916   1.19744887 -0.35565171 -0.75726292  0.2958303
  -0.47565119 -0.10588196 -0.2509306   0.18533279]
 [-0.39115139 -0.70529762  0.31158645  0.67414286  0.38461698 -0.95149181
   0.25559017  0.43634283 -0.0925802  -0.08482589]
 [-0.66532663 -0.41807472  0.28433433  0.78639086  0.52088791 -0.72075142
  -0.61257593  0.04780705  0.82791516 -0.5254725 ]
 [-0.51523638 -0.01101876 -0.26110855 -0.40528721  1.25700354 -1.0100179
   0.2378279   0.68108086  0.0063864   0.79601935]]
```