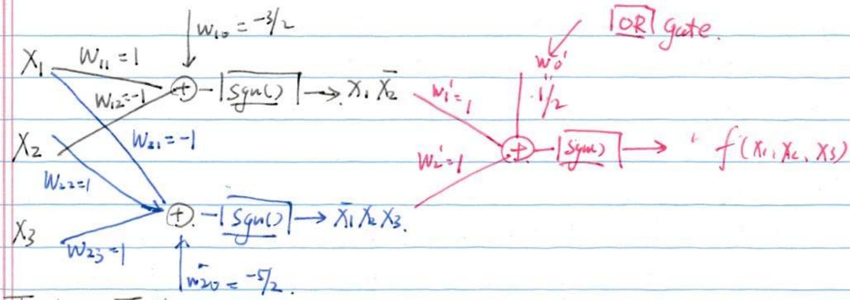
1.



Truth Table.

X_1	X_2	X_3	$X_1\bar{X}_2$	$\bar{X}_1X_2X_3$	$f(X_1,X_2,X_3)$
T	T	T	F	F	F
T	T	F	F	F	F
T	F	T	T	F	T
T	F	F	T	F	T
F	T	T	F	T	T
F	T	F	F	F	F
F	F	T	F	F	F
F	F	F	F	F	F

⇓

X_1	X_2	X_3	$X_1\bar{X}_2$	$\bar{X}_1X_2X_3$	$f(X)$
1	1	1	-1	-1	-1
1	1	-1	-1	-1	-1
1	-1	1	1	-1	1
1	-1	-1	1	-1	1
-1	1	1	-1	1	1
-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1
-1	-1	-1	-1	-1	-1

For the 1st layer, we have,

$$\begin{cases} W_{10} + W_{11} + W_{12} < 0 \ ; & W_{10} + W_{11} - W_{12} > 0. \\ W_{10} - W_{11} + W_{12} < 0 & W_{10} - W_{11} - W_{12} < 0. \end{cases}$$

$\exists$ a set of solution s.t. $w_{10} = -3/2$. $w_{11} = 1$ $w_{12} = -1$.

Also

$$\begin{cases} W_{20} + W_{21} + W_{22} + W_{23} < 0 \ ; & W_{20} + W_{21} + W_{22} - W_{23} < 0. \\ W_{20} + W_{21} - W_{22} + W_{23} < 0 \ ; & W_{20} + W_{21} - W_{22} - W_{23} < 0. \\ W_{20} - W_{21} + W_{22} + W_{23} > 0 \ ; & W_{20} - W_{21} + W_{22} - W_{23} < 0. \\ W_{20} - W_{21} - W_{22} + W_{23} < 0 \ ; & W_{20} - W_{21} - W_{22} - W_{23} < 0. \end{cases}$$

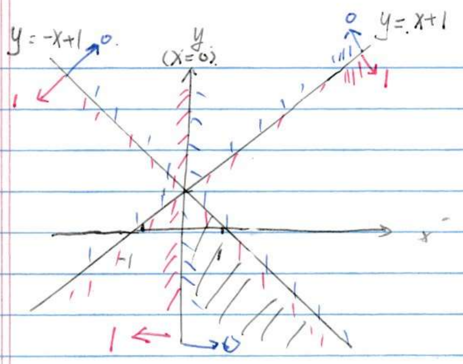
$\exists$ a set of solution s.t. $W_{20} = -5/2$, $W_{21} = -1$, $W_{22} = 1$ $W_{23} = 1$

For 2nd layer.

$$\begin{cases} W_0' - W_1' - W_2' < 0. \ ; & W_0' + W_1' - W_2' > 0. \\ W_0' - W_1' + W_2' > 0. \end{cases}$$

$\exists$ a set of solution: $W_0' = 1/2$, $W_1' = 1$, $W_2' = 1$.

2.



From the 1st layer, we can get 3 decision boundaries.

$$\begin{cases} y = x+1 & \langle 1 \rangle \\ y = -x+1 & \langle 2 \rangle \\ x = 0. & \langle 3 \rangle. \end{cases}$$

and like in the above graph, the "1" region (including the boundary) is sketched in red, "0" region is sketched in blue.

In the 2nd layer, to make $z=1$, the 3 outputs from the 1st layer need to be $\{1, 1, 0\}$. Hence, the target region is actually the region at the "1" side w.r.t. $\langle 1\rangle$ and $\langle 2\rangle$, but on the "0" side for $\langle 3\rangle$.
and it's block in blank in the above figure.

CS 559 Hwk 1

September 21, 2017

Q.3

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [2]: np.random.seed(42)
```

(b) Pick (your code should pick it) w0 uniformly at random on [1/4 , 1/4].

(c) Pick w1 uniformly at random on [1, 1].

(d) Pick w2 uniformly at random on [1, 1].

```
In [3]: # (b),(c),(d)
        w = np.zeros(3)
        w[0] = np.random.uniform(-.25,.25)
        w[1] = np.random.uniform(-1,1)
        w[2] = np.random.uniform(-1,1)
```

(e) Write in your report the numbers [w0, w1, w2] you have picked.

```
In [4]: # (e)
        w

Out[4]: array([-0.06272994,  0.90142861,  0.46398788])
```

(f) Pick n = 100 vectors x1, . . . , xn independently and uniformly at random on [1, 1]2, call the collection of these vectors S.

```
In [5]: # (f)
        S = np.random.uniform(-1,1,(100, 2))
```

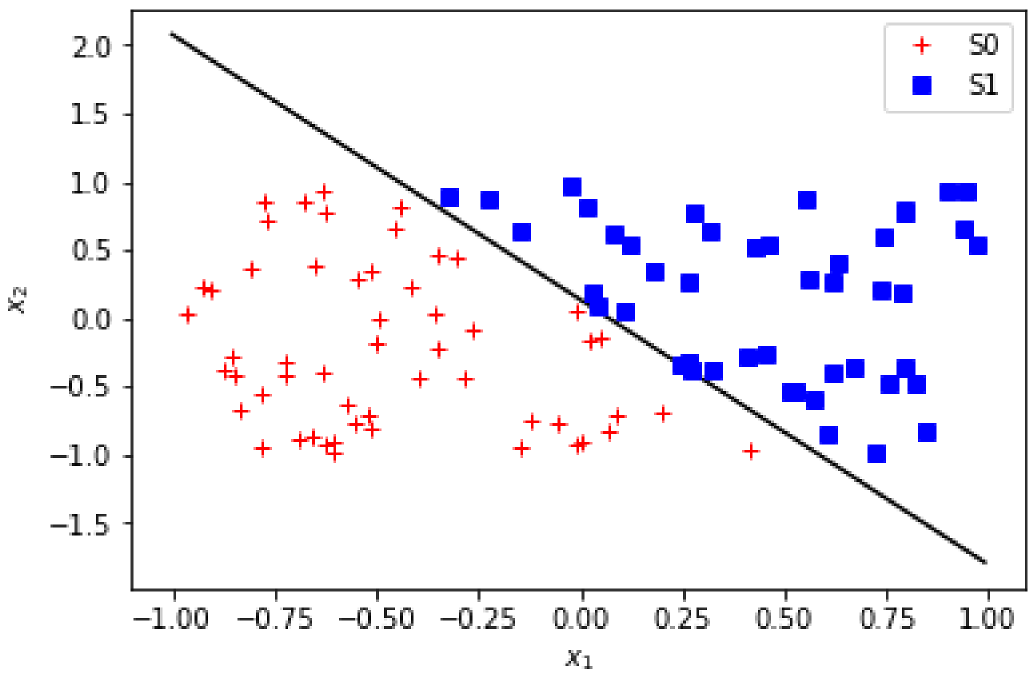
(g) Let S1 S denote the collection of all x = [x1 x2] S satisfying [1 x1 x2][w0 w1, w2]T 0.

(h) Let S0 S denote the collection of all x = [x1 x2] S satisfying [1 x1 x2][w0 w1, w2]T < 0.

```
In [6]: # (g)(h)
        Sone = np.hstack((np.ones((100,1)),S))
        temp = Sone.dot(w)>=0
        Sone_labeled = np.hstack((Sone,temp.reshape((100,1))))
        S1 = Sone_labeled[Sone_labeled[:,3]==1,1:3]
        S0 = Sone_labeled[Sone_labeled[:,3]!=1,1:3]
```

(i) In one plot, show the line w0 + w1x1 + w2x2 = 0, with x1 being the "x-axis" and x2 being the "y-axis." In the same plot, show all the points in S1 and all the points in S0. Use different symbols for S0 and S1. Indicate which points belong to which class. Given the ω, x_1 serves as the x-axis and x_2 serves as the y-aixs, we should have the boundary line like: $y = -\frac{\omega_1}{\omega_2}x - \frac{\omega_0}{\omega_2}$.

In [36]:
```
# (i)
x = np.arange(-1,1,0.01)
plt.plot(x,-x*w[1]/w[2]-w[0]/w[2],'k-')
ps0 = plt.plot(S0[:,0],S0[:,1],'r+',label = 'S0')
ps1 = plt.plot(S1[:,0],S1[:,1],'bs',label = 'S1')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend()
plt.show()
```



(j) Use the perceptron training algorithm to find the weights that can separate the two classes S0 and S1.

i. Use the training parameter = 1.

In [231]:
```
## (j)
# [i]
np.random.seed(44)
eta = 1
```

ii. Pick w0 , w1 , w2 independently and uniformly at random on [1, 1]. Write them in your report.

```
In [232]: # [ii]
          wprime0 = np.zeros(3)
          wprime0[0] = np.random.uniform(-1,1)
          wprime0[1] = np.random.uniform(-1,1)
          wprime0[2] = np.random.uniform(-1,1)
          wprime0

Out[232]: array([ 0.6696843 , -0.79040779,  0.48928096])
```

iii. Record the number of misclassifications if we use the weights [w0 , w1 , w2].

```
In [233]: # [iii]
          S0one = np.hstack((np.ones((np.shape(S0)[0],1)),S0))
          S1one = np.hstack((np.ones((np.shape(S1)[0],1)),S1))
          count0 = np.sum(S0one.dot(wprime0)>=0) + np.sum(S1one.dot(wprime0)<0)
          count0

Out[233]: 63
```

iv.After one epoch of the perceptron training algorithm, you will find a new set of weights [w0, w1, w2].

```
In [234]: # [iv]
          wtemp = wprime0
          for i in range(0,100):
              wtemp += eta * Sone_labeled[i,0:3]* (Sone_labeled[i,3]-np.heaviside(Sone_labeled[i
          # result from epoch 1
          wprime1 = wtemp
          wprime1

Out[234]: array([-0.3303157 ,  3.29102147,  0.5143063 ])
```

v. Record the number of misclassifications if we use the weights [w0, w1, w2].

```
In [235]: # [v]
          count1 = np.sum(S0one.dot(wprime1)>=0) + np.sum(S1one.dot(wprime1)<0)
          count1

Out[235]: 6
```

vi. Do another epoch of the perceptron training algorithm, find a new set of weights, record the number of misclassifications, and so on, until convergence.

```
In [236]: # [vi]
          # epoch 2
          wtemp = wprime1
```

```
    for i in range(0,100):
        wtemp += eta * Sone_labeled[i,0:3] * (Sone_labeled[i,3]-np.heaviside(Sone_labeled[
    # result from epoch 2
    wprime2 = wtemp
    wprime2
```

```
Out[236]: array([-0.3303157 ,  3.81725553,  1.77925678])
```

```
In [237]: # record the  number of misclassifications after epoch 2
          count2 = np.sum(S0one.dot(wprime2)>=0) + np.sum(S1one.dot(wprime2)<0)
          count2
```

```
Out[237]: 1
```

```
In [238]: # write a loop to run PTA until converge(NO more misclassification)
          itern = 7
          count = np.empty((1,itern+1))
          count[:,0] = count0
          wall = np.empty((3,itern+1))
          wall[:,0] = wprime0.reshape((1,3))

          for k in range(0,itern):
              wtemp = wall[:,k]
              for i in range(0,100):
                  wtemp += eta * Sone_labeled[i,0:3] * (Sone_labeled[i,3]-np.heaviside(Sone_labe

              counttemp = np.sum(S0one.dot(wtemp)>=0) + np.sum(S1one.dot(wtemp)<0)
              wall[:,k+1] = wtemp
              count[:,k+1] = counttemp
```

vii. Write down the final weights you obtain in your report. How does these weights compare to the "optimal" weights [w0, w1, w2]? We know that the creterion of convergence in this situation is the count of misclassification becomes 0. So we show the

```
In [239]: count_eta_1 = count
          count_eta_1
```

```
Out[239]: array([[ 63.,   0.,   0.,   0.,   0.,   0.,   0.,   0.]])
```

Thus, we just get the latest epoch's weights as our final weights.

```
In [240]: # final weights
          w_final_1 = wall[:,-1]
          w_final_1
```

```
Out[240]: array([-0.3303157 ,  4.14741448,  2.01640985])
```

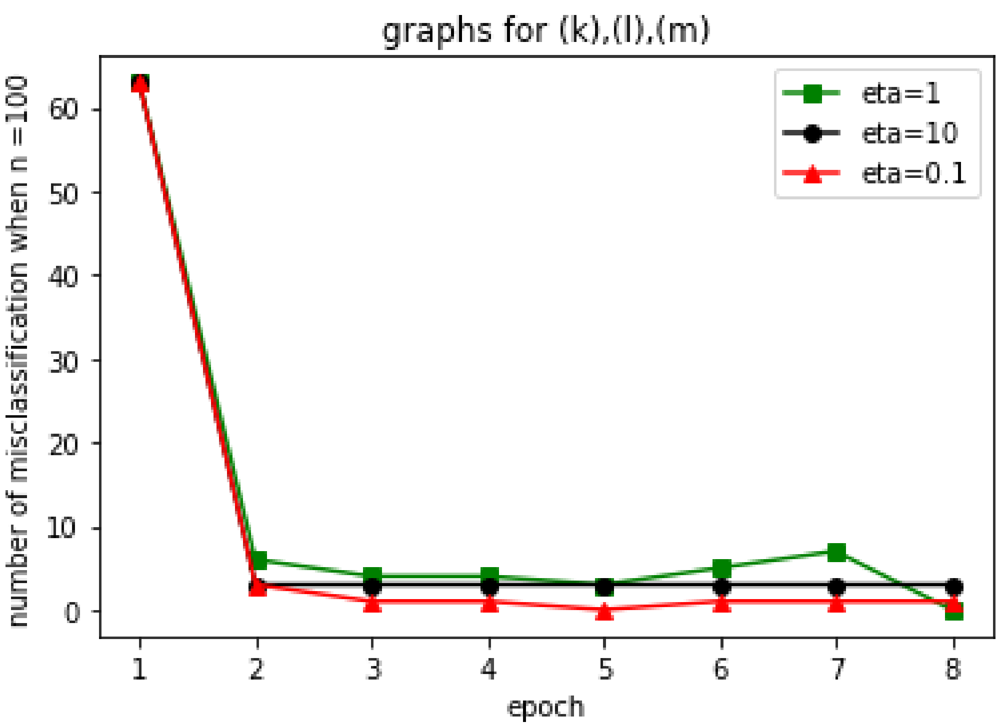
```
In [241]: # compare the final weights with the optimal weights
          w_final_1 == w
```

```
Out[241]: array([False, False, False], dtype=bool)
```

Clearly, the final weights are different to the optimal weights. But they could be the same if we manipulate the random seed. After all, the two classes of points can be linearly separated in more than one way, so there is not a unique weight in terms of merely no misclassific

(k) Regarding the previous step, draw a graph that shows the epoch number vs the number of mis- classifications.

```
In [250]: x = np.arange(1,(itern+2),1)
          plt.plot(x,count_eta_1[0],'gs-',label = 'eta=1')
          plt.plot(x,count_eta_10[0],'ko-',label = 'eta=10')
          plt.plot(x,count_eta_01[0],'r^-',label = 'eta=0.1')
          plt.xlabel('epoch')
          plt.ylabel('number of misclassification when n =100')
          plt.title('graphs for (k),(l),(m)')
          plt.legend()
          plt.show()
```



(l) Repeat the same experiment with = 10. Do not change w0, w1, w2, S, w0 , w1 , w2 . As in the case = 1, draw a graph that shows the epoch number vs the number of misclassifications.

```
In [242]: # write a loop to run PTA until converge(NO more misclassification)
          eta = 10
```

```
itern = 7
count = np.empty((1,itern+1))
count[:,0] = count0
wall = np.empty((3,itern+1))
wall[:,0] = wprime0.reshape((1,3))

for k in range(0,itern):
    wtemp = wall[:,k]
    for i in range(0,100):
        wtemp += eta * Sone_labeled[i,0:3] * (Sone_labeled[i,3]-np.heaviside(Sone_labe

    counttemp = np.sum(S0one.dot(wtemp)>=0) + np.sum(S1one.dot(wtemp)<0)
    wall[:,k+1] = wtemp
    count[:,k+1] = counttemp

# see the mis-classified count
count_eta_10 = count
count_eta_10
```

Out[242]:
```
array([[ 63.,   2.,   1.,   0.,   0.,   0.,   0.,   0.]])
```

In [243]:
```
# get the final weights
w_final_10 = wall[:,-1]
w_final_10
```

Out[243]:
```
array([ -0.3303157 ,  26.4552763 ,  13.31221826])
```

(m) Repeat the same experiment with = 0.1. Do not change w0, w1, w2, S, w0 , w1 , w2 . As in the case = 1, draw a graph that shows the epoch number vs the number of misclassifications.

In [244]:
```
# write a loop to run PTA until converge(NO more misclassification)
eta = 0.1
itern = 7
count = np.empty((1,itern+1))
count[:,0] = count0
wall = np.empty((3,itern+1))
wall[:,0] = wprime0.reshape((1,3))

for k in range(0,itern):
    wtemp = wall[:,k]
    for i in range(0,100):
        wtemp += eta * Sone_labeled[i,0:3] * (Sone_labeled[i,3]-np.heaviside(Sone_labe

    counttemp = np.sum(S0one.dot(wtemp)>=0) + np.sum(S1one.dot(wtemp)<0)
    wall[:,k+1] = wtemp
    count[:,k+1] = counttemp

# see the mis-classified count
```

```
count_eta_01 = count
count_eta_01
```

```
Out[244]: array([[ 63.,   0.,   0.,   0.,   0.,   0.,   0.,   0.]])
```

(n) Comment on how the changes in effect the number of epochs needed until convergence. Given the random seed, different choices between the learning rates do not post any practical differences in the coverging speed. However, in practice, if we are working with more complicated networks, choosing a not too large or too small is very important. If the learning rate is too high, we might have a hard time of finding a converging point. Whilst, if the learning rate is set too low, we are likely to be "trapped" in local minima and have a extreme long converging process.

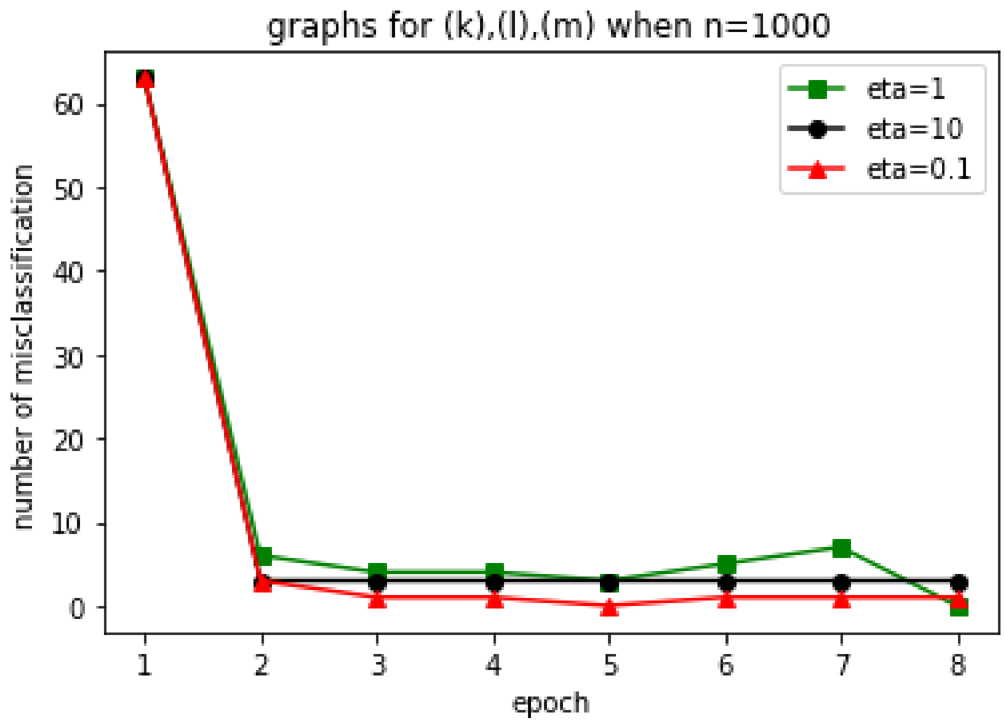
(o) Comment on whether we would get the exact same results (in terms of the effects of on training performance) if we had started with different w0, w1, w2, S, w0 , w1 , w2 . When the random seed changes, which flips how the weights and data are initilized, dramatic changes on the final weights choices, converging speed across different ηs should be expected. This is sensible given there exists infinit many of weights that can get the classification job done.

(p) Do the same experiments with n = 1000 samples. Comment on the differences compared to n = 100.

```
In [249]: np.random.seed(42)
          S = np.random.uniform(-1,1,(1000, 2))
          Sone = np.hstack((np.ones((1000,1)),S))
          temp = Sone.dot(w)>=0
          Sone_labeled = np.hstack((Sone,temp.reshape((1000,1))))
          S1 = Sone_labeled[Sone_labeled[:,3]==1,1:3]
          S0 = Sone_labeled[Sone_labeled[:,3]!=1,1:3]
          np.random.seed(44)
          wprime0 = np.zeros(3)
          wprime0[0] = np.random.uniform(-1,1)
          wprime0[1] = np.random.uniform(-1,1)
          wprime0[2] = np.random.uniform(-1,1)
          # write a loop to run PTA until converge(NO more misclassification)
          eta = 1
          itern = 7
          count = np.empty((1,itern+1))
          count[:,0] = count0
          wall = np.empty((3,itern+1))
          wall[:,0] = wprime0.reshape((1,3))

          for k in range(0,itern):
              wtemp = wall[:,k]
              for i in range(0,100):
                  wtemp += eta * Sone_labeled[i,0:3] * (Sone_labeled[i,3]-np.heaviside(Sone_labe

              counttemp = np.sum(S0one.dot(wtemp)>=0) + np.sum(S1one.dot(wtemp)<0)
              wall[:,k+1] = wtemp
```

```
    count[:,k+1] = counttemp
count_eta_1 = count
w_final_1 = wall[:,-1]
# write a loop to run PTA until converge(NO more misclassification)
eta = 10
itern = 7
count = np.empty((1,itern+1))
count[:,0] = count0
wall = np.empty((3,itern+1))
wall[:,0] = wprime0.reshape((1,3))

for k in range(0,itern):
    wtemp = wall[:,k]
    for i in range(0,100):
        wtemp += eta * Sone_labeled[i,0:3] * (Sone_labeled[i,3]-np.heaviside(Sone_labe

    counttemp = np.sum(S0one.dot(wtemp)>=0) + np.sum(S1one.dot(wtemp)<0)
    wall[:,k+1] = wtemp
    count[:,k+1] = counttemp
# see the mis-classified count
count_eta_10 = count
# get the final weights
w_final_10 = wall[:,-1]
# write a loop to run PTA until converge(NO more misclassification)
eta = 0.1
itern = 7
count = np.empty((1,itern+1))
count[:,0] = count0
wall = np.empty((3,itern+1))
wall[:,0] = wprime0.reshape((1,3))

for k in range(0,itern):
    wtemp = wall[:,k]
    for i in range(0,100):
        wtemp += eta * Sone_labeled[i,0:3] * (Sone_labeled[i,3]-np.heaviside(Sone_labe

    counttemp = np.sum(S0one.dot(wtemp)>=0) + np.sum(S1one.dot(wtemp)<0)
    wall[:,k+1] = wtemp
    count[:,k+1] = counttemp

# see the mis-classified count
count_eta_01 = count
w_final_01 = wall[:,-1]
x = np.arange(1,(itern+2),1)
plt.plot(x,count_eta_1[0],'gs-',label = 'eta=1')
plt.plot(x,count_eta_10[0],'ko-',label = 'eta=10')
plt.plot(x,count_eta_01[0],'r^-',label = 'eta=0.1')
plt.xlabel('epoch')
```

```
plt.ylabel('number of misclassification')
plt.title('graphs for (k),(l),(m) when n=1000')
plt.legend()
plt.show()
```



By setting the same random seeds, we can gurantee that the weight vectors are initialized exactly the same as they were in n=100 case. From the above plot, we can clearly observe the converging speed slows down due to the increased sample size. And different η values causes more tangible disparities in their convergence and final weight values. As a result, we generally need more epochs to find the converging point to deal with the increased sample size.