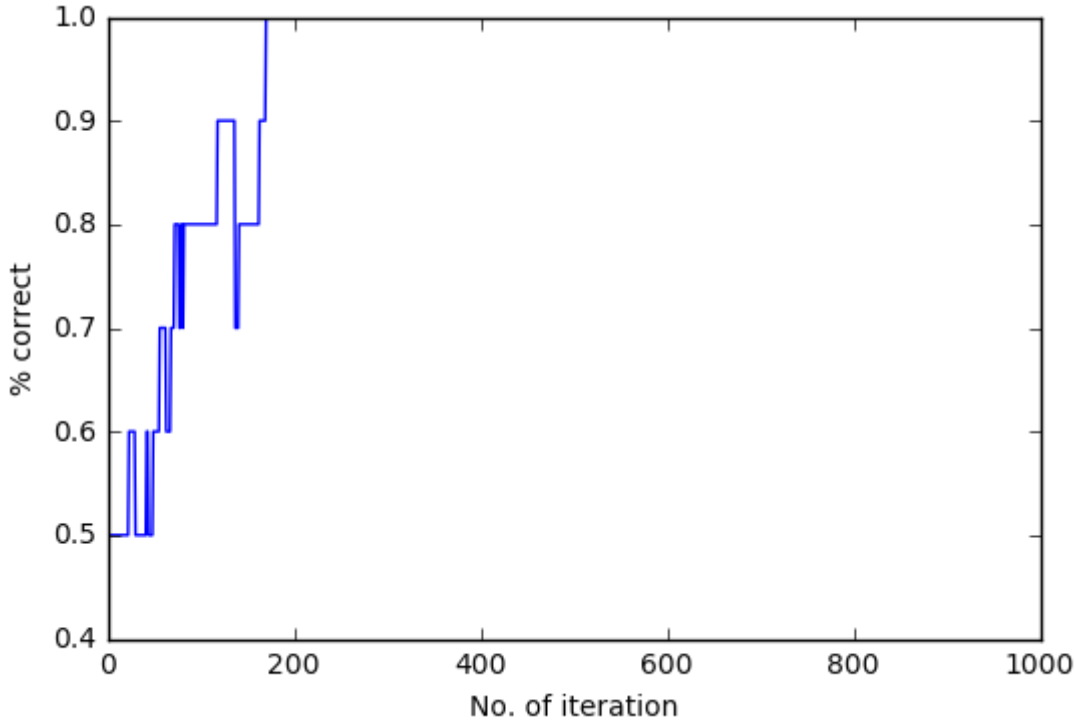
Qn1 Learning to classify random patterns

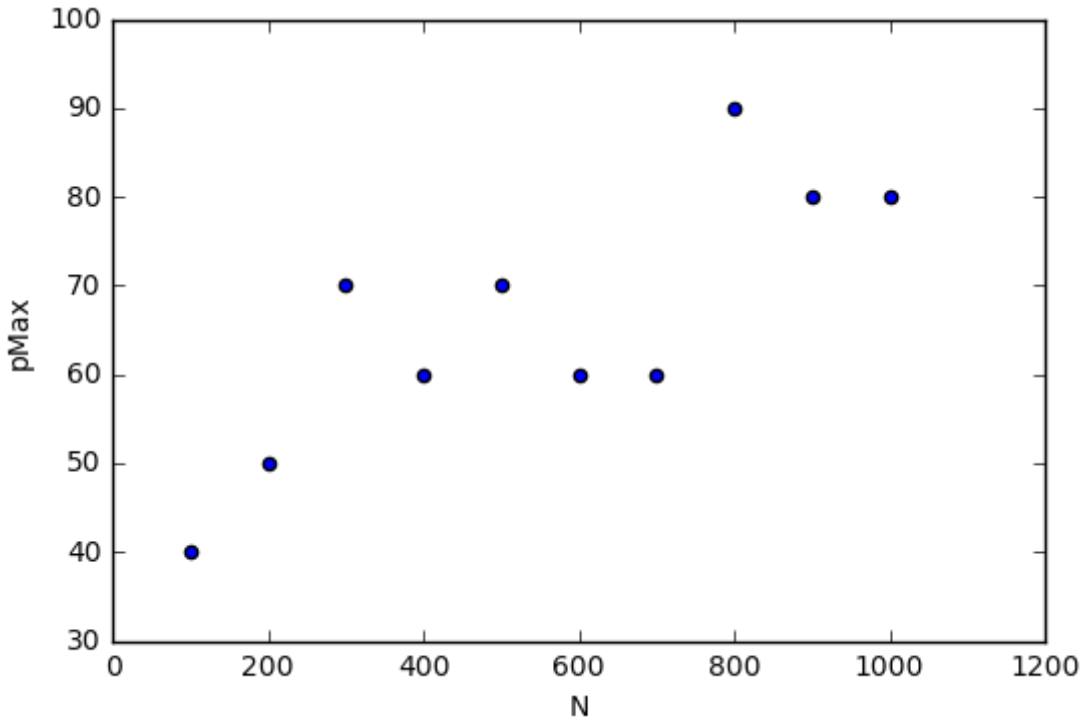
In [242]:
```
def perceptron(p,R,Sai,w,theta):
    # generated class
    Ro = np.sign(np.dot(Sai,w) - theta)
    alpha = 0.01 # learning rate
    score = (Ro == R) # performance
    correct = sum(score)/p # percentage correct

    # randomly pick a pattern to test & update score if Ro != R
    pick = np.random.randint(0,p, size=1)
    if score[pick] == False:
        w = w+ (alpha*R[pick]*Sai[pick]).flatten()
        theta = theta - alpha*R[pick]
    return(w,theta,correct)

def PN(p,N):
    # desired class
    R = np.sign(np.random.randn(p)-0.5)
    # patterns
    Sai =np.sign(np.random.randn(p*N)-0.5).reshape(p,N)
    #weights for each neuron
    w = np.random.randn(N)
    theta = np.random.randn(1)
    C = np.zeros(1000)
    for n in np.arange(0,1000):
        w,theta,correct = perceptron(p,R,Sai,w,theta)
        C[n] = correct
    return(C)

N = 100
p = 10
C=PN(p,N)

plt.plot(C)
plt.xlabel('No. of iteration')
plt.ylabel('% correct')
plt.show()
```



Plot pmax vs N and show that it is approximately linear.

Answer: as shown in the plot below Pmax vs N is approximately linear.

In [243]:
```
Nn = np.linspace(100,1000,10)
Pmax = np.zeros(10)
for i,n in enumerate(Nn):
    n = n.astype(int)
    P = np.arange(10, 2*n ,10)
    for p in P:
        C=PN(p,n)
        if sum(C==1) ==0:
            Pmax[i]=  p
            break
plt.scatter(Nn,Pmax)
plt.xlabel('N')
plt.ylabel('pMax')
plt.show()
```

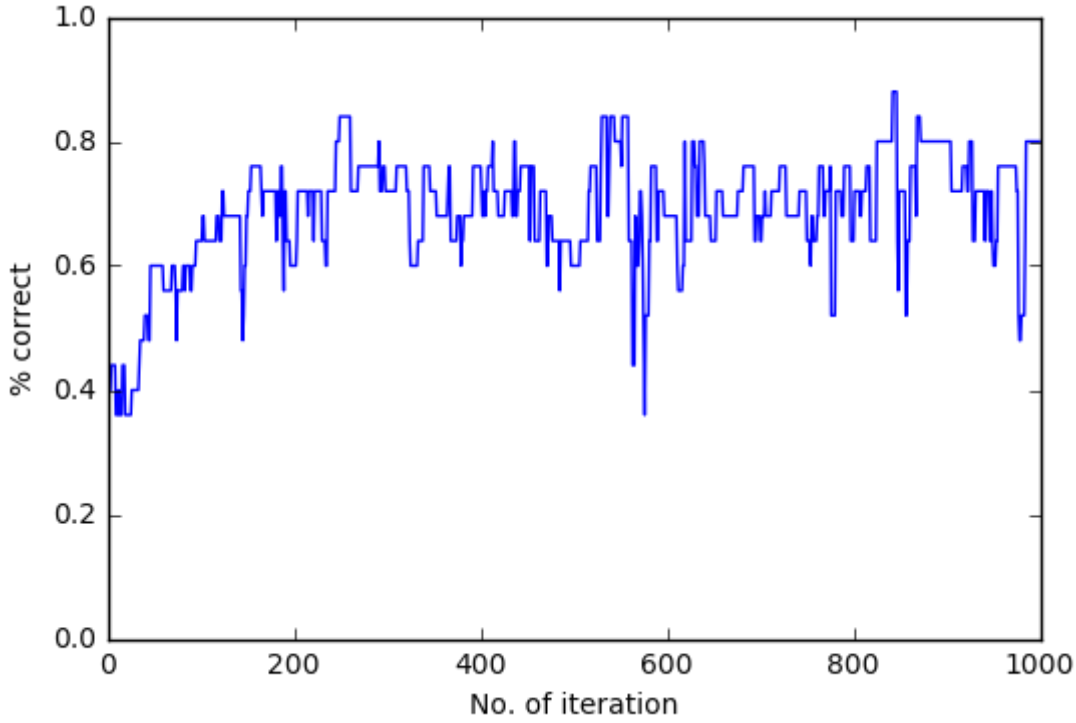


Qn2 Non-linearly separable patterns

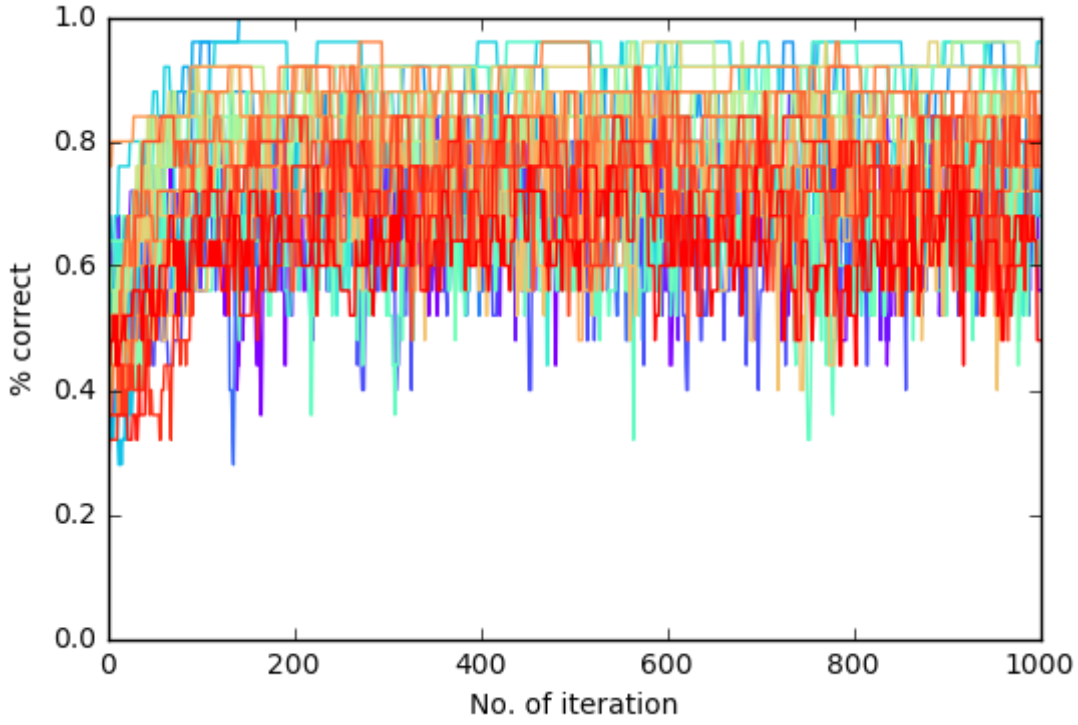
Show that even when N is very large (it can be arbitrarily large), the fraction of errors typically never goes to zero (repeat the experiment many times for different choices of random patterns). Show that the situation gets even worse when p increases. at non-linear separability is not a pathological situations, but it is what typically happens in ”realistic” situations.

In [244]:

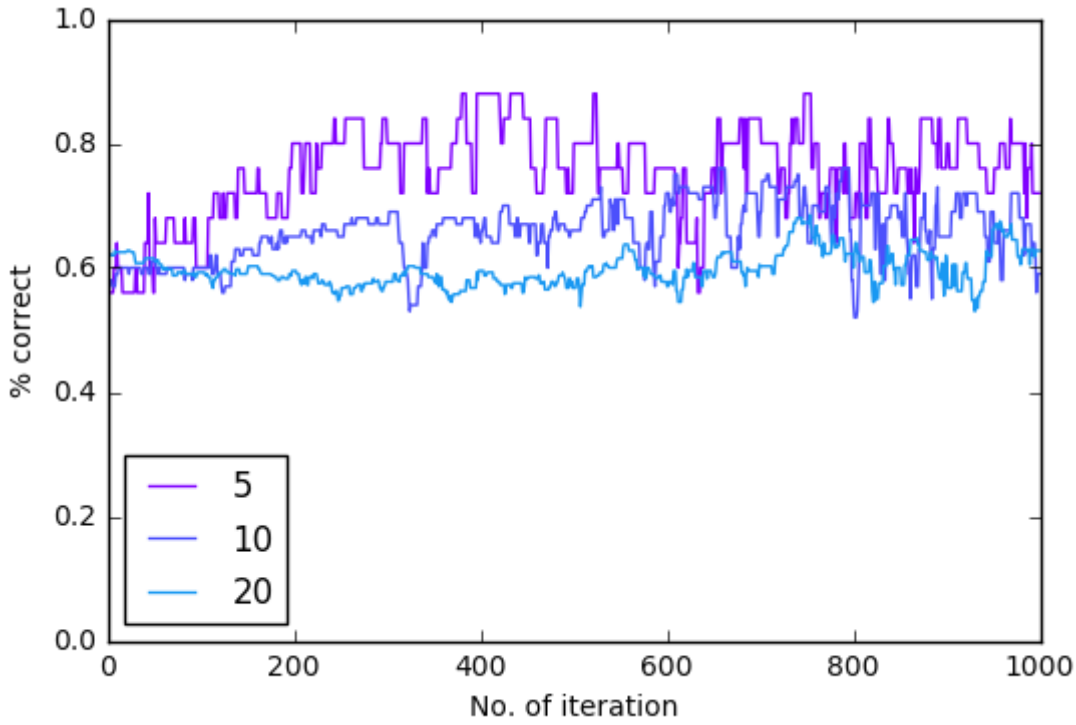
```
import itertools

def perceptrons(p,R,Sai,w,theta):
    # generated class
    Ro = np.sign(np.dot(Sai,w) - theta)
    alpha = 0.01 # learning rate
    score = (Ro == R) # performance
    correct = sum(score)/(p**2) # percentage correct

    # randomly pick a pattern to test & update score if Ro != R
    pick = np.random.randint(0,p**2, size=1)
    if score[pick] == False:
        w = w+ (alpha*R[pick]*Sai[pick]).flatten()
        theta = theta - alpha*R[pick]
    return(w,theta,correct)

def PNs(p,N):
    # desired class
    R = np.sign(np.random.randn(p**2)-0.5)
    # patterns
    Sai =np.sign(np.random.randn(p*N)-0.5).reshape(p,N)
    Sai= np.array(list(itertools.product(Sai, Sai)))
    Sai = Sai.flatten().reshape(p**2,2*N)
    #weights for each neuron
    w = np.random.randn(2*N)
    theta = np.random.randn(1)
    C = np.zeros(1000)
    for n in np.arange(0,1000):
        w,theta,correct = perceptrons(p,R,Sai,w,theta)
        C[n] = correct
    return(C)

N = 100
p = 5
C=PNs(p,N)

plt.plot(C)
plt.xlabel('No. of iteration')
plt.ylabel('% correct')
plt.ylim(0,1)
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
% correct
0
200
400
600
800
1000
No. of iteration

In [246]:
```
# large N
# as shown in the plot, having large N does not help
import matplotlib as mpl

# simulate very large N by repeating the experiment for N = 200 for many times
Nl= (np.ones(30)*200).astype(int)
p =5
#Nl = np.arange(100,1000,200)
colors = mpl.cm.rainbow(np.linspace(0, 1, len(Nl)))
fig, ax = plt.subplots()
for color, N in zip(colors, Nl):
    C=PNs(p,N)
    ax.plot(C, color = color)
plt.xlabel('No. of iteration')
plt.ylabel('% correct')
plt.ylim(0,1)
#plt.legend(Nl, loc ='best')
plt.show()
```

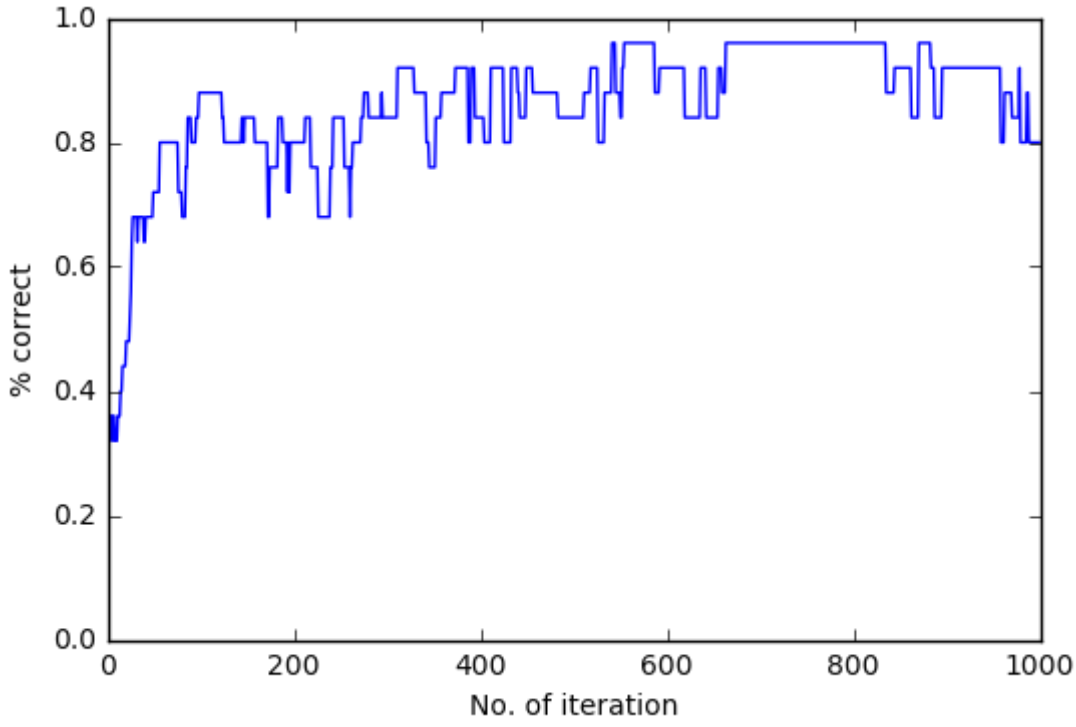


In [247]:
```
# larger p
# as shown in the plot, having larger p makes it worse

N = 100
Pl = [5,10,20]
colors = mpl.cm.rainbow(np.linspace(0, 3, len(Nl)))
fig, ax = plt.subplots()
for color, p in zip(colors, Pl):
    C=PNs(p,N)
    ax.plot(C, color = color)
plt.xlabel('No. of iteration')
plt.ylabel('% correct')
plt.ylim(0,1)
plt.legend(Pl, loc ='best')
plt.show()
```

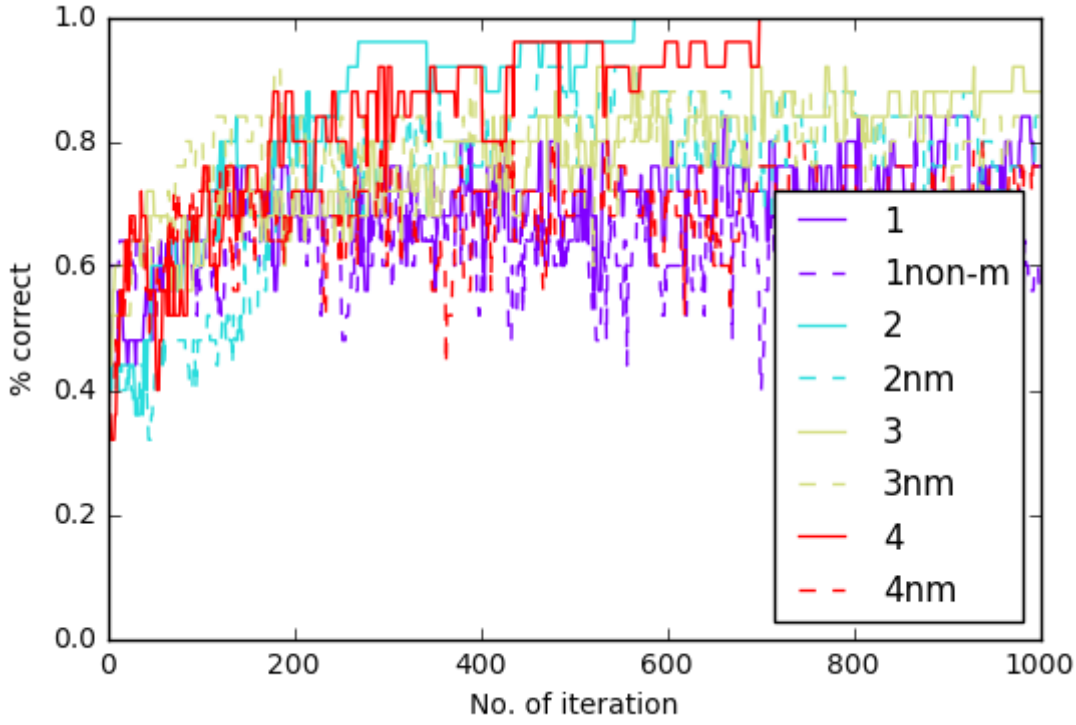


Qn3: Solving the problem of non linear separability with mixed selectivity neurons.

In [248]:
```
def PNm(p,N,beta):
    Mn = beta*(p**2)
    # desired class
    R = np.sign(np.random.randn(p**2)-0.5)
    # patterns
    Sai =np.sign(np.random.randn(p*N)-0.5).reshape(p,N)
    Sai= np.array(list(itertools.product(Sai, Sai)))
    Sai = Sai.flatten().reshape(p**2,2*N)
    S = np.sign(((Sai[:,np.arange(0,Mn)] == 1) & (Sai[:,np.arange(N,N+Mn)] ==
1)).astype(int)-0.5)
    Sai = np.hstack((S, Sai))
    #weights for each neuron
    w = np.random.randn(Mn+2*N)
    theta = np.random.randn(1)
    C = np.zeros(1000)
    for n in np.arange(0,1000):
        w,theta,correct = perceptrons(p,R,Sai,w,theta)
        C[n] = correct
    return(C)

N = 100
p = 5
beta = 1
C=PNm(p,N,1)

plt.plot(C)
plt.xlabel('No. of iteration')
plt.ylabel('% correct')
plt.ylim(0,1)
plt.show()
```

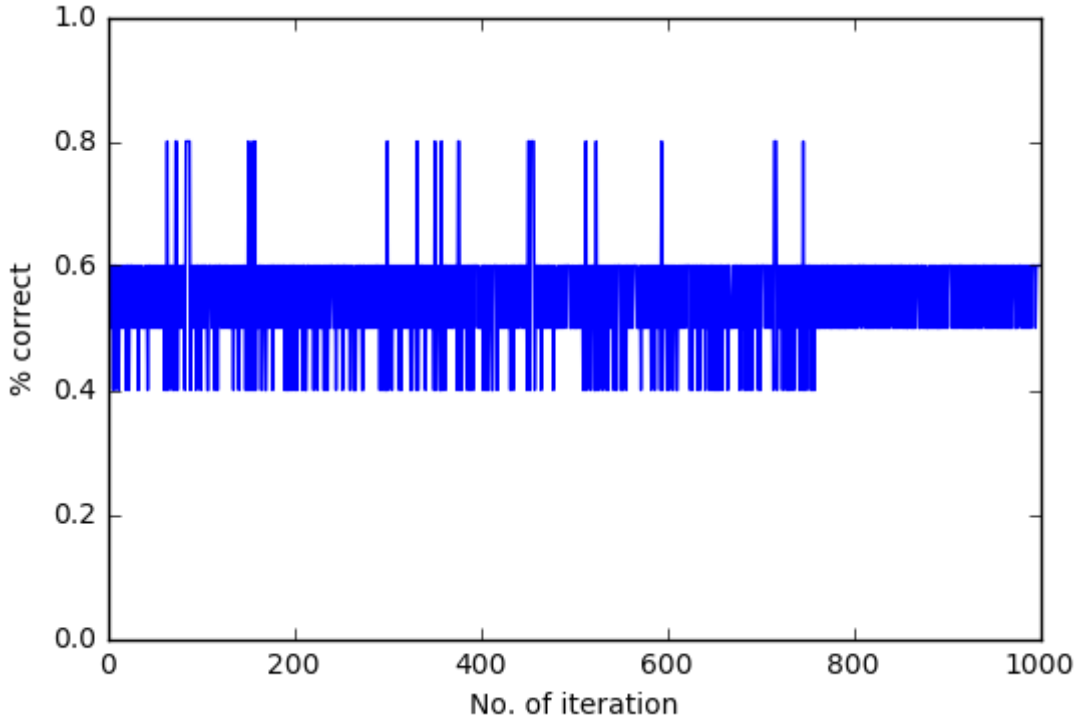


In [249]:
```
#Show that the error actually goes to zero as M grows
# and that it is much smaller than in the case in which the M mixed selectivit
y neurons are replaced with non-mixing neurons.

# As shown on the plot, larger M gets 100% correct, and mixed-selectivity neur
ons perform better than non-mixing neurons.

# a function that replated M cells with non-mixing cells
def PNnonm(p,N,beta):
    Mn = beta*(p**2)
    # desired class
    R = np.sign(np.random.randn(p**2)-0.5)
    # patterns
    Sai =np.sign(np.random.randn(p*N)-0.5).reshape(p,N)
    Sai= np.array(list(itertools.product(Sai, Sai)))
    Sai = Sai.flatten().reshape(p**2,2*N)
    S = Sai[:,np.arange(0,Mn)]
    Sai = np.hstack((S, Sai))
    #weights for each neuron
    w = np.random.randn(Mn+2*N)
    theta = np.random.randn(1)
    C = np.zeros(1000)
    for n in np.arange(0,1000):
        w,theta,correct = perceptrons(p,R,Sai,w,theta)
        C[n] = correct
    return(C)

N = 100
p = 5
Beta = [1,2,3,4]
legend = [1,'1non-m',2,'2nm',3,'3nm',4,'4nm']
colors = mpl.cm.rainbow(np.linspace(0, 1, len(Beta)))
fig, ax = plt.subplots()
for color, b in zip(colors, Beta):
    C=PNm(p,N,b)
    Cnonm=PNnonm(p,N,b)
    ax.plot(C, color = color)
    ax.plot(Cnonm, color = color, ls = '--')
plt.xlabel('No. of iteration')
plt.ylabel('% correct')
plt.ylim(0,1)
plt.legend(legend, loc ='best')
plt.show()
```

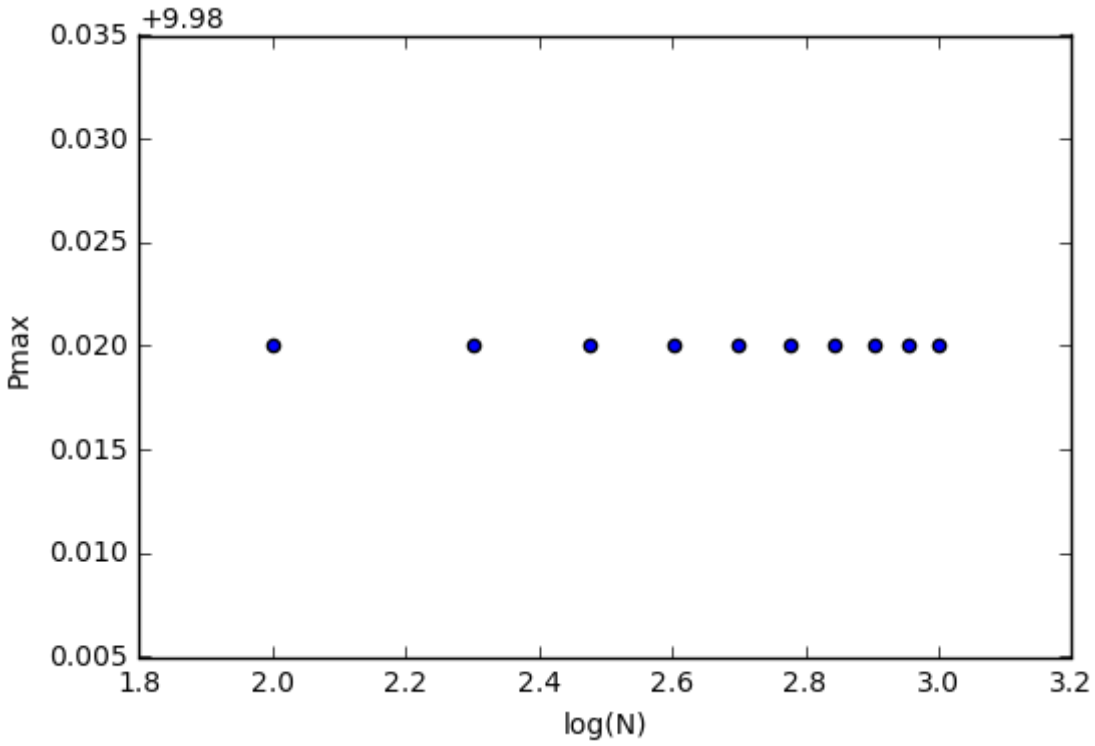


Qn4: Learning with binary synapses

In [384]:
```
def perceptronb(q,p,R,Sai,w,theta):
    # generated class
    Ro = np.sign(np.dot(Sai,w) - theta)
    alpha = 0.03 # learning rate
    score = (Ro == R) # performance
    correct = sum(score)/p # percentage correct
    # randomly pick a pattern to test & update score if Ro != R
    for pick in np.arange(0,p):
        #pick = 4 #np.random.randint(0,p, size=1)
        if score[pick] == False:
            prob = np.random.random(1)
            if prob <= q:
                w = np.sign(((R[pick]*Sai[pick]).flatten() ==1).astype(int)-0.
5)
            theta = theta - alpha*R[pick]
    return(w,theta,correct)

def PNb(p,N,q):
    # desired class
    R = np.sign(np.random.randn(p)-0.5)
    # patterns
    Sai =np.sign(np.random.randn(p*N)-0.5).reshape(p,N)
    #weights for each neuron
    w = np.sign(np.random.randn(N)-0.5)
    theta = np.random.randn(1)
    C = np.zeros(1000)
    for n in np.arange(0,1000):
        w,theta,correct = perceptronb(q,p,R,Sai,w,theta)
        C[n] = correct
    return(C)


q =  0.3
N = 100
p = 10
C=PNb(p,N,q)

plt.plot(C)
plt.xlabel('No. of iteration')
plt.ylabel('% correct')
plt.ylim(0,1)
plt.show()
```

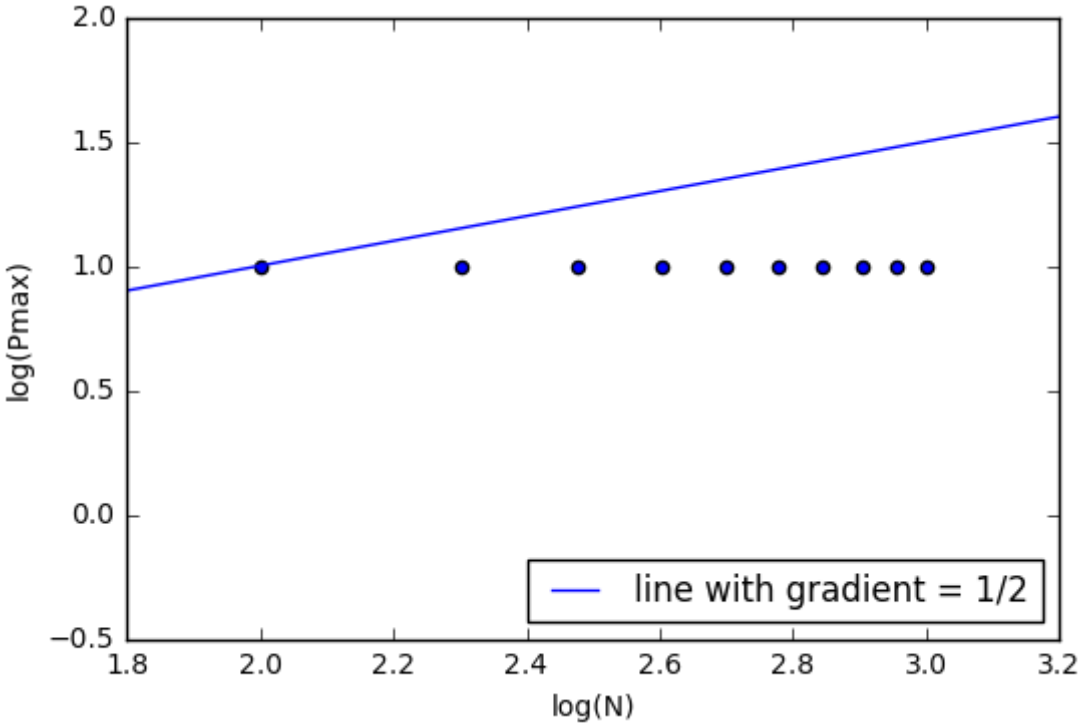

1.0
0.8
0.6
0.4
0.2
0.0
% correct
0
200
400
600
800
1000
No. of iteration

In [386]:
```
# finding pmax for binary synapses
Nn = np.linspace(100,1000,10)
Pmax = np.zeros(10)
q =0.9
for i,n in enumerate(Nn):
    n = n.astype(int)
    P = np.arange(10, 2*n ,5)
    for p in P:
        C=PNb(p,n,q)
        if sum(C==1) ==0:
            Pmax[i]= p
            break
plt.scatter(np.log10(Nn),Pmax)
plt.ylabel('Pmax')
plt.xlabel('log(N)')
plt.show()
```



Qn 2. When q = a/√N, with a = 10, show that the maximum number of random inputs scales as √N: plot pmax vs N on a log-log scale and show that it is a line with a slope 1/2

In [385]:
```
Nn = np.linspace(100,1000,10)
Pmax = np.zeros(10)
for i,n in enumerate(Nn):
    q = 10/np.sqrt(N)
    n = n.astype(int)
    P = np.arange(10, 2*n ,10)
    for p in P:
        C=PNb(p,n,q)
        if sum(C==1) ==0:
            Pmax[i]= p
            break
plt.scatter(np.log10(Nn),np.log10(Pmax))
plt.ylabel('log(Pmax)')
plt.xlabel('log(N)')
plt.plot(np.linspace(0,3.2,100), 0.5*np.linspace(0,3.2,100))
plt.xlim(1.8,3.2)
plt.legend(['line with gradient = 1/2'],loc='lower right')
plt.show()
```



In [ ]:

In [ ]: