Universität Bonn, AIS - Robot Learning, SS17

Assignment 01

Bhargavi Mahesh - 9029542

Mohammad Wasil - 9029719

Gabriela Cortés - 9028191

In [2]:

```
import numpy as np
```

Exercise 1.1

Given is six-armed bandit, as introduced in the lecture.

The first arm shall sample its reward uniformly from the interval [2, 3).

The second arm shall sample its reward uniformly from [-1, 5).

The third arm shall sample its reward uniformly from the interval [1, 5).

The fourth arm shall sample its reward uniformly from [-2, 4).

The fifth arm shall sample its reward uniformly from [0, 3).

The sixth arm shall sample its reward uniformly from [2, 6).

What is the expected reward when actions are chosen uniformly?

Answer

$E(R) = \sum_{t=1}^{T} \mu_{a_{i,t}}$ where $E(R)$ is the expected reward, $\mu_{a_{i,t}}$ is the mean value of the rewards of $a_{i,t}$ and $a_{i,t} = a_1, a_2, a_3, a_4, a_5, a_6$ at time step t.

When t tends to infinite and selecting the arms in a uniform distribution, the expected reward will be the sum of the mean of the reward probability distribution of each arm.

Total numerical expected reward is 14.

Exercise 1.2

Implement the six-armed bandit from 1.1) and compute the sample average reward for 100 uniformly chosen actions.

Compare this to your expectation from 1.1).

In [135]:

```
k= 6                 #number of arms
pulls = 100         #number of pulls
Q_a= np.zeros(k)     # estimated mean reward of actions a_i at time step t
A = np.zeros(pulls)  # action selected at time step t
R = np.zeros(pulls)  #reward at time step t
C_a= np.zeros(k)     #counts of actions

for j in range(pulls):

    #action selection
    q= np.int_(np.random.uniform(0,k))

    #save the arm selected
    A[j]=q

    #count of arm selected
    C_a[q]+=1

    #actions rewards
    r_a =[np.random.uniform(2,3),np.random.uniform(-1,5),np.random.uniform(1,5),np.
          np.random.uniform(0,3),np.random.uniform(2,6)]
    R= r_a[q]

    Q_a[q]+=(R-Q_a[q])/C_a[q]

best_arm = np.argmax(Q_a)
print "Arms rewards 0-5 (1-6): ", best_arm
print "Arms rewards 0-5 (1-6): ",Q_a
print sum(Q_a)
```

```
Arms rewards 0-5 (1-6):  5
Arms rewards 0-5 (1-6):  [ 2.5576817   1.63445476  2.73264033  0.45473
448  1.29620408  4.2054643 ]
12.8811796529
```

With only 100 pulls, the accumulated reward is not exactly the same as previously calculated expected reward.

Exercise 1.3

Initialize $Q(a_i) = 0$ and chose 1000 actions according to an ε-greedy selection strategy (ε=0.1). Update your action values by computing the sample average reward of each action recursively.

For every 100 actions show the percentage of choosing arm 1, arm 2, arm 3, arm 4, arm 5, and arm 6 as well as the resulting average reward.

In [40]:

```
#explore (select randomly between 1 and 6) if generated random nuber less than e
e = 0.1
#greedy choose the maximum reward if generated random number morthan 0.1
g = 0.9
Q_a = np.zeros(k)
C_a = np.zeros(k)
actions = 0
percentage = np.zeros(6)
index = 0
exp=0
hundred_count =0
for i in range(1000):
    actions = actions + 1
    randomNumber = np.random.uniform(0, 1)
    r_a = np.array([np.random.uniform(2,3),np.random.uniform(-1,5),np.random.unifor
                    np.random.uniform(-2,4),np.random.uniform(0,3),np.random.unifo

    #exploration 10%
    if (randomNumber <= e):
        index = np.random.randint(0,k)
    #exploitation 90%
    else:
        maxProb = Q_a[0]
        maxIndex = 0
        for i in range(6):
            if maxProb < Q_a[i]:
                maxProb = Q_a[i]
                index = i

    C_a[index] = C_a[index] + 1
    Q_a[index]+= (r_a[index]-Q_a[index])/C_a[index]


    if actions == 100:
        hundred_count+=1

        percentage = C_a/hundred_count

        print "\nPercentage of choosing arms 1-6:",percentage
        print "Rewards of arms 1-6:",Q_a
        actions = 0
```

```
Percentage of choosing arms 1-6: [ 12.   2.  38.  14.   6.  28.]
Rewards of arms 1-6: [ 2.51179078  1.24674965  3.30371367  0.80792609
0.79550605  3.956586  ]

Percentage of choosing arms 1-6: [  7.    3.   20.5   7.5   4.   58. ]
Rewards of arms 1-6: [ 2.51106493  1.26545341  3.24503731  0.62530825
0.75121144  4.0113156 ]

Percentage of choosing arms 1-6: [  5.           2.66666667  15.
     5.           3.          69.33333333]
Rewards of arms 1-6: [ 2.48447053  1.95249279  3.15128565  0.62530825
0.94748982  3.98524758]

Percentage of choosing arms 1-6: [  4.5    2.    12.25   4.25   3.
 74.  ]
Rewards of arms 1-6: [ 2.48438002  1.95249279  3.23810222  0.785281
```

```
 0.98355929  3.99280514]


Percentage of choosing arms 1-6: [  3.6   2.   10.2   3.8   2.8  77.6]
Rewards of arms 1-6: [ 2.48438002  1.79343741  3.19140084  0.77285804
0.97471393  3.99145824]

Percentage of choosing arms 1-6: [  3.16666667   1.83333333   8.666666
67   3.5          2.5         80.33333333]
Rewards of arms 1-6: [ 2.4652053   1.75158667  3.17092904  0.52358448
1.10151108  4.01701273]

Percentage of choosing arms 1-6: [  3.14285714   1.85714286   7.571428
57   3.28571429   2.42857143
  81.71428571]
Rewards of arms 1-6: [ 2.50430757  1.74002179  3.20217889  0.56120745
1.11405635  4.04453284]

Percentage of choosing arms 1-6: [  2.875   1.75    6.875   3.25    2.
5    82.75 ]
Rewards of arms 1-6: [ 2.51979792  1.6697368   3.1494463   0.76249561
1.16021849  4.04986286]

Percentage of choosing arms 1-6: [  2.77777778   1.77777778   6.333333
33   3.           2.22222222
  83.88888889]
Rewards of arms 1-6: [ 2.50330993  1.75443861  3.13350491  0.80919083
1.16021849  4.01942047]

Percentage of choosing arms 1-6: [  2.5   1.8   5.7   3.    2.   85. ]
Rewards of arms 1-6: [ 2.50330993  1.54356387  3.13350491  0.72710864
1.16021849  4.02298655]
```

Exercise 1.4

Redo the experiment, but after 500 steps, sample the rewards of the third arm uniformly from [6, 8).

Compare updating action values by computing the sample average reward of each action recursively (as done in 1.3) with using a constant learning rate α=0.01.

For every 100 actions show the percentage of choosing arm 1, arm 2, arm 3, arm 4, arm 5, and arm 6 as well as the resulting average reward.

In [80]:

```
e = 0.1
g = 0.9
Q_a = np.zeros(k)
C_a = np.zeros(k)
actions = 0
percentage = np.zeros(6)
index = 0
hundred_count =0
for i in range(1000):
    actions = actions + 1
    randomNumber = np.random.uniform(0, 1)
    if hundred_count < 5:
        r_a = np.array([np.random.uniform(2,3),np.random.uniform(-1,5),np.random.un
                    np.random.uniform(-2,4),np.random.uniform(0,3),np.random.unifo
    else:
        r_a = np.array([np.random.uniform(2,3),np.random.uniform(-1,5),np.random.un
                    np.random.uniform(-2,4),np.random.uniform(0,3),np.random.unifo
    #exploration 10%
    if (randomNumber <= e):
        index = np.random.randint(0,k)
    #exploitation 90%
    else:
        maxProb = Q_a[0]
        maxIndex = 0
        for i in range(6):
            if maxProb < Q_a[i]:
                maxProb = Q_a[i]
                index = i

    C_a[index] = C_a[index] + 1
    Q_a[index]+= (r_a[index]-Q_a[index])/C_a[index]


    if actions == 100:
        hundred_count+=1
        percentage = C_a/hundred_count

        print "\nPercentage of choosing arms 1-6:",percentage
        print "Rewards of arms 1-6:",Q_a
        actions = 0
```

```
Percentage of choosing arms 1-6: [  1.   0.   3.  10.   2.  84.]
Rewards of arms 1-6: [ 2.90168601  0.          2.19822334  0.47166912
0.74010356  3.99753355]

Percentage of choosing arms 1-6: [  2.    0.5   3.    5.    1.5  88. ]
Rewards of arms 1-6: [ 2.69743184  2.02622873  2.33765158  0.47166912
0.93330567  3.97644457]

Percentage of choosing arms 1-6: [  2.66666667   1.           2.333333
33   3.33333333   2.33333333
  88.33333333]
Rewards of arms 1-6: [ 2.63261605  2.40295663  2.5867324   0.47166912
0.95222358  3.96398137]

Percentage of choosing arms 1-6: [  2.25   1.     2.     2.75   1.75
 90.25]
Rewards of arms 1-6: [ 2.57307274  2.22905865  2.77322486  0.5749514
```

```
 0.95222358  3.98023012]


 Percentage of choosing arms 1-6: [  2.6   1.4   1.8   2.4   1.4  90.4]
 Rewards of arms 1-6: [ 2.50931866  2.46412199  2.66378672  0.72403537
 0.95222358  3.96844916]

 Percentage of choosing arms 1-6: [  2.5          1.16666667   1.833333
 33   2.16666667   1.5         90.83333333]
 Rewards of arms 1-6: [ 2.50156944  2.46412199  3.54568999  0.86627118
 1.0367765   3.96063223]

 Percentage of choosing arms 1-6: [  2.14285714   1.28571429   1.571428
 57   2.14285714   1.42857143
   91.42857143]
 Rewards of arms 1-6: [ 2.50156944  1.96679836  3.54568999  0.8081976
 1.22010608  3.92486041]

 Percentage of choosing arms 1-6: [  2.      1.375   7.75    2.25    1.
 375  85.25 ]
 Rewards of arms 1-6: [ 2.48455564  1.80936237  6.34054458  0.53088815
 1.11612271  3.93357914]

 Percentage of choosing arms 1-6: [  1.77777778   1.44444444  17.222222
 22   2.33333333   1.22222222  76.        ]
 Rewards of arms 1-6: [ 2.48455564  1.53089846  6.79294541  0.45668542
 1.11612271  3.92967281]

 Percentage of choosing arms 1-6: [  1.9   1.4  24.7   2.2   1.2  68.6]
 Rewards of arms 1-6: [ 2.45655362  1.50827613  6.86522928  0.51679922
 1.03342913  3.92672086]
```

In [109]:

```
#constant learning rate

e = 0.1
g = 0.9
alpha = 0.01
Q_a = np.zeros(k)
C_a = np.zeros(k)
actions = 0
percentage = np.zeros(6)
index = 0
hundred_count =0
for i in range(1000):
    actions = actions + 1
    randomNumber = np.random.uniform(0, 1)
    if hundred_count < 5:
        r_a = np.array([np.random.uniform(2,3),np.random.uniform(-1,5),np.random.un
                    np.random.uniform(-2,4),np.random.uniform(0,3),np.random.unifo
    else:
        r_a = np.array([np.random.uniform(2,3),np.random.uniform(-1,5),np.random.un
                    np.random.uniform(-2,4),np.random.uniform(0,3),np.random.unifo
    #exploration 10%
    if (randomNumber <= e):
        index = np.random.randint(0,k)
    #exploitation 90%
    else:
        maxProb = Q_a[0]
        maxIndex = 0
        for i in range(6):
            if maxProb < Q_a[i]:
                maxProb = Q_a[i]
                index = i

    C_a[index] = C_a[index] + 1
    Q_a[index]+= (r_a[index]-Q_a[index])*alpha


    if actions == 100:
        hundred_count+=1
        percentage = C_a/hundred_count
        print "\nPercentage of choosing arms 1-6:",percentage
        print "Rewards of arms 1-6:",Q_a
        actions = 0
```

```
Percentage of choosing arms 1-6: [  1.   1.  93.   2.   1.   2.]
Rewards of arms 1-6: [  2.72969730e-02   2.34298858e-02   1.78706872e+
00  -5.55089282e-04
   1.75116993e-02   6.13380771e-02]

Percentage of choosing arms 1-6: [  1.   2.  92.   1.   1.   3.]
Rewards of arms 1-6: [  4.91242113e-02   3.66712662e-02   2.50350156e+
00  -5.55089282e-04
   4.14203021e-02   2.22309201e-01]

Percentage of choosing arms 1-6: [  0.66666667   2.          92.666666
67   1.          1.          2.66666667]
Rewards of arms 1-6: [ 0.04912421  0.07899205  2.84247328  0.02255354
```

```
 0.05271574  0.32818566]


 Percentage of choosing arms 1-6: [  0.5    2.    92.25   1.5    1.5
 2.25]
 Rewards of arms 1-6: [ 0.04912421  0.09749661  2.98066088  0.04736695
 0.10962776  0.38234537]

 Percentage of choosing arms 1-6: [  1.    1.8  92.4   1.2   1.6   2. ]
 Rewards of arms 1-6: [ 0.1131545   0.13328606  3.01432102  0.04736695
 0.11552324  0.42949704]

 Percentage of choosing arms 1-6: [  1.           1.5         92.5
      1.16666667   1.5          2.33333333]
 Rewards of arms 1-6: [ 0.13685366  0.13328606  5.4655855   0.0433786
 0.1412639   0.53696552]

 Percentage of choosing arms 1-6: [  1.           1.28571429  92.571428
 57   1.42857143   1.28571429
    2.42857143]
 Rewards of arms 1-6: [ 0.15955344  0.13328606  6.4143881   0.03582242
 0.1412639   0.64662356]

 Percentage of choosing arms 1-6: [  1.     1.5   92.25   1.5    1.5
 2.25]
 Rewards of arms 1-6: [ 0.18595583  0.16418445  6.82594655  0.06134541
 0.19552068  0.6733159 ]

 Percentage of choosing arms 1-6: [  0.88888889   1.55555556  92.333333
 33   1.55555556   1.66666667   2.         ]
 Rewards of arms 1-6: [ 0.18595583  0.21125837  6.89217394  0.07733688
 0.25902264  0.6733159 ]

 Percentage of choosing arms 1-6: [  0.9   1.5  92.7   1.5   1.6   1.8]
 Rewards of arms 1-6: [ 0.20651609  0.20866464  6.94890207  0.11180264
 0.28429335  0.6733159 ]
```

In [165]:

```
#constant learning rate with 1,00,000 steps

e = 0.1
g = 0.9
alpha = 0.01
Q_a = np.zeros(k)
C_a = np.zeros(k)
actions = 0
percentage = np.zeros(6)
index = 0
hundred_count =0
for i in range(100000):
    actions = actions + 1
    randomNumber = np.random.uniform(0, 1)
    if hundred_count < 500:
        r_a = np.array([np.random.uniform(2,3),np.random.uniform(-1,5),np.random.un
                    np.random.uniform(-2,4),np.random.uniform(0,3),np.random.unifo
    else:
        r_a = np.array([np.random.uniform(2,3),np.random.uniform(-1,5),np.random.un
                    np.random.uniform(-2,4),np.random.uniform(0,3),np.random.unifo
    #exploration 10%
    if (randomNumber <= e):
        index = np.random.randint(0,k)
    #exploitation 90%
    else:
        maxProb = Q_a[0]
        maxIndex = 0
        for i in range(6):
            if maxProb < Q_a[i]:
                maxProb = Q_a[i]
                index = i

    C_a[index] = C_a[index] + 1
    Q_a[index]+= (r_a[index]-Q_a[index])*alpha


    if actions == 100:
        hundred_count+=1
        percentage = C_a/hundred_count

        actions = 0
print "\nPercentage of choosing arms 1-6:",percentage
print "Rewards of arms 1-6:",Q_a
```

```
Percentage of choosing arms 1-6: [  1.756   1.682  51.574   1.684   1.
663  41.641]
Rewards of arms 1-6: [ 2.48749309  1.82817758  7.04024128  0.92555191
1.51372821  3.84421537]
```

If the constant learning rate, alpha is 0.01,then it eventually learns that third arm is the best. This can be noticed clearly when steps are 1,00,000. No matter how the initial bias is, third arm mostly gets the highest percentage in the end(even after introducing new distribution for third arm after 50,000 steps).

Exercise 1.5

Modify your implementation by using an optimistic initialization $Q(a_i) = 10$ and a greedy action selection strategy, still using a constant learning rate α =0.01.

For every 100 actions show the percentage of choosing arm 1, arm 2, arm 3, arm 4, arm 5, and arm 6 as well as the resulting average reward.

Compare this to your result from 1.4).

In [184]:

```
k=6
e = 0.1
g = 0.9
alpha = 0.01
Q_a = np.ones(k)
Q_a = Q_a*10
C_a = np.zeros(k)
actions = 0
percentage = np.zeros(6)
index = 0
hundred_count =0
for i in range(1000):
    actions = actions + 1
    randomNumber = np.random.uniform(0, 1)
    if hundred_count < 5:
        r_a = np.array([np.random.uniform(2,3),np.random.uniform(-1,5),np.random.un
                    np.random.uniform(-2,4),np.random.uniform(0,3),np.random.unifo
    else:
        r_a = np.array([np.random.uniform(2,3),np.random.uniform(-1,5),np.random.un
                    np.random.uniform(-2,4),np.random.uniform(0,3),np.random.unifo
    #greedy selection strategy
    maxProb = Q_a[0]
    maxIndex = 0
    for i in range(6):
        if maxProb < Q_a[i]:
            maxProb = Q_a[i]
            index = i

    C_a[index] = C_a[index] + 1
    Q_a[index]+= (r_a[index]-Q_a[index])*alpha


    if actions == 100:
        hundred_count+=1
        percentage = C_a/hundred_count

        print "\nPercentage of choosing arms 1-6:",percentage
        print "Rewards of arms 1-6:",Q_a
        actions = 0
```

```
Percentage of choosing arms 1-6: [  1.   1.   1.   1.   1.  95.]
Rewards of arms 1-6: [ 9.92792066  9.92006259  9.91762303  9.90817751
9.90827333  6.33501775]

Percentage of choosing arms 1-6: [  0.5   0.5   0.5   0.5   0.5  97.5]
Rewards of arms 1-6: [ 9.92792066  9.92006259  9.91762303  9.90817751
9.90827333  4.80811757]

Percentage of choosing arms 1-6: [  0.33333333   0.33333333   0.333333
33   0.33333333   0.33333333
  98.33333333]
Rewards of arms 1-6: [ 9.92792066  9.92006259  9.91762303  9.90817751
9.90827333  4.40371261]

Percentage of choosing arms 1-6: [  0.25   0.25   0.25   0.25   0.25
 98.75]
Rewards of arms 1-6: [ 9.92792066  9.92006259  9.91762303  9.90817751
9.90827333  4.06450199]
```

```
Percentage of choosing arms 1-6: [  0.2   0.2   0.2   0.2   0.2  99. ]
Rewards of arms 1-6: [ 9.92792066  9.92006259  9.91762303  9.90817751
9.90827333  4.00157199]

Percentage of choosing arms 1-6: [  0.16666667   0.16666667   0.166666
67   0.16666667   0.16666667
  99.16666667]
Rewards of arms 1-6: [ 9.92792066  9.92006259  9.91762303  9.90817751
9.90827333  3.93817126]

Percentage of choosing arms 1-6: [  0.14285714   0.14285714   0.142857
14   0.14285714   0.14285714
  99.28571429]
Rewards of arms 1-6: [ 9.92792066  9.92006259  9.91762303  9.90817751
9.90827333  4.02133286]

Percentage of choosing arms 1-6: [  0.125   0.125   0.125   0.125   0.
125  99.375]
Rewards of arms 1-6: [ 9.92792066  9.92006259  9.91762303  9.90817751
9.90827333  4.00690199]

Percentage of choosing arms 1-6: [  0.11111111   0.11111111   0.111111
11   0.11111111   0.11111111
  99.44444444]
Rewards of arms 1-6: [ 9.92792066  9.92006259  9.91762303  9.90817751
9.90827333  4.16350677]

Percentage of choosing arms 1-6: [  0.1   0.1   0.1   0.1   0.1  99.5]
Rewards of arms 1-6: [ 9.92792066  9.92006259  9.91762303  9.90817751
9.90827333  4.0399091 ]
```

mostly biased towards the initial choice although encourages exploration initially

In [ ]: