linear

December 4, 2017

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

1 Environment description

```
In [2]: class linear_environment:
            def __init__(self):
                self.n_states = 21          # number of states
                self.n_actions = 2         # number of actions  left/right
                self.reward = np.zeros([self.n_states, self.n_actions])
                self.terminal = np.zeros(self.n_states, dtype=np.int)          # 1 if terminal s
                self.next_state = np.stack([np.arange(self.n_states)-1, \
                                            np.arange(self.n_states)+1],1)        # next_state
                self.init_state = 10    # initial state
                self.reward[1,0] = 1    # rewards only at terminal states
                self.reward[-2,1] = 1
                self.terminal[0] = 1    # only two terminal states
                self.terminal[-1] = 1
                self.next_state[0,0] = 0
                self.next_state[-1,1] = self.n_states-1

        # an instance of the environment
        env = linear_environment()
```

2 Hyperparameters

```
In [3]: n_episodes = 1       # number of episodes to run
        max_steps = 1000         # max. # of steps to run in each episode
        alpha = 0.2           # learning rate
        gamma = 0.9           # discount factor
```

```
In [4]: class epsilon_profile: pass
        # constant epsilon = 1
        epsilon1 = epsilon_profile()
        epsilon1.init = 1.     # initial epsilon in e-greedy
        epsilon1.final = 1.    # final epsilon in e-greedy
```

```
epsilon1.dec_episode = 0
epsilon1.dec_step = 0.

# linearly decreasing epsilon
epsilon2 = epsilon_profile()
epsilon2.init = 1.    # initial epsilon in e-greedy
epsilon2.final = 0.   # final epsilon in e-greedy
epsilon2.dec_episode = 1. / n_episodes  # amount of decrement in each episode
epsilon2.dec_step = 0.
```

3 Train function for Q table

```
In [5]: def train(n_episodes=None, eps=None, plot=False):
            Q, n_steps, sum_rewards = Q_learning_train(env, n_episodes, max_steps,\
                                                       alpha, gamma, eps)
            print('Q(s,a)')
            np.set_printoptions(precision=6)
            print(Q)
            if not plot:
                for k in range(n_episodes):
                    print('%2d: %.2f steps %2d' % (k, sum_rewards[k],n_steps[k]))
            else:
                fig = plt.figure(num=None, figsize=(6, 4), dpi=130, facecolor='w', \
                                                                    edgecolor='k')
                ax2 = fig.add_subplot(111)
                plt.xlabel('Number of episodes')
                plt.ylabel('Number of steps before the termination')
                x = np.linspace(0,len(n_steps),num=len(n_steps))
                plt.plot(x, n_steps, linewidth=1, c='m')
                plt.grid(True)
                plt.show()
            print("train average n_steps = %d" % np.mean(n_steps))
            return Q
```

4 Test function with trained Q table

```
In [6]: def test(Q):
            test_n_episodes = 1  # number of episodes to run
            test_max_steps = 1000   # max. # of steps to run in each episode
            test_epsilon = 0.    # test epsilon
            test_n_steps, test_sum_rewards, s, a, sn, r = Q_test(Q, env, test_n_episodes,\
                                                        test_max_steps, test_epsilon)
            print('Test results')
            print("sum_rewards = %d, n_steps = %d"%(test_sum_rewards[0], test_n_steps[0]))
```

5 1. a) $\epsilon = 1$

As can be seen from results below, during testing always $n_{steps} \neq 10$

This can be explained by the Q table itself. Since $n_{episodes} = 1$ only one end of the linear environment can be reached and hence only one reward could be received during the training while taken action leads to the terminal state. Therefore why only one value in the Q table is updated and the others stay 0.

That's why during the testing, the walk begins at $s_{init} = 10$ and the Q table is 0 in the neighborhood, hence greedy policy w.r.t. Q values is not useful, and turns into a random walk. And in random walk in order to deviate on 10 positions from the beginning state it takes roughly $10^2 = 100$ steps

```
In [7]: print("1a experiment")
        for i in range(5):
            Q = train(n_episodes=1, eps=epsilon1)
            test(Q)
            print(" ------------------------ ")
```

```
1a experiment
Q(s,a)
[[ 0.   0. ]
 [ 0.2  0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]]
 0: 1.00 steps 16
train average n_steps = 16
Test results
sum_rewards = 1, n_steps = 18
 ------------------------
Q(s,a)
[[ 0.   0. ]
```

```
 [ 0.2  0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]]
 0: 1.00 steps 108
train average n_steps = 108
Test results
sum_rewards = 1, n_steps = 198
 ------------------------
Q(s,a)
[[ 0.   0. ]
 [ 0.2  0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]]
 0: 1.00 steps 266
```

```
train average n_steps = 266
Test results
sum_rewards = 1, n_steps = 44
 ------------------------
Q(s,a)
[[ 0.   0. ]
 [ 0.2  0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]]
 0: 1.00 steps 62
train average n_steps = 62
Test results
sum_rewards = 1, n_steps = 222
 ------------------------
Q(s,a)
[[ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
```

```
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0. ]
 [ 0.   0.2]
 [ 0.   0. ]]
 0: 1.00 steps 134
train average n_steps = 134
Test results
sum_rewards = 1, n_steps = 62
 ------------------------
```

6 1. b) $\epsilon = 1$

As can be seen from results below, during testing sometimes $n_{steps} = 10$

This can be explained by the Q table itself. Since $n_{episodes} = 5$ more state-action values can be updated in Q table, also since $\epsilon = 1$, the agent always explores and for action selection it never relies on Q values during the training, therefore its just a random walk. Therefore after the first episode, when the pre-terminal state-action value is updated in one of the ends of Q table, that q value is influencing the values in other states of Q table for teh next episodes, as the exploration takes place according to

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$$

Depending on whether most of the state-action pairs (to the left or to the right from the initial state) were updated during training or not, Q table is going to be efficient for finding the shortest path or not.

That's why during the testing, the walk begins at $s_{init} = 10$ and whenever the Q table wasn't updated in the neighborhood, a number of states to complete the linear environment is not 10. However, whenever the state-action space was explored enough, greedy policy w.r.t. Q values is useful in guiding the agent until the terminal state in 10 steps.

```
In [8]: print("1b experiment")
        for i in range(5):
            Q = train(n_episodes=5, eps=epsilon1)
            test(Q)
            print(" ------------------------ ")
```

```
1b experiment
Q(s,a)
[[  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   7.278008e-09]
 [  5.038621e-10   4.789800e-08]
 [  8.273291e-09   1.997478e-07]
 [  5.674496e-08   6.506374e-07]
 [  1.247944e-07   4.799784e-06]
```

```
 [  2.666547e-07   2.995170e-05]
 [  6.695995e-06   1.022877e-04]
 [  2.755139e-05   2.691710e-04]
 [  9.796664e-05   1.153201e-03]
 [  2.147551e-04   1.557015e-02]
 [  3.712011e-03   4.597552e-02]
 [  1.100835e-02   1.261459e-01]
 [  8.876832e-02   1.723584e-01]
 [  1.412041e-01   2.118302e-01]
 [  1.469879e-01   3.265828e-01]
 [  2.376583e-01   4.352691e-01]
 [  2.508740e-01   6.723200e-01]
 [  0.000000e+00   0.000000e+00]]
 0: 1.00 steps 20
 1: 1.00 steps 48
 2: 1.00 steps 72
 3: 1.00 steps 346
 4: 1.00 steps 114
train average n_steps = 120
Test results
sum_rewards = 1, n_steps = 10
 ------------------------
Q(s,a)
[[  0.000000e+00   0.000000e+00]
 [  4.880000e-01   0.000000e+00]
 [  9.360000e-02   0.000000e+00]
 [  6.480000e-03   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   2.099520e-04]
 [  0.000000e+00   2.099520e-03]
 [  2.099520e-04   1.581120e-02]
 [  3.032640e-03   6.480000e-02]
 [  6.480000e-03   3.600000e-01]
 [  0.000000e+00   0.000000e+00]]
 0: 1.00 steps 164
 1: 1.00 steps 116
 2: 1.00 steps 46
 3: 1.00 steps 14
```

```
 4: 1.00 steps 198
train average n_steps = 107
Test results
sum_rewards = 1, n_steps = 10
 ------------------------
Q(s,a)
[[  0.000000e+00   0.000000e+00]
 [  3.600000e-01   6.480000e-03]
 [  6.480000e-02   2.624446e-12]
 [  4.724003e-13   4.901299e-11]
 [  2.261223e-11   8.100142e-11]
 [  1.458026e-11   4.500079e-10]
 [  1.170020e-10   1.388913e-09]
 [  2.500044e-10   7.716185e-09]
 [  1.487264e-09   3.983217e-08]
 [  7.716185e-09   2.044258e-07]
 [  5.692825e-08   7.571950e-07]
 [  2.756594e-07   1.587509e-06]
 [  5.293432e-07   3.614547e-06]
 [  6.506185e-07   2.286710e-05]
 [  1.054978e-05   1.829102e-04]
 [  3.292383e-05   3.049175e-03]
 [  8.712571e-04   7.735565e-03]
 [  1.116945e-03   3.779136e-02]
 [  7.138368e-03   1.396800e-01]
 [  1.684800e-02   4.880000e-01]
 [  0.000000e+00   0.000000e+00]]
 0: 1.00 steps 44
 1: 1.00 steps 182
 2: 1.00 steps 78
 3: 1.00 steps 70
 4: 1.00 steps 106
train average n_steps = 96
Test results
sum_rewards = 1, n_steps = 10
 ------------------------
Q(s,a)
[[  0.000000e+00   0.000000e+00]
 [  2.000000e-01   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
```

```
 [  0.000000e+00   2.066979e-05]
 [  1.573372e-06   1.154720e-04]
 [  3.547800e-05   4.399043e-04]
 [  9.426521e-05   2.895735e-03]
 [  3.586522e-04   1.414736e-02]
 [  4.311515e-03   4.747508e-02]
 [  2.024943e-02   8.913947e-02]
 [  3.770526e-02   2.834865e-01]
 [  8.301243e-02   5.904000e-01]
 [  0.000000e+00   0.000000e+00]]
 0: 1.00 steps 52
 1: 1.00 steps 132
 2: 1.00 steps 46
 3: 1.00 steps 56
 4: 1.00 steps 196
train average n_steps = 96
Test results
sum_rewards = 1, n_steps = 10
 ------------------------
Q(s,a)
[[  0.000000e+00   0.000000e+00]
 [  3.600000e-01   0.000000e+00]
 [  3.600000e-02   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   0.000000e+00]
 [  0.000000e+00   2.995350e-07]
 [  5.391631e-08   3.878390e-06]
 [  9.377382e-07   1.415068e-05]
 [  4.236972e-06   4.948813e-05]
 [  2.017826e-05   1.081499e-04]
 [  5.569513e-05   1.724847e-04]
 [  7.116846e-05   3.322665e-04]
 [  1.706480e-04   6.197783e-04]
 [  2.482137e-04   3.032640e-03]
 [  5.458752e-04   9.296571e-02]
 [  3.137859e-02   2.711002e-01]
 [  1.065022e-01   4.880000e-01]
 [  0.000000e+00   0.000000e+00]]
 0: 1.00 steps 58
 1: 1.00 steps 66
 2: 1.00 steps 104
 3: 1.00 steps 156
 4: 1.00 steps 160
train average n_steps = 108
Test results
```

```
sum_rewards = 1, n_steps = 10
 ------------------------
```

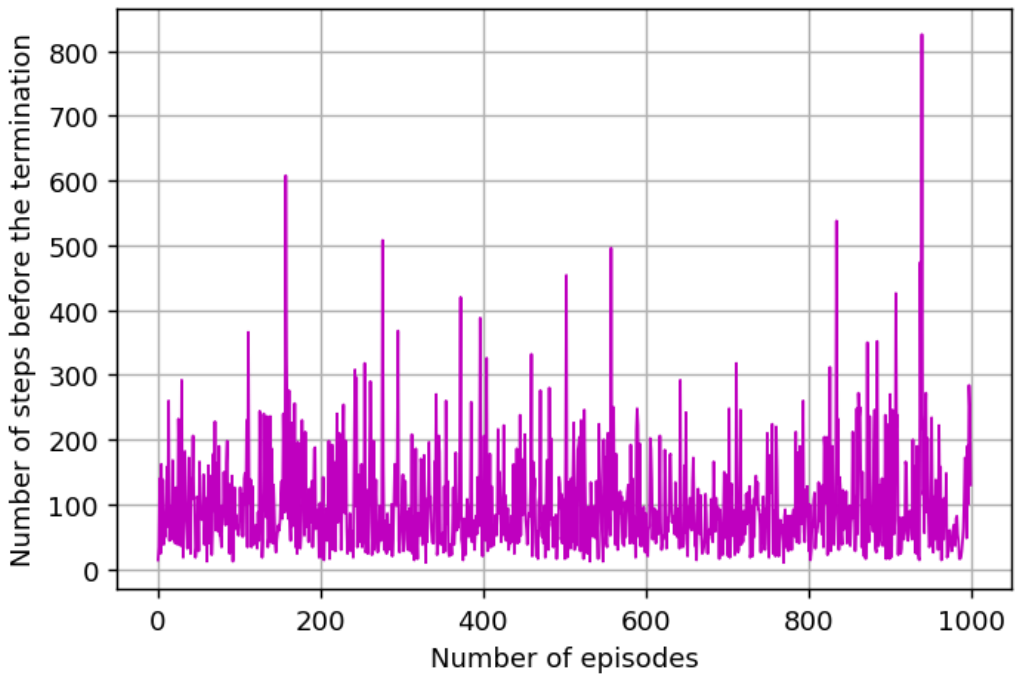
7 1. c) $\epsilon = 1$

This can be explained by the Q table. Since $n_{episodes} = 1000$ all state-action values get updates in Q table, it is due to $\epsilon = 1$ which turns the walk into a random walk, sinse the agent always explores and for action selection, it never relies on Q values during the training. Since it is a random walk during training, in order to deviate on 10 positions from the beginning state it takes roughly $10^2 = 100$ steps, which is confirmed empirically.

That's why during the testing, the greedy policy w.r.t. Q values is useful in guiding the agent until the terminal state in shortest way of $n_{steps} = 10$.

```
In [9]: Q = train(n_episodes=1000, eps=epsilon1, plot=True)
        print('\n')
        test(Q)
        for i in range(8):
            print('\n')
```

```
Q(s,a)
[[ 0.         0.      ]
 [ 1.         0.81    ]
 [ 0.9        0.729   ]
 [ 0.81       0.6561  ]
 [ 0.729      0.59049 ]
 [ 0.6561     0.531441]
 [ 0.59049    0.478297]
 [ 0.531441   0.430467]
 [ 0.478297   0.38742 ]
 [ 0.430467   0.348678]
 [ 0.38742    0.38742 ]
 [ 0.348678   0.430467]
 [ 0.38742    0.478297]
 [ 0.430467   0.531441]
 [ 0.478297   0.59049 ]
 [ 0.531441   0.6561  ]
 [ 0.59049    0.729   ]
 [ 0.6561     0.81    ]
 [ 0.729      0.9     ]
 [ 0.81       1.      ]
 [ 0.         0.      ]]
```




```
train average n_steps = 97


Test results
sum_rewards = 1, n_steps = 10
```

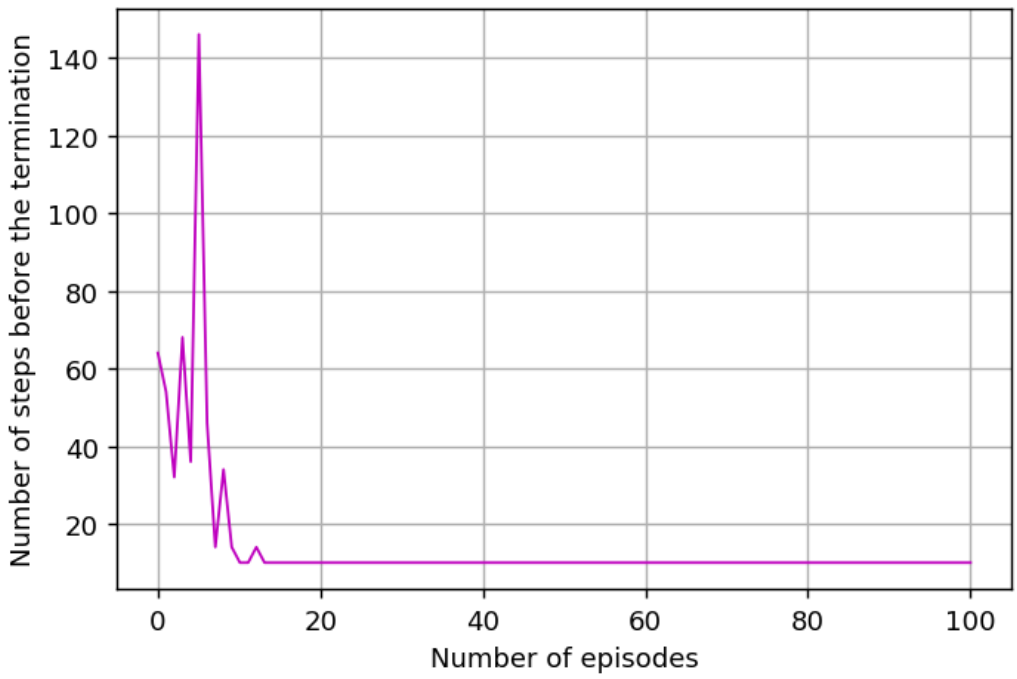
8 2. a) $\epsilon = 1, \quad \epsilon \to 0$

As can be seen from results below, during testing $n_{steps} = 10$

Also we can see that in this case due to exploration that is linearly replaced by exploitation only a half of Q table is updated fully since in the beginning the agent explores random state-action pairs (in the random walk setting) but with time as $\epsilon \to 0$ it starts to rely on Q function values and act greedy on them. That's why Q table is sparse, but it is enough for converging to the optimal value of steps at the end of training.

```
In [10]: Q = train(n_episodes=100, eps=epsilon2, plot=True)
         print('\n')
         test(Q)
```

```
Q(s,a)
[[ 0.        0.      ]
 [ 0.488     0.      ]
 [ 0.0936    0.      ]
 [ 0.00648   0.      ]
 [ 0.        0.      ]
 [ 0.        0.      ]
 [ 0.        0.      ]
 [ 0.        0.      ]
 [ 0.        0.      ]
 [ 0.        0.      ]
 [ 0.        0.38607 ]
 [ 0.        0.4299  ]
 [ 0.        0.478087]
 [ 0.        0.531373]
 [ 0.        0.590471]
 [ 0.        0.656096]
 [ 0.        0.728999]
 [ 0.        0.81    ]
 [ 0.        0.9     ]
 [ 0.        1.      ]
 [ 0.        0.      ]]
```




```
train average n_steps = 14


Test results
sum_rewards = 1, n_steps = 10
```