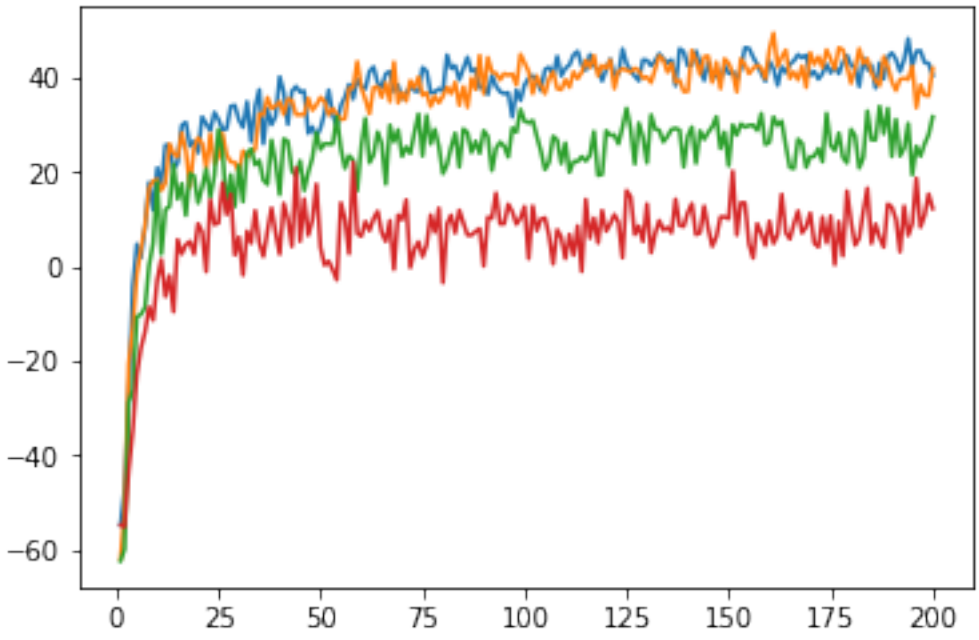
RLmaze

June 5, 2018

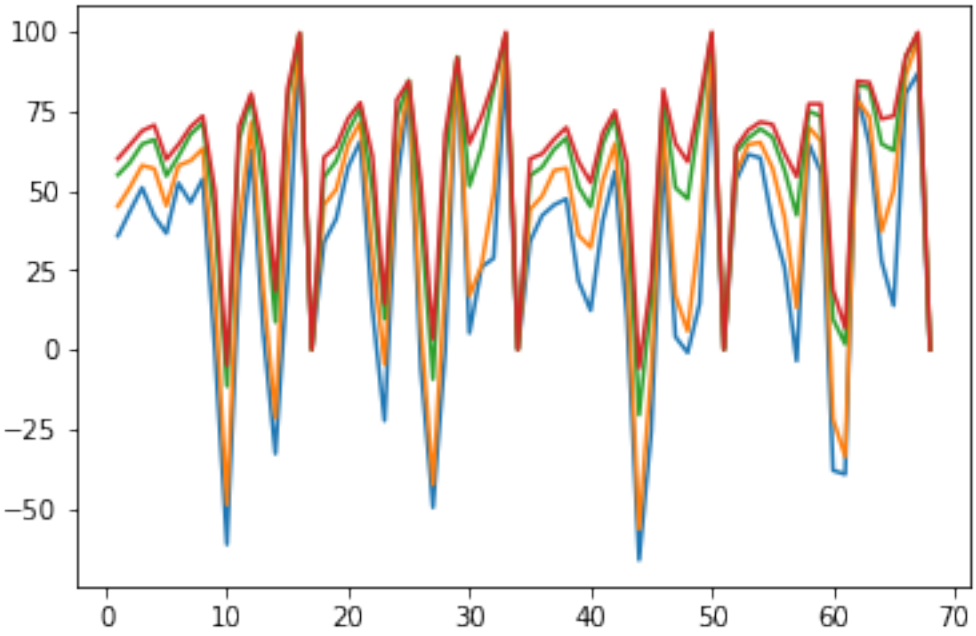
```
In [3]: %run TestRLmaze.py

0.05
0.1
0.3
0.5


In [4]: %matplotlib inline

In [36]: x = np.linspace(1,200,200)

In [37]: plt.plot(x,mean_reward[0,:],x,mean_reward[1,:],x,mean_reward[2,:],x,mean_reward[3,:])

Out[37]: [<matplotlib.lines.Line2D at 0x1c4dc4e7908>,
          <matplotlib.lines.Line2D at 0x1c4dc4e7a58>,
          <matplotlib.lines.Line2D at 0x1c4dc4e7ef0>,
          <matplotlib.lines.Line2D at 0x1c4dc4fb358>]
```



From the graph we can tell that when increasing the exploration rate, the average accumulated rewards decreases. It is because if the epsilon is large, the chance of the learning system not taking the optimal action is higher. This leads to a low average reward.

```
In [38]: x = np.linspace(1,mdp.nActions*mdp.nStates, mdp.nActions*mdp.nStates)
         plt.plot(x, mean_Q[0,:,:].reshape(mdp.nActions*mdp.nStates),
                  x, mean_Q[1,:,:].reshape(mdp.nActions*mdp.nStates),
                  x, mean_Q[2,:,:].reshape(mdp.nActions*mdp.nStates),
                  x, mean_Q[3,:,:].reshape(mdp.nActions*mdp.nStates))
```

```
Out[38]: [<matplotlib.lines.Line2D at 0x1c4dc6ff7b8>,
          <matplotlib.lines.Line2D at 0x1c4dc6ff908>,
          <matplotlib.lines.Line2D at 0x1c4dc6ffd30>,
          <matplotlib.lines.Line2D at 0x1c4dc754198>]
```



For Q values, we notice that average Q values are higher when epsilon is larger.

```
In [6]: np.amax(mean_Q[0,:,:],axis=0)
```

```
Out[6]: array([ 54.66096897,  62.98237051,  62.20576446,  68.88748888,
                37.35874309,  53.40520693,  67.21702105,  80.22943255,
                12.35209438, -38.50617095,  76.83206833,  91.28818339,
                29.46427074,  18.97631442,  80.6239287 ,  96.87430482,   0.        ])
```

```
In [7]: np.amax(mean_Q[1,:,:],axis=0)
```

```
Out[7]: array([ 56.31039563,  63.65422472,  65.22325731,  72.68111327,
                43.24761887,  56.93652034,  72.02282996,  81.10081005,
                18.88206251, -36.93115144,  79.26985302,  90.17483254,
                34.76442091,  35.44258075,  87.21827359,  98.84844281,   0.        ])

In [9]: np.amax(mean_Q[2,:,:],axis=0)

Out[9]: array([ 61.09944196,  67.08139369,  71.7725864 ,  75.10585665,
                57.14468902,  62.50486054,  74.95621896,  83.79845393,
                46.58761385,   2.40464077,  82.92240405,  92.06438459,
                60.27539678,  61.74201367,  91.7787288 ,  99.34837052,   0.        ])

In [10]: np.amax(mean_Q[3,:,:],axis=0)

Out[10]: array([ 63.39391498,  68.14487916,  73.67559975,  77.87693455,
                 63.15277581,  64.27687926,  77.75421798,  85.02314978,
                 58.62229274,   8.02908339,  84.29917054,  92.69306627,
                 72.21362976,  73.06620638,  92.96666898,  99.53519656,   0.        ])
```

By comparing the optimal state values with results in assignment part 1, we find when eplison = 0.3, the optimal state values are the closest.

Also, the optimal policy from part 1 is [3 3 3 1 3 3 3 1 1 3 3 1 3 3 3 0 0]. To obtain the policy from the average of 100 trials, we think the policy when epsilon = 0.3 is close to the optimal policy that we get from part 1.

```
In [15]: # policy epsilon = 0.05
         np.argmax(mean_Q[0,:,:],axis=0)

Out[15]: array([3, 3, 1, 1, 0, 0, 3, 1, 2, 3, 3, 1, 3, 1, 3, 0, 0], dtype=int64)

In [16]: # policy epsilon = 0.1
         np.argmax(mean_Q[1,:,:],axis=0)

Out[16]: array([3, 3, 1, 1, 0, 0, 3, 1, 2, 3, 3, 1, 3, 3, 3, 0, 0], dtype=int64)

In [17]: # policy epsilon = 0.3
         np.argmax(mean_Q[2,:,:],axis=0)

Out[17]: array([3, 3, 1, 1, 3, 0, 3, 1, 2, 3, 3, 1, 3, 3, 3, 0, 0], dtype=int64)

In [18]: # policy epsilon = 0.5
         np.argmax(mean_Q[1,:,:],axis=0)

Out[18]: array([3, 3, 1, 1, 0, 0, 3, 1, 2, 3, 3, 1, 3, 3, 3, 0, 0], dtype=int64)
```

If the epsilon is too small, then the system cannot find an optimal solution. If the epsilon is too large, the system will have a hard time converging to that optimal solution.