ELEN_6885_HW4_Part_1_2_3

November 19, 2019

1 ELEN 6885 Reinforcement Learning Coding Assignment (Part 1, 2, 3)

1.1 Taxi Problem Overview

Please put your code into the block marked by: ############################# YOUR CODE STARTS HERE YOUR CODE ENDS HERE ############################# You should not edit anything outside of the block.

[0]:

```

```

2 Playing with the environment

Run the cell below to get a feel for the environment by moving your agent(the taxi) by taking one of the actions at each step.

[0]:

```
from gym.wrappers import Monitor
import gym
import random
import numpy as np
```

[0]:

```
"""
You can test your game now.
Input range from 0 to 5:
    0 : South (Down)
    1 : North (Up)
    2 : East (Right)
    3 : West (Left)
    4: Pick up
    5: Drop off
    6: exit_game
"""
GAME = "Taxi-v3"
env = gym.make(GAME)
env = Monitor(env, "taxi_simple", force=True)
s = env.reset()
steps = 100
```

```
for step in range(steps):
    env.render()
    action = int(input("Please type in the next action:"))
    if action==6:
        break
    s, r, done, info = env.step(action)
    print('state:',s)
    print('reward:',r)
    print('Is state terminal?:',done)
    print('info:',info)

# close environment and monitor
env.close()
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
Please type in the next action:6
```

2.1 1.1 Incremental implementation of average

We've finished the incremental implementation of average for you. Please call the function to estimate with 1/step step size and fixed step size to compare the difference between these two on a simulated Bandit problem.

[0]:
```
def estimate(OldEstimate, StepSize, Target):
    '''An incremental implementation of average.
    OldEstimate : float
    StepSize : float
    Target : float
    '''
    NewEstimate = OldEstimate + StepSize * (Target - OldEstimate)
    return NewEstimate
```

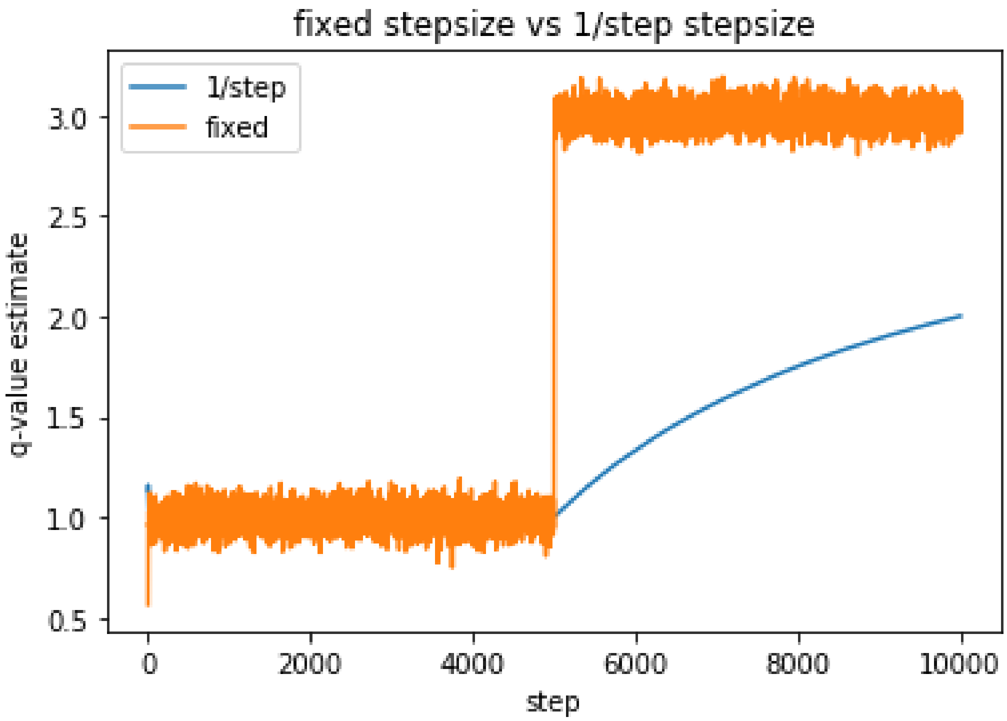
[0]:
```
random.seed(6885)
numTimeStep = 10000
q_h = np.zeros(numTimeStep + 1) # Q Value estimate with 1/step step size
q_f = np.zeros(numTimeStep + 1) # Q value estimate with fixed step size
FixedStepSize = 0.5 #A large number to exaggerate the difference
for step in range(1, numTimeStep + 1):
    if step < numTimeStep / 2:
        r = random.gauss(mu = 1, sigma = 0.1)
    else:
```

```
        r = random.gauss(mu = 3, sigma = 0.1)

    #TIPS: Call function estimate defined in ./helpers/utils.py
    ############################
    # YOUR CODE STARTS HERE
    q_f[step]=estimate(q_f[step-1],FixedStepSize,r)
    q_h[step]=estimate(q_h[step-1],1/step,r)

    # YOUR CODE ENDS HERE
    ############################

q_h = q_h[1:]
q_f = q_f[1:]
```

Plot the two Q value estimates. (Please include a title, labels on both axes, and legends)

[0]:
```
import matplotlib.pyplot as plt
############################
# YOUR CODE STARTS HERE
plt.plot(q_h)
plt.plot(q_f)
plt.xlabel('step')
plt.ylabel('q-value estimate')
plt.title('fixed stepsize vs 1/step stepsize')
plt.legend(['1/step','fixed'])


# YOUR CODE ENDS HERE
############################
```

[0]:
```
<matplotlib.legend.Legend at 0x7ff13326a908>
```



2.2 1.2 ϵ-Greedy for Exploration

In Reinforcement Learning, we are always faced with the dilemma of exploration and exploitation. ϵ-Greedy is a trade-off between them. You are supposed to implement Greedy and ϵ-Greedy. We combine these two policies in one function by treating Greedy as ϵ-Greedy where $\epsilon = 0$. Edit the function epsilon_greedy the following block.

[0]:
```python
def epsilon_greedy(value, e, seed = None):
    '''
    Implement Epsilon-Greedy policy.

    Inputs:
    value: numpy ndarray
    A vector of values of actions to choose from
    e: float
    Epsilon
    seed: None or int
    Assign an integer value to remove the randomness

    Outputs:
    action: int
    Index of the chosen action
    '''
    assert len(value.shape) == 1
```

```
    assert 0 <= e <= 1

    if seed != None:
          np.random.seed(seed)

    ############################
    # YOUR CODE STARTS HERE
    n=len(value)
    ############################
    # YOUR CODE STARTS HERE
    np.random.seed(0)
    randProb = np.random.random()  # Pick random probability between 0-1
    if randProb < e:
        a = np.random.choice(len(value))  # Select random action
    else:
        maxAction = np.argmax(value)  # Find max value estimate
        action = np.where(value == np.argmax(value))[0]
        if len(action) == 0:
            a = maxAction
        else:
            a = np.random.choice(action)

    # YOUR CODE ENDS HERE
    ############################
    return a
```

[0]:
```
np.random.seed(6885) #Set the seed forreproducability
q = np.random.normal(0, 1, size = 5)
############################
# YOUR CODE STARTS HERE
greedy_action=epsilon_greedy(q, 0, seed = None)
e_greedy_action=epsilon_greedy(q, 0, seed = None)
# YOUR CODE ENDS HERE
############################
print('Values:')
print(q)
print('Greedy Choice =', greedy_action)
print('Epsilon-Greedy Choice =', e_greedy_action)
```

```
Values:
[ 0.61264537  0.27923079 -0.84600857  0.05469574 -1.09990968]
Greedy Choice = 0
Epsilon-Greedy Choice = 0
```

You should get the following results: Values: [0.61264537 0.27923079 -0.84600857 0.05469574 -1.09990968] Greedy Choice = 0 Epsilon-Greedy Choice = 0

2.3 1.3 Exploration VS. Exploitation

Try to reproduce Figure 2.2 (the upper one is enough) of the Sutton's book based on the experiment described in Chapter 2.3.

[0]:
```
# Do the experiment and record average reward acquired in each time step
############################
# YOUR CODE STARTS HERE
import numpy as np
import matplotlib.pyplot as plt


Arms = 10  # n number of bandits
iterations = 2000  # number of repeated iterations
plays = 1000
kAction = np.zeros(Arms)          # count of actions taken at time t
rSum = np.zeros(Arms)

valEstimates = np.zeros(Arms)

scoreArr = np.zeros(plays)
scoreArr1 = np.zeros(plays)
scoreArr2 = np.zeros(plays)
rAvg=np.zeros(plays)

def action(valEstimates,e):
    randProb = np.random.random()  # Pick random probability between 0-1
    if randProb < e:
        a = np.random.choice(len(valEstimates))  # Select random action
    else:
        maxAction = np.argmax(valEstimates)  # Find max value estimate
        action = np.where(valEstimates == np.argmax(valEstimates))[0]
        if len(action) == 0:
            a = maxAction
        else:
            a = np.random.choice(action)
    return a

for iIter in range(iterations):

    q_star = np.random.normal(0, 1, 10)
    #actionT = None  # Store last action
    kAction = np.zeros(Arms)
    kAction1 = np.zeros(Arms)
    kAction2 = np.zeros(Arms)# count of actions taken at time t
    rSum = np.zeros(Arms)
    rSum1 = np.zeros(Arms)
    rSum2 = np.zeros(Arms)
```

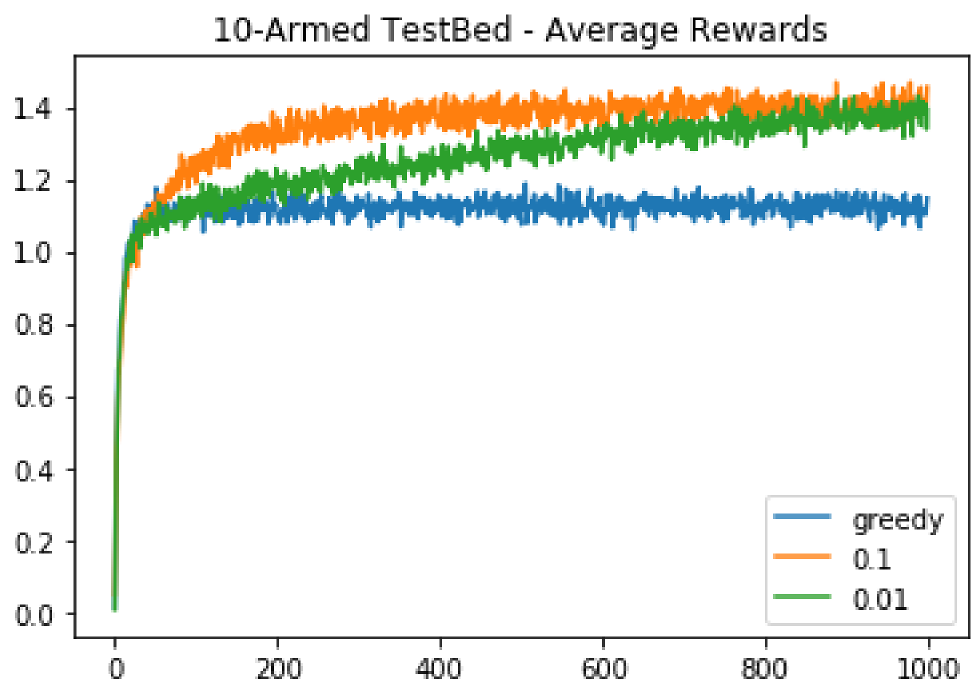
```
    valEstimates = np.zeros(Arms)
    valEstimates1 = np.zeros(Arms)
    valEstimates2 = np.zeros(Arms)
    if (iIter % 100) == 0:
        print("Completed Iterations: ", iIter)
    for jPlays in range(plays):
        actionT = action(valEstimates,e=0)
        actionT1 = action(valEstimates1, e=0.1)
        actionT2 = action(valEstimates2, e=0.01)
        rewardT = np.random.normal(q_star[actionT], 1)
        rewardT1 = np.random.normal(q_star[actionT1], 1)
        rewardT2 = np.random.normal(q_star[actionT2], 1)
        At = actionT
        At1 = actionT1
        At2 = actionT2
        kAction[At] += 1
        kAction1[At1] += 1
        kAction2[At2] += 1# Add 1 to action selection
        rSum[At] += rewardT
        rSum1[At1] += rewardT1
        rSum2[At2] += rewardT2# Add reward to sum array
        valEstimates[At] = rSum[At] / kAction[At]
        valEstimates1[At1] = rSum1[At1] / kAction1[At1]
        valEstimates2[At2] = rSum2[At2] / kAction2[At2]
        scoreArr[jPlays] += rewardT
        scoreArr1[jPlays] += rewardT1
        scoreArr2[jPlays] += rewardT2
rAvg = scoreArr / iterations
rAvg1 = scoreArr1/ iterations
rAvg2 = scoreArr2 / iterations

# YOUR CODE ENDS HERE
############################
```

```
Completed Iterations:  0
Completed Iterations:  100
Completed Iterations:  200
Completed Iterations:  300
Completed Iterations:  400
Completed Iterations:  500
Completed Iterations:  600
Completed Iterations:  700
Completed Iterations:  800
Completed Iterations:  900
Completed Iterations:  1000
Completed Iterations:  1100
Completed Iterations:  1200
```

```
Completed Iterations:  1300
Completed Iterations:  1400
Completed Iterations:  1500
Completed Iterations:  1600
Completed Iterations:  1700
Completed Iterations:  1800
Completed Iterations:  1900
```

[0]:

```
# Plot the average reward
############################
# YOUR CODE STARTS HERE
plt.title("10-Armed TestBed - Average Rewards")
plt.plot(rAvg)
plt.plot(rAvg1)
plt.plot(rAvg2)
plt.legend(['greedy','0.1','0.01'],loc=4)
plt.show()

# YOUR CODE ENDS HERE
############################
```



3 Question 2

In this question, you will implement the value iteration and policy iteration algorithms to solve the Taxi game problem

3.1 2.1 Model-based RL: value iteration

For this part, you need to implement the helper functions action_evaluation(env, gamma, v), and extract_policy(env, v, gamma) in utils.py. Understand action_selection(q) which we have implemented. Use these helper functions to implement the value_iteration algorithm below.

[0]:
```
import numpy as np
from helpers import utils
def value_iteration(env, gamma, max_iteration, theta):
    """
    Implement value iteration algorithm. You should use extract_policy to for␣
 ↪extracting the policy.

    Parameters
    ----------
    env: OpenAI env.
            env.P: dictionary
                    the transition probabilities of the environment
                    P[state][action] is tuples with (probability, nextstate,␣
 ↪reward, terminal)
            env.nS: int
                    number of states
            env.nA: int
                    number of actions
    gamma: float
            Discount factor.
    max_iteration: int
            The maximum number of iterations to run before stopping.
    theta: float
            Determines when value function has converged.
    Returns:
    ----------
    value function: np.ndarray
    policy: np.ndarray
    """
    nS = env.nS
    nA = env.nA
    V = np.zeros(env.nS)
    ############################
    # YOUR CODE STARTS HERE
    for i in range(max_iteration):
      q=utils.action_evaluation(env, gamma, V)
      #print(q)
```

```
        policy=np.argmax(q, axis = 1)
        #q=action_evaluation(env, gamma, v)
        delta=[]
        for s in range(nS):
            temp=V[s]
            #print(temp)
            V[s]=max(q[s][:])
            #print(V[s])
            #print(abs(temp-V[s]))
            delta.append(abs(temp-V[s]))
        if max(delta)<theta:
                break
    # YOUR CODE ENDS HERE
    ############################

    return V, policy
```

After implementing the above function, read and understand the functions implemented in evaluation_utils.py, which we will use to evaluate our value iteration policy

[0]:
```
from helpers import evaluation_utils
import gym
GAME = "Taxi-v3"
env = gym.make(GAME)
V_vi, policy_vi = value_iteration(env, gamma=0.95, max_iteration=6000,␣
 ↪theta=1e-5)
# visualize how the agent performs with the policy generated from value␣
 ↪iteration
evaluation_utils.render_episode(env, policy_vi)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
```

```
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
+---------+
```

```
|R: | : :G|
| : | : : |
| : : : : |
|_| : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| :_: : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : | : : |
| : :_: : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : |_: : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: |_: :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
```

```
|R: | :_:G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
Episode reward: 5.000000
```

[0]:
```python
# evaluate the performance of value iteration over 100 episodes
evaluation_utils.avg_performance(env, policy_vi)
```

[0]:
```
8.121212121212121
```

3.2 2.2 Model-based RL: policy iteration

In this part, you are supposed to implement policy iteration to solve the Taxi game problem.

[0]:
```python
#from helpers import utils
def policy_iteration(env, gamma, max_iteration, theta):
    """Implement Policy iteration algorithm.

    You should use the policy_evaluation and policy_improvement methods to
    implement this method.

    Parameters
    ----------
    env: OpenAI env.
            env.P: dictionary
                    the transition probabilities of the environment
                    P[state][action] is tuples with (probability, nextstate,␣
 ↪reward, terminal)
```

```
            env.nS: int
                    number of states
            env.nA: int
                    number of actions
    gamma: float
            Discount factor.
    max_iteration: int
            The maximum number of iterations to run before stopping.
    theta: float
            Determines when value function has converged.
    Returns:
    ----------
    value function: np.ndarray
    policy: np.ndarray
    """

    V0= np.zeros(env.nS)
    policy = np.zeros(env.nS, dtype=int)
    ############################
    # YOUR CODE STARTS HERE
    nS = env.nS
    nA = env.nA
    policy_stable=['b']
    while True:
        if 'b'  not in policy_stable:
            break
        else:
            #delta=[]
            #for i in range(max_iteration):
                #for s in range(nS):
                    #temp1=V[s]
                    #V = policy_evaluation(env, policy, gamma, theta)
                    #delta.append(abs(temp1 - V[s]))
                #if max(delta) < theta:
                    #break
            V = policy_evaluation(env, policy, gamma, theta,max_iteration,V0)
            policy, policy_stable = policy_improvement(env, V, policy, gamma)


    # YOUR CODE ENDS HERE
    ############################

    return V, policy


def policy_evaluation(env, policy, gamma, theta,max_iteration,V):
    """Evaluate the value function from a given policy.
```

```
    Parameters
    ----------
    env: OpenAI env.
            env.P: dictionary
                    the transition probabilities of the environment
                    P[state][action] is tuples with (probability, nextstate,␣
↪reward, terminal)
            env.nS: int
                    number of states
            env.nA: int
                    number of actions

    gamma: float
            Discount factor.
    policy: np.array
            The policy to evaluate. Maps states to actions.
    max_iteration: int
            The maximum number of iterations to run before stopping.
    theta: float
            Determines when value function has converged.
    Returns
    -------
    value function: np.ndarray
            The value function from the given policy.
    """
    #V = np.zeros(env.nS)###
    ############################
    # YOUR CODE STARTS HERE
    nS = env.nS
    nA = env.nA
    #v = np.zeros(nS)
    P = env.P
    for i in range(max_iteration):
        delta = []
        for s in range(nS):
            temp1 = V[s]
            v_s = 0
            for i in range(len(P[s][policy[s]])):
                next_state_tuple = P[s][policy[s]][i]
                v_next_state = V[next_state_tuple[1]]
                p_next_state = next_state_tuple[0]
                reward_next_state = next_state_tuple[2]
                v_s += p_next_state * (reward_next_state + gamma * v_next_state)
                delta.append(abs(temp1 - V[s]))
            V[s] = v_s
        if max(delta) < theta:
```

```
            break

     # YOUR CODE ENDS HERE
    ############################

    return V


def policy_improvement(env, value_from_policy, policy, gamma):
    """Given the value function from policy, improve the policy.

    Parameters
    ----------
    env: OpenAI env
            env.P: dictionary
                    the transition probabilities of the environment
                    P[state][action] is tuples with (probability, nextstate,␣
 ↪reward, terminal)
            env.nS: int
                    number of states
            env.nA: int
                    number of actions

    value_from_policy: np.ndarray
            The value calculated from the policy
    policy: np.array
            The previous policy.
    gamma: float
            Discount factor.

    Returns
    -------
    new policy: np.ndarray
            An array of integers. Each integer is the optimal action to take
            in that state according to the environment dynamics and the
            given value function.
    stable policy: bool
            True if the optimal policy is found, otherwise false
    """
    ############################
    # YOUR CODE STARTS HERE
    nS = env.nS
    nA = env.nA
    q = np.zeros((nS, nA))
    new_policy = np.zeros(env.nS,dtype=int)
    P = env.P
    temp=[]
```

```
    policy_stable = []
    for s in range(nS):
        for a in range(nA):
            q_s_a = 0
            for i in range(len(P[s][a])):
                next_state_tuple = P[s][a][i]
                v_next_state = value_from_policy[next_state_tuple[1]]
                p_next_state = next_state_tuple[0]
                reward_next_state = next_state_tuple[2]
                q_s_a += p_next_state * (reward_next_state + gamma *␣
 ↪v_next_state)
            q[s][a] = q_s_a
        temp.append(policy[s])
        new_policy[s]= int(np.argmax(q[s]))
        if new_policy[s]==temp[s]:
            policy_stable.append('a')
        else:
            policy_stable.append('b')
   # print(new_policy)
    # YOUR CODE ENDS HERE
    ############################

    return new_policy, policy_stable
```

[0]:
```
## Testing out policy iteration policy for one episode
from helpers import evaluation_utils
import gym
GAME = "Taxi-v3"
env = gym.make(GAME)
#evaluation_utils.render_episode(env, policy_vi)
V_pi, policy_pi = policy_iteration(env, gamma=0.95, max_iteration=6000,␣
 ↪theta=1e-5)
```

[0]:
```
# visualize how the agent performs with the policy generated from policy␣
 ↪iteration
evaluation_utils.render_episode(env, policy_pi)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
```

```
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
```

```
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : |_: |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : :_: |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : :_: : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| : | : : |
| :_: : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| :_| : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R:_| : :G|
```

```
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
Episode reward: 6.000000
```

[0]:

```
# evaluate the performance of policy iteration over 100 episodes
print(evaluation_utils.avg_performance(env, policy_pi))
```

```
8.383838383838384
```

4 Part 3: Q-learning and SARSA

4.1 3.1 Model-free RL: Q-learning

In this part, you will implement Q-learning.

[0]:

```
def epsilon_greedy(value, e, seed=None):
    assert len(value.shape) == 1


    assert 0 <= e <= 1

    if seed != None:
        np.random.seed(seed)
    #n = len(value)
    #np.random.seed(0)
    randProb = np.random.random()  # Pick random probability between 0-1
    if randProb < e:
        a = np.random.choice(len(value))  # Select random action
```

```
    else:
        maxAction = np.argmax(value)  # Find max value estimate
        action = np.where(value == np.argmax(value))[0]
        if len(action) == 0:
            a = maxAction
        else:
            a = np.random.choice(action)
    return a



def QLearning(env, num_episodes, gamma, lr, e):
    """
    Implement the Q-learning algorithm following the epsilon-greedy exploration.
    Inputs:
    env: OpenAI Gym environment
            env.P: dictionary
                    P[state][action] are tuples of tuples tuples with␣
 ↪(probability, nextstate, reward, terminal)
                    probability: float
                    nextstate: int
                    reward: float
                    terminal: boolean
            env.nS: int
                    number of states
            env.nA: int
                    number of actions
    num_episodes: int
            Number of episodes of training
    gamma: float
            Discount factor.
    lr: float
            Learning rate.
    e: float
            Epsilon value used in the epsilon-greedy method.
    Outputs:
    Q: numpy.ndarray
    """
    nS = env.nS
    P = env.P
    nA = env.nA
    Q = np.zeros((env.nS, env.nA))
    reward_com=[]
    ############################
    # YOUR CODE STARTS HERE
    for i in range(num_episodes):
        #print(Q)
```

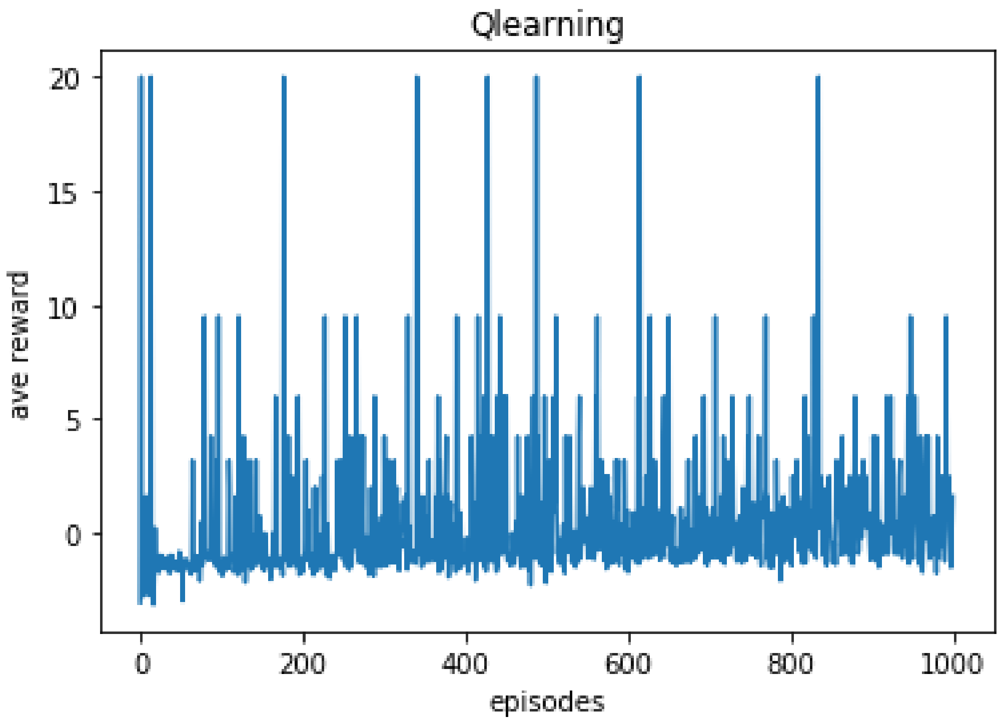
```
        reward=0
        state = np.random.randint(0, nS)
        states=0
        while True:

            action = epsilon_greedy(Q[state], e,seed=None)
            n0=len(P[state][action])
            p_next_state = np.zeros(len(P[state][action]),dtype=int)
            next_state=np.zeros(len(P[state][action]),dtype=int)
            for i in range(len(P[state][action])):
              next_state_tuple = P[state][action][i]
              next_state[i]=next_state_tuple[1]
              #print('$$',next_state_tuple[2])
              p_next_state[i] = next_state_tuple[0]
              #reward_next_state = next_state_tuple[2]

            new_state = np.random.choice(next_state, p=p_next_state.ravel())
            num=int(np.argwhere(next_state==new_state))
            reward_next_state=P[state][action][num][2]
            states+=1
            terminal_state = P[state][action][num][3]
            Q[state][action] += lr * (reward_next_state + gamma *␣
↪max(Q[new_state])- Q[state][action])
            reward+=reward_next_state
            state = new_state
            if  terminal_state==True:
              reward_ave=reward/states
              break

        reward_com.append(reward_ave)

# YOUR CODE ENDS HERE
############################
    #print(reward_com)
    return Q,reward_com



    # YOUR CODE ENDS HERE
    ############################
```

[255]:

```
import matplotlib.pyplot as plt
Q,reward_com = QLearning(env = env.env, num_episodes = 1000, gamma = 1, lr = 0.
↪1, e = 0.1)

print('Action values:')
print(Q)

############################
# YOUR CODE STARTS HERE
plt.plot(reward_com)
plt.xlabel('episodes')
plt.ylabel('ave reward')
plt.title("Qlearning")
```

```
Action values:
[[ 5.19337550e+00 -1.00000000e-01 -1.00000000e-01  3.08443570e+00
   9.13028229e+01  0.00000000e+00]
 [-3.23969166e+00 -3.58930933e+00 -3.67201534e+00 -3.55985212e+00
   9.53536173e+00 -6.29542100e+00]
 [-1.03280583e+00 -3.29505878e-02 -1.45270334e+00 -1.57747241e-01
   3.14578010e+01 -3.47432407e-03]
 …
 [-8.93662800e-01 -4.37554944e-01 -8.00000000e-01 -8.38818893e-01
  -1.00000000e+00 -1.91000000e+00]
 [-2.60000000e+00 -2.55861093e+00 -2.58903520e+00 -2.59902722e+00
  -3.00000000e+00 -3.00000000e+00]
 [ 0.00000000e+00 -1.10000000e-01 -1.00000000e-01  1.58556448e+01
   0.00000000e+00 -1.00000000e+00]]
```

[255]: Text(0.5, 1.0, 'Qlearning')




```
[2]: #Uncomment the following to evaluate your result, comment them when you␣
     ↪generate the pdf
     #from helpers import utils

     #env = gym.make('Taxi-v3')
     #policy_estimate = utils.action_selection(Q)

     #evaluation_utils.render_episode(env, policy_estimate)
```

4.2 3.2 Model-free RL: SARSA

In this part, you will implement Sarsa.

```
[0]: def SARSA(env, num_episodes, gamma, lr, e):
         """
         Implement the SARSA algorithm following epsilon-greedy exploration.
         Inputs:
         env: OpenAI Gym environment
                 env.P: dictionary
                         P[state][action] are tuples of tuples tuples with␣
     ↪(probability, nextstate, reward, terminal)
                         probability: float
                         nextstate: int
```

```
                    reward: float
                    terminal: boolean
            env.nS: int
                    number of states
            env.nA: int
                    number of actions
    num_episodes: int
            Number of episodes of training
    gamma: float
            Discount factor.
    lr: float
            Learning rate.
    e: float
            Epsilon value used in the epsilon-greedy method.
    Outputs:
    Q: numpy.ndarray
            State-action values
    """
    nS = env.nS
    P = env.P
    nA = env.nA
    Q = np.zeros((env.nS, env.nA))
    ############################
    # YOUR CODE STARTS HERE
    reward_com = []
    for i in range(num_episodes):
        #print(Q)
        reward = 0
        states=0
        state = np.random.randint(0, nS)
        while True:
           action = epsilon_greedy(Q[state], e,seed=None)
           n0=len(P[state][action])
           p_next_state = np.zeros(len(P[state][action]),dtype=int)
           next_state=np.zeros(len(P[state][action]),dtype=int)
           for i in range(len(P[state][action])):
             next_state_tuple = P[state][action][i]
             next_state[i]=next_state_tuple[1]
             p_next_state[i] = next_state_tuple[0]
             #reward_next_state = next_state_tuple[2]

           new_state = np.random.choice(next_state, p=p_next_state.ravel())
           num=int(np.argwhere(next_state==new_state))
             #print(state)
           reward_next_state=P[state][action][num][2]
           terminal_state = P[state][action][num][3]
             #print( reward_next_state)
```

```
            action_next=epsilon_greedy(Q[new_state], e,seed=None)
            Q[state][action] += lr * (reward_next_state + gamma *␣
↪Q[new_state][action_next]- Q[state][action])
            #Q[state][action] += lr * (reward_next_state + gamma *␣
↪max(Q[new_state])- Q[state][action])
            #print(Q[state][action])
            #print(Q[new_state][action_next])
            #pdb.set_trace()
            reward += reward_next_state
            states += 1
            state = new_state
            if  terminal_state==True:
              reward_ave = reward / states
              break

        reward_com.append(reward_ave)
    # YOUR CODE ENDS HERE
    ############################

    return Q,reward_com

def epsilon_greedy(value, e, seed=None):
    assert len(value.shape) == 1


    assert 0 <= e <= 1

    if seed != None:
        np.random.seed(seed)
    #n = len(value)
    #np.random.seed(0)
    randProb = np.random.random()  # Pick random probability between 0-1
    if randProb < e:
        a = np.random.choice(len(value))  # Select random action
    else:
        maxAction = np.argmax(value)  # Find max value estimate
        action = np.where(value == np.argmax(value))[0]
        if len(action) == 0:
            a = maxAction
        else:
            a = np.random.choice(action)
    return a
```

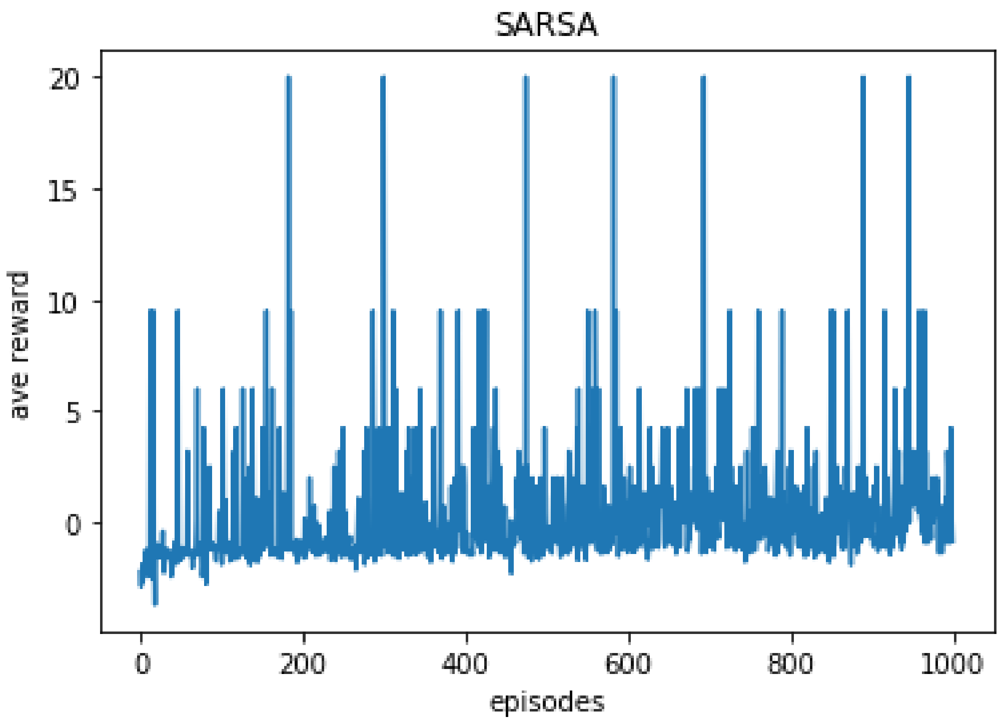
[0]:
```
def render_episode_Q(env, Q):
    """Renders one episode for Q functionon environment.


      Parameters
```

```
    ----------
    env: gym.core.Environment
      Environment to play Q function on.
    Q: np.array of shape [env.nS x env.nA]
      state-action values.
  """

  episode_reward = 0
  state = env.reset()
  done = False
  while not done:
      env.render()
      time.sleep(0.5)
      action = np.argmax(Q[state])
      state, reward, done, _ = env.step(action)
      episode_reward += reward

  print ("Episode reward: %f" %episode_reward)
```

[0]:
```
Q,reward_com = SARSA(env = env.env, num_episodes = 1000, gamma = 1, lr = 0.1, e␣
 ↪= 0.1)
print('Action values:')
print(Q)
plt.plot(reward_com)
plt.xlabel('episodes')
plt.ylabel('ave reward')
plt.title("SARSA")
```

```
Action values:
[[ 3.13426689e+00  2.78800349e+00  1.63642898e-02 -2.00000000e-01
   4.31518569e+01  1.80921795e+00]
 [-3.82198590e+00 -3.76951696e+00 -3.83169281e+00 -3.74401558e+00
  -1.12011272e+00 -4.79122233e+00]
 [-1.49408909e+00 -1.13406515e-01 -1.44254142e+00 -1.37799730e+00
   2.64419050e+01 -1.54021897e+00]
 …
 [-1.00000000e+00 -9.12288460e-01 -9.90000000e-01  3.57393073e+00
  -1.98531261e+00 -1.00000000e+00]
 [-2.86697419e+00 -2.85807880e+00 -2.91009163e+00 -5.78758333e-01
  -3.97074853e+00 -3.00000000e+00]
 [-2.00000000e-01 -1.86254179e-01 -2.00000000e-01  1.78387360e+01
  -1.00000000e+00 -1.00000000e+00]]
```

[0]:
```
Text(0.5, 1.0, 'SARSA')
```



[1]:
```
# Uncomment the following to evaluate your result, comment them when you
↪generate the pdf
#from helpers import utils
#env = gym.make('Taxi-v3')
#policy_estimate = utils.action_selection(Q)
#evaluation_utils.render_episode(env, policy_estimate)
```

[ ]:
```

```