qg_2172_HW4_Part_123

November 25, 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.

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.

[7]:
```
from gym.wrappers import Monitor
import gym
import random
import numpy as np
from google.colab import files

uploaded = files.upload()
```

```
<IPython.core.display.HTML object>


Saving utils.py to utils.py
Saving evaluation_utils.py to evaluation_utils.py
```

[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: |
+---------+
```

```
       ␣
↪---------------------------------------------------------------------------

       KeyboardInterrupt                         Traceback (most recent call␣
↪last)

       /usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in␣
↪_input_request(self, prompt, ident, parent, password)
       729             try:
   --> 730                 ident, reply = self.session.recv(self.stdin_socket,␣
↪0)
       731             except Exception:


       /usr/local/lib/python3.6/dist-packages/jupyter_client/session.py in␣
↪recv(self, socket, mode, content, copy)
       802         try:
```

```
  --> 803             msg_list = socket.recv_multipart(mode, copy=copy)
      804         except zmq.ZMQError as e:


      /usr/local/lib/python3.6/dist-packages/zmq/sugar/socket.py in␣
↪recv_multipart(self, flags, copy, track)
      465         """
  --> 466         parts = [self.recv(flags, copy=copy, track=track)]
      467         # have first part already, only loop while more to receive


      zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()


      zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()


      zmq/backend/cython/socket.pyx in zmq.backend.cython.socket._recv_copy()


      /usr/local/lib/python3.6/dist-packages/zmq/backend/cython/checkrc.pxd in␣
↪zmq.backend.cython.checkrc._check_rc()


      KeyboardInterrupt:


  During handling of the above exception, another exception occurred:


      KeyboardInterrupt                         Traceback (most recent call␣
↪last)

      <ipython-input-2-5951a1d297f9> in <module>()
       17 for step in range(steps):
       18     env.render()
  ---> 19     action = int(input("Please type in the next action:"))
       20     if action==6:
       21         break


      /usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in␣
↪raw_input(self, prompt)
      703             self._parent_ident,
      704             self._parent_header,
  --> 705             password=False,
      706         )
```

```
    707


    /usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in
 ↪_input_request(self, prompt, ident, parent, password)
    733             except KeyboardInterrupt:
    734                 # re-raise KeyboardInterrupt, to truncate traceback
--> 735                 raise KeyboardInterrupt
    736             else:
    737                 break


    KeyboardInterrupt:
```

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.

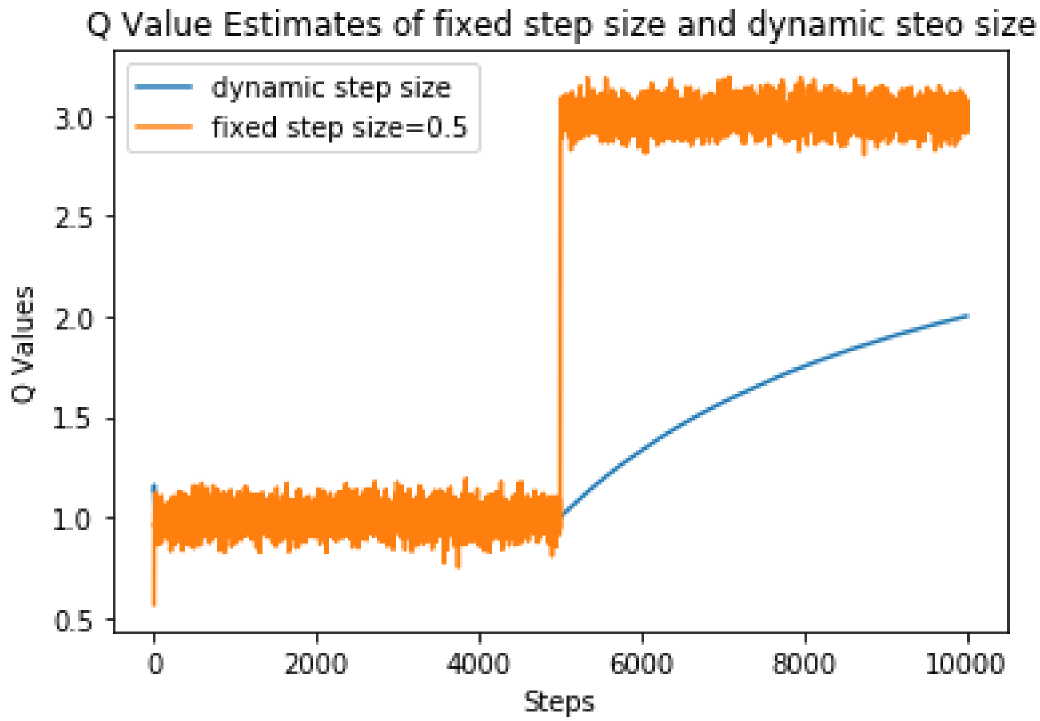
[0]:
```
def estimate(OldEstimate, StepSize, Target):
    '''An incremental implementation of average.
    OldEstimate : float
    StepSize : float
    Target : float
    '''
    NewEstimate = OldEstimate + StepSize * (Target - OldEstimate)
    return NewEstimate
```

[0]:
```
import random
import numpy as np
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 ./RLalgs/utils.py
    ############################
    # YOUR CODE STARTS HERE
    q_h[step]=estimate(q_h[step-1],1/step,r)
```

```
    q_f[step]=estimate(q_f[step-1],FixedStepSize,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
%matplotlib inline
############################
# YOUR CODE STARTS HERE
plt.title('Q Value Estimates of fixed step size and dynamic steo size')

plt.plot(q_h, label="dynamic step size")
plt.plot(q_f, label="fixed step size=0.5")
plt.legend()

plt.xlabel('Steps')
plt.ylabel('Q Values')

plt.show()


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



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]:
```
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
    probability=np.full(value.shape,e/value.shape[0])
    max_value_index=np.argmax(value)
    probability[max_value_index]+=1-e
    action=np.random.choice(value.shape[0],1,p=probability)



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

    return action
```

[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,None)
e_greedy_action=epsilon_greedy(q,0.1,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

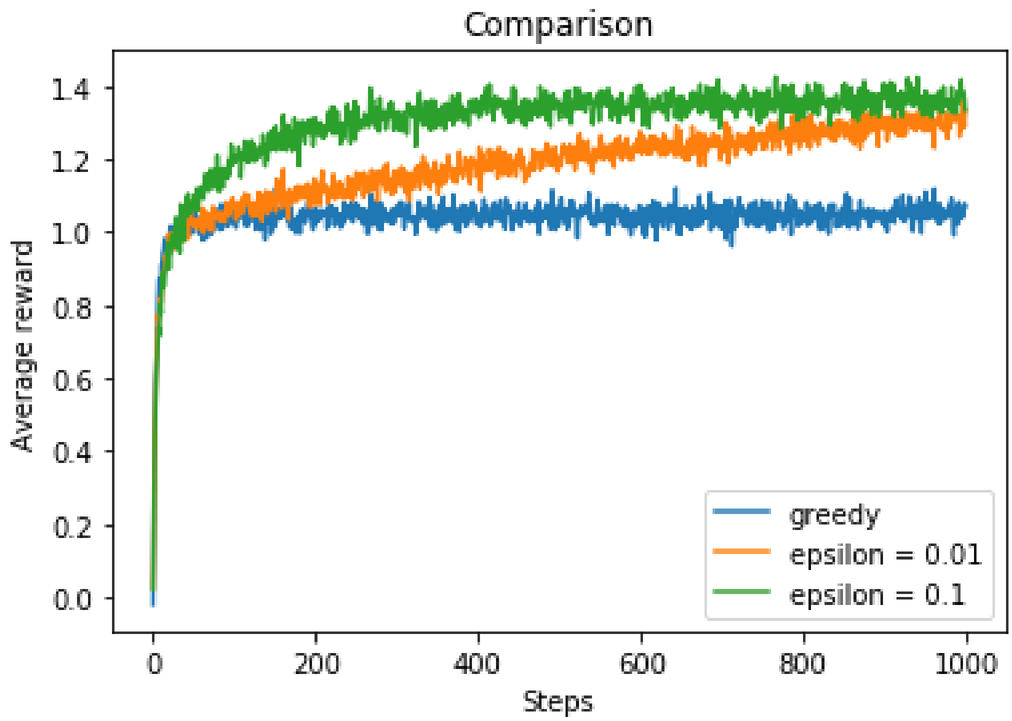
Try to reproduce Figure 2.2 (the upper one is enough) of the Sutton's book based on the experiment described in Chapter 2.3.

[0]:
```
# # Do the experiment and record average reward acquired in each time step
# ############################
# # YOUR CODE STARTS HERE
```

```
# Returns the action-value for each action at the current time step
def Q_cal(actions):
  array_size=len(actions)
  Q=[] # used to store Q value for every state

  # action[i][0] represents total reward of action i
  # action[i][1] represents number of actions taken of action i
  # for example action[3][10,5] means for action 5, it has been taken 5 times,␣
 ↪with a total reward of 10

  # update Q
  for i in range(array_size):
    if actions[i][1]==0:
      Q.append(0) # If this action hasn't been taken yet, return 0
    else:
      Q.append(actions[i][0]/float(actions[i][1]))# total reward/ number of␣
 ↪action
  return Q


def epsilon_greedy_iteration(values,e,iteration_num):
  array_size=len(values)
  actions=[]
  rewards=[]

  # initialize every action's Q value
  for i in range(array_size):
    actions.append([0.0,0])#( [0,0.0],[0,0.0],[0.0.0]...)

  for j in range(iteration_num):
    r=np.random.rand(1) # random reward
    # greedy policy: epsilon=e
    if r>e:
      action_to_take=np.argmax(Q_cal(actions)) # greedy
    else:
      action_to_take=np.random.randint(0,array_size) # equal probability for␣
 ↪evey action

    reward=np.random.normal(values[action_to_take],scale=1.0,size=1)[0] #␣
 ↪arbitry reward~(value,1)
    actions[action_to_take][0]+=reward # uodate Q value of this action
    actions[action_to_take][1]+=1

    # record each action's average reward for each step
    rewards.append(reward)
  return rewards # return a 1*n convex that store reward for each step (with␣
 ↪different actions)
```

```
def run(e,steps):
  reward_total=0
  rewards=[]
  for i in range(2000): # to obtain a stable value
    values=np.random.normal(0,1,size=10)
    reward=epsilon_greedy_iteration(values,e,steps)
    rewards.append(reward)
  means=np.mean(rewards,axis=0) # return a 1*n convex that stores mean reward␣
↪from step 1 to step n after trying 2000 times
  return means
# # YOUR CODE ENDS HERE
# ############################
```

[0]:
```
# Plot the average reward
############################
# YOUR CODE STARTS HERE
import matplotlib.pyplot as plt
%matplotlib inline
greedy=run(0.0,1000)
epsilon_001 = run(0.01, 1000)
epsilon_010 = run(0.1, 1000)
plt.plot(range(0, 1000), greedy,label='greedy')
plt.plot(range(0, 1000), epsilon_001,label='epsilon = 0.01')
plt.plot(range(0, 1000), epsilon_010,label='epsilon = 0.1')
plt.xlabel('Steps')
plt.ylabel('Average reward')
plt.title('Comparison')
plt.legend()
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
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
    """
    V = np.zeros(env.nS,dtype=float)
    ############################
    # YOUR CODE STARTS HERE
    policy=np.zeros(env.nS,dtype=int)
    for i in range(max_iteration):
        Vprime=np.zeros(env.nS,dtype=float)

        for state in range(env.nS):
            for action in range(env.nA):
                V_tmp=0
                for probability,nextstate,reward,terminal in env.
↪P[state][action]: # directly find the maximum V value for each state
                    V_tmp=V_tmp+probability*(reward+gamma*V[nextstate])
                if (Vprime[state]<V_tmp):
                    Vprime[state]=V_tmp # update real-time maximum V value for␣
↪each state
        if np.all(np.abs(V-Vprime)<theta): # judge when to converge and break␣
↪if so
            break
        V=Vprime.copy() # update V

    # Find optimal policy by given optimal V
    new_policy=np.zeros(env.nS,dtype=int)
    for state in range(env.nS):
        current_max_v=-999999
        tmp=np.zeros(env.nA,dtype=float) # save real-time V value for one state␣
↪according to different actions
        for action in range(env.nA):
          for probability,nextstate,reward,terminal in env.P[state][action]:
```

```
                tmp[action]=tmp[action]+probability*(reward+gamma*V[nextstate]) #␣
↪calculate current V for current action
            if(tmp[action]>current_max_v):# judge whether it is the maximum V␣
↪value for this state until now
                current_max_v=tmp[action] # If so, update ir
                new_policy[state]=action  # If so, update policy

    policy=new_policy.copy()



    # 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]:
```
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: |
+---------+
  (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: 6.000000
```

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

[0]:
```
8.282828282828282
```

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]:
```
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.
```

```python
    Returns:
    ----------
    value function: np.ndarray
    policy: np.ndarray
    """

    V = np.zeros(env.nS,dtype=int)
    policy = np.zeros(env.nS, dtype=int)

    for state in range(env.nS):
        policy[state]=1 # intial policy for every state
    ############################
    # YOUR CODE STARTS HERE
    for i in range(max_iteration):
        Vprime=policy_evaluation(env,policy,gamma,theta) # policy evaluation
        policy_improved,flag=policy_improvement(env,Vprime,policy,gamma) # policy improvement
        if np.all(np.abs(V-Vprime)<theta): # judge when to converge and break if so
            break
        V=Vprime.copy() # update V value for each state
        policy=policy_improved.copy() # update policy for each state


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

    return V, policy


def policy_evaluation(env, policy, gamma, theta):
    """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
    Vprime=np.zeros(env.nS) # value of next state

    while True:
        Vprime=np.zeros(env.nS,dtype=float)
        for state in range(env.nS):
            action=policy[state] # choose action according to policy
            for probability, nextstate, reward, terminal in env.
 ↪P[state][action]:
                ␣
 ↪Vprime[state]=Vprime[state]+probability*(reward+gamma*V[nextstate]) # update␣
 ↪V value for each state
        if np.all(np.abs(V-Vprime)<theta): # judge when to converge and break␣
 ↪if so
            break
        V=Vprime.copy() # update current V


    # 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
    new_policy=np.zeros(env.nS,dtype=int)

    policy_stable=True

    for state in range(env.nS):
        current_max_v=-999999
        tmp=np.zeros(env.nA,dtype=float) # save real-time V value for one state␣
↪according to different actions
        for action in range(env.nA):
          for probability,nextstate,reward,terminal in env.P[state][action]:
             ␣
↪tmp[action]=tmp[action]+probability*(reward+gamma*value_from_policy[nextstate])␣
↪# calculate current V for current action
          if(tmp[action]>current_max_v):# judge whether it is the maximum V␣
↪value for this state until now
              current_max_v=tmp[action] # If so, update ir
              new_policy[state]=action  # If so, update policy

    for i in new_policy: # If there exists one state that has no optimal policy␣
↪then break and return False!
        if new_policy[i]==0:
            policy_stable=False
```

```
            break

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

    return new_policy, policy_stable
```

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

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

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

```
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|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: 4.000000
```

[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: |
+---------+
  (South)
+---------+
|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: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
```

```
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
Episode reward: 11.000000
```

[0]:
```
# evaluate the performance of policy iteration over 100 episodes
print(evaluation_utils.avg_performance(env, policy_pi))
```

```
8.272727272727273
```

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 QLearning(env, num_episodes, gamma, lr, e):
    """Implement the Q-learning algorithm following the epsilon-greedy␣
 ↪exploration.
    Update Q at the end of every episode.

    Parameters
    ----------
    env: gym.core.Environment
      Environment to compute Q function
    num_episodes: int
      Number of episodes of training.
    gamma: float
      Discount factor.
    learning_rate: float
      Learning rate.
    e: float
      Epsilon value used in the epsilon-greedy method.


    Returns
    -------
    np.array
```

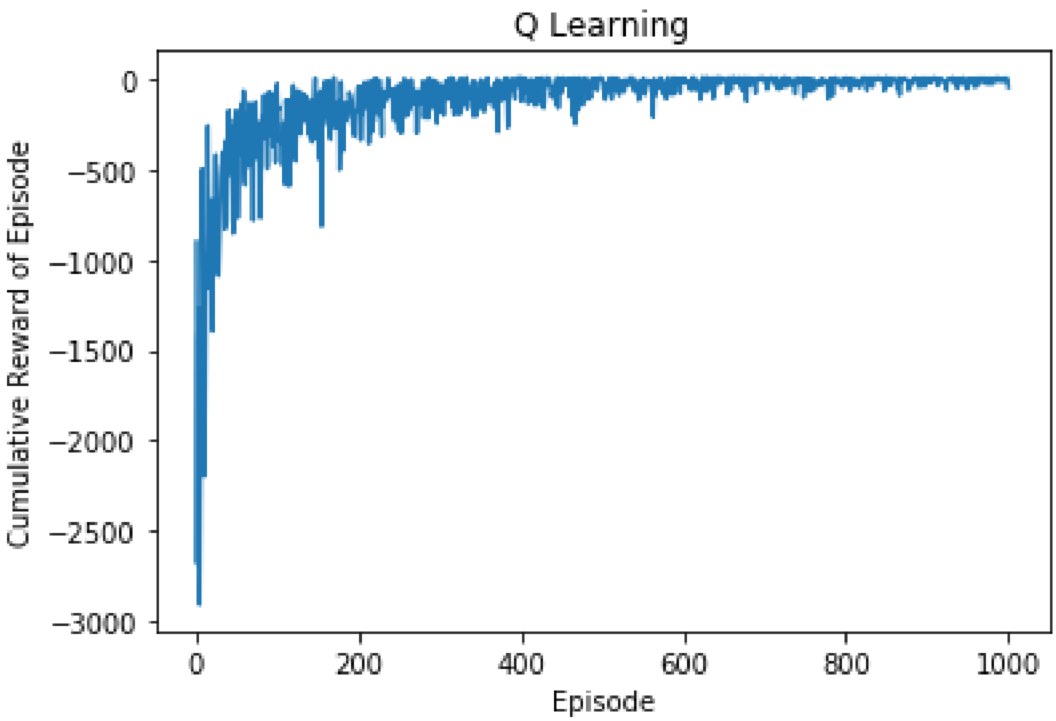
```
      An array of shape [env.nS x env.nA] representing state, action values
    """

    ############################
    # YOUR CODE STARTS HERE
    Q=np.zeros((env.nS,env.nA)) # intializee every state's Q value to 0
    Q_rewards=[] # store cumulative reward for each step

    for i in range(num_episodes):
      state=env.reset() # reset the enviroment
      reward_for_this=0
      while True:
        random_value=np.random.random()
        # epsilon-greedy policy
        if random_value>e:
          action=np.argmax(Q[state]) # choose the maximum value
        else:
          action=np.random.randint(env.nA)
        state_n, reward, done, _ = env.step(action)
        reward_for_this+=reward
        # for q learning, direct choose the action that has the maximum reward
        Q[state][action]=Q[state][action]+lr*(reward+gamma*np.
 ↪max(Q[state_n])-Q[state][action])
        state=state_n # update current state
        if done: # exit if one episode ends, break!!!!!
          break
      Q_rewards.append(reward_for_this)
    # YOUR CODE ENDS HERE
    ############################

    #return np.zeros((env.nS, env.nA))
    return Q,Q_rewards
```

[11]:
```
env = gym.make("Taxi-v3")
Q,Q_rewards = QLearning(env = env.env, num_episodes = 1000, gamma = 1, lr = 0.
 ↪1, e = 0.1)
print('Action values:')
print(Q)

import matplotlib.pyplot as plt
%matplotlib inline
plt.title('Q Learning')
plt.xlabel('Episode')
plt.ylabel('Cumulative Reward of Episode')
plt.plot(range(1,1001),Q_rewards)
```

```
Action values:
[[ 0.           0.           0.           0.           0.           0.        ]
```

```
[-1.47958384 -3.89193768 -4.20760934 -3.35222425  9.72012492 -4.4645559 ]
[-1.9794029  -0.4017635  -1.84399352 -1.7998     14.2203126  -2.97189524]
...
[-1.1        -1.00877608 -1.1         0.44346311 -1.93       -2.        ]
[-2.88024057 -2.85582085 -2.8924057  -2.26924164 -3.91231215 -6.03047715]
[ 0.37397956 -0.2        -0.2         7.78581602 -1.70920151 -1.        ]]
```

[11]: [<matplotlib.lines.Line2D at 0x7fce1858c160>]



[13]:
```python
# Uncomment the following to evaluate your result, comment them when you
↪generate the pdf
import evaluation_utils
from utils import action_selection
env = gym.make('Taxi-v3')
policy_estimate = action_selection(Q)
#render(env, policy_estimate)
evaluation_utils.render_episode(env,policy_estimate)
```

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

```
+---------+

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

```
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
|_| : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
Episode reward: 11.000000
```

4.2 3.2 Model-free RL: SARSA

In this part, you will implement Sarsa.

[0]:
```python
def SARSA(env, num_episodes, gamma, lr, e):
    """Implement the SARSA algorithm following epsilon-greedy exploration.
    Update Q at the end of every episode.

    Parameters
    ----------
    env: gym.core.Environment
      Environment to compute Q function
```

```
    num_episodes: int
      Number of episodes of training
    gamma: float
      Discount factor.
    learning_rate: float
      Learning rate.
    e: float
      Epsilon value used in the epsilon-greedy method.


    Returns
    -------
    np.array
      An array of shape [env.nS x env.nA] representing state-action values
    """

    ############################
    # YOUR CODE STARTS HERE
    Q=np.zeros((env.nS,env.nA)) # initialize every state's Q value for each␣
↪action
    Q_rewards=[]
    for i in range(num_episodes):
      state=env.reset() # reset the environment
      # epsilon-greedy policy
      if np.random.random()>e:
        action=np.argmax(Q[state])
      else:
        action=np.random.randint(env.nA)
      reward_for_this=0
      # for SARSA, when choosing next step's action, following the policy of␣
↪current action
      # eplison-greedy policy
      while True:
        state_n,reward,done,_=env.step(action)
        if np.random.random()>e:
          action_n=np.argmax(Q[state_n])
        else:
          action_n=np.random.randint(env.nA)
        # update Q[state][action]
        ␣
↪Q[state][action]=Q[state][action]+lr*(reward+gamma*Q[state_n][action_n]-Q[state][action])
        reward_for_this+=reward
        state=state_n # update state
        action=action_n # update action
        if done:
          break
      Q_rewards.append(reward_for_this)
```

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

    #return np.ones((env.nS, env.nA))
    return Q, Q_rewards
```

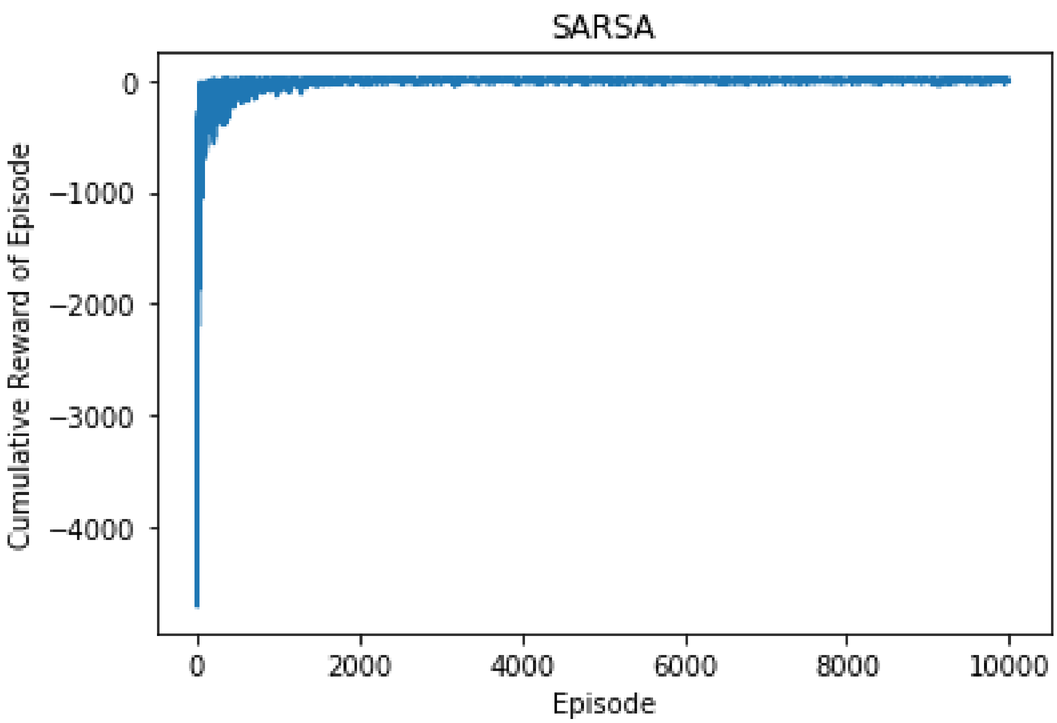
[0]:
```
import time
def render_episode_Q(env, Q):
    """Renders one episode for Q functionon environment.

      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)
```

[25]:
```
Q,Q_rewards = SARSA(env = env.env, num_episodes = 10000, gamma = 1, lr = 0.1, e␣
 ↪= 0.1)
print('Action values:')
print(Q)

import matplotlib.pyplot as plt
%matplotlib inline
plt.title('SARSA')
plt.xlabel('Episode')
plt.ylabel('Cumulative Reward of Episode')
plt.plot(range(1,10001),Q_rewards)
```

```
Action values:
[[ 0.          0.          0.          0.          0.          0.        ]
 [ 0.57748287  0.96270149 -1.91369218  2.19509429  6.68944349 -7.37264556]
 [ 4.39720668  7.42154763  5.98620518  6.67607814 12.72281144  0.07580237]
 ...
```

```
[10.02880443 14.29725852  9.05688499  4.85573763  0.64193794 -0.12591817]
[-3.3763171   2.85969628 -3.25709774 -3.66525514 -5.44813421 -6.00780589]
[-0.361       3.96131186  4.44763346 18.73542553 -1.9        -2.57087522]]
```

[25]: [<matplotlib.lines.Line2D at 0x7fce17bcd080>]



[26]:
```python
# Uncomment the following to evaluate your result, comment them when you
 ↪generate the pdf
from utils import action_selection
import evaluation_utils
env = gym.make('Taxi-v3')
#policy_estimate = action_selection(Q)
#render(env, policy_estimate)
render_episode_Q(env,Q)
```

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

+---------+
```

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

```
|R: | : :G|
| : | :_: |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | :_:G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
Episode reward: 11.000000
```