E 6885: Reinforcement Learning

Tashrif Billah (tb2699)

PhD Student, Electrical Engineering

1. (1)

For value iteration, we have used the following algorithm in Lecture-3, page 17-

Initialize array v arbitrarily (e.g., $v(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $temp \leftarrow v(s)$
    $v(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$
    $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \gamma v(s')\Big]$

Please see the attached code vi.py.

1.(2)
On the other hand, for policy iteration, we have used the following algorithm from Lecture-3, page 12-

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $temp \leftarrow v(s)$
   $v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \Big[r(s, \pi(s), s') + \gamma v(s') \Big]$
   $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $temp \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s, a) \Big[r(s, a, s') + \gamma v(s') \Big]$
   If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ $false$
   If *policy-stable*, then stop and return v and π; else go to 2

Please see the attached code pi.py.

In the main functions, they produce outputs for optimal value function and optimal policy. Also, they print the average rewards on 100 episodes using the optimal policy.

1. (3) The optimal value function for both the value iteration and policy iteration-

```
[ 119.877   82.823  146.66   122.89    55.584   82.823   79.786   96.385
   72.797   55.584  146.66    96.385   48.215   55.584   79.786  122.89
  127.239   88.234  155.432  130.411  112.883   77.682  138.327  115.746
   59.562   88.234   85.038  102.511   68.158   51.805  138.327   90.566
   51.805   59.562   85.038  130.411  119.877   93.931  146.66   138.327
   93.931   63.75   115.746   96.385   72.797  106.239  102.511  122.89
   63.75    48.215  130.411   85.038   55.584   63.75    90.566  138.327
   99.927  112.883  138.327  146.66    88.234   59.562  108.959   90.566
   77.682  112.883  108.959  130.411   59.562   44.804  122.89    79.786
   59.562   68.158   96.385  146.66    93.931  119.877  130.411  155.432
   82.823   55.584  102.511   85.038   82.823  119.877  115.746  138.327
   55.584   41.564  115.746   74.796   55.584   63.75    90.566  138.327
   88.234  127.239  122.89   146.66   112.883   77.682  138.327  115.746
   59.562   88.234   85.038  102.511   77.682   59.562  155.432  102.511
   51.805   59.562   85.038  130.411  119.877   93.931  164.665  138.327
  106.239   72.797  130.411  108.959   63.75    93.931   90.566  108.959
   72.797   55.584  146.66    96.385   55.584   63.75    90.566  138.327
  112.883   99.927  155.432  146.66    99.927   68.158  122.89   102.511
   68.158   99.927   96.385  115.746   68.158   51.805  138.327   90.566
   59.562   68.158   96.385  146.66   106.239  106.239  146.66   155.432
   93.931   63.75   115.746   96.385   72.797  106.239  102.511  122.89
   63.75    48.215  130.411   85.038   63.75    72.797  102.511  155.432
   99.927  112.883  138.327  164.665   88.234   59.562  108.959   90.566
   77.682  112.883  108.959  130.411   59.562   44.804  122.89    79.786
   59.562   68.158   96.385  146.66    93.931  119.877  130.411  155.432
  106.239   72.797  130.411  108.959   55.584   82.823   79.786   96.385
   82.823   63.75   164.665  108.959   55.584   63.75    90.566  138.327
  112.883   88.234  174.385  146.66    99.927   68.158  122.89   102.511
   59.562   88.234   85.038  102.511   77.682   59.562  155.432  102.511
   59.562   68.158   96.385  146.66   106.239   93.931  164.665  155.432
```

```
 93.931   63.75   115.746   96.385   72.797  106.239  102.511  122.89
 63.75    48.215  130.411   85.038   63.75    72.797  102.511  155.432
 99.927  112.883  138.327  164.665   88.234   59.562  108.959   90.566
 77.682  112.883  108.959  130.411   59.562   44.804  122.89    79.786
 59.562   68.158   96.385  146.66    93.931  119.877  130.411  155.432
106.239   72.797  130.411  108.959   55.584   82.823   79.786   96.385
 82.823   63.75   164.665  108.959   55.584   63.75    90.566  138.327
112.883   88.234  174.385  146.66    99.927   68.158  122.89   102.511
 59.562   88.234   85.038  102.511   77.682   59.562  155.432  102.511
 59.562   68.158   96.385  146.66   106.239   93.931  164.665  155.432
 93.931   63.75   115.746   96.385   63.75    93.931   90.566  108.959
 72.797   55.584  146.66    96.385   63.75    72.797  102.511  155.432
 99.927   99.927  155.432  164.665   88.234   59.562  108.959   90.566
 68.158   99.927   96.385  115.746   68.158   51.805  138.327   90.566
 68.158   77.682  108.959  164.665   93.931  106.239  146.66   174.385
 82.823   55.584  102.511   85.038   72.797  106.239  102.511  122.89
 63.75    48.215  130.411   85.038   63.75    72.797  102.511  155.432
 88.234  112.883  138.327  164.665   99.927   68.158  122.89   102.511
 51.805   77.682   74.796   90.566   88.234   68.158  174.385  115.746
 51.805   59.562   85.038  130.411  106.239   82.823  184.615  138.327
 93.931   63.75   115.746   96.385   55.584   82.823   79.786   96.385
 72.797   55.584  146.66    96.385   55.584   63.75    90.566  138.327
 99.927   88.234  155.432  146.66    88.234   59.562  108.959   90.566
 59.562   88.234   85.038  102.511   68.158   51.805  138.327   90.566
 59.562   68.158   96.385  146.66    93.931   93.931  146.66   155.432

 82.823   55.584  102.511   85.038   63.75    93.931   90.566  108.959
 63.75    48.215  130.411   85.038   72.797   82.823  115.746  174.385
 88.234   99.927  138.327  184.615   77.682   51.805   96.385   79.786
 68.158   99.927   96.385  115.746   59.562   44.804  122.89    79.786
 68.158   77.682  108.959  164.665   82.823  106.239  130.411  174.385
 93.931   63.75   115.746   96.385   48.215   72.797   70.057   85.038
 93.931   72.797  184.615  122.89    48.215   55.584   79.786  122.89
 99.927   77.682  195.385  130.411   88.234   59.562  108.959   90.566
 51.805   77.682   74.796   90.566   68.158   51.805  138.327   90.566
 51.805   59.562   85.038  130.411   93.931   82.823  146.66   138.327
 82.823   55.584  102.511   85.038   55.584   82.823   79.786   96.385
 63.75    48.215  130.411   85.038   55.584   63.75    90.566  138.327
 88.234   88.234  138.327  146.66    77.682   51.805   96.385   79.786
 59.562   88.234   85.038  102.511   59.562   44.804  122.89    79.786
 77.682   88.234  122.89   184.615   82.823   93.931  130.411  195.385
 72.797   48.215   90.566   74.796   63.75    93.931   90.566  108.959
 55.584   41.564  115.746   74.796   72.797   82.823  115.746  174.385
 77.682   99.927  122.891  184.615]
```

The above set of values are part of a $500x1$ dimension vector corresponding to 500 states. However, Python console plots them in the above format to fit in the terminal.

The optimal policy for both the value iteration and policy iteration-

$\pi^*(a|s) =$

```
[4 4 4 4 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0 0 3
 0 0 0 0 0 0 0 2 2 2 2 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 2 2 2 2 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 4 4 4 4 0 0 0 0 0 0 0 0 0 5 0 0 1 1 1 1 2 2 2 2 0 0 0
 0 0 0 0 0 1 2 0 0 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 1 2 0 0 3 3 3 3 2 2 2 2
 0 0 0 0 0 0 0 0 3 2 0 0 3 3 3 3 2 2 2 2 0 0 0 0 0 0 0 0 3 2 0 0 3 3 3 3 1
 1 1 1 0 0 0 0 0 0 0 0 3 1 0 0 1 1 1 1 2 2 2 2 0 0 0 0 2 2 2 2 1 2 0 2 1 1
 1 1 2 2 2 2 3 3 3 3 2 2 2 2 1 2 3 2 1 1 1 1 2 2 2 2 3 3 3 3 2 2 2 2 1 2 3
 2 1 1 1 1 2 2 2 2 3 3 3 3 0 0 0 0 1 2 3 0 1 1 1 1 1 1 1 1 3 3 3 3 0 0 0 0
 1 1 3 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 1 1 1 1 1 2 2 2 2 1 1 1 1 2
 2 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1
 1 1 0 0 0 0 1 2 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 0 1 1 1 1 1 1
 1 4 4 4 4 1 1 1 1 1 1 5 1 1 1 1 1 2 2 2 2 1 1 1 1 2 2 2 2 1 2 1 2 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 4 4 4 4 1 2 1 5 1
 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 1 1 1 3]
```

The above set of actions are part of a $500x1$ dimension vector corresponding to 500 states. However, Python console plots them in the above format to fit in the terminal.

Result comparison

Algorithm	Average reward	# of iteration to converge	Convergence time in seconds
Value iteration	8.6868	146	0.6191
Policy iteration	8.5757	17	4.6816

Note: If the code is run, it may produce slightly different value for the above, given the state space is probabilistic.

Discussion

1. The value iteration takes more number of iteration while policy iteration takes less. But, the time of convergence of the latter is greater than the former.

2. The above observation is justified, because in each iteration of the policy iteration, a complete value iteration is performed. This step is called policy evaluation. Thereby, a lot of value iteration yields a larger convergence time for policy iteration.

3. However, the average reward of the both the policies are nearly same. It implies that, whatever policy we choose, that would yield the same result. This attribute is further justified by the exact similarity of the optimal value function and optimal policy yielded by both value and policy iteration.

2.(4)

For SARSA learning, we have used the following policy from Lecture-5, slide-21

With probability $1-\epsilon$ choose the greedy action

With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Please see the attached code ql_sarsa.py.

The output is produced in required format. Since the $Q(s,a)$ is of big dimension 500x6, the output is not shown here. Please see the attached QL and SARSA Output.txt

2.(5)
For Q learning, we have used the same greedy policy as above. However, for the complete algorithm, we have used the following pseudocode from page 145, Sutton's book-

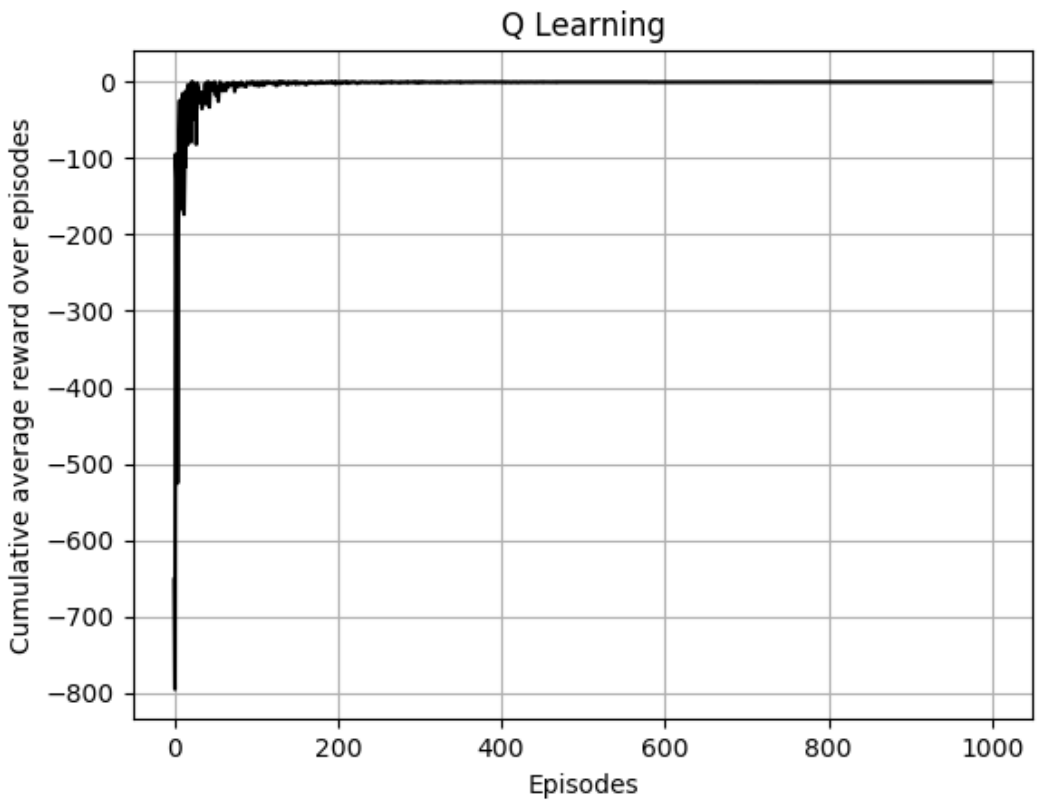
With probability $1-\epsilon$ choose the greedy action
With probability ϵ choose an action at random

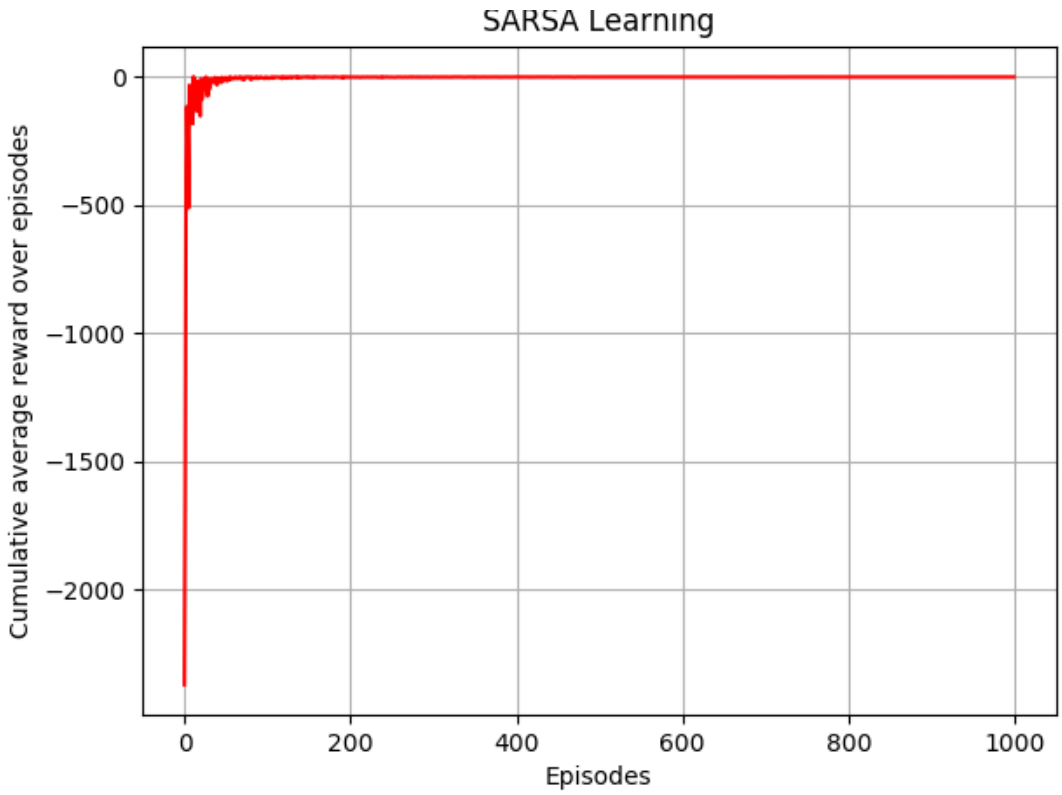
$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$;
  until S is terminal

Please see the attached code ql_sarsa.py.

The output is produced in required format. Since the $Q(s,a)$ is of big dimension 500x6, the output is not shown here. Please see the attached QL and SARSA Output.txt


Q Learning
Cumulative average reward over episodes
0
−100
−200
−300
−400
−500
−600
−700
−800
0
200
400
600
800
1000
Episodes


SARSA Learning
Cumulative average reward over episodes
0
−500
−1000
−1500
−2000
0
200
400
600
800
1000
Episodes

Convergence time comparison

Algorithm	Convergence time in seconds
Q-Learning	4.2066
SARSA Learning	4.3681

Note: If the code is run, it may produce slightly different value for the above, given the state space is probabilistic.

Discussion

1. As we have taken, the cumulative average reward is given by-

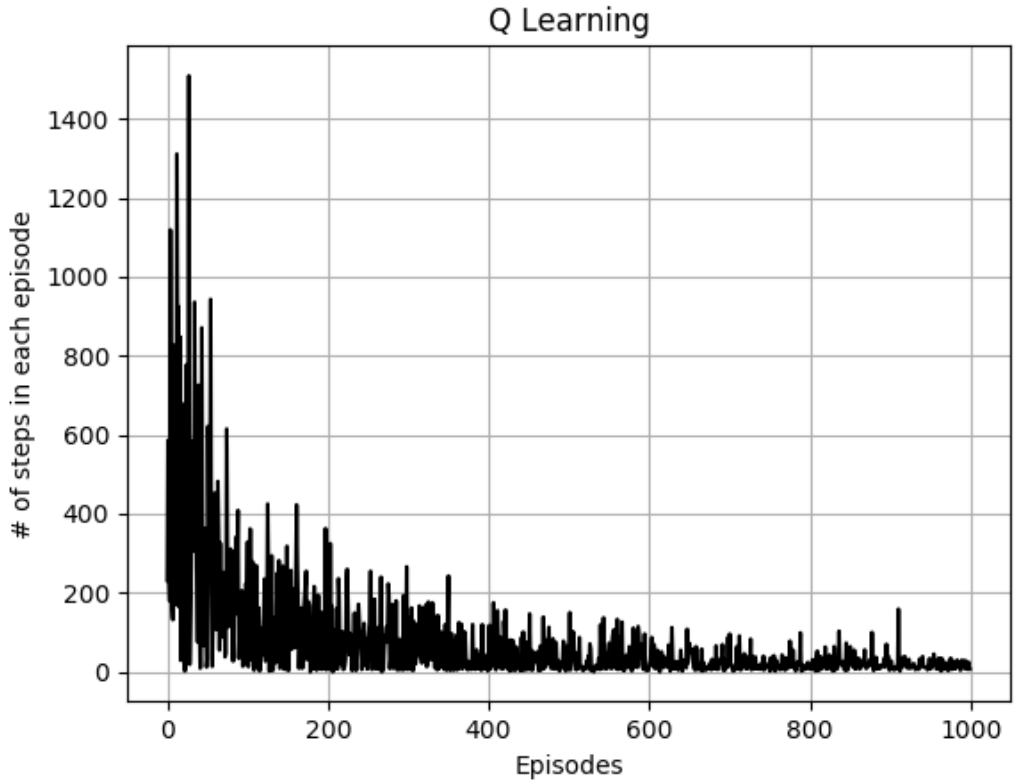
$$R_{cavg} = \frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{L_i} R_l(s, a, s')$$

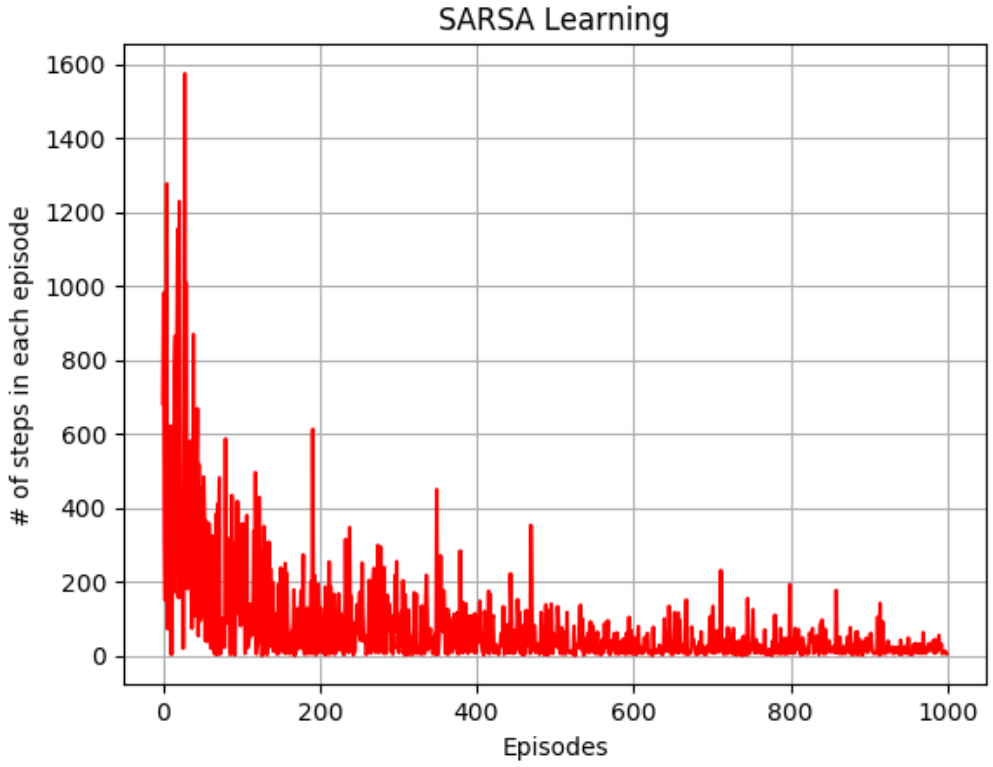
where, n is the number of iteration performed up to any point, and L_i is the length of i^{th} episode, $i = 1, 2, \ldots, n$. $R_l(s, a, s')$ is the reward at l^{th} step of the i^{th} episode.

2. As we can see in the vertical axis of the graph, the Q learning cumulative reward graph converges to the steady state value quicker than the SARSA learning.

3. The action-value function $Q(s, a)$ for Q-learning approximates q^*, the optimal action-value function, irrespective of the policy being followed. This attribute is conducive to the early convergence of Q learning compared to SARSA.

2.(7)


Q Learning
# of steps in each episode
Episodes
0
200
400
600
800
1000
1200
1400


SARSA Learning
# of steps in each episode
Episodes
0
200
400
600
800
1000
1200
1400
1600

Discussion

The SARSA learning algorithm takes more number of steps to complete an episode compared to Q learning as can be seen in the above figures. After an initial transient, Q-learning learns values for the optimal policy. However, due to greedy choice of $Q(s,a)$ to update at each step, it sometimes makes a bad decision by running into more negative rewards in future.

On the other hand, SARSA learning algorithm makes greedy action selection at each step of an episode and learns the longer but safer path towards the terminal state. For this reason, SARSA has relatively more number of steps in each episode compared to the Q learning algorithm.

Note: The above description is the expected behavior for both the learning algorithms. However, the above two figures do not completely point it out. Rather, for both the learning algorithms, the number of steps decay look similar.