CSE4/510: Reinforcement Learning

Project 1 - Building Reinforcement Learning Environment

Vinita Venktesh Chappar
University at Buffalo
Buffalo, New York
vchappar@buffalo.edu

1 Abstract

In this project, a reinforcement learning algorithm SARSA is implemented to enable an agent to learn and solve an environment such that we get a maximized reward. The agent learns about the environment by taking suitable actions and progressing onto the next states in the environment. After enough exploration, the agent can then take optimal path to solve the environment based on his previous knowledge.

2 Modules of the project

2.1 Environment

An environment in RL is the physical world in which the agent acts in. This environment can be of two types:

2.1.1 Deterministic

A deterministic environment is the one in which agent's current state and current action uniquely determine the outcome. This type of environment is fully observable.

2.1.2 Stochastic

An environment is called Stochastic if it is random in nature. This means that the current state and action do not determine the outcome. This type of environment is difficult to analyse as you never know in which state the agent could end up being. Here, you have to store the outcomes of previous states as well to be able to predict your agent's next state.
Example: Dice games. You cannot predict your next state by your dice throwing action.

In this project we have a 5x5 grid world. We have obstacles and a goal state. Rewards are assigned such that the agent knows to get the maximized reward it has to reach the goal state. Moreover, most of the times the agent will take a random action based on the action set to add some randomness to the environment
Following is the defined grid env:

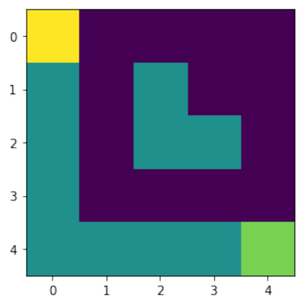


Figure 1: Environment in reset state

Blocks in the env:

- yellow block - Agent's current position
- green block - Goal state
- grey blocks - Obstacles

2.2 Actions

Actions define the agent's behaviour of moving in a certain direction.
The Action set is defined as:
Action = A ∈ { down, up, right, left }
In numerical format:
Action = A ∈ { 0, 1, 2, 3 }

2.3 States

In a 5x5 grid environment, we have 25 states. We have starting state at (0,0). The goal state is at (4,4). We also have obstacles as shown in the environment diagram.

S0	S1	S2	S3	S4
S5	S6	S7	S8	S9
S10	S11	S12	S13	S14
S15	S16	S17	S18	S19
S20	S21	S22	S23	S24

Table 1: State Set

2.4 Rewards

Rewards are returned outcomes when the agent takes certain action and end's up in a certain state.
Here, the reward set is defined as:
R ∈ {-20, -10, 0, +100}

- The **-20** reward is returned when the agent hits an **obstacle** state. This will add onto the agent's experience that it should avoid this state.

- **+100** reward is given when agent reaches the **goal state**.

- Now, the **-10** reward is given in two cases - when the agent takes an action and ends up in the **same state** or when it **goes away from the goal state**. This is added to ensure that agent always keeps moving forward towards the goal state and does not stay idle.

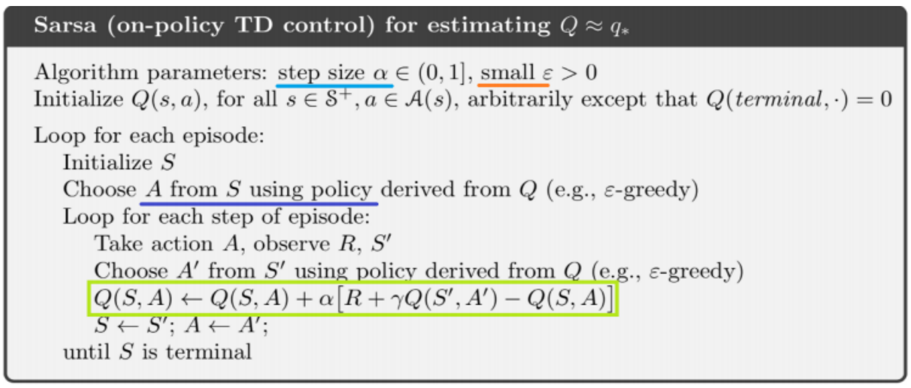
- The rest of the states return just a 0.

3 SARSA Algorithm

SARSA: State, Action, Reward, nextState, nextAction

SARSA is a reinforcement learning algorithm which enables agent to explore the environment and find the optimal path to reach the goal state. We calculate the action value function Q for each action and store it in a table called as Q-table. This Q-table will be used by the agent while testing. It will find the next action and state using the Q-table to find the optimal policy.

Here, main function of updating the Q value depends on current state **'S1'**, current action choosen **'A1'**, reward returned **'R'**, the next state **'S2'** and next action choosen **'A2'**.
Hence the acronym **'SARSA'**.
Below is the pseudo code for the SARSA algorithm:



```
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal,\cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

Figure 2: Pseudo code adapted as given in "Lecture 6: Temporal Difference TD(0), SARSA, Q-learning" pdf

In this project, the ε - greedy policy for choosing the action is implemented. The epsilon greedy policy works as follows -

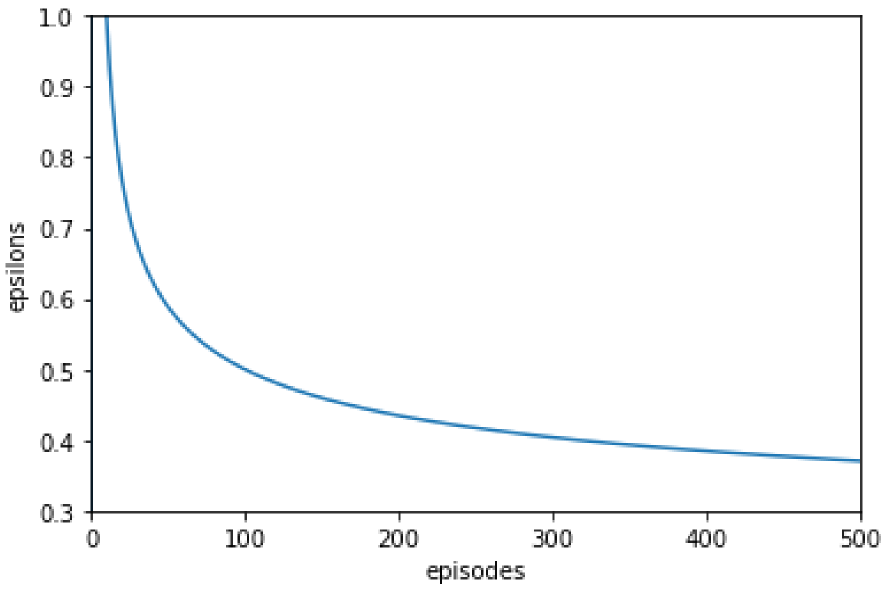
- For ε, execute a random action (Exploration)
- For 1 - ε, execute optimal action (Action which has maximum Q value in the Q table) (Exploitation)

Here, another thing to consider was that, as number of episodes increase, the randomness of the actions should reduce. Since towards the end of episodes we want the agent to converge more towards the optimal policy.

To achieve this, the **get_epsilon** method is implemented which reduces the value of epsilon as the number of episodes increases. This will reduce the chance of agent picking a random action when it has sufficiently trained.

epsilon = 1 / math.log10(episode + 0.00001)

In the below graph, you can see epsilon values reducing as the episodes increases. Min epsilon value is around 0.3



Some features of SARSA algorithm:

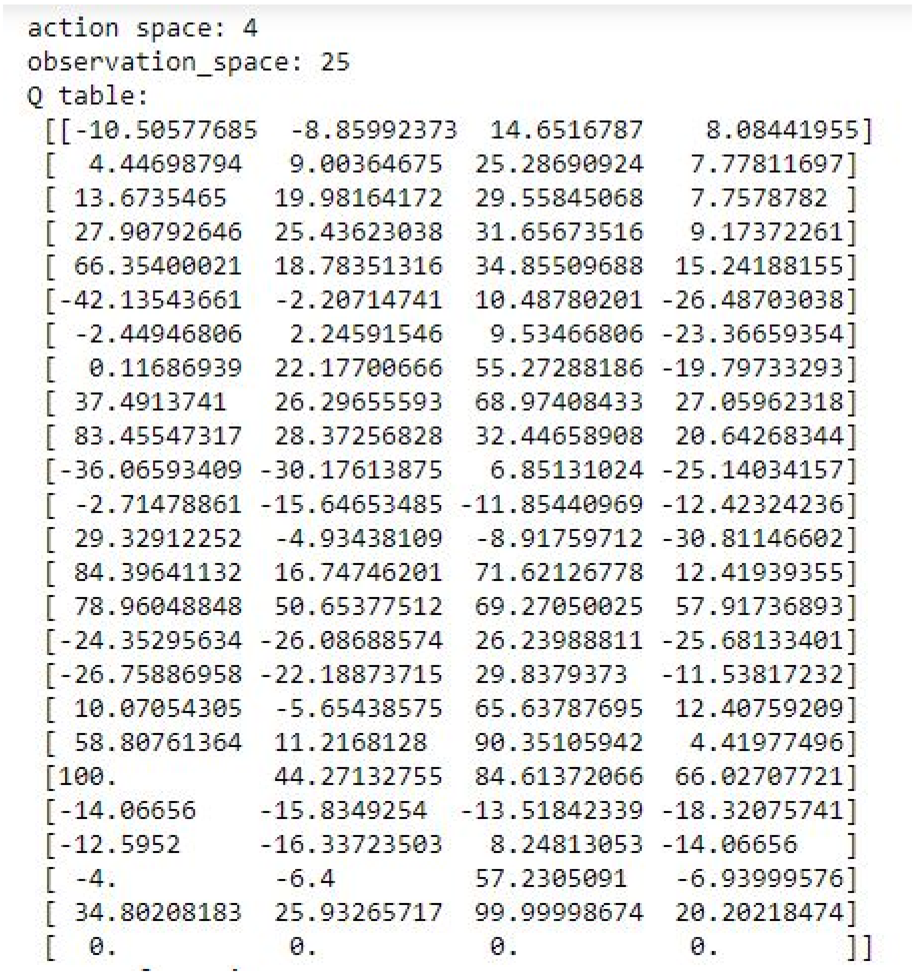
- On policy algorithm - This means that while learning an optimal policy, it will consider the current estimate of the policy and generate an optimal behaviour towards it.
- Iterative algorithm - SARSA keeps on updating the Qtable iteratively with each episode.
- It converges to an optimal policy as long as each state-action pair is visited 'n' number of times, 'n' preferably being 'infinite'.

Difference between Q-learning and SARSA:

- In Q-learning for estimating Value of Q, it directly approximates the optimal action-val function independently of the policy being followed (i.e it is an Off policy algorithm)
- QLearning will only seem to follow e-greedy policy since it has the max function in the Q value estimation formula.
- Sarsa converges faster than Q learning. So in situations where agent's performance matters most, Sarsa is more preferable.

4 Results

After executing the learn method, the final Q table is returned with optimal Q values for each state-action pair.



```
action space: 4
observation_space: 25
Q table:
 [[-10.50577685  -8.85992373  14.6516787    8.08441955]
 [  4.44698794   9.00364675  25.28690924   7.77811697]
 [ 13.6735465   19.98164172  29.55845068   7.7578782 ]
 [ 27.90792646  25.43623038  31.65673516   9.17372261]
 [ 66.35400021  18.78351316  34.85509688  15.24188155]
 [-42.13543661  -2.20714741  10.48780201 -26.48703038]
 [ -2.44946806   2.24591546   9.53466806 -23.36659354]
 [  0.11686939  22.17700666  55.27288186 -19.79733293]
 [ 37.4913741   26.29655593  68.97408433  27.05962318]
 [ 83.45547317  28.37256828  32.44658908  20.64268344]
 [-36.06593409 -30.17613875   6.85131024 -25.14034157]
 [ -2.71478861 -15.64653485 -11.85440969 -12.42324236]
 [ 29.32912252  -4.93438109  -8.91759712 -30.81146602]
 [ 84.39641132  16.74746201  71.62126778  12.41939355]
 [ 78.96048848  50.65377512  69.27050025  57.91736893]
 [-24.35295634 -26.08688574  26.23988811 -25.68133401]
 [-26.75886958 -22.18873715  29.8379373  -11.53817232]
 [ 10.07054305  -5.65438575  65.63787695  12.40759209]
 [ 58.80761364  11.2168128   90.35105942   4.41977496]
 [100.          44.27132755  84.61372066  66.02707721]
 [-14.06656    -15.8349254  -13.51842339 -18.32075741]
 [-12.5952     -16.33723503   8.24813053 -14.06656   ]
 [ -4.          -6.4         57.2305091   -6.93999576]
 [ 34.80208183  25.93265717  99.99998674  20.20218474]
 [  0.           0.           0.           0.        ]]
```

Figure 3: Final Q-table after 500 episodes

After the learning phase of the agent, we get to the testing phase. As shown in the figures below, we get an optimal path from the agent and at the goal state we get a total reward of +100. Hence, the agent has successfully learnt the environment and avoided the obstacles to get the maximum reward possible.

Result simulation:

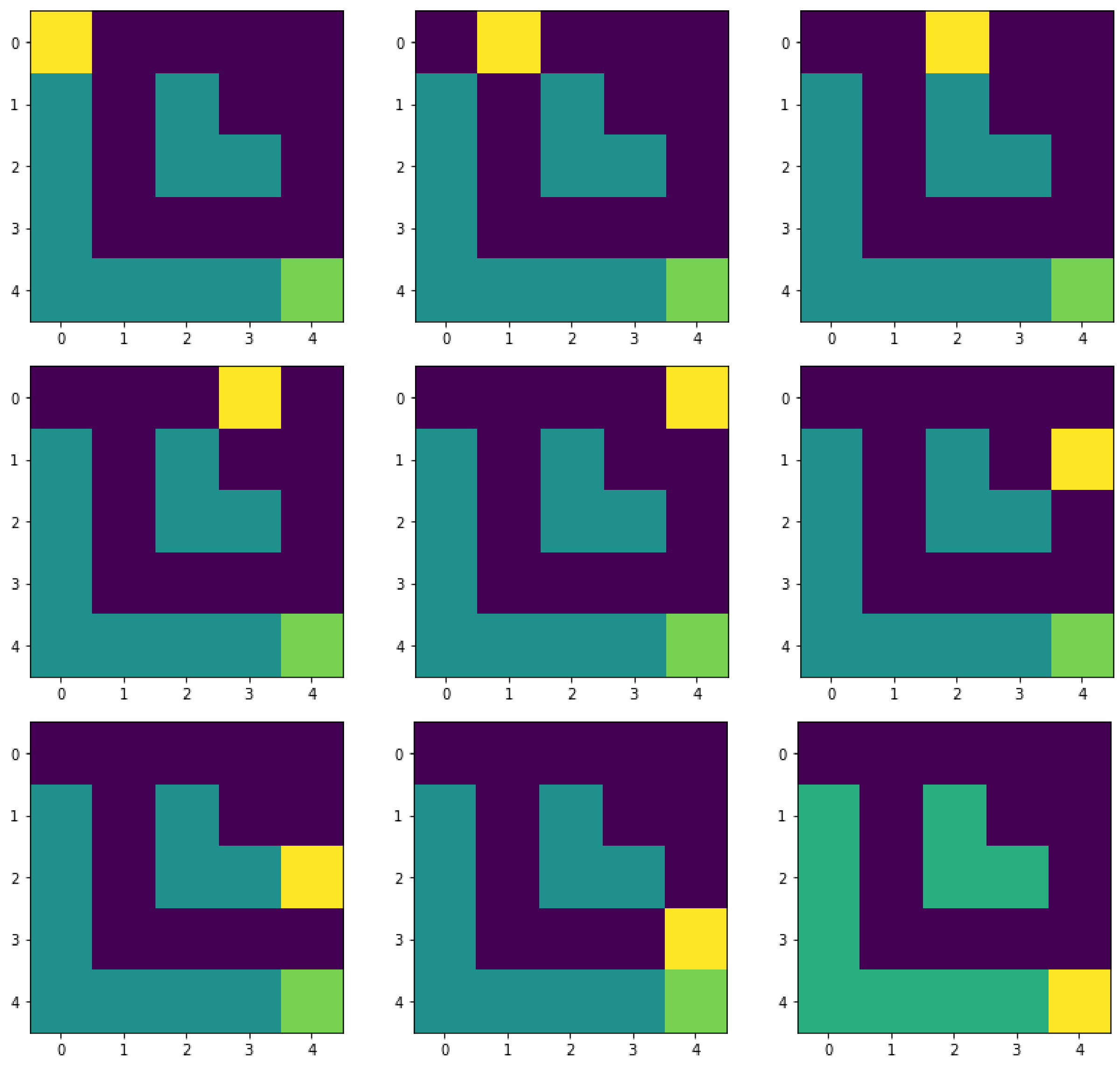
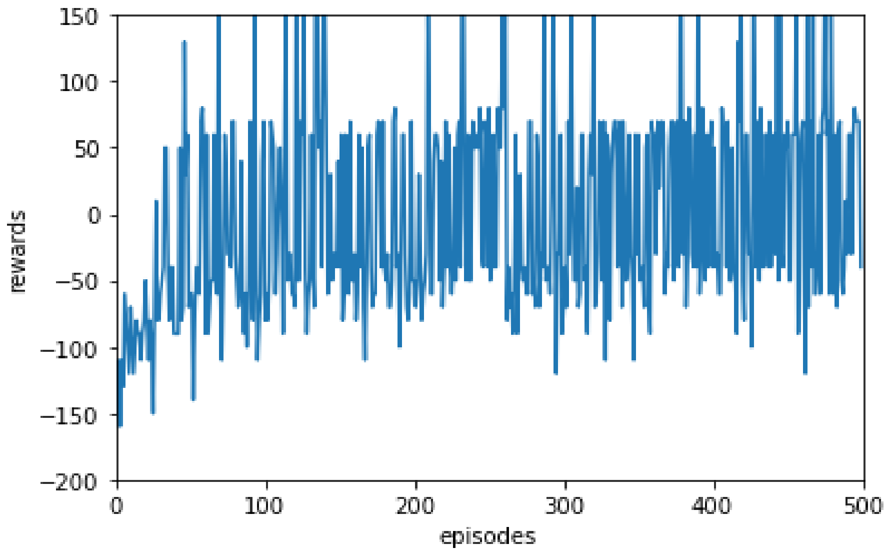


Figure 4: Grid world solution

In the below graph, we can see that as the episodes increase, the total reward increases showing that our agent is learning the environment.



5 References

[1] Lecture slides, CSE510 Introduction to Reinforcement Learning, Summer 2020

[2] Decayed epsilon greedy, https://www.oreilly.com/library/view/r-machine-learning/9781789807943/
242b-4e03-9063-2e9ad758b274.xhtml

[3] Richard S. Sutton and Andrew G. Barto, "Reinforcement learning: An introduction", Second Edition, MIT Press, 2019