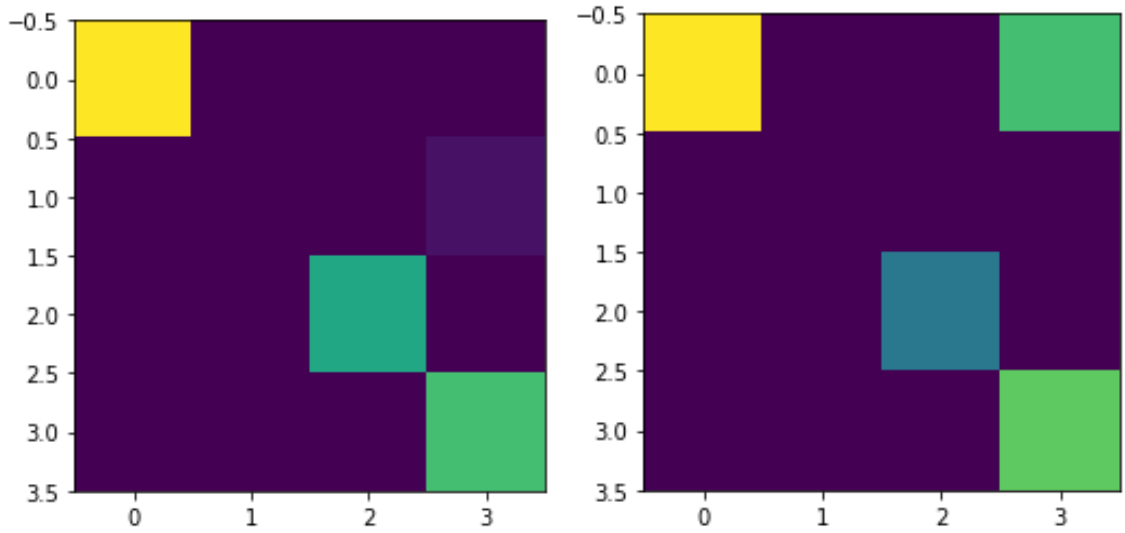
Machine Learning Assignment 4

1. Specification of Environment:

- Total states = 16 (a 4x4 grid)
- We have 4 rewards which are 0,1,2, -0.5
- Number of actions: 4 (up, down, left, right)
- Agent starts position is (0,0) and goal position is (3,3)

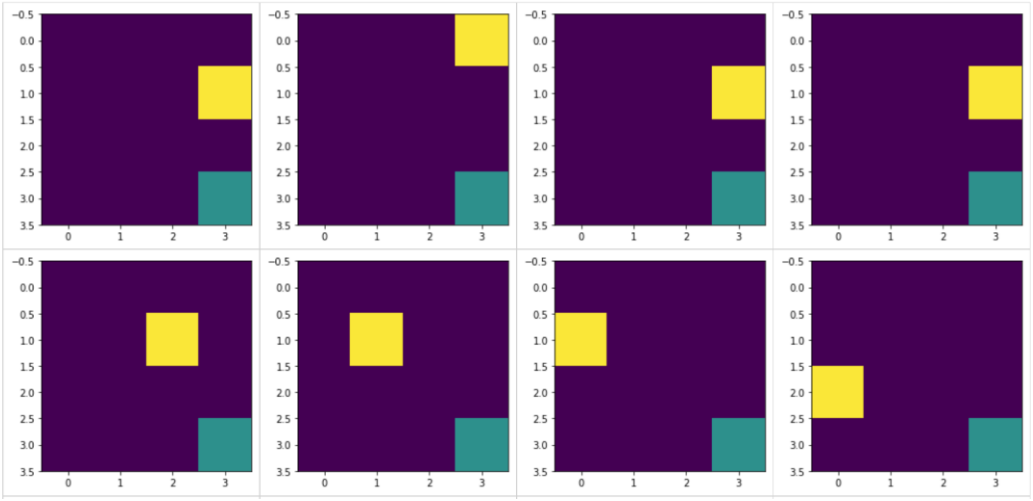
2. Visualization of the Environment



The different Rewards are represented by different color in the environment

Simulation for 10 different random time steps

```
1.85
0
1.85
0
1.85
0
0
0
0
0
0
0
0
1
0
0
-0.5
-0.5
-0.5
Total Reward accumulated  5.050000000000001
```

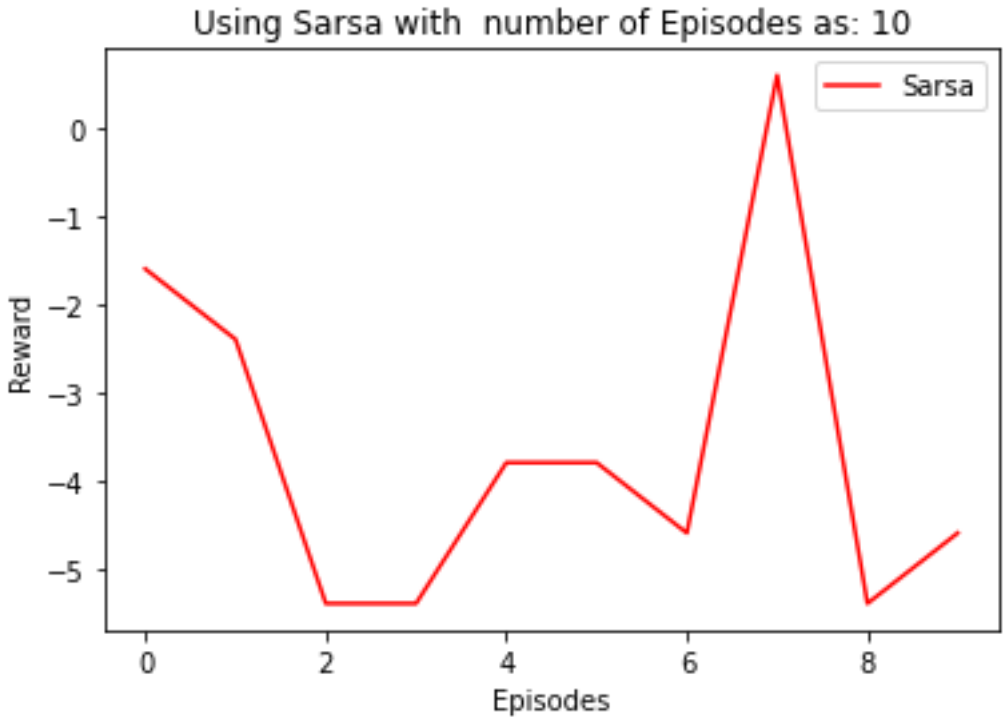


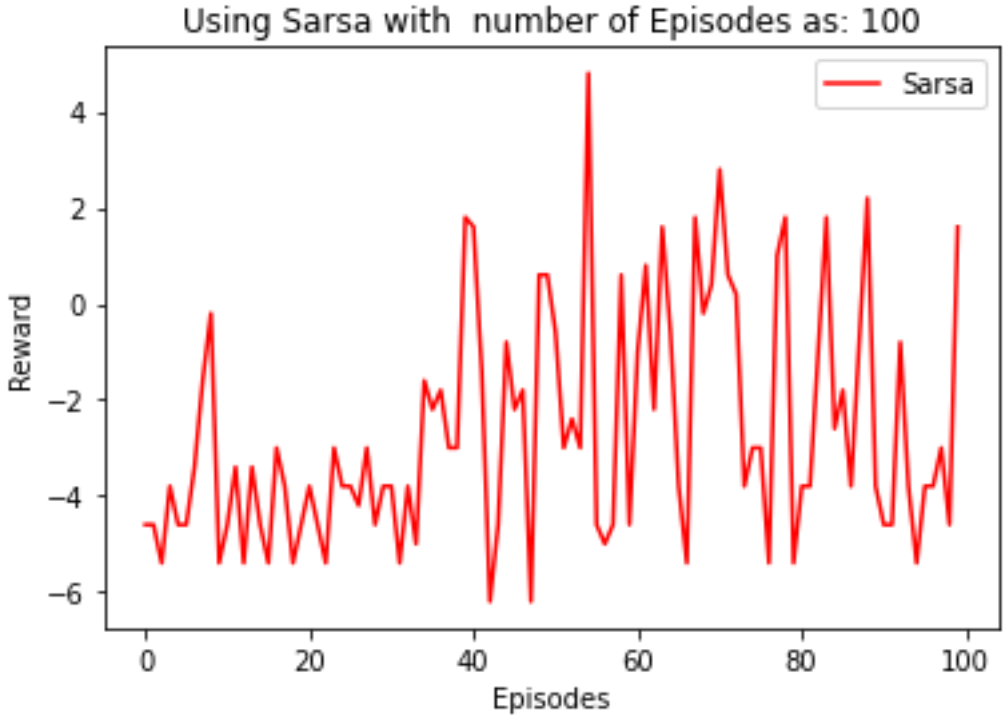
3. Safety in environment

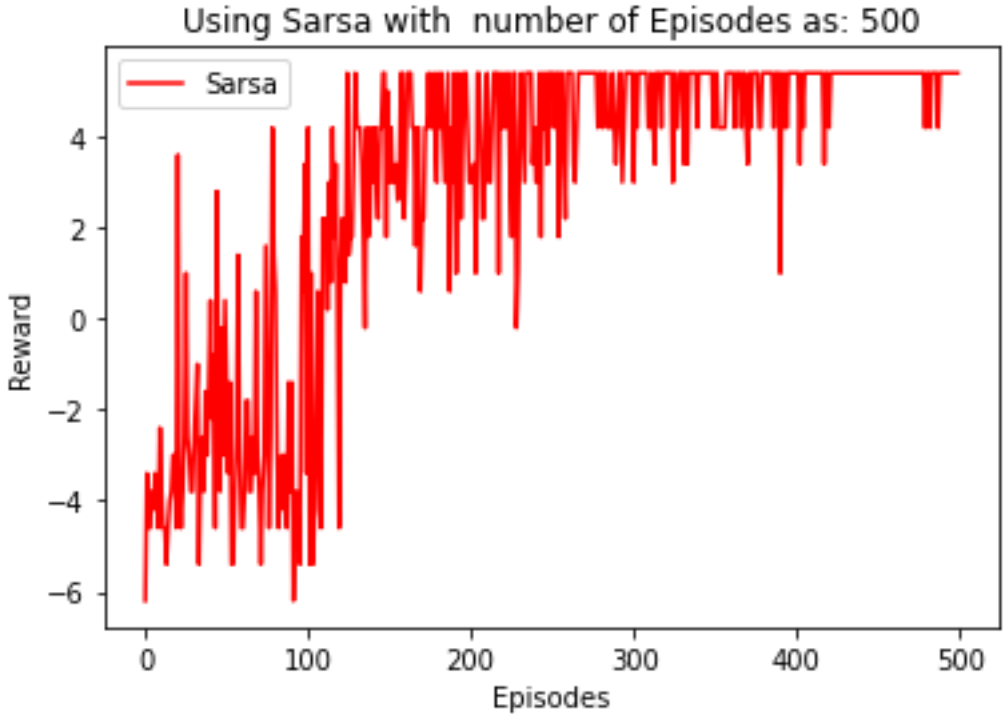
- **The most important thing to ensure the safety of our RL environment is that our agent should never take any action outside of the defined action state we can ensure that by having a deterministic setting. Also, the agent must not at any point of time get outside the environment, we can ensure this by using the clip function from the NumPy library.**

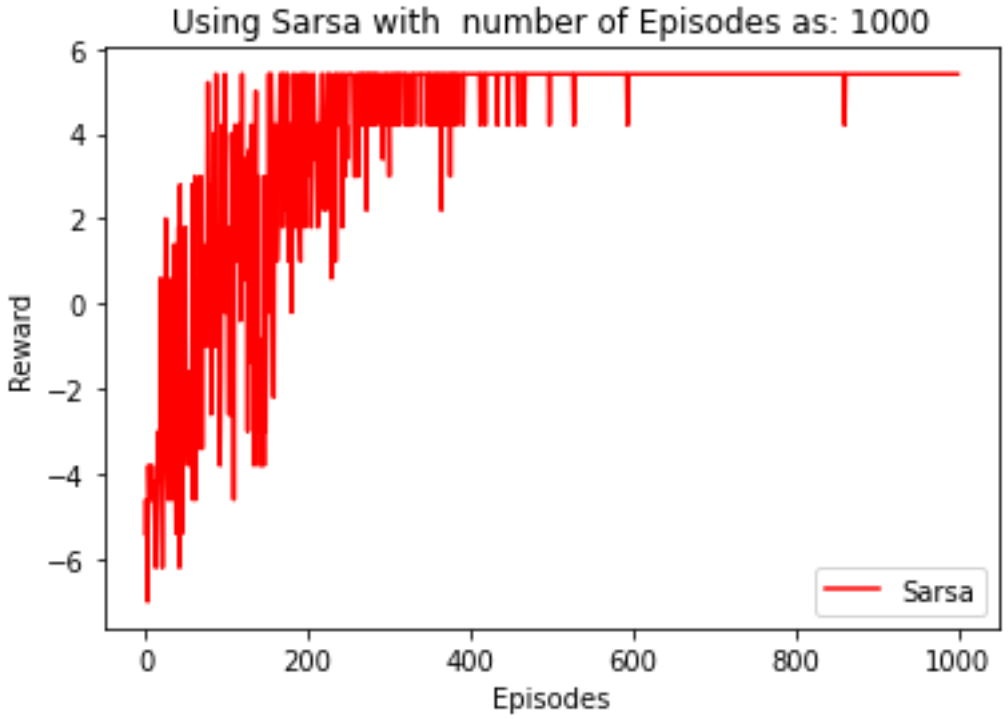
1. **Results**

Sarsa was able to converge after 500 episodes

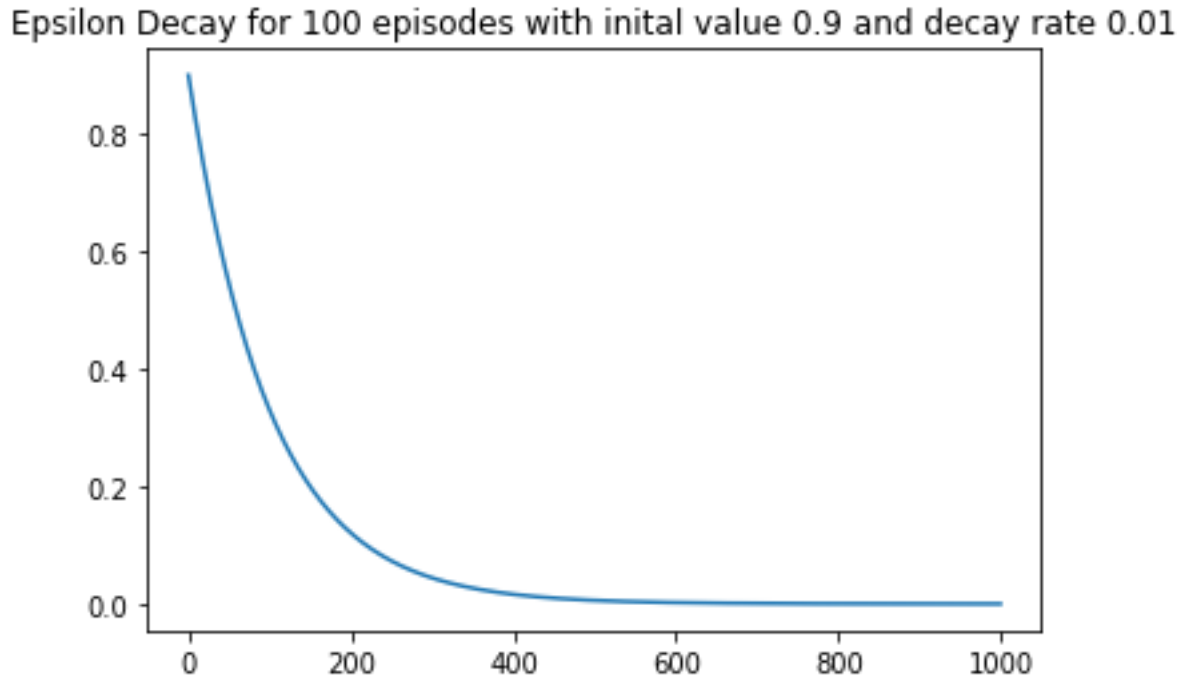




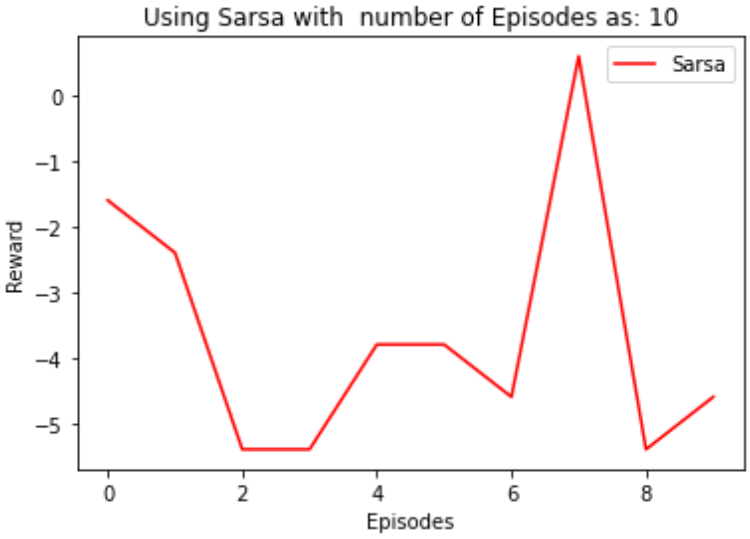

Using Sarsa with number of Episodes as: 500
Sarsa
Reward
Episodes
4
2
0
−2
−4
−6
0
100
200
300
400
500


Using Sarsa with number of Episodes as: 1000
Sarsa
Reward
Episodes
6
4
2
0
−2
−4
−6
0
200
400
600
800
1000

2. Plot for epsilon Decay



Environment Run for 10 episodes



Interpretation of Results

With 10 episodes we see that there is no significant learning and greedy action also does not always yield the best result.

The environment starts to stabilize with SARSA after 500 episodes and by 1000 episodes we see a completely flat line meaning that the agent has successfully learnt the best policy.

Q learning

Model free tabulation method for reinforcement learning that relies on temporal difference learning.

Q Learning is a value-based reinforcement learning algorithm where the values also known as q values are updated by using a value function for example the Bellman Equation. The Q in q learning stands for quality it represents the utility of a particular action in a particular state with respect to the objective of maximizing future cumulative reward.

It is an off-policy model free learning algorithm

It uses the temporal differences to update the Q values

Q new (S(t), A(t) Q(s(t),A(t))+ α .[r(t) + γ. Max(Q(s(t+1),A) – Q(S(t),A(t)]

Policy Improvement is done with the temporal difference learning where the q or quality values are updated.

It Can calculate expected utility of an action without a model and a policy However, Learning is expensive for the agent in the beginning specially if we have a lot of

possible states, in such a case we must keep discount factor as close to 1 as possible.

The update function used was

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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

Here alpha was the learning rate and gamma the decay rate which penalizes actions which have no net impact, and it is particularly useful for cases where we have a time constrained environment.

Sarsa is Q learning but with a slight modification to the update function

SARSA stands for state action reward state action, it is a modified q learning algorithm where target policy is same as behavior policy.

We have the tuple state action reward state action (s, a, r, s', a')

As Target policy is same as behavior policy SARSA is an on-policy learning algorithm. The 2 consecutive state action pairs and the immediate reward received by the agent while transitioning from the first state to the next state determines the updated Q value

Update function

Q(s_{t},a_{t}) = Q(s_{t},a_{t}) + α (r_{t+1}+ γ Q(s_{t+1},a_{t+1})-Q(s_{t},a_{t}))

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

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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

The learning is faster in SARSA as it takes action optimal in the next step thus policy formulation and learning is faster.

Sarsa can be trapped in a local minima and can keep on trying to find the best policy till perpetuity.

Hyperparameter tuning and Bonus Not Attempted No time :(

References

References

1.https://ikvibhav.medium.com/open-aigym-simple-sarsa-and-q-learning-reinforcement-learning-implementations-7d5ea6f1ff9 2.https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html 3.https://en.wikipedia.org/wiki/State%E2%80%93action%E2%80%93reward%E2%80%93state%E2%80%93action 4.Reinforcement Learning Project and assignment https://github.com/namantejaswi/Reinforcement_Learning_HW
5.https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf