NAIVE REINFORCEMENT LEARNING ON ATARI GAMES

Mentor :- Dr. Manish Kumar Bajpai (Assistant Professor, CSE Dept. IIITDM Jabalpur)

Pravesh Vajpayee (2018194)

Kshitij Kumar Singh (2018124)

Raj Agrawal Mohit (2018201)

INTRODUCTION AND PROBLEM STATEMENT

In this project , Two RL algorithms we have selected to compare their performance in this project are SARSA, naive Q-learning on different Atari games, and analyze them in order to identify factors which might influence the performance of these algorithms on different types of environments.

ENVIRONMENT

• Taxi-v2 as a deterministic environment with limited observation space.

1. Taxi-v2

Taxi-v2 is a deterministic, single-agent game where the agent's job is simply to pick up the passenger at one loca-tion and drop him off in another. Agents receive +20 points for a successful dropoff, and lose 1 point for every timestep it takes. There is also a 10 point penalty for illegal pick-up and drop-off actions. [2] There is no randomness that is out of agent's control, as there are no moving parts in this game other than the agent.

Taxi- V0

APPROACH

We will be implementing it with 2 RL algorithms i.e. Naive Q Learning , SARSA .

Naive Q learning

Naive Q learning is an algorithm which learns the value of a state (s) - action (a) pair, which we can represent by Q(s, a) , and store it in a table, which we call Q table. Q learning algorithm learns Q(s, a) by updating arg maxa Q(s, a) for any state s the agent is in. Q(s, a) is updated after each action the agent takes, and we assume there is a reward signal after each action. Therefore, the update equation for naive Q-learning is:

Q(s, a) ← Q(s, a) +α(R(s) + γ arg maxa Q(s 0 , a0) − Q(s, a)).

```
Initialize Q(s, a) arbitrarily
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) <-- Q(s, a) + α [r + γ max_α,Q(s', a') - Q(s, a)]
      s <-- s';
   until s is terminal
```

SARSA

Similar to Naive Q-learning, SARSA simulate Bellman update closely, but with a slightly different update equation, which is Q(s, a) ← Q(s, a) + α (R(s) + γ Q(s 0 , a0) − Q(s, a)). With much similarity to naive Q-learning, the difference between these two is the removal of the max operator in SARSA because SARSA is an on-policy algorithm. SARSA keeps updating its Q table which is used for action selection to interact with the environment it is in. However, naive Q-learning is an off-policy algorithm, meaning the action selection algorithm is not necessarily updated, but the Q table is updated with each action taken with respect to the update equation.

```
Initialize Q(s, a) arbitrarily
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) <-- Q(s, a) + α[r + γQ(s', a') - Q(s, a)]
      s <-- s'; a <-- a';
   until s is terminal
```

IMPLEMENTATION

Two jupyter notebook files have been attached along with this report in which Q learning and SARSA had been implemented through episodic learning.

RESULT

Based on the performance of Naive Q Learning (Figure 1) and SARSA (Figure 2) on Taxi-v2, we can see that while there index exists a trend where the sum of rewards over an episode increases as the algorithm is trained, and both seem to be converging to a similar policy, SARSA seems to converge much faster than Naive Q in this case. This difference in convergence rate can be explained by the effect of epsilon-greedy implementation, where SARSA converges faster as it takes the epsilon probability of random action into account when updating its policy.

With either algorithm, there does not seem to be any difference between their initial performance and final performance. Our hypothesis is that the problem may lie in the non-markovian nature of this environment, as our implementation only takes into account the current state when deciding which action to take.

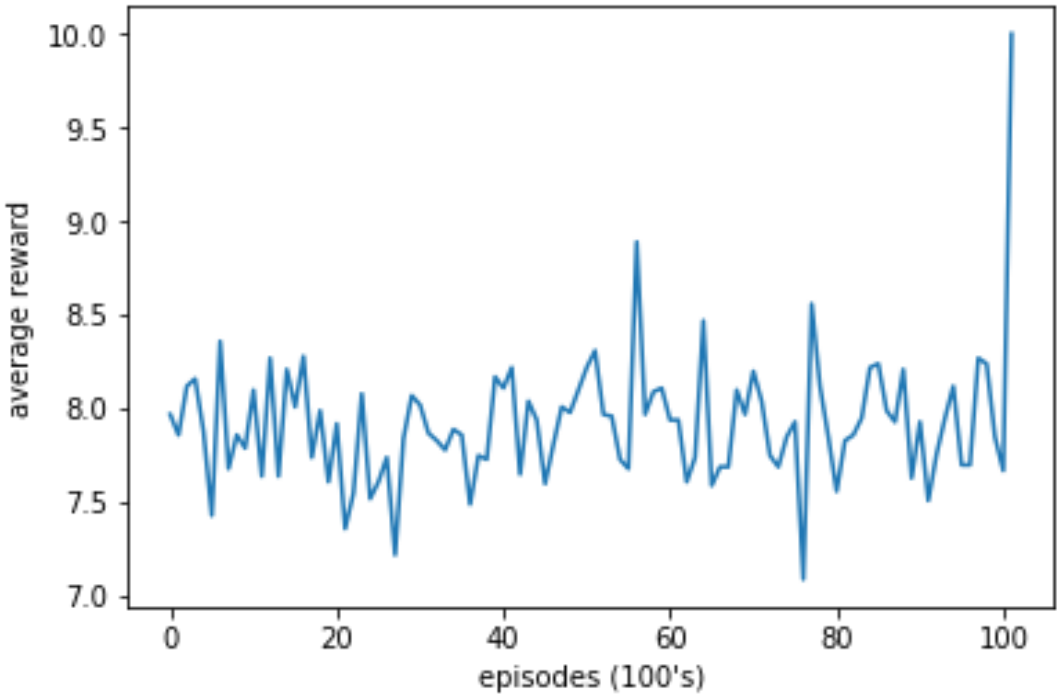
With a non-markovian environment, this leads to a situation where the value of an action given a state cannot be learned properly, as the value actually depends on the history of

past states and not the current state. Another possibility is that the state space is too big for either algorithm to show any trends within the episodes we trained it for. This seems less likely as we've trained it for 1000 episodes. However, this possibility can be further explored simply by training for a longer amount of time.

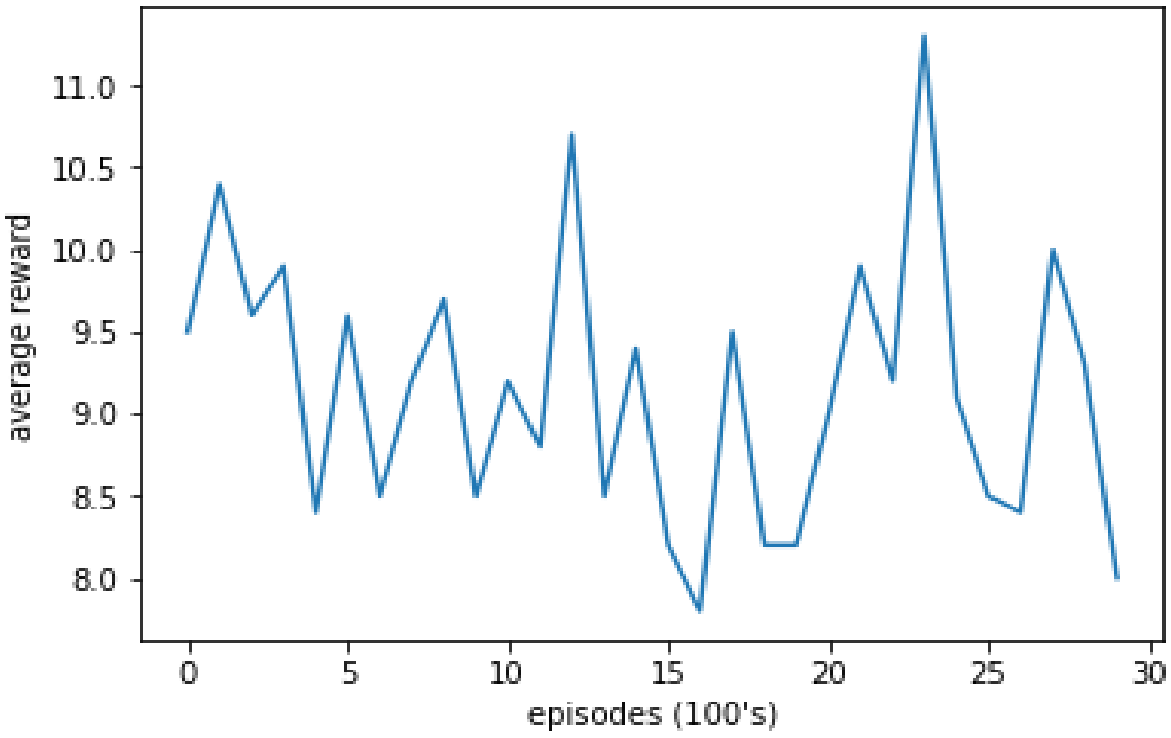
Performance of Naive Q in Breakout seems in line with our hypothesis. Although Breakout is deterministic and single-agent, just like Taxi-v2, Naive Q does not show any improvement in performance after training.

Improvement in performance by Deep Q-Learning using Experience Replay further supports our hypothesis, as using Experience Replay allows it to use a history of states to determine action, solving the problem of non-markovian property

OUTPUT IMAGES



Q Learning Reward Vs Episodes plot



SARSA Algorithm Rewards Vs Episodes

REFERENCES

[1] Andrew G.Barto Richard S.Sutton, Reinforcement Learning An Introduction, The MIT Press, 2018.
[2] “Taxi-v2,” 2011.
[3] Peter Norvig Stuart Russel, Artificial Intelligence A Modern Approach, Pearson, 2010. [4] “Monte carlo tree search,” 2010.
[5] “Implementing deep reinforcement learning models with tensorflow + openai gym,” 2018. [6] “Playing atari with deep reinforcement learning,” 2013.

[7] https://en.wikipedia.org/wiki/Wikipedia
[8] geeksforgeeks.org
[9] www.cse.unsw.edu.au