Reinforcement Learning using Sarsa

Manish Vishnoi, Sanay Devi, Alisa Pinchuk, Aditya Rohilla
1215218290, 1213169956, 1201546974, 1215187714

***Abstract*—In the past few decades, research in artificial intelligence has grown extensively, producing a number of simple to sophisticated learning algorithms and techniques utilized everyday for various use cases from fraud detection, recommendation systems, stock tradings and autonomous vehicles. The purpose of this paper is to compare two model free reinforcement learning algorithms - Q-learning and SARSA(λ) with Linear Function Approximation. Both models optimize their learning and actions from environment inputs in real time. We apply these algorithms on the pacman project created by UC Berkeley, and run number of performance tests. The results showcase performance matrices with reward in last 100 iterations and average score after learning.**

I. Introduction

REinforcement Learning (RL) refers to goal oriented methods and algorithms aimed at achieving a certain goal. These algorithms start from initial ground states and learn from their experiences to form a policy that achieves maximum reward. Based on the action taken, these algorithms get penalized for the wrong actions and get rewarded for the right ones. RL algorithms learn like humans, sometimes waiting to see the outcome of their actions, hence they work in a delayed return environment, where the outcome is observed after a series of actions.Reinforcement Algorithms work well in ambiguous and unknown situations, making them one of the best AI algorithms available for learning in real life situations [2].

Recently, RL algorithms defeated the world champions at the game of Go, showing the prowess of its applications RL algorithms can be divided as model-based and model-free [5].The model-based algorithms try to learn the underlying model of the environment to form it's policy, where as model-free algorithms rely on hit and trial methods to update their knowledge. Extra space is required by model-based algorithms to store the transition probabilities between states and hence they are not very useful as the state space grows. However, there is no such restriction on model-free algorithms; therefore, they are used more widely.

In particular, this paper compares two model-free reinforcement learning algorithms, the Q-learning and SARSA algorithm with Linear Function Approximation. In order to check which algorithm performs better, both are provided with the same parameters and environment. Q-learning is an off-policy algorithm, i.e agent learns the values of optimal policy independent of actions. Particularly, Q-learning updates its policy from next states greedy actions. Q-learning ignores current policy, i.e ignores current action but updates policy to the action that maximizes the value of the next state-action pair[5].

$$Q(s,a) = Q_k(s,a) + \alpha[R^{'} + \gamma \max_a Q_k(s^{'}, a^{'}) - Q_k(s,a)]$$

On the other hand, SARSA(λ) is an on-policy algorithm. It updates its policy using the Q-value of next state and current policys action. SARSA estimates the return for state-action pairs assuming the current policy continues to be followed.

$$Q(s,a) = Q_k(s,a) + \alpha[R^{'} + \gamma Q_k(s^{'}, a^{'}) - Q_k(s,a)]$$

In this project, we aim to understand how the algorithms performance changes when using SARSA and Q-learning in the pacman project. In order to do so, we implemented SARSA algorithm with Linear Function Approximation and integrated it into an existing pacman project created by U.C. Berkeley. The implementation process is discussed in more detail the first part of Section 3 of this paper. Next, we run performance tests on existing Q-learning algorithm (built in individual project 3 of the class) and SARSA implementation. Finally, in second part of section 3, we analyzed the performance results, and present our findings.

II. Background

As mentioned previously, SARSA is an on-policy algorithm. In SARSA, the agent currently in state S, takes an action A, getting a reward of R. Then moving to state S, it takes another action A and gets the reward R as seen in Figure 1. Now it goes back and updates the value of action A chosen in state S.

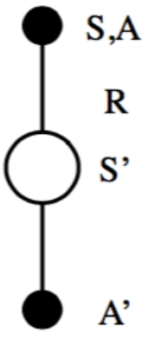

Fig. 1. SARSA state representation

This updating of values takes a lot of time and space as for calculating value of current state S, the values of all future

states is needed. This is where Temporal Difference (TD) learning comes in to picture. TD Learning is a model-free reinforcement learning method to estimate the final reward which is calculated at each state and the state-action value is updated for every step of the way making it much faster [3].

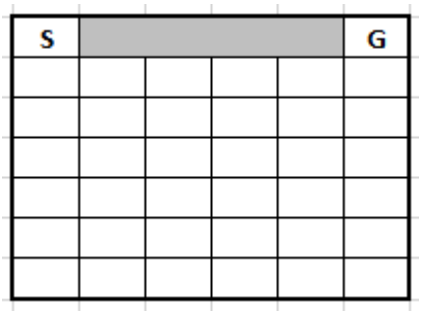

Fig. 2. Pacman State

Consider the Pacman Domain in Figure 1. The Pacman is currently in state S and wants to reach the Goal to get maximum reward. The grey area represents fire pits and the dots represents the area where Pacman can move. Now, using Q-learning the action chosen should have the highest value however there is still some randomness due to exploration. The pacman may know the best move but occasionally may jump into the fire pit because of Randomness. Q learning assumes that the pacman is following an optimal policy so that the actions will converge such that the optimal path is along the fire pits. Now using SARSA, after falling into the fire pits multiple times, the Pacman realizes that it is not the best choice to tread along the fire pits but it would be optimal to move as far as possible away from it. So the Pacman will now go all the way south and then move west and finally north to reach the Goal.

Algorithm 1 Sarsa Algorithm

1: initial $Q(s,a)$ set arbitrarily
2: **for** each episode **do**
3: intialize s
4: $a(s) - >$using policy derived from Q
5: **for** each step in episode **do**
6: $a'(s') - >$using policy derived from Q
7: $Q(s,a) = Q(s,a) + alpha * (R + discount * Q(s',a') - Q(s,a))$
8: $s = s'$
9: $a = a'$
10: **end for**
11: **end for**

Fig. 3. SARSA Algorithm

III. Implementation Details and Results

After doing extensive research on SARSA algorithm and reviewing test results between SARSA and Q-learning methods from prior research work, we moved to the implementation phase of the project. As mentioned, we used previously developed code of Q-learning from individual project phase 3 as a codebase for implementing SARSA algorithm. SARSA code lies in sarsalearningAgents.py that includes SarsaLearningAgent (with alpha=0.5, and gamma=0.9), PacmanSarsaAgent(where, alpha=0.2, and gamma=0.8), and ApproximateSarsaAgent (where agent learns weights for features of states and action pairs) classes [6]. In both SarsaLearningAgent and ApproximateSarsaAgent, update function was developed following SARSA algorithm as seen in Figure 1 line 7. To get new action a', getAction function on nextState was called. The getAction function used policy that was derived from Q, where a check was done on whether the current action produced max reward at that specific state; in either case, an action with max reward was chosen. To run the code, pacman.py should be called, as it is the main function that holds logic for pacman game.

The next phase of the project was to run number of tests in order to compare SARSA and Q-learning agents. The README file provides with details on how each test was run. First, Q-learning tests were run to build a baseline for SARSA, by executing ApproximateQAgent class with an argument specifying number of episodes ranging from 100 to 2000.

TABLE I
AVG. REWARD IN LAST 100 ITERATIONS

Iterations	SARSA	Q-learning
100	-511.78	-510.11
500	-170.73	-309.5
800	61.96	-218.38
1200	469.68	64.32
1300	460.76	136.68
1600	469.74	197.02
2000	499.84	175.93

Table I clearly shows that SARSA achieves positive reward earlier than Q-learning. Also, we can observe that once SARSA learned the correct policy it sticks to it rather than exploring other policies. As being an on-policy algorithm, there are no exploring or exploration state. SARSA learns by improving a single policy and that benefits it to have better rewards per iteration than Q-learning. The reward representation closer to score helps us to correctly determine number of iterations needed to learn pacman. We can see from the reward function that around 1200 iterations SARSA would converge, while Q-learning would run another 100 episodes and converge around 1300 iterations; however, to confirm those assumptions we would need to check the score values. Note, for Q-learning, we would also need to wait for agent to finish it's exploration and start exploitation before we could confirm when convergence occurs. You can get a clear picture of rewards in Figure 4.

From Figure 4, we can clearly see that SARSA learned much faster than Q-learning. One more point about SARSA is that as it learns from single policy, variation in rewards is very less. You can see from figure that even before saturation SARSA was learning with an increasing curve, on the other

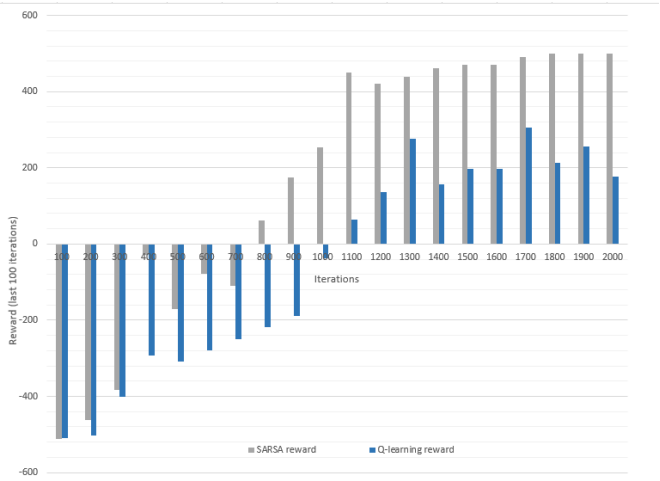

Fig. 4. Reward comparison between SARSA and Q-learning w.r.t iterations.

hand as q-learning adapts a new policy with each iteration, before saturation it fluctuates a lot between rewards.

Now, we can compare scores obtained after each iteration. As seen in Table 2, SARSA converged almost 100 iterations before Q-learning.

TABLE II
AVG. SCORE

Iterations	SARSA	Q-learning
100	-509.70	-512.5
500	-7.5	-7.8
800	193	-107.3
1200	499.4	401.8
1300	501.4	500.6
1600	500.6	499.4
2000	499.8	499.4

Also, we see that SARSA performed better in terms of consistency, as the average score stayed around 500 throughout all iterations from 1200 to 2000.

From Table II, we observe that SARSA's average reward and average score were almost the same while Q-learning has a big jump from average reward to the average score.

TABLE III
OVERALL REWARD

Iterations	SARSA	Q-learning
100	-511.78	-510.11
500	-312.37	-403.25
800	-210.5	-345.38
1200	-23.36	-232.41
1300	3.97	-193.26
1600	112.35	-122.61
2000	178.35	-50.41

This is expected as SARSA is an on-policy algorithm, so with each round of episodes, we will see real time spikes for both reward and score values; while Q-learning is trying different policies during exploration phase; therefore, its rewards and scores are varying. Another observation was that Q-learning's score varies a lot before converging while SARSA has an exponential curve for its average score.

In Table III, we observe the overall reward for various number of iterations. Similar results are seen, where SARSA algorithm produces positive overall reward starting at 1300 iterations and increases to almost 200 at 2000 iterations. Q-learning algorithm however still gives negative reward even at 2000 iterations. This table showcases that even high rewards are important in the use case, SARSA algorithm should be used.

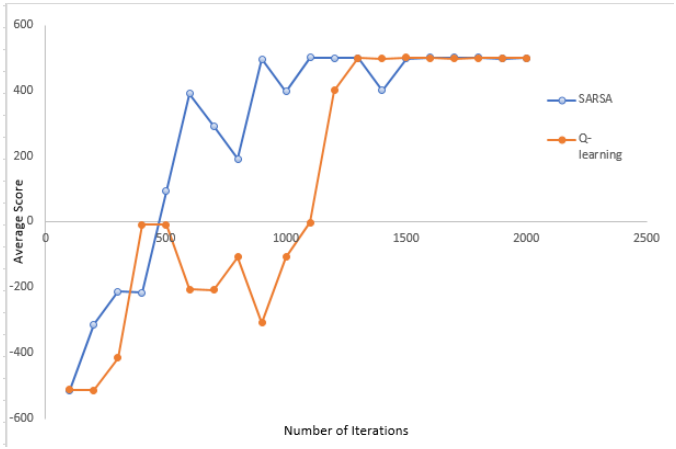

Fig. 5. Score comparison between SARSA and Q-learning w.r.t iterations.

As we can see in Figure 5, when the number of iterations reach to around 600, SARSA produced a higher score of around 400, whereas Q-learning achieved a score of around negative 200. This observation is key to understanding the major difference between the two algorithms.

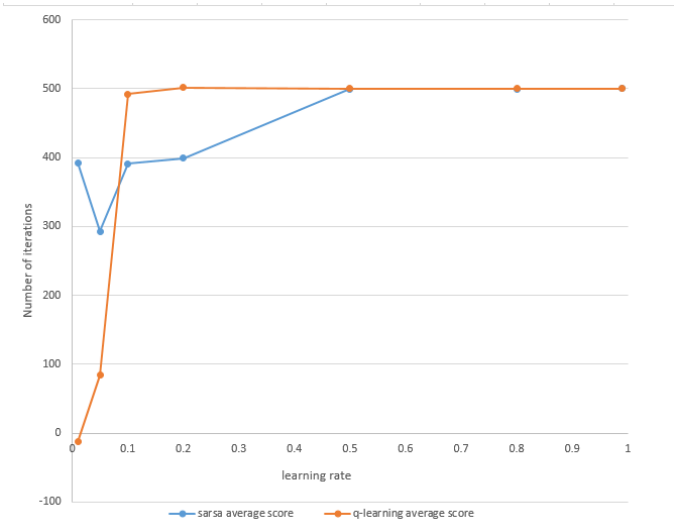

Fig. 6. Score comparison between SARSA and Q-learning w.r.t learning rates (for 1200 iterations).

In SARSA the agent from the start is very careful, avoiding high negative rewards whereas Q-learning strives towards finding the optimal solution without considering the negative rewards along the path. Similar pattern can be observed as the number of iterations increases. In the end however, around 2000 iterations as an example, Q-learning and SARSA perform almost equally, achieving similar average rewards.

Knowing the number of iterations where SARSA is converging, another test was run analyzing average score while varying learning rate. Number of iterations for this test was fixed at 1200, and learning rate varied from 0.01 to 0.99. From Figure 6, we can see that Q-learning algorithm is more dependent on learning rate than SARSA. Average score for Q-learning jumped from negative value to almost 500 when learning rate went from 0.01 to 0.1. On the other hand, average score for SARSA algorithm was already at 300 range at 0.01 learning rate.

IV. Conclusion

In conclusion, this paper identifies the motivation behind SARSA, its implementation details and comparison results with Q-learning. We implemented SARSA with Linear Function Approximation and compared its results with Q-learning developed as part of pacman project. After performing various tests on both Q-learning and SARSA with similar parameters for alpha and gamma, we found that SARSA performed better with average rewards and the time required to explore the maze was smaller as that compared to Q-learning. Initially, we proposed a decay in an epsilon value, which is also known as the epsilon-greedy method, where we found that SARSA did perform better than Q-learning. Without the decaying factor, we found that Q-learning And SARSA with linear feature selection perform almost equally, with SARSA beating Q-learning by 100 iterations.

We have demonstrated the overall suitability for SARSA which is an on-policy reinforcement learning, implying that the agent follows changing optimal policies, whereas Q-learning follows an off-policy learning [1].

What we learnt is that in SARSA if there is a high negative reward near the optimal path, it will not select optimal path right away, as it avoids the dangerous/unsafe path and learns the environment slowly via exploration. Q-learning, on the other hand, would take the optimal path into consideration while still exploring. Motivating this discussion further we can consider a real world-example where SARSA can be chosen over Q-learning. If we are training a robot to walk through a maze with actual fire pits, then to prevent the robot from real damage and incurring a loss of money and time, we could use SARSA so that the robot carefully selects its actions avoiding high risk paths [4].

Future work may include (i) Implementation of SARSA on a real-world example and seeing how it fairs (ii) Compare SARSA with Deep-Q-Networks(DQN) (iii) Compare SARSA with Deep-Deterministic-Policy Gradient(DDPG).

V. Team Effectiveness

Alisa Pinchuk - 25%
Major contributor of the implementation of the SARSA algorithm as well as creator of the tests that were run to compare Q-learning vs SARSA algorithm. Executed number of performance tests and collected results for Q-learning vs SARSA. Major contributing author of the team's project report.

Aditya Rohilla - 25%
Did extensive research in previous research done using SARSA, as well as implementation strategy, and the major differences between SARSA and Q-learning algorithms. Major contributing author of the team's project report.

Manish Vishnoi - 25%
Contributor of the implementation of the SARSA algorithm. Major contributor in testing efforts that compared SARSA to Q-learning by executing the tests, collecting results, and presenting them in a reader friendly form. Contributing author of the team's project report.

Sanay Devi - 25%
Did extensive research on SARSA with linear function approximation implementation algorithm, gathered details on differences between SARSA and Q-learning algorithms, as well as its use cases. Analyzed provided test results. Major contributing author of the team's project report.

References

[1] Adesh Gautam. "Introduction to Reinforcement Learning (Coding SARSA)-Part 4": The Startup. https://medium.com/swlh/introduction-to-reinforcement-learning-coding-sarsa-part-4-2d64d6e37617.
[2] C. K. Go, Bryan Lao, Junichiro Yoshimoto, Kazushi Ikeda, "A Reinforcement Learning Approach to the Shepherding Tas Using Sarsa". *Nara Institute of Science and Technology.*, 2016. https://medium.com/swlh/introduction-to-reinforcement-learning-coding-sarsa-part-4-2d64d6e37617.
[3] Dialogue Systems Group, "Temporal-difference methods" Cambridge University Engineering Department. http://mi.eng.cam.ac.uk/~mg436/LectureSlides/MLSALT7/L3.pdf.
[4] Ignazio Aleo, Paolo Arena, Luca Patane, "Sarsa-based reinforcement learning for motion planning in Serial Manipulators", 2010. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5596282.
[5] Rafipoor, Hossein. 2013. Reinforcement Learning Part 2: SARSA vs Q-Learning. Studywolf. https://studywolf.wordpress.com/2013/07/01/reinforcement-learning-sarsa-vs-q-learning/.
[6] Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning: An Introduction". Second Edition. pp. 105-110. MIT Press, Cambridge, MA. November 5, 2017. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.