Reinforcement Learning using Sarsa and Q-Learning for Decision Making CSE 571

Vinayak Kothari, Anirudh Kaushik, Ryan Kittle, Andrew Xi

***Abstract*—This report describes the work on Episodic Semi-Gradient Sarsa and True Online Sarsa agents. These agents are used to improve upon the Q-learning agent, which was provided to us for our fourth project on Reinforcement Learning. In both cases, the greedy policy is approximated by using the ε-greedy policy, which selects a random policy ε% of the time and selects the best known policy in all other cases. However, there are many places where these three algorithms diverge, creating large differences in their implementations and results. Both agents were successfully implemented with substantially better results than the original Q-learning agent on certain maps, both in regards to the Q values obtained throughout training and the reward/win rate acquiring throughout testing. This project exemplifies the effect that small differences in algorithms can make in reinforcement learning models, and how these minute changes can be incorporated in different ways to improve ones results.**

***Index Terms*—Q-learning, Pacman, Episodic Semi-gradient Sarsa, True Online Sarsa, Reinforcement learning**

I. Introduction

REINFORCEMENT learning has been the primary point-of-interest in AI and machine learning in recent years. Using the reinforcement learning model, agents are able to learn about their actions and environment through trial and error, slowly making improvements on its approach as it optimizes its strategy. Researchers can create reward functions for their agents, awarding desirable behavior and punishing undesirable actions. The agent is able to pursue long-term goals designated by the designer, making its own decisions to maximize the given reward function [1]. There are many types of reinforcement learning models, with each having its own pros and cons. However, we will focus on three primary algorithms throughout this report: Q-Learning, Semi-Gradient One-Step Sarsa, and True Online Sarsa(λ).

Starting off with Q-Learning, an agent always selects the greedy action when making decisions. In our Project 4 implementation, we used ε-greedy decision making to ensure a small amount of exploration will occur throughout its learning. Additionally, Q-Learning is an off-policy learning algorithm; it does not necessarily need to follow the policy that it is learning. Instead, off-policy learning methods allow an agent to learn about a policy and its effectiveness while following an entirely different policy in its actual decision making.

Sarsa differs from Q-learning in two main ways. To begin, Sarsa always uses ε-greedy decision making when creating its policy. Although we implemented the same decision making policy for our Q-Learning model in Project 4, the algorithm traditionally uses the greedy decision for its entire policy. With this, Sarsa is typically more exploratory in its training. Secondly, Sarsa is an on-policy algorithm, meaning it always follows the policy it is learning about. This creates differences in Q value calculations and policy updates, which will be explored more later in the report.

Semi-Gradient Sarsa expands upon the Sarsa model by incorporating the *gradient* of the Q function. This gradient term is the partial derivative of the Q function with respect to the weight w. To incorporate this into our model while following the design choices laid out in the Pacman Project 4, our gradient is represented by the feature vector used for linear approximation. This allows the agent to make more informed choices by looking at the most recent history of its choices. Using this model, the agent can estimate $\hat{q} \approx q_*$ to make its decisions.

Finally, True Online Sarsa (λ) expands upon this idea further by incorporating an *eligibility trace* to determine the proper amount of steps to consider when making future decisions. This allows the agent to fully maximize its actions by incorporating more than just its most recent step in its decision making. This again creates changes in its Q value calculation, allowing the agent to estimate $w^T x \approx q_*$.

This report describes the work done and results of implementing the Episodic Semi-Gradient Sarsa and True Online Sarsa algorithms described in Reinforcement Learning, 2nd Edition [2]. The motivation for this project, which was performed by 4 team members, was to improve upon the Q-learning agent which used the linear function approximation from Project 4. While Sarsa and Q-Learning are similar in their approaches, there are multiple key differences that can have quite a large impact on learning rate and overall results; while small summaries have been described in the Introduction above, we will speak on these differences and their effects in detail throughout this report.

II. Technical Approach

Starting off with Q-learning as a baseline, the algorithm is implemented as follows.

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

Fig. 1: Algorithm for Q-Learning, taken from [2]

As discussed earlier in the report, our implementation of Q-learning uses an ε-greedy policy when determining its future actions. This allows our agent to perform a random action $\varepsilon\%$ of the time, promoting exploration of the space. This improves overall decision making by ensuring there is potential for finding different policies as opposed to simply improving the current policy.

On the other hand, the algorithm always uses the greedy action when updating the Q value for state-action pairs. This is where the idea of off-policy learning comes into play. Although actions are sometimes chosen at random, the Q values are always updated according to the greedy decision. All of this was implemented during our fourth reinforcement learning project, so we did not need to make any adjustments for this project or report.

For implementation part we also updated our feature extractor to truncated n-step BFS search which would make it's next move based on the number of scared ghosts and their positions, the number of non scared ghost and their positions, closest power pellet, and closest food. We considered n to be $(width\_of\_grid + height\_of\_grid)/2$. This improved our algorithms' performance in all three learning methods: semi gradient SARSA, true online SARSA, and Q learning with linear feature approximation. Before applying this extractor our agent was just looking for food and running away from ghost if it was one step nearby. The agent was not thinking as effectively as possible, so we made the aforementioned changes through the feature extractor.

Let's now move onto the general Sarsa algorithm and delve deeper into the differences between the two algorithms. The implementation for Sarsa is found below.

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., ε-greedy)
  Loop 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

Fig. 2: Algorithm for Sarsa, taken from [2]

Similar to Q-learning, Sarsa uses an ε-greedy policy when determining its future actions. Again, this promotes exploration within the environment and can improve the overall decision-making policy by allowing for better policies to be found through random actions.

However, Sarsa is an on-policy learning model, meaning it will always follow the policy it is learning about. Because of that, the Q values are not updated using the greedy actions as they are in Q learning. Instead, Sarsa calculates the Q' by using the same ε-greedy policy as its decision making for actions. Since the action is chosen using the same policy as the Q value updates, the action does not need to be updated for each step of each episode; it will be the same action either way.

Moving onto Episodic Semi-Gradient Sarsa, the algorithm is shown below.

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

Fig. 3: Algorithm for Episodic Semi-Gradient Sarsa, taken from [2]

As you can see, the algorithm is fairly similar to the implementation of basic Sarsa, with the primary difference coming from the incorporation of gradient descent. The general gradient-descent update for action-value prediction is

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha[U_t - \hat{q}(S_t, A_t, \mathbf{w}_t)]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \quad (1)$$

In the case of the Episodic Semi-Gradient One-Step Sarsa (or simply Episodic Semi-Gradient Sarsa), this becomes

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha[R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t) \quad (2)$$

Improving upon the general Sarsa algorithm, episodic semi-gradient one step Sarsa adds the gradient term $(\nabla\hat{q}(S_t, A_t, \mathbf{w}_t))$ which is the partial derivative of the Q function with respect to $\mathbf{w}$. This extra term in the update function allows for the agent to take into account its most recent action when deciding its next action. With this, the agent can determine how effective its previous action was, and use this value to maximize the reward for its next decision.

This ideology of incorporating past decisions is taken a step further in the True Online Sarsa (λ) method, with the algorithm being shown below.

True online Sarsa(λ) for estimating $\mathbf{w}^\top\mathbf{x} \approx q_\pi$ or q_*

Input: a feature function $\mathbf{x} : \mathcal{S}^+ \times \mathcal{A} \to \mathbb{R}^d$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$, small $\varepsilon > 0$
Initialize: $\mathbf{w} \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or ε-greedy according to $\hat{q}(S, \cdot, \mathbf{w})$
  $\mathbf{x} \leftarrow \mathbf{x}(S, A)$
  $\mathbf{z} \leftarrow \mathbf{0}$
  $Q_{old} \leftarrow 0$
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose $A' \sim \pi(\cdot|S')$ or ε-greedy according to $\hat{q}(S', \cdot, \mathbf{w})$
    $\mathbf{x}' \leftarrow \mathbf{x}(S', A')$
    $Q \leftarrow \mathbf{w}^\top\mathbf{x}$
    $Q' \leftarrow \mathbf{w}^\top\mathbf{x}'$
    $\delta \leftarrow R + \gamma Q' - Q$
    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \big(1 - \alpha\gamma\lambda\mathbf{z}^\top\mathbf{x}\big)\,\mathbf{x}$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + Q - Q_{old})\mathbf{z} - \alpha(Q - Q_{old})\mathbf{x}$
    $Q_{old} \leftarrow Q'$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    $A \leftarrow A'$
  until S' is terminal

Fig. 4: Algorithm for true online Sarsa, taken from [2]

The update step for true online Sarsa is

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + Q - Q_{old})\mathbf{z} - \alpha(Q - Q_{old})\mathbf{x} \quad (3)$$

In this case, $\mathbf{x}$ is the feature function with $\mathbf{x}(terminal, \cdot) = 0$. For our implementation of True Online Sarsa, the feature vector accepts a state-action pair rather than just a state. This was simple to implement, and only required a couple smaller changes within the overall codebase to account for the added parameter.

Episodic semi-gradient one-step Sarsa is a special case of the more generalized episodic semi-gradient n-step Sarsa; it simply takes into account the single most recent action as opposed to the n most recent actions. The primary difference between the n-step Sarsa and the true online Sarsa is that while n-step varies the update length (n), true online Sarsa varies a trace parameter called λ. The eligibility trace weights earlier action values less so they fade away in a sense. This allows the agent to view all actions taken so far, but focus on more recent ones. In total, all of the aforementioned variants of Sarsa take into account the agent's recent history when updating the Q value and choosing future actions.

III. Results and Analysis

As for testing our implementation, we varied the number of training episodes and used 25 testing iterations for analyzing the outcome between two Pacman layouts, mediumClassic and powerClassic.

By adjusting the number of our results and analysis will be broken up into two main sections: training improvements and testing improvements.

A. Training Improvements

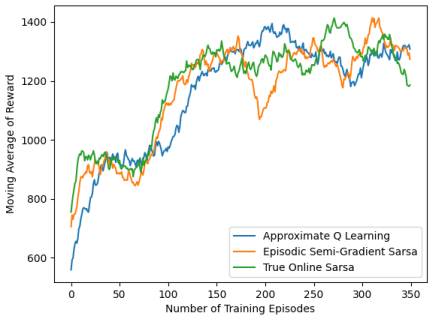

Fig. 5: Plot of training episodes vs rewards for Q-learning (blue), episodic semi-gradient Sarsa (orange), and true online Sarsa (green) for the mediumClassic map

As shown in Fig. 7, the two Sarsa agents are consistently ahead of the Q-learning agent. For example, the Sarsa agents reach an average reward of 1200 at about 100 training episodes, while it takes the Q-learning agent about 130 episodes to reach the same average reward, representing a 23% improvement over the Q-learning agent. The episodic semi-gradient Sarsa agent has a significant dip in its average reward at about 200 training episodes—the cause for this is not 100% clear, but we believe that it has to do with the fact that it only looks back 1 step in the past, so it gets stuck on a single poor action. This is not present in the true online Sarsa agent because it is able to view all actions, albeit with diminishing weight.

We used the mediumClassic map because we felt it was most representative of a typical Pacman environment. The three learning methods all converge as the number of training episodes increases. However, initially the two Sarsa agents converge faster. The true online Sarsa agent converges slightly faster than the episodic semi-gradient Sarsa agent.

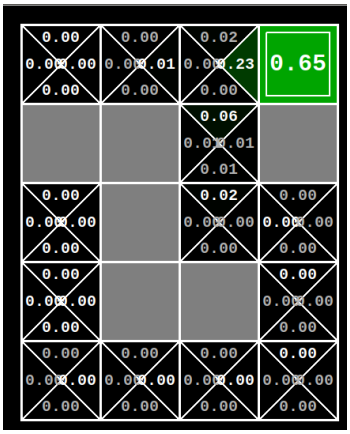

Fig. 6: Results for Approximate Q learning with identity features in Maze grid with
0.2 noise , 0.5 epsilon , 10 iterations. We infer that it is far away from converging. Time taken 5 minutes 3 seconds

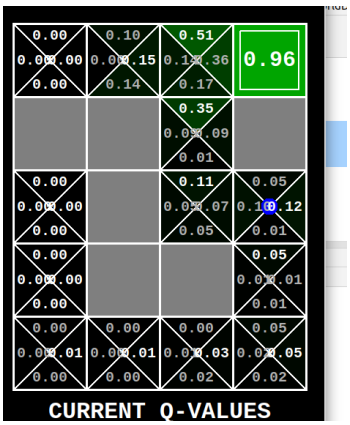

Fig. 7: Results for True Online SARSA Lamda with identity features in Maze grid with
0.2 noise , 0.5 epsilon , 10 iterations. We infer it is much closer to convergence value. Time taken 1 minute 23 seconds

B. Testing Improvements

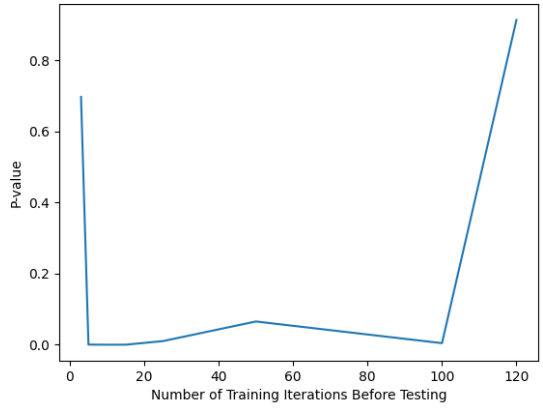

Fig. 8: Plot of P value vs training iterations for powerClassic

In Fig. 8, we can see that for the powerClassic map, the P-value is initially very low. A P-value below 0.05 corresponds to a statistically significant difference. Therefore, based on the graph, we can see that up to 100 iterations, the three agents are statistically different from each other. That changes around 120 iterations, though, because the normal Q-value agent has now had enough time to catch up to the two Sarsa agents.

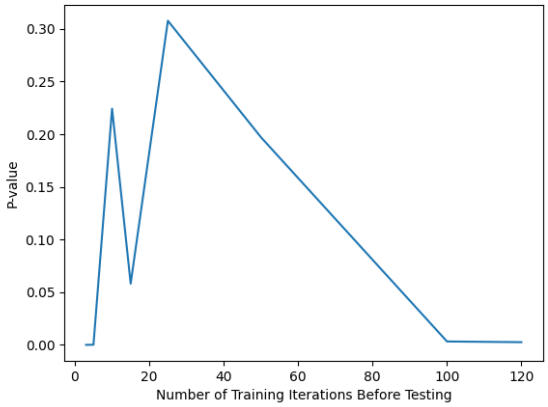

Fig. 9: Plot of P value vs training iterations for mediumClassic

In Fig. 9, we can see that there is not as much of a statistically significant difference initially. We believe that this is because the mediumClassic grid is smaller and easier to evaluate; with fewer power pellets, fewer ghosts, and a smaller overall size, it is easier for all three algorithms to converge and therefore they all perform similarly. However, after 25 iterations, the p-value decreases, signifying a statistically significant difference. We believe that this is caused by the fact that the ghosts are more important in the feature extractor function. Since ghosts are not as important for the mediumClassic map, their position does not matter as much. Instead, the difference will mostly depend on the learning method used. Since both Sarsa agents are better than the Q-learning method, they will pull ahead as the number of training iterations increases.

IV. Discussion

As seen in the results above, the episodic semi-gradient Sarsa and true online Sarsa agents both converge faster than the Q-learning agent, with the true online Sarsa agent outperforming the rest. This is as expected, because it has access to more of the steps taken than the single step episodic semi-gradient Sarsa.

Overall the two Sarsa agents perform better than the Q-learning agent and learn faster.

V. Conclusion

In the general Q-learning algorithm, agents will always select the greedy action when making decisions. Q-learning is an off-policy learning algorithm, while the Sarsa algorithms are on-policy. The general Sarsa algorithm uses an ε-greedy decision process while (most) Q-learning implementations use an always-greedy process, which means that in traditional implementations of both algorithms, Sarsa agents will be more exploratory than Q-learning agents when training. Episodic semi-gradient Sarsa takes the basic Sarsa model and adds the gradient of the Q function ($\nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$), giving the agent the ability to look at its most recent choices in addition to the current choice, which allows it to make a more informed decision. True online Sarsa (λ), on the other hand, keeps track of all steps but uses an eligibility trace to "fade" their importance when making decisions.

As shown by our results comparing both the training and testing stages of our algorithms, semi-gradient Sarsa provides for a nice improvement over the Q-learning algorithm, and improved even further with our true online Sarsa implementation. In training, we can see that the true online Sarsa agent converges fastest of the three methods, with semi-gradient Sarsa only slightly behind. Both agents allow for over a 23% improvement over Q-learning when analyzing the convergence of training rewards. Throughout the testing phase, our Sarsa agents are shown to be statistically different from Q-learning through our ANOVA tests, which compare the models after different number of training iterations. Although expected, these results show that only a few small changes in algorithm design can lead to great improvements in the overall reinforcement learning system.

REFERENCES

[1] J. M. Carew, "What is reinforcement learning? A comprehensive overview," SearchEnterpriseAI, 29-Mar-2021. [Online]. Available: https://searchenterpriseai.techtarget.com/definition/reinforcement-learning. [Accessed: 08-Dec-2021].

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction, second edition*. The MIT Press, 2020.