Considerations For SARSA and Expected SARSA: Card Game Scenario

Kwaku Agyapong
Data Science and Artificial Intelligence
University of Cote d'Azur

***Abstract*—This study reviews Temporal Difference for the control problem and studies two special cases of temporal difference namely SARSA and Expected SARSA. We consider a card game scenario where we make considerations for using either case to find the best policy. The study concludes that if we are interested in a stationary or stable policy, then Expected SARSA almost certainly is best but if we care about randomness of the process then SARSA is used for finding the best policy.**

Keywords—temporal difference, agent, policy, goal, action

I. INTRODUCTION

Temporal Difference (TD) learning is a combination of Monte Carlo and dynamic programming (DP) ideas. TD methods learn directly from raw experience without a model of the environment's dynamics and update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap). DP, TD, and Monte Carlo methods all use some variation of generalized policy iteration (GPI) for the problem of finding an optimal policy. The difference is how each approaches the prediction problem. Both TD and Monte Carlo methods use experience to solve the prediction problem. Given some experience following a policy π, both methods update their estimate V of v_π the nonterminal states S_t occurring in that experience. Whereas Monte Carlo methods must wait until the end of the episode to determine the increment to $V(S_t)$ as only then is G_t known, TD methods need to wait only until the next time step. At time t + 1 they immediately form a target and make a useful update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$. The two updates for TD and MC are expressed in equations 1 and 2 respectively:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (1)$$

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \quad (2)$$

From equations (1) and (2), the target for the Monte Carlo update is G_t, whereas the target for the TD update is $R_{t+1} + \gamma V(S_{t+1})$. The TD in (1) is called one-step TD or TD(0). TD(0) bases its update in part on an existing estimate V, as such it is seen as a bootstrapping method. Just as the DP and MC targets are seen as estimates, TD target is also an estimate because it combines the sampling of Monte Carlo with the bootstrapping of DP. This can be seen from equation (3) where the TD target samples the expected values and uses the current estimate V instead of the true v_π.

$$v_\pi = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \quad (3)$$

$R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ from equation (1) is referred to as the TD error and measures the difference between the estimated value of S_t and the better estimate $R_{t+1} + \gamma V(S_{t+1})$. At each time t, TD error is the error made at that time. Because the TD error depends on the next state and next reward, it is not actually available until one time step later. That is, TD error is the error in $V(S_t)$, available at time t + 1. TD methods update their estimates based in part on other estimates. They learn a guess from a guess, that is, they bootstrap. TD methods have an advantage over DP methods in that they do not require a model of the environment, of its reward and next-state probability distributions. An advantage of TD methods over Monte Carlo methods is that they are naturally implemented in an online, fully incremental fashion. With Monte Carlo methods one must wait until the end of an episode, because only then is the return known, whereas with TD methods one need wait only one time step. Also, some Monte Carlo methods must ignore or discount episodes on which experimental actions are taken, which can greatly slow learning. TD methods are much less susceptible to these problems because they learn from each transition regardless of what subsequent actions are taken. Although TD estimates are based on the guess of the current estimate, it guarantees convergence to the optimal or current answer because for any fixed policy π, TD(0) has been proved to converge to v_π, in the mean for a constant step-size parameter if it is sufficiently small, and with probability 1 if the step-size parameter decreases according to the usual stochastic approximation conditions. TD methods have usually been found to converge faster than constant-α MC methods on stochastic tasks. If only a finite amount of experience is available, say 10 episodes or 100 time steps, a common approach with incremental learning methods is to present the experience repeatedly until the method converges upon an answer. From equations (1) and (2), the updates are computed for every time step t at which a nonterminal state is visited, but the value function is changed only once, by the sum of all the increments. Then all the available experience is processed again with the new value function to produce a new overall increment, and so on, until the value function converges. This is known as batch updating because updates are made only after processing each complete batch of training data. Under batch updating, TD(0) converges deterministically to a single answer independent of the step-size parameter, α, as long as α is chosen to be sufficiently small. The constant-α MC method also converges deterministically under the same conditions, but to a different answer.

SARSA and Expected SARSA are TD prediction methods used for the control problem. In both cases there is a trade-off between exploitation and exploration. TD methods generally fall into two classes; on-policy and off policy. SARSA is an on-policy whereas Expected SARSA can behave as an instance of both classes.

Objectives of the study:

1. To differentiate between the SARSA and Expected SARSA
2. Make decision on which instance to use for our card game

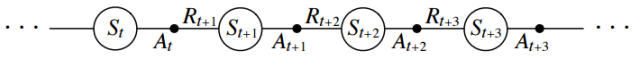
Hypothesis:

1. SARSA is useful for introducing randomness in our card game
2. Expected SARSA is useful for stability in our card game

II. LITERATURE REVIEV

A. *SARSA*

The first step is to learn an action-value function rather than a state-value function. Particularly, for an on-policy method we must estimate $q_\pi(s, a)$ for the current behavior policy π and for all states s and actions a. This can be done using essentially the same TD method described above for learning v_π. An episode consists of an alternating sequence of states and state–action pairs as shown in fig (1):



In SARSA, we consider a transition not from a state to state (as happens during TD(0)), but we consider transitions from state–action pair to state–action pair, and learn the values of state–action pairs. However, the theorems assuring the convergence of state values under TD(0) also apply to the corresponding algorithm for action values:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma\, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (4)$$

This update is done after every transition from a nonterminal state S_t. If S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1})$ is defined as zero. This rule uses every element of the quintuple of events, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, that make up a transition from one state–action pair to the next. This quintuple gives rise to the name SARSA for the algorithm. As in all on-policy methods, we continually estimate q_π for the behavior policy π, and at the same time change π toward greediness with respect to q_π . SARSA is an on-policy learning algorithm because it updates its policy based on actions actually taken by the agent. At each time step, SARSA learns by observing the current state, taking an action based on its current policy (e.g., an epsilon-greedy policy), receiving a reward, transitioning to a new state, and taking another action based on the updated policy. It is an iterative algorithm that learns from the agent's experiences in the environment, updating its policy based on the observed rewards and transitions.

It is particularly suitable for scenarios where the agent directly interacts with the environment and needs to learn an optimal policy while considering the impact of its own actions. The convergence properties of the SARSA algorithm depend on the nature of the policy's dependence on Q. For example, one could use ε-greedy or ε -soft policies. SARSA converges with probability 1 to an optimal policy and action-value function as long as all state–action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy (which can be arranged, for example, with ε -greedy policies by setting $\varepsilon = 1/t$).

B. *Expected SARSA*

Expected SARSA behaves like Q-Learning but instead of using the maximum over next state–action pairs it uses the expected value, taking into account how likely each action is under the current policy. It considers the expected value of the next action rather than just the value of the action actually taken. Expected SARSA is also an on-policy learning algorithm, meaning it updates its policy based on actions taken by the agent. However, unlike SARSA, which directly uses the value of the next action, Expected SARSA considers the expected value of all possible actions in the next state. Equation (4) provides the update rule for Expected SARSA:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma\, E_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)]$$

$$\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \textstyle\sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \quad (5)$$

This update rule computes the expected value of the next state-action pair by considering the sum of the values of all possible actions in the next state, weighted by their probabilities of being chosen under the agent's policy. This provides a more accurate estimate of the future rewards and helps the agent make more informed decisions. Given the next state, S_{t+1}, this algorithm moves deterministically in the same direction as SARSA moves in expectation, and thus it is called Expected SARSA. Expected SARSA is more complex computationally than SARSA but, in return, it eliminates the variance due to the random selection of A_{t+1} . Given the same amount of experience we might expect it to perform slightly better than SARSA, and indeed it generally does. Expected SARSA is particularly useful in scenarios where the agent needs to take into account the uncertainty in its actions and their outcomes. By considering the expected value of the next action, it can make more robust decisions and learn a better policy over time.

The key difference between SARSA and Expected SARSA lies in how they estimate the value of future state-action pairs during the learning process. SARSA updates its Q-values (action-values) using the value of the action actually taken in the next state. It directly considers the value of the next action according to the agent's policy. Expected SARSA computes the expected value of all possible actions in the next state and uses this expectation to update its Q-values. It considers the average value of all possible actions in the next state, weighted by their probabilities under the agent's policy. SARSA updates its Q-values by directly considering the value of the action taken in the next state, while Expected SARSA computes the expected value of all possible actions in the next state and updates its Q-values accordingly. Expected SARSA tends to provide more stable and conservative updates, especially in environments with high stochasticity or uncertainty, because it accounts for the variability in future actions. However, it may require more computational resources due to the need to compute the expected value.

Important to note is the fact that Expected SARSA might use a policy different from the target policy π to generate behavior, in which case it becomes an off-policy algorithm. For example, suppose π is the greedy policy while behavior is more exploratory, Expected SARSA becomes Q-Learning which is an off-policy algorithm.

III. METHODOLOGY

The card game has a set of rules defining how players interact with the cards and make moves. Players aim to follow specific sequences, patterns, or combinations to achieve victory.

States: States in the card game represent the current configuration of the game, including the cards in play, the players' hands, and the overall game state.

Environment: The environment is the card game itself, with dynamic interactions based on player moves and the evolving state of the game. It involves transitions between states based on players' actions and decisions.

Objectives/Rewards:

- Teaching Objective: The primary objective is to teach users how to play the card game effectively, guiding them towards making optimal decisions.
- Reward System: Rewards are provided based on the quality of moves made by users. Positive rewards for correct moves, negative rewards for mistakes, and cumulative rewards over time. Successfully executing a card combination results in a favorable outcome. Positive reinforcement are in the form of points, progression, or verbal praise, encouraging the player to repeat successful strategies. Making a suboptimal move exposes the player to a disadvantageous position. Negative feedback, such as loss of points, setback in progression, or verbal cues indicating a mistake, prompting the player to reconsider their decision-making process. Consistently applying effective strategies and making sound decisions throughout multiple game sessions lead to cumulative increase in points, progression to higher levels of difficulty or complexity, or unlocking additional features or content within the game, indicating mastery and proficiency in playing the card game effectively.

Actions: Actions represent the possible moves or decisions that a player can make during their turn. Actions include playing a card, making a strategic decision, or following a specific game rule. For example, a player can choose to play a specific card from their hand onto the game board. A player selects a card and places it in a valid position on the game board. Decision by player include deciding whether to prioritize attacking the opponent or defending one's own position. A player can choose a strategic approach based on the current game state and their objective. Players adhere to a rule that dictates the order in which certain actions can be performed ensuring compliance with the established rules of the game, such as turn order or card interactions.

By providing clear examples of actions and rewards, the reinforcement learning agent can effectively guide users through the learning process, reinforcing desirable behaviors and correcting mistakes to facilitate skill development and mastery of the card game.

A. SARSA Considerations

Quick Adaptation - SARSA allows for quick adaptation to the game dynamics, facilitating a dynamic learning experience.

Exploratory Learning - The occasional fluctuations introduce users to different strategies and aspects of the game.

Unpredictability - Fluctuations may lead to unpredictable learning experiences, which might not be suitable for all users.

Risk of Confusion - Rapid changes may overwhelm new users, potentially leading to confusion.

Conclusion: SARSA is suitable for users who prefer a dynamic and exploratory learning experience. Initial high rewards can positively impact user engagement and motivation.

B. Considerations for Expected SARSA

Stability - Expected SARSA provides a more stable learning process, reducing the impact of noise and fluctuations.

Consistent Guidance - Users receive consistent and reliable suggestions for optimal moves.

Slower Adaptation - Expected SARSA may adapt more slowly to rapid changes in user behavior or the game environment.

Limited Exploration - The algorithm leans towards exploitation, potentially limiting the variety of strategies explored.

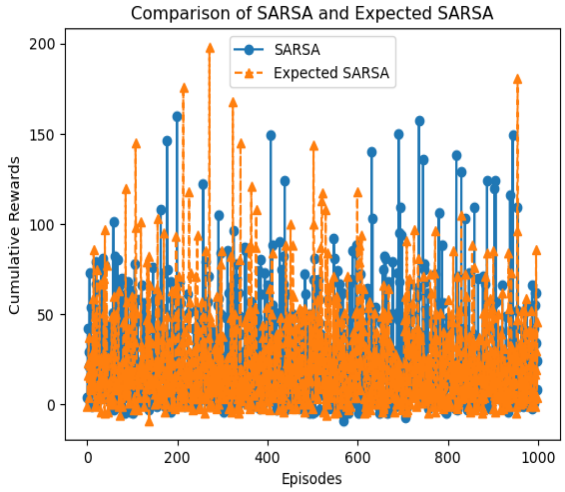
Conclusion - Expected SARSA is suitable for users who prefer a predictable and guided learning experience. Users looking for steady and reliable assistance in mastering the game might benefit from Expected SARSA.

IV. RESULTS

The resulting plots from the card game scenario are shown in this section. The aim of the agent is to train users on how to make the best possible moves to get the highest rewards. In each case the aim of the agent is to find the best action state pairs for our algorithm.

A. Cumulative/Total Rewards per Episodes

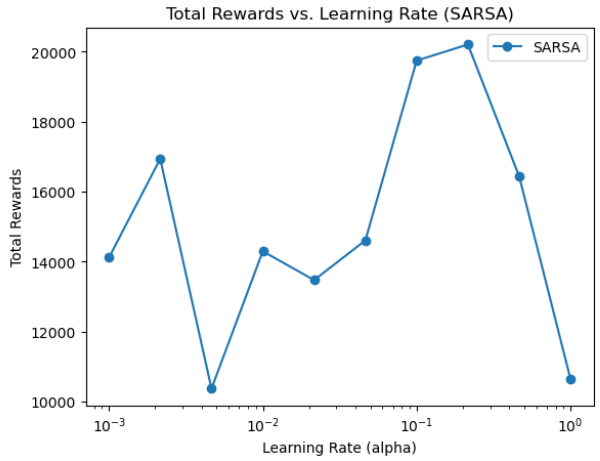
The plot shows the total rewards at each episode during the game. The agent finds the best action state pairs. The plot shows a lot of variability in the rewards for SARSA and this can be explained by the fact that following the epsilon-greedy policy, the agent finds the best policy by computing the updates based on the next action that gives the highest reward in the next state. Thus, the agent follows an exploratory behavior by observing the current state and taking an action that gives the highest rewards and uses this to update the policy. Expected SARSA on the other hand shows stable rewards because it uses the expected values of all possible actions in the next state to update its policy there by removing the variability.



The feedback from users is important in this case because if users are interested in taking risks for higher rewards or are interested in moving from one stage of the game to the other at a much faster rate, then it is important to implement the SARSA algorithm for the best policy because it allows for randomness. If, however, the feedback form users are that they want to be able to learn to play the game at more predictable pace where they receive regular instructions on moves to make and allows them to move from one stage of the game through a gradual process of taking predictable actions, then the developers need to consider expected SARSA for the problem of finding the best policy as it removes randomness and allows for exploitation.

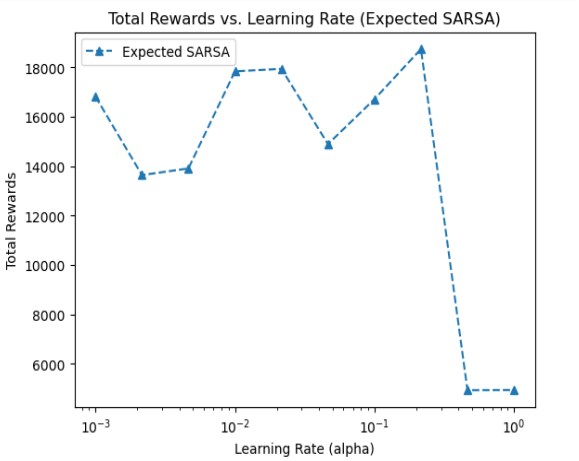
B. Total Rewards per Learning Rate (SARSA)

The plot shows the rewards from each episode against the learning rate. It indicates the best learning rate for SARSA in this game scenario is approximately 0.2. Learning rates above this leads to a drop in rewards per each episode and leads also to slow convergence of the algorithm.



C. Total Rewards per Learning Rate (Expected SARSA)

The plot shows the total rewards at each episode and learning rate for Expected SARSA. The plot shows a learning rate of approximately 0.2 ensures the highest reward for the case of Expected SARSA. Any rate above this drastically reduces the rewards and will lead to slow convergence of the algorithm. It is interesting to note that Expected SARSA is highly sensitive to a learning rate above 0.2 because of the drastic drop in total rewards.



V. Conclusions

SARSA and Expected SARSA are temporal difference algorithms for the control problem. SARSA makes use of the next action to find the best policy whereas Expected SARSA takes into account the expected value of all the possibles action in the next state. In deciding what to use for the control problem when building a game that uses an agent to teach users the best possible moves to make to achieve the highest rewards and also to progress from one stage to another, the developers must take into account the feedback of users. If users are adventurous and prefer to move quickly between stages of the card game, then developers should employ SARSA as it allows for exploration and randomness in actions taken. However, if the users are conservative and want a gradual process to learn the best possible set of actions then Expected SARSA should be used as it allows for exploitation and removes randomness and variability.

References

[1] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed., Cambridge, MA : The MIT Press, 2018, pp. 119–140.