Reinforcement Learning Algorithm for Adaptive Control - Evaluating Off-policy Q-Learning and On-policy Double SARSA Algorithms in the CartPole Environment

---

Abstract

This study presents a comparative analysis of Double SARSA and Q-learning, two prominent reinforcement learning algorithms, within the framework of the CartPole problem, a standard in adaptive control systems. Double SARSA, an enhancement of the SARSA algorithm, addresses Q-learning's overestimation bias by utilizing dual Q-tables updated alternately to provide an unbiased estimate of subsequent state-action values. Conversely, Q-learning employs a single Q-table updated based on the maximum Q-value of the next state, which can lead to optimistic value estimations. The performance of both algorithms is evaluated through code that measures training curves, Q-value heatmaps, learned policies, and reward distributions.These metrics provide an understanding of the decision-making processes and overall effectiveness of each algorithm. Additionally, this study includes a sensitivity analysis on Double SARSA due to its demonstrated robustness in training performance which proves to be slightly better than q-learning. The results highlight the contrasts of Double SARSA and Q-learning, focusing on their suitability for adaptive and self learning control tasks.

1. Introduction

In the rapidly evolving field of artificial intelligence, reinforcement learning (RL) has emerged as a critical methodology for systems to learn optimal behaviors through interaction with their environment. This report focuses on a comparative analysis of two significant reinforcement learning algorithms: Q-learning, an off-policy approach, and Double SARSA, an on-policy method. The study focuses on the CartPole problem, a task in the OpenAI Gym toolkit **[1]** that challenges algorithms to balance a pole on a moving cart, with the objective of establishing a self-learning system which is the main focus of our module “Adaptive Systems”.

The CartPole problem is a simple adaptive control system problem where success is measured by the algorithm's ability to maintain an unstable system in equilibrium. It serves as a platform to evaluate the strategic decision-making capabilities, adaptability, and efficiency of different RL algorithms under controlled conditions. Q-learning is a model-free, off-policy reinforcement learning algorithm that determines the optimal action based on the agent's current state. It operates independently of a fixed policy, hence its classification as off-policy. This approach does not rely on a predictive model of the environment; instead, it learns through trial and error, utilizing predicted outcomes to decide the best course of action without directly using a reward system for learning.**[2]**

Sarsa is a simple on-policy algorithm similar to Q-learning, which also learns action values at each step. Unlike Q-learning, however, Sarsa bases its updates purely on the states visited and actions taken, aligning closely with the actual policy followed. Consequently, Sarsa's action-value estimates do not converge with a constant learning rate, but with a sufficiently small rate, the policy can stabilize, balancing exploration and exploitation effectively.

For instance, using an epsilon-greedy policy, Sarsa tends to converge to a cautious strategy, avoiding states near others with high negative rewards. This strategy proactively considers the risk of random actions, steering the policy away from potentially risky "edges."**[3]**

Q-learning and Double SARSA are selected for their contrasting approaches: Q-learning's potential for rapid policy optimization through greedy maximization of expected rewards, and Double SARSA's strategy to mitigate overestimation by averaging updates across two Q-tables. This distinction is crucial for understanding how each algorithm processes and integrates environmental feedback to refine its policy.

This report evaluates how these algorithms perform on the CartPole environment, using metrics such as training progress, and episode duration. Emphasis is placed on sensitivity analysis for Double SARSA to examine its robustness and responsiveness to variations in learning rate and discount factor due to this algorithm's stable nature. This analysis enhances the understanding of how different reinforcement learning strategies perform in a standard test environment and also provides insights that could inform the development of more sophisticated algorithms capable of tackling a wider range of adaptive systems.

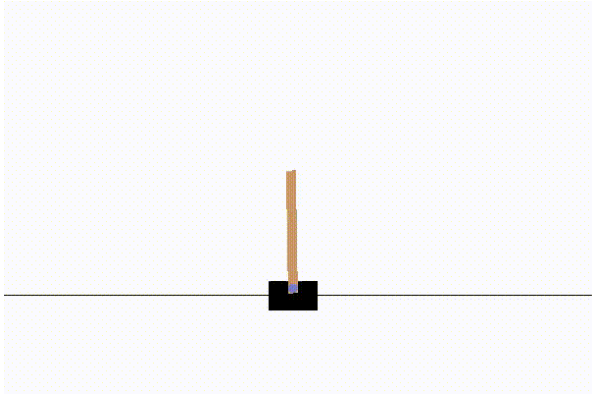


Figure 1. Cartpole Environment - Pygame simulation

2. Methods

2.1 CartPole Problem Setup

The experiments were conducted using the CartPole-v1 environment from the OpenAI Gym library. The objective is to balance an inverted pendulum (pole) on a movable cart by applying forces to either side of the cart. The environment consists of a cart-pole system where the action space is a 1D array with values 0 (push cart left) or 1 (push cart right). The observation space is a 4D array representing the cart's position, velocity, pole angle, and angular velocity, with specific ranges provided. The goal is to keep the pole balanced by pushing the cart left or right based on these observations. A reward of +1 is given for each time step the pole remains upright, with a threshold of 475 to solve the environment. An episode terminates if the pole angle exceeds ±12°, the cart position goes beyond ±2.4, or the episode length surpasses 500 steps. Each episode begins with the observations randomly initialized within the range of (-0.05, 0.05), providing varied starting conditions. The objective is to maximize the reward by keeping the pole upright as long as possible within these constraints through appropriate actions.

2.2 Algorithm Implementation

Two reinforcement learning algorithms, Q-learning and Double SARSA, were employed and compared in this study.

2.2.1 Q-learning

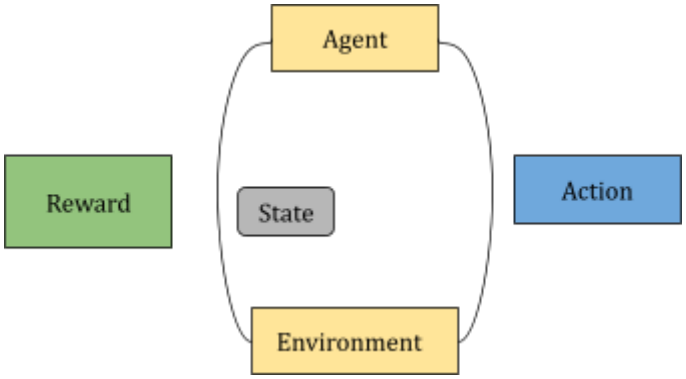


Figure 2. Q-learning schematic showing how the agent and environment are set-up **[2]**

Q-learning is an off-policy algorithm that aims to learn an optimal action-value function (Q-function) by iteratively updating the value estimates for each state-action pair. The Q-function is updated using the following equation:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1}, a_t)) \ldots\ldots \textbf{Equation. 1 [4]}$$

Where:
- s, a, s_t, a_t are the current state, action, next state, and next action, respectively.
- r is the immediate reward obtained after taking action a in state s.
- α is the learning rate parameter.
- γ is the discount factor parameter.

The pseudocode for Q-learning is as follows:**[4]**

```
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., epsilon-greedy)
        Take action a, observe r, s'
        Q(s, a) <- Q(s, a) + alpha * (r + gamma * max_a' Q(s', a') - Q(s, a))
        s <- s'
    until s is terminal
```

2.2.2 Double SARSA

Double SARSA is an extension of the SARSA algorithm, designed to address the overestimation bias present in Q-learning. It maintains two separate Q-tables, denoted as Q1 and Q2, and updates them alternately based on the other Q-table's estimate of the next state-action value. The update rules for Double SARSA are as follows:

With 50% chance, update either Q1:

$$Q_1(s,a) \leftarrow Q_1(s,a) + \alpha(r + \gamma Q_2(s', a') - Q_1(s,a)) \text{ ….. } \textbf{Equation. 2 [3]}$$

Or Q2;

$$Q_2(s,a) \leftarrow Q_2(s,a) + \alpha(r + \gamma Q_1(s', a') - Q_2(s,a)) \text{ ….. } \textbf{Equation. 3 [3]}$$

Where:
- s, a, s', a' are the current state, action, next state, and next action, respectively.
- r is the immediate reward obtained after taking action a in state s.
- α is the learning rate parameter.
- γ is the discount factor parameter.

The pseudocode for Double SARSA is: **[5]**

```
Initialize Q1(s, a) and Q2(s, a) arbitrarily
Repeat (for each episode):
    Initialize s
    Choose a from s using policy derived from Q1 and Q2 (e.g., epsilon-greedy)
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q1 and Q2 (e.g., epsilon-greedy)
        With probability 0.5:
            Q1(s, a) <- Q1(s, a) + alpha * (r + gamma * Q2(s', a') - Q1(s, a))
        Otherwise:
            Q2(s, a) <- Q2(s, a) + alpha * (r + gamma * Q1(s', a') - Q2(s, a))
        s <- s'; a <- a'
    until s is terminal
```

2.2.3 State-Action Representation

To handle the continuous state space in the CartPole problem, both algorithms discretized the state variables (cart position, cart velocity, pole angle, and angular velocity) into a fixed number of bins. This discretization allowed for a tabular representation of the Q-functions, where each state-action pair was associated with a specific Q-value.

2.2.4 Exploration Strategy

Both algorithms employed an epsilon-greedy exploration strategy to balance exploration and exploitation. With a probability of epsilon, the agent selected a random action to explore the environment; otherwise, it chose the action with the highest estimated Q-value (exploitation). The value of epsilon was gradually decreased over the course of training to shift the focus from exploration to exploitation.

2.2.5 Evaluation

To thoroughly assess the performance of Double SARSA and Q-learning algorithms in the CartPole-v1 environment, several visual tools and plots will be employed. These include Q-value heatmaps and policy heatmaps, which provide insights into the expected rewards and strategic decisions across various states, respectively. “Episode Length Over Time Plots” will be used to track the progression and consistency of the agent's performance. Policy visualizations when the cart is stationary will focus on the strategies from a neutral state. Together, these visualizations will offer a comprehensive analysis of the algorithms' effectiveness, stability, and adaptability in a dynamic learning environment.

2.2.6 Hyper- Parameter Setup

The hyper-parameters will be standardized across both algorithmic environments to ensure a consistent basis for comparison. For the sensitivity analysis, adjustments will be made to the learning rate and discount factor to explore their impact on the Double SARSA Algorithm.

Hyper-Parameter	**Value**	**Description**	**Purpose**
pos_space	np.linspace(-2.4, 2.4, 10)	Discrete bins for the cart's position.	Creates 10 evenly spaced values representing the cart's position range.
vel_space	np.linspace(-4, 4, 10)	Discrete bins for the cart's velocity.	Defines 10 bins for the range of the cart's velocities.
ang_space	np.linspace(-.2095, .2095, 10)	Discrete bins for the pole's angle.	Segments the pole's angle into 10 discrete bins.
ang_vel_space	np.linspace(-4, 4, 10)	Discrete bins for the pole's angular velocity.	Divides the pole's angular velocity into 10 categories.
learning_rate	0.1	The step size at each iteration.	Controls how much the Q-values are updated during learning.
discount_factor	0.99	The discount factor for future rewards.	Determines the importance of future rewards in the update of Q-values.
epsilon	1	Exploration rate.	Initially set to 1 for maximum exploration, it controls the

			trade-off between exploration and exploitation.
epsilon_decay_rate	0.00001	Rate at which epsilon decreases.	Gradually reduces the exploration rate to shift focus towards exploitation of learned policies.

2.3 Third-Party Libraries

he experiments relied on the following third-party libraries:

1. OpenAI Gym: This library is for developing and comparing reinforcement learning algorithms.
2. NumPy: A library for scientific computing in Python
3. Matplotlib and Seaborn: These were used to generate plots and visualizations for analyzing the algorithms' performance plots and Q-value heatmaps.
4. Pygame: This was used to display the trained cartpole environment using 'human render' mode.

3. Results

This section presents a comparative analysis of the results obtained from implementing two reinforcement learning algorithms, Double SARSA and Q-learning, within the CartPole-v1 environment. The effectiveness and learning dynamics of each algorithm are evaluated through an examination of the learned policies, Q-value heatmaps, and performance metrics over episodes. By analyzing these main plots, this analysis aims to highlight distinct characteristics and outcomes of each algorithm, offering insights into their relative strengths and weaknesses in balancing the pole on the cart.

3.1 Comparative Analysis- Double SARSA and Q- learning

3.1.1 Performance over Episodes

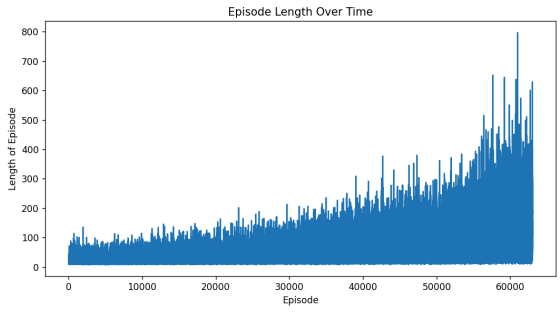
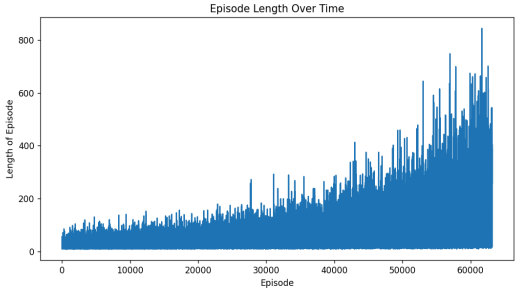
Double SARSA	Q- learning
	

Figure 3. Episode length comparison of double SARSA algorithm (left) and q-learning algorithm (right).

The graph provides a depiction of episode lengths throughout the course of training, indicating the number of time steps the agent was able to maintain the pole in an upright position before an episode terminated. An upward trend in episode length is evident as training progresses, with initial episodes being considerably shorter and later episodes extending significantly longer.

The graph demonstrates a similar increasing trend in episode lengths as observed with the Q-learning algorithm, suggesting that the Double SARSA algorithm also progressively improves in maintaining the balance of the pole over extended periods as training progresses. However, a notable distinction is observed in the training dynamics of the two algorithms. For Double SARSA, the episode length exhibits a smoother progression, particularly from **30,000 to 50,000** episodes, indicating a more gradual and consistent increase in both the maximum and average episode lengths. This pattern likely reflects a more stable learning process, potentially influenced by an optimal set of hyperparameters that foster steady improvement.

In contrast, the Q-learning algorithm, while also demonstrating an overall increase in episode lengths, shows more pronounced spikes and greater variability in the maximum lengths of episodes. These spikes might indicate instances where the learning algorithm achieved exceptional performance, but the frequent fluctuations could also suggest a less stable learning process. This variability indicates that while occasionally hitting high performance peaks, Q- learning lacks the consistency seen in Double SARSA, due to different exploration strategies or learning rate adjustments.

3.1.2 Q- value Divergence

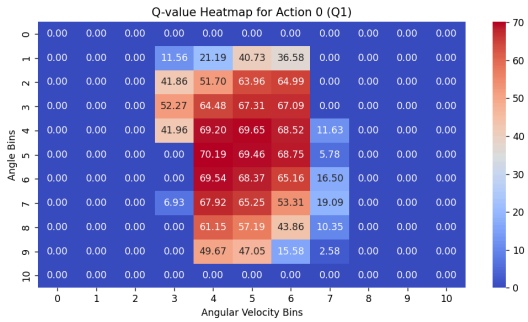
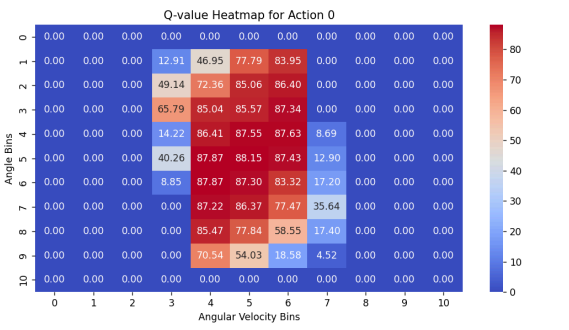
Double SARSA	Q- Learning
	

Figure 4. Q value heat map comparison of double SARSA algorithm (left) and q-learning algorithm (right).

The Q-value heatmap from Q-learning displays several high Q-values particularly concentrated across the middle range of angular velocity and angle bins, with the highest values prominently peaking around bins 4 to 5. This suggests that Q-learning has identified these states as particularly advantageous for taking Action 0, with a strong preference evident under conditions of moderate angular velocities, peaking at up to 87.63.

In contrast, the heatmap from Double SARSA presents a different pattern, where the highest Q-values, peaking at around 69.65, are more broadly distributed across the angular velocity spectrum, particularly between bins 3 to 6. In comparative implications, Q-learning often adopts an aggressive optimization strategy that targets the highest Q-values, resulting in higher values within its heatmap. While this approach may efficiently exploit specific states, it risks overfitting **[3]** and might not generalize robustly across varied environments.

In contrast, Double SARSA's method of updating two separate Q-tables and averaging them tends to yield more stable learning outcomes. This stability is evidenced by lower, more uniformly distributed Q-values, indicating a learning process that is more resilient to environmental variations and less susceptible to the overestimation bias typical of Q-learning. Behaviorally, while Q-learning appears to prioritize actions within a narrower scope of scenarios, particularly those involving moderate angular velocities, Double SARSA displays a broader distribution of Q-values across a wide range of states. This suggests a conservative yet adaptable strategy, potentially enabling more effective navigation through diverse and changing conditions.

3.1.3 Policy Comparison

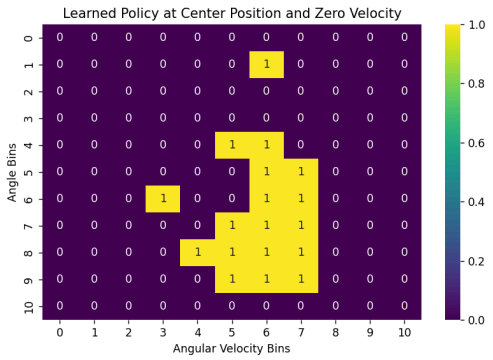
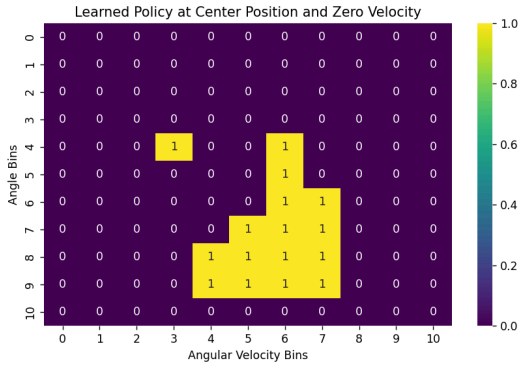
Double SARSA	Q- Learning
	

Figure 5. Learned Policy comparison of double SARSA algorithm (left) and q-learning algorithm (right).

Both Double SARSA and Q-learning algorithms demonstrate similar strategies with certain differences in their approach to handling the CartPole dynamics. They have each learned to push the cart to the right (action 1) in scenarios where such an action is deemed most corrective, particularly as the pole's angular velocity increases, indicating a shared understanding of how to counteract potential falls.

Additionally, both algorithms exhibit conservatism at lower velocities, favoring action 0, which suggests a common strategy for managing situations with less dynamic urgency. However, there are distinct variations in risk handling between the two; Q-learning appears slightly more aggressive or experimental in its application of action 1 at lower velocities. This could be attributed to its tendency to maximize Q-values based on expected rewards, leading to a different risk assessment compared to Double SARSA's method of averaging Q-values from two tables, which tends to produce more conservative estimates and decisions. Overall, In terms of developing a strategy, both algorithms perform in a similar fashion due to the same values of hyperparameters being used in the python code.

3.2 Analyses of Results- Double SARSA and Q-learning

3.2.1 Convergence Analysis - Exploration and Exploitation Strategy

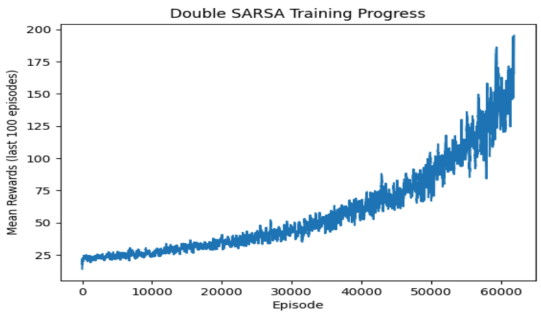
Double SARSA	Q- learning
	

Figure 6. Training Progress comparison of double SARSA algorithm (left) and q-learning algorithm (right).

The analysis of the "Training Progress" plot for both algorithms; Double SARSA and Q-learning can be seen to reveal a stable convergence toward an effective policy, achieved over approximately 40,000 episodes. This convergence is characterized by a gradual increase in mean rewards and a reduction in the variance and rate of change of these rewards, indicating learning. The overall trend shows smooth progression, with occasional fluctuations that aligns with the robustness criteria outlined in the OpenAI Gym documentation. The agent is expected to handle minor perturbations in initial conditions adeptly and develop a consistent policy that maximizes the duration of each episode.

A key factor contributing to the smooth convergence and robust policy development is the balance between exploration and exploitation, managed through an epsilon-greedy strategy. Initially, the agent predominantly explores the environment, facilitated by a high epsilon value. This exploration allows the agent to experience various states and update its Q-values accordingly. The relevant code snippet demonstrates this strategy:

```
if is_training and rng.random() < epsilon:
    action = env.action_space.sample()  # Exploration
else:
    action = np.argmax(q[state_p, state_v, state_a, state_av, :])
# Exploitation
```

As training progresses, epsilon gradually decays, shifting the agent's behavior from exploration to exploitation of the learned policy. This shift is governed by the following line of code:

```
epsilon = max(epsilon - epsilon_decay_rate, 0)
```

By the time of convergence, the average value of epsilon was noted to be 0.3 for both the algorithms, marking a significant transition from exploratory to informed decision-making behavior.

Although there remains some variability in episode lengths, indicative of continued exploration and adaptation, a notable decrease in this variability is observed in later episodes. This reduction points toward the stabilization of the agent's performance, suggesting that the underlying Q-values are converging towards optimality.

In summary, the agent's training within the CartPole-v1 environment demonstrates effective learning and policy development, marked by an upward trend in episode lengths and decreased variance, aligning with a successful balance between exploratory actions and the exploitation of a gradually maturing strategy. This progression underlines the capability of the epsilon-greedy approach in facilitating both robust and adaptable behaviors in dynamic learning environments.

3.2.2 Algorithm Efficiency and Handling Discrete Decisions

The efficiency of both Q-learning and Double SARSA, as well as their ability to handle discrete decisions, are pivotal in their application to environments like CartPole-v1, which is characterized by finite, discrete state and action spaces. These algorithms are well-suited for environments where the choices are limited and well-defined, due to their fundamental design that excels under such constraints.

In the CartPole-v1 environment, the action space is binary, consisting of two discrete options: moving the cart left or right. This simplicity aids in reducing computational complexity and enhances the decision-making process. The implementation of Q-learning is illustrated by the following code snippet, which selects actions based on the maximum Q-value and updates these values based on observed and anticipated future rewards:

```
# Action selection based on maximum Q-value
action = np.argmax(q[state_p, state_v, state_a, state_av, :])

# Q-value update based on observed reward and maximum future reward
q[state_p, state_v, state_a, state_av, action] += learning_rate_a * (
   reward + discount_factor_g * np.max(q[new_state_p, new_state_v, new_state_a,
new_state_av, :]) - q[state_p, state_v, state_a, state_av, action]
)
```

Similarly, Double SARSA's implementation uses a pair of Q-tables to reduce overestimation bias by alternating which table is updated, thus potentially increasing the stability and accuracy of the learning process. The implementation of Double SARSA in the same environment involves discrete decision handling as follows:

```
# Combined Q-table for action selection
combined_q = (q1 + q2) / 2
action = np.argmax(combined_q[state_p, state_v, state_a, state_av, :])

# Alternating Q-table updates
if np.random.rand() < 0.5:
 new_action = np.argmax(q1[new_state_p, new_state_v, new_state_a, new_state_av, :])
 q1[state_p, state_v, state_a, state_av, action] += learning_rate * (
  reward + discount_factor * q2[new_state_p, new_state_v, new_state_a, new_state_av,
new_action]
 - q1[state_p, state_v, state_a, state_av, action]
 )
else:
```

```
 new_action = np.argmax(q2[new_state_p, new_state_v, new_state_a, new_state_av, :])
 q2[state_p, state_v, state_a, state_av, action] += learning_rate * (
  reward + discount_factor * q1[new_state_p, new_state_v, new_state_a, new_state_av,
new_action]
 - q2[state_p, state_v, state_a, state_av, action]
 )
```

Both methods incrementally update their respective Q-values with observed and anticipated future rewards, directing the agent toward actions that extend episode durations and increase cumulative rewards. These updates are particularly effective in environments with discrete decisions, like CartPole-v1, providing a mechanism for the agent to learn the value of each action in various states and subsequently make choices.

However, it's crucial to recognize that the efficiency of these algorithms may decline as the complexity of the environment increases. In scenarios where the state and action spaces expand, the resources required for storing and updating the Q-tables can grow, potentially challenging the practical application of these methods.

3.2.3 Sensitivity Analysis of Double SARSA Algorithm

The choice to conduct a sensitivity analysis specifically on the Double SARSA algorithm, rather than Q-learning, was influenced by several key factors. Firstly, Double SARSA and Q-learning have fundamental differences in their update mechanisms; Double SARSA's on-policy approach contrasts with Q-learning's off-policy method, which can lead to more stable and robust policies in environments with varied reward dynamics and state transitions. Secondly, Double SARSA's method of averaging two Q-tables may offer more generalizable and stable outcomes, making it a valuable subject for understanding the impact of hyperparameter changes in realistic scenarios with inherent noise and uncertainty. The section examines the effect of change of learning rate and discount factor on the performance of the Double SARSA algorithm.

3.2.3.1 Learning Rate

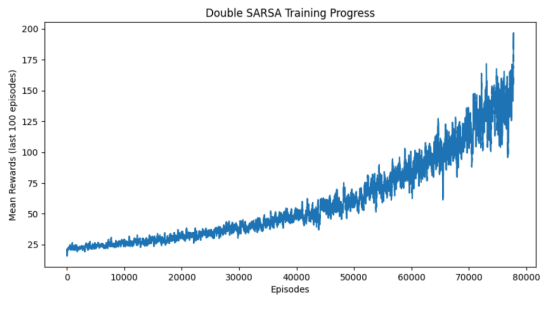
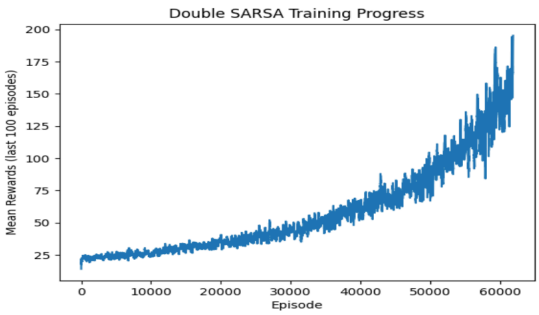
Learning Rate= 0.5	Learning Rate= 0.1
	

Figure 7. Training Progress comparison at lr-0.5 (left) and lr=0.1 (right) of double SARSA algorithm

Higher learning rates may lead to faster convergence but potentially unstable behavior, while lower rates can result in slower learning. In a sensitivity analysis of the Double SARSA algorithm, two learning rates, 0.5 and 0.1, were assessed to determine their impact on training performance. The analysis revealed significant differences

in how each rate affected the learning dynamics. At a learning rate of 0.5, the training showed a volatile yet rapid improvement in performance, with mean rewards increasing sharply over 100 episodes. This rapid adaptation, however, was accompanied by increased fluctuations in reward patterns, indicating potential instability or overfitting **[3]**. In contrast, a learning rate of 0.1 led to a more consistent and gradual improvement, suggesting that a slower rate fosters more stable learning, albeit at the cost of slower convergence to optimal policies.

The total number of episodes required to stabilize the learning outcomes further illustrated the effects of the learning rate. With a rate of 0.5, more than 80,000 episodes were needed, whereas only about 60,000 episodes were required at a rate of 0.1. This indicates that a higher learning rate, while speeding up initial learning phases, may lead to frequent policy overshoots and require more episodes to achieve policy refinement and stability. On the other hand, a lower learning rate promotes efficiency and earlier stabilization, likely due to smaller, more conservative updates to Q-values.

These findings underscore the importance of choosing an appropriate learning rate based on the specific environmental dynamics and learning objectives. A higher rate might be suitable for dynamic environments requiring rapid adaptation, whereas a lower rate could be beneficial in more stable settings where gradual, consistent learning is preferable.

3.2.3.2 Discount Factor

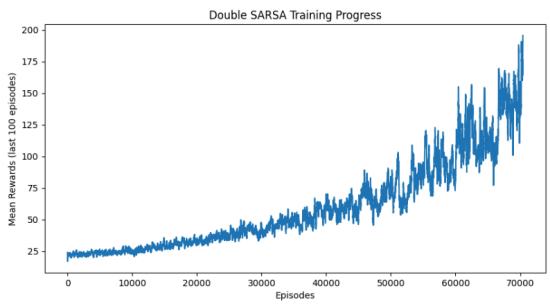
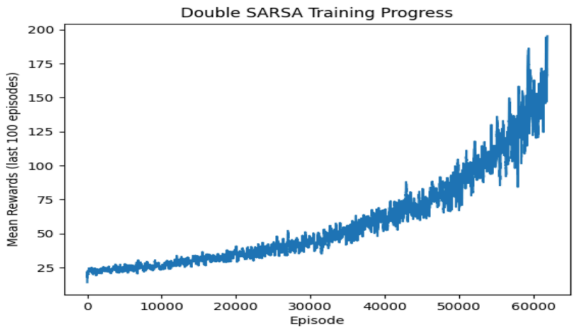
Discount Rate= 0.6	Discount Rate= 0.99
	

Figure 8. Training Progress comparison at df=0.6 (left) and df=0.99 (right) of double SARSA algorithm

The discount factor determines the importance of future rewards, impacting the agent's decision-making and the learned policy. In the sensitivity analysis focusing on the discount factor of the Double SARSA algorithm, different settings are assessed; a high discount factor of 0.99 versus a lower one at 0.6. The training outcomes provide a clear illustration of how the valuation of future rewards influences the learning trajectory and decision-making process of an agent.

With a discount factor of 0.99, the training progress displayed a strong and steady upward trend in mean rewards, indicative of the agent's strategy to weigh future rewards nearly as heavily as immediate ones. This approach encourages comprehensive, long-term strategic planning, as the agent learns to perform actions that promise sustained benefits, leading to more consistent and smooth improvement over time. The smoother progression with a higher discount factor suggests that the agent is effectively optimizing for the long-term, reflecting a mature understanding of the environment's reward dynamics.

Conversely, setting the discount factor to 0.6 resulted in a more erratic learning curve with noticeable fluctuations in performance. This lower discount factor prioritizes immediate rewards over future gains, which tends to produce a strategy that may neglect the fuller potential of future rewards. As a result, the learning process exhibits increased volatility and less consistency, demonstrating how a focus on short-term gains can lead to frequent adjustments in policy and a less stable approach to maximizing rewards.

These contrasting outcomes underscore the critical role of the discount factor in reinforcement learning. A higher discount factor is well-suited for scenarios where long-term benefits are crucial and the environment allows for strategic foresight. In contrast, a lower discount factor might be preferable in situations where immediate results are more significant or when the future is too unpredictable. Adjusting the discount factor thus allows for fine-tuning the agent's focus between immediate and future rewards, optimizing learning and performance according to the specific challenges and objectives of different tasks and environments.

4. Discussion

Based on the presented results and analysis, several key points can be highlighted from this study regarding the performance and characteristics of the reinforcement learning algorithms applied to the CartPole environment. Both algorithms' performance had a degree of similarity to them due to standardization of hyper-parameters across the analyses. The heat maps revealed that the agent effectively learned to prioritize actions that stabilize the pole's position, particularly in states with higher angular velocities. Additionally, the episode length trend over time showcased the agent's ability to improve its performance and maintain the pole's balance for longer durations as training progressed. The gradual increase in episode lengths, accompanied by a reduction in variability, indicated convergence towards an effective policy. Furthermore, the training progress plot illustrated the agent's successful learning, as the mean reward surpassed and sustained above the threshold of 195, indicating a stabilized environment. The learned policy heatmap also highlighted the agent's adaptability, consistently choosing actions to recenter the pole at extreme angles while responding to variations.

Compared to Q-learning, the Double SARSA algorithm exhibited a smoother progression in episode lengths during training, suggesting a more stable and consistent learning process. The Q-value heatmap for Double SARSA displayed a broader distribution of moderately high Q-values across various states, indicating a more conservative and adaptable approach to state-action valuation. While the learned policies of both algorithms shared similarities in handling CartPole dynamics, Double SARSA appeared slightly more conservative in its risk assessment, due to its averaging method for Q-value updates. The contrasting behaviors of these algorithms can be attributed to their fundamental differences in update mechanisms – Q-learning follows an off-policy approach, while Double SARSA is an on-policy algorithm. The off-policy nature of Q-learning allows it to update Q-values based on the maximum expected future reward, potentially leading to more aggressive optimization. In contrast, Double SARSA's on-policy updates, which consider the next action chosen by the policy, tend to yield more conservative and stable estimates, contributing to its smoother learning curve and broader distribution of Q-values.

The sensitivity analysis on the Double SARSA algorithm highlighted the impact of hyperparameters, particularly the learning rate and discount factor, on the learning dynamics and convergence. Higher learning rates led to

faster initial learning but increased fluctuations and potential instability, while lower rates promoted more gradual and stable learning at the cost of slower convergence. Furthermore, a higher discount factor (e.g., 0.99) encouraged long-term strategic planning and resulted in smoother improvement over time, while a lower discount factor (e.g., 0.6) prioritized immediate rewards, leading to more erratic spikes in the learning curve.

Overall, both Q-learning and Double SARSA demonstrated effective learning and policy development, as evidenced by the upward trends in episode lengths and successful task completion. However, Double SARSA exhibited a more stable learning process, potentially due to its averaging method and reduced susceptibility to overestimation bias. The Q-value divergence between the two algorithms highlighted their different optimization strategies, with Q-learning showing more aggressive optimization and Double SARSA favoring stability and adaptability across a broader range of scenarios.

References

[1] Cart Pole. (n.d.). Gym Documentation. Retrieved May 2, 2024, from https://www.gymlibrary.dev/environments/classic_control/cart_pole/

[2] Guide, S., & Banoula, M. (2023, November 6). What is Q-Learning: Everything you Need to Know. Simplilearn.com. Retrieved May 2, 2024, from https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning#what_is_qlearning

[3] Ganger, M. , Duryea, E. and Hu, W. (2016) Double Sarsa and Double Expected Sarsa with Shallow and Deep Learning. Journal of Data Analysis and Information Processing, 4, 159-176. doi: 10.4236/jdaip.2016.44014.

[4] Tesauro, G. (n.d.). Q-learning. Wikipedia. Retrieved May 3, 2024, from https://en.wikipedia.org/wiki/Q-learning

[5] guide, s. (2020, April 24). Learning with Deep SARSA & OpenAI Gym | by Gelana Tostaeva | The Startup. Medium. Retrieved May 3, 2024, from https://medium.com/swlh/learning-with-deep-sarsa-openai-gym-c9a470d027a

---

Abstract: 162 Words
Intro: 457 Words
Results: 2315 Words
Discussion: 514 Words;
Total= 3448 WC (with abstract)