# Contents

# 1 Introduction

The objective of this project is to evaluate and compare the performance of agents trained using various reinforcement learning algorithms. Specifically, the study examines three algorithms categorized under both tabular and approximative methods:

- SARSA($\lambda$) with Fixed Exploration (limited exploration),
- SARSA($\lambda$) with Decayed Exploration (enhanced exploration), and
- Semi-Gradient SARSA($\lambda$) (Approximation method)

These algorithms were employed to train agents in the Connect X environment, a strategic game that requires effective decision-making. The trained agents were tested against two types of opponents: **Random**, which selects moves arbitrarily, and **Negamax**, a more strategic adversary utilizing a search-based algorithm. Performance evaluation was conducted based on win rates and average rewards, providing insights into how the learning process evolves over time.
This study offers a comparative analysis of different reinforcement learning strategies, particularly in balancing exploration and exploitation, and examines how these choices influence an agent's ability to adapt and compete against varying levels of opponents.

# 2 Methodology

## 2.1 Environment Setup

The experimental environment for training and evaluating the agents is designed based on the Connect X game, a strategic turn-based game that requires players to align a specified number of pieces in a row to secure a win. The specific configuration of the environment used in this study consists of:

- Grid Dimensions: 4 rows $\times$ 5 columns
- Winning Condition: A player must align three pieces in a row (horizontally, vertically, or diagonally) to achieve victory.

To effectively assess the learning performance of different reinforcement learning algorithms, agents were trained for a predefined number of episodes. The training process was structured as follows:

- SARSA($\lambda$) with Fixed Exploration (limited exploration) and SARSA($\lambda$) with Decayed Exploration (enhanced exploration) were trained for a total of $5,000$ episodes each.
- Semi-Gradient SARSA($\lambda$) (which utilizes function approximation) was trained for a longer duration, with a total of $10,000$ episodes, to allow the agent to generalize better over a larger state space.

This setup ensures that each algorithm has a sufficient number of training iterations to develop an effective policy while allowing for a comparative analysis of their learning efficiencies. The difference in training episodes between tabular and approximative methods accounts for the additional complexity introduced by function approximation, which typically requires more training data to converge to an optimal strategy.

## 2.2 Chosen Algorithms

The first algorithm follows the State-Action-Reward-State-Action (SARSA) framework with eligibility traces, where $\lambda$ controls the extent to which past experiences influence the learning process. In this variant, the agent maintains a fixed exploration rate, meaning it selects a predefined percentage of random actions for exploration while relying on the learned policy for the remaining actions (exploitation). This approach ensures a consistent level of exploration throughout training, allowing the agent to continuously test alternative strategies rather than fully converging to a deterministic policy early on.
The second algorithm introduces a decaying exploration rate, where the agent initially explores more frequently but gradually reduces exploration as training progresses. At the beginning of the learning process,

the agent takes a large number of random actions, allowing it to explore a wide range of state-action pairs. Over time, the exploration rate decays following a predefined schedule, leading the agent to increasingly exploit its learned policy. This strategy helps strike a balance between early exploration (to gather diverse experiences) and later exploitation (to optimize decision-making based on learned patterns).
Unlike traditional SARSA($\lambda$), which relies on tabular methods to store and update Q-values, the third algorithm (Semi-Gradient SARSA ($\lambda$) utilizes function approximation to estimate the value function, making it suitable for environments with large or continuous state spaces. Instead of maintaining a lookup table for state-action values, the agent employs a neural network as a function approximator, updating its parameters using gradient descent based on the temporal-difference (TD) error.

### 2.3 Training parameters

To optimize the performance of the reinforcement learning agents, a set of training parameters was carefully selected. These parameters influence the learning dynamics, convergence speed, and overall effectiveness of the trained policy. Additionally, a grid search approach was employed to systematically tune the hyperparameters. The key training parameters used in this study are as follows:

1. Learning Rate ($\alpha$): The learning rate ($\alpha$) determines how much new information overrides previously learned values during Q-value updates. A value of 0.1 was selected to balance learning stability and adaptability, preventing overly aggressive updates while allowing steady improvements in the agent's policy.

2. Discount Factor ($\gamma$): The discount factor ($\gamma$) dictates the importance of future rewards relative to immediate rewards. A high value of 0.99 was chosen to prioritize long-term reward maximization, encouraging the agent to develop strategies that optimize cumulative performance rather than short-term gains.

3. Exploration Rate ($\epsilon$, Epsilon-Greedy Strategy): To balance exploration and exploitation, the epsilon-greedy strategy was implemented. The agent selects a random action with probability $\epsilon = 0.1$, while with probability 1 - $\epsilon = 0.9$, it follows the learned policy. This ensures that while the agent predominantly exploits its knowledge, it still explores alternative actions to discover potentially better strategies.

4. Eligibility Trace Decay ($\lambda$): The eligibility trace decay parameter ($\lambda$) defines how past experiences influence current updates in the SARSA($\lambda$) algorithm. A decay value of 0.9 was chosen to allow the agent to retain significant traces of past experiences, leading to faster and more efficient learning by associating previous actions with long-term rewards.

### 2.4 Evaluation Metrics

To assess the performance of the trained agents, a set of quantitative evaluation metrics was employed. These metrics provide insights into the agent's ability to learn optimal strategies, its consistency in decision-making, and the effectiveness of the reinforcement learning algorithms used. The following key evaluation metrics were considered:

- Win Rate: The win rate is a fundamental metric that measures the agent's overall effectiveness in gameplay. It is defined as the proportion of games won by the agent against its opponents over a series of evaluation matches. A higher win rate indicates that the agent has successfully learned a competitive policy, while a lower win rate suggests that further optimization may be required.

- Temporal-Difference ($TD$) Error: TD Error represents the discrepancy between the predicted Q-value (expected future reward) and the actual reward received. It serves as an indicator of how accurately the agent's value function approximates true state-action values. A high TD error suggests that the agent's policy is still adjusting, whereas a decreasing TD error indicates that learning is stabilizing.

- Reward Variance: The reward variance measures the fluctuation in rewards obtained by the agent across different episodes. A low variance suggests that the agent has learned a stable and consistent strategy, while a high variance may indicate instability or excessive exploration.

## 3 Results

### 3.1 Agent 1: SARSA($\lambda$) with Fixed Exploration

Agent 1 was trained using the SARSA($\lambda$) algorithm with Fixed Exploration ($\epsilon = 0.1$), a reinforcement learning method that incorporates eligibility traces along with a constant exploration rate. In this approach, the agent follows a fixed epsilon-greedy strategy, where it selects a random action with a predefined probability ($\epsilon = 0.1$) and exploits its learned policy for the remaining actions (with a probability of 1 - $\epsilon$). This strategy ensures a balance between exploration of the environment and exploitation of the knowledge gained from previous experiences.
The agent underwent training over 5,000 episodes. During this process, its performance was evaluated over the course of 500 episodes, with key metrics—such as win rate and reward variance—being tracked at every 10-episode interval. This regular evaluation allowed for a detailed analysis of the agent's progress and provided insights into the stability and effectiveness of its learning over time.
For the final validation of the trained agent's performance, it was tested by playing against two types of opponents: a Random opponent, which selects actions at random, and a Negamax opponent, which utilizes a more strategic search-based algorithm. The agent's performance was assessed through 40 test games, where its ability to adapt, generalize, and compete against both random and strategic opponents was scrutinized.

#### 3.1.1 Win Rate Over Episodes

In the first 100 episodes, the agent's win rate fluctuates between 0.90 and 0.98, indicating rapid improvement in performance. Initially, the win rate increases steadily, but minor fluctuations suggest ongoing learning and adjustments. From episode 100 to 200, the win rate stabilizes around 0.92, showing that the agent has found a relatively strong strategy. However, occasional dips indicate that the agent is still refining its approach and may not have reached optimal performance yet. In the final 300 episodes, the win rate remains consistently around 0.92 to 0.93, demonstrating that the agent has achieved a stable level of performance. The minimal fluctuation suggests that learning has plateaued, and the agent has settled into a reliable strategy with little further improvement.

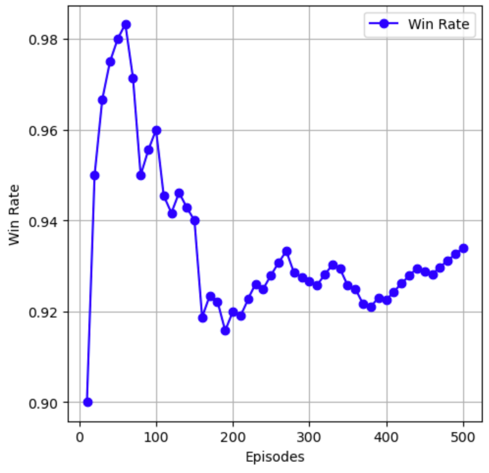

Figure 1: Agent 1 Win Rate over 500 episodes

#### 3.1.2 Reward Variance Over Time

Early on, the reward variance fluctuates between 0.00 and 0.64, indicating that the agent's performance is still inconsistent. Lower variance (0.00) suggests the agent is consistently winning, while higher values (0.36–0.64) indicate occasional losses or draws. From episode 100 to 200, the reward variance continues to show some fluctuation, with values ranging between 0.00 and 0.96. A variance of 0.96 (at episode 160) suggests a high level of inconsistency in rewards, possibly due to changes in the agent's strategy or external randomness in the environment. However, episodes with 0.00 variance indicate stretches of consistent wins. In the final 300 episodes, the reward variance generally stabilizes around 0.00 to 0.64, with occasional jumps. The continued presence of 0.36 and 0.64 suggests that while the agent has achieved stable performance, there are still some unpredictable elements causing occasional variations in rewards

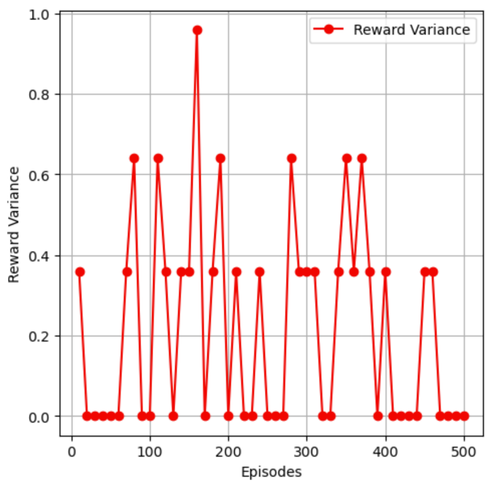

Figure 2: Agent 1 reward variation over 500 episodes

Overall, the SARSA algorithm enables the agent to learn a relatively strong policy, as reflected in the win rate stabilizing around 0.92 to 0.93 and the frequent perfect rewards of 1.00 in later episodes. However, the agent experiences fluctuations in reward variance, especially in the early and middle stages of training, indicating instability in decision-making. These fluctuations suggest that a fixed exploration strategy may be limiting the agent's ability to fully optimize its policy. Implementing a decaying exploration rate could help by allowing the agent to shift from exploration to exploitation more effectively. Despite these variations, the agent achieves consistent performance, maintaining a high win rate and strong overall learning progress.

## 3.2 Agent 2: SARSA($\lambda$) with Decayed Exploration

Agent 2 was trained using the SARSA($\lambda$) algorithm with Decayed Exploration, a reinforcement learning approach that integrates eligibility traces with an exploration strategy that decays over time. This method allows the agent to begin with a high level of exploration and gradually shift towards exploiting its learned policy as the training progresses. The decaying exploration rate ensures that in the early stages of training, the agent is encouraged to explore the environment broadly, and as the training continues, the agent increasingly focuses on exploiting the strategies it has learned, leading to more efficient decision-making.
The training process followed a similar structure to that of the previous SARSA($\lambda$) agent, with the key difference being the decay of the exploration rate over time. Initially, the agent explored a wider range of actions, which led to some variability in performance. However, as exploration decreased, the agent increasingly exploited its learned strategies, resulting in more stable and consistent performance. The decay factor helped to strike a balance between exploration and exploitation, ensuring that the agent did not prematurely converge to a suboptimal policy.

### 3.2.1 Win Rate Over Episodes

Throughout the first 10 episodes, the agent demonstrates a flawless performance with a win rate of 1.00, suggesting that it either quickly learns an optimal strategy or the initial environment is relatively easy. However, by episode 20, the win rate drops slightly to 0.95, possibly indicating increased exploration or a shift in environmental dynamics. As training progresses, the agent adapts, and its win rate gradually recovers to 0.97 by episode 30 and 0.98 by episode 50, showing signs of refinement. By episode 70, the agent stabilizes at 0.99, maintaining this near-perfect success rate consistently until episode 200. The agent quickly regains stability, continuing to perform at 0.99 for most of the training process from episodes 200 to 400. Finally, around episode 410, the agent reaches peak performance again, with a win rate returning to 1.00, which it maintains until the end of training at episode 500. This suggests that the agent has fully optimized its strategy, achieving perfect mastery of the task.

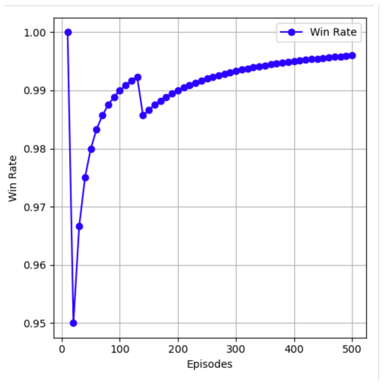

Figure 3: Agent 2 Win Rate over 500 episodes

### 3.2.2 Reward Variance Over Time

During the initial exploration phase the reward variance remains at 0.00, indicating that the agent consistently receives the same rewards across episodes. However, by episode 20, the reward variance spikes to 0.36, suggesting variability in the agent's performance, possibly due to exploration or adapting to new strategies. As training progresses, the agent improves, and by episode 30, the reward variance drops back to 0.00, signifying stable and consistent performance. From episodes 30 to 130, the agent maintains this stability, reinforcing the idea that it has found an effective policy with minimal fluctuations in rewards. A slight deviation occurs around episode 140, where the reward variance rises again to 0.36, mirroring a dip in the average reward. This suggests that the agent might be experimenting with different strategies or encountering more diverse situations. However, this fluctuation is short-lived, as the reward variance returns to 0.00 from episode 150 onward, indicating that the agent has regained consistency. From episodes 150 to 500, the reward variance remains at 0.00, confirming that the agent has fully optimized its policy. The absence of variability during this period signifies that the agent consistently receives maximum rewards, demonstrating mastery over the environment.

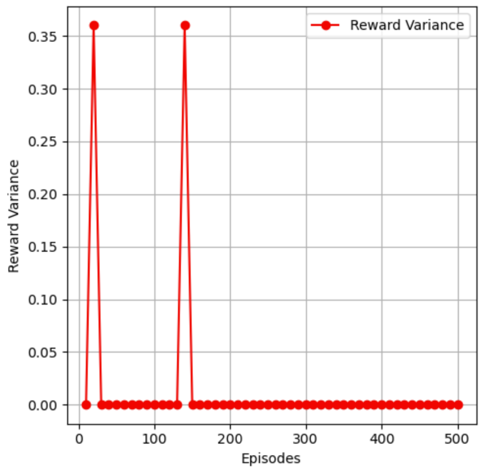

Figure 4: Agent 2 reward variation over 500 episodes

Overall, the agent demonstrates effective learning in the training process with consistent performance throughout the whole episodes. As training progresses, the agent enters a phase of exploration, resulting in fluctuations in performance, but it is able to stabilize and return to a high level of performance by the end of training.

## 3.3 Agent 3: Semi-Gradient SARSA($\lambda$) (Approximation method)

Agent 3 was trained using the Semi-Gradient SARSA($\lambda$) algorithm, a variant of SARSA($\lambda$) that utilizes function approximation to estimate the value function. The agent utilizes eligibility traces in conjunction with the approximated Q-values to update its policy. These approximations allow the agent to handle environments with large state spaces more efficiently. To approximate the Q-function, a neural network-based Q-function was implemented. The input layer accepts the state representation, which is a vector of dimension. This state representation encodes the current environment conditions, which the agent uses to decide on its actions. The network includes two fully connected hidden layers with 64 and 32 neurons, respectively. These layers use ReLU activations to introduce non-linearity into the model, enabling the network to learn complex patterns and representations from the state-action space. The output layer consists of a number of neurons equal to the number of possible actions, corresponding to the action-value estimates for each action. This allows the agent to evaluate which actions are most beneficial based on its learned policy.

### 3.3.1 Win Rate Over Episodes

Over the course of 500 episodes, the agent showed steady improvement in its win rate. Starting at 0.60 after 10 episodes, it progressively increased to 0.80 by episode 50, marking a significant turning point. Although there were slight fluctuations in the win rate, it remained relatively stable at 0.81-0.82 from episode 150 onward. The highest win rate of 0.83 was achieved at episode 250, but after that, the win rate stabilized around 0.80. By episode 500, the agent's performance had leveled off, reflecting a consistent and reliable learning process. In summary, the agent's win rate improved steadily, reaching a peak at episode 250, and ultimately settled at a strong and stable 0.80 by episode 500.

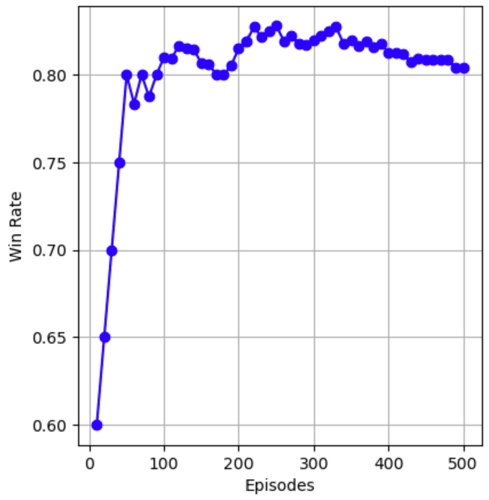

Figure 5: Agent 3 Win Rate over 500 episodes

### 3.3.2 Reward Variance Over Time

Reward variance started high at 0.96 after 10 episodes, indicating a high variability in the rewards. As the agent improved, the variance dropped significantly to 0.00 after 50 episodes, reflecting more consistent performance. This consistency remained stable, with reward variance fluctuating between 0.00 and 0.36 in

the later episodes, signifying that the agent's performance became more predictable and stable.
In summary, as the agent progressed through training, it showed a steady improvement in stabilizing reward variance, indicating better performance and more consistent decision-making. These trends highlight the agent's learning process, where it became more accurate in its predictions and its performance became more reliable.

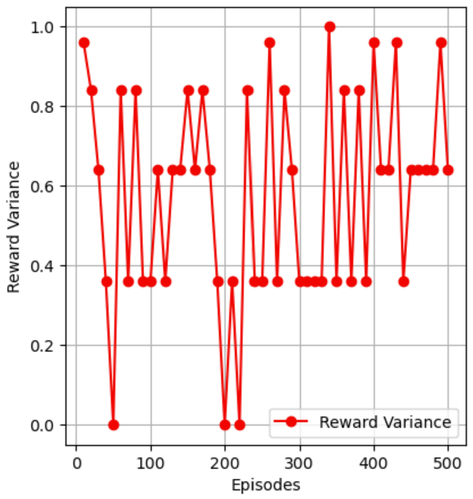

Figure 6: Agent 3 reward variation over 500 episodes

## 4 Conclusion

The prioritization of agents is determined based on their performance when tested against both a random agent and a negamax agent. The random agent serves as a baseline, providing a comparison for assessing the agent's ability to make meaningful decisions, while the negamax agent represents a more strategic opponent, offering a higher level of challenge. By evaluating each agent's success rate and win percentage, it is possible to gauge their proficiency and overall learning progress. The agents that demonstrate superior performance, especially in terms of strategic decision-making and consistency across a range of scenarios, are prioritized for further evaluation or deployment. This approach ensures that only agents capable of handling both random and more strategic opponents are considered for continued training or application.

Table 1: Comparative Performance of Agents Against Random and Negamax Opponents

| **Agents** | **Performance** | **Against Random** | **Against Negamax** |
|---|---|---|---|
| SARSA($\lambda$) with Fixed Exploration (Few Explorations) | Wins | 37 | 7 |
| | Losses | 3 | 33 |
| | Draws | 0 | 0 |
| SARSA($\lambda$) with Decayed Factor (More Explorations) | Wins | 40 | 40 |
| | Losses | 0 | 0 |
| | Draws | 0 | 0 |
| Semi-Gradient SARSA($\lambda$) | Wins | 32 | 0 |
| | Losses | 8 | 40 |
| | Draws | 0 | 0 |

SARSA($\lambda$) with Decayed Exploration proved to be the most robust agent, performing well both against the Random agent (100% win rate) and the Negamax agent (100% win rate). The decayed exploration strategy allowed it to learn and adapt more efficiently compared to SARSA($\lambda$) with fixed exploration.
Semi-Gradient SARSA($\lambda$) demonstrated weaker performance, particularly against Negamax, where it failed to secure any wins. This suggests that Semi-Gradient SARSA($\lambda$) might not be suitable for environments

requiring strong strategic depth or adversarial play, as it seems less capable of handling the complexity of agents like Negamax. In conclusion, SARSA($\lambda$) with Decayed Exploration appears to be the most effective approach in this experiment, especially in balancing exploration and exploitation in both simpler and more complex environments. However, further refinements and tuning might be necessary for more challenging adversarial agents, especially when compared to Negamax's deterministic and strategic behavior.