Abstract

Reinforcement learning (RL) enables agents to learn optimal strategies through trial and error, making it particularly effective in dynamic and complex environments like board games. Connect Four is a popular two-player abstract strategy game. The objective of the game is to connect four of one's own discs of the same colour vertically, horizontally, or diagonally. RL has shown great success in optimising game strategies. In this research, we aim to apply RL techniques to optimise Connect Four game strategies.

In this work, we review different RL methodologies that form the state of the art, implement a RL model that can learn optimal Connect Four game strategies, and fine-tune different RL parameters to optimise the performance of the model.

The RL implementation consists of an architecture that manages game state, allows players to drop pieces, and checks for winning conditions. A Q-learning agent learns optimal strategies through reinforcement learning, utilising a Q-table to evaluate actions based on rewards received from gameplay. The agent employs an epsilon-greedy strategy for action selection, balancing exploration and exploitation, and updates its Q-values based on the outcomes of its actions. The training process involves simulating multiple game episodes, enabling the agent to adapt and improve its performance over time.

The Q-learning agent implemented for Connect Four successfully learned to master the game by optimising its Q-values and leveraging reinforcement learning concepts. The observed metrics show a clear learning progression, with the agent improving both in terms of winning games and making more efficient, optimal moves. The results illustrate the potential of Q-learning as an effective approach to training agents in competitive environments with complex state spaces.

1. Problem statement

Connect-Four is a two-player, abstract strategy game created by Howard Wexler and first marketed in 1974 [1].

Classically, it is played on a 6-row, 7-column vertical board. Competing players choose a colour and then take turns dropping coloured discs into the playing board. The pieces fall vertically to occupy the lowest available space within the selected column. The first player to form a horizontal, vertical or diagonal sequence of four tokens of his(her) selected colour, wins.

Connect-Four is an adversarial, zero-sum game with perfect information. The positions to evaluate are in trillions, but the game is proved to be computationally solved using minimax on an 8-ply database with alpha-beta pruning [6]. It has been studied extensively in the field of artificial intelligence due to its complexity and challenging nature. As part of this work, we review and explore how Reinforcement Learning (RL) techniques could be employed to implement a Connect-Four game. We further evaluate how different implementation choices and parameters impact its performance.

2. Background

2.1 Reinforcement Learning

An essential mechanism of human learning is by interacting with our environment. We make choices, observe consequences, and adapt future choices based on the experience we build. The domain of **Reinforcement Learning** (RL) is the structured, computational model of Machine Learning inspired from this key pathway of human learning. It is a goal-finding paradigm, which learns from the consequences (aka, rewards) from previous choices and makes successively better, goal-friendly choices with time. RL differs from **Supervised Learning** - which relies on labelled data, i.e. an input state and a correct action for the given state, and also from **Unsupervised Learning** which focuses on the extraction of structure in volumes of seemingly unstructured, unlabelled data [2].

RL formulations are typically modelled as interactions between an **agent** and its **environment** at discrete time-steps. These are elegantly represented through a **Markov Decision Process** framework **(S, A, P, R)** (Figure 2.1), where **S** enumerates a set of environmental states, **A** enumerates the set of the agent's actions, **P** (AxS => R) models the probability of the system transitioning from a given state to another given state under influence of a given action by the agent, and **R** models the Reward obtained with the corresponding transition [3]. The objective of the Markov Decision Process is to build a policy function that could specify the specific action which the agent would use when in a given state, so as to maximise the cumulative reward across time.

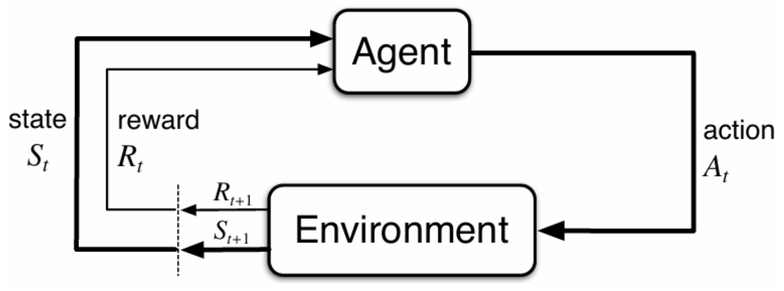


Figure 2.1 Markov Decision Process

Source: Barto, Sutton, Reinforcement Learning

2.2 Reinforcement Learning Methods

RL has been effectively applied to various application areas that require a non-human agent to learn from its actions 'on the job'. This includes application of RL to board games like Tic-tac-toe [4] and Snake [5]. Methods of implementing RL-based solutions typically map to one of two broad classes:

- **Model-Based RL**, where the agent creates a model of the environment to predict outcomes and make decisions. In situations where the environment is predictable, model-based methods can learn rapidly and effectively even before they are exposed to the real environment.

  Some popular Model-Based RL algorithms are Monte-Carlo Tree Search (MCTS) [7] and Temporal Difference Learning (TDL) [8]. The MCTS algorithm plans several steps forward by looking ahead at possible moves and outcomes, while the TDL approach learns from the delta between what the internal model predicted and what was encountered in the real environment.

- **Model-Free RL**, where the agent does not have a model of the environment; rather, it learns by trial-and-error from the reward it gets after each action. The interaction cost for such models is high, but they are effective in environments that are dynamic.

  Some popular Model-Free RL algorithms are Q-Learning [9], Deep Q Networks (Q-Learning with deep learning) and SARSA (Q-Learning that factors in the next action's impact).

3. Methodology

To implement a RL-based solution to Connect-Four, we chose the **Q-Learning** Algorithm.

3.1 Q-Learning

The goal of Q-learning is to learn a policy π(s) that instructs the agent on what action to take in any given state. Q-learning finds an optimal policy that maximizes the expected value of the total reward over all successive steps starting from the current state. A typical implementation of π(s) could be the Epsilon-Greedy policy which, based on a user-specified probability of exploration, either returns the action which maximizes the potential reward (Q value), or returns a random action – thus supporting exploitation as well as exploration.

"Q" (or quality of an action in a given state), is an abstract function that returns the reward used as the reinforcement. Initially, Q is initialised to an arbitrary fixed value. Then, at each time t, the agent selects an action a, observes a reward r, enters a new state st+1, and Q is updated, using the weighted average of the old value and the new information based on rewards earned.

The core algorithm is as follows:

```
Q-Learning -
Input:
      A: Agent Action Set
      S: Agent State Set
      T: Maximum time-steps
      $\alpha$: Learning Rate
      $\gamma$: Discount Factor
      $\epsilon$: Exploration Probability
Output: Q
1. Initialize Q(s,a) = 0, for all s $\in$ S, a $\in$ A
2. Initialize $s_t$
3. Initialize t <- 0
4. while (t < T), do:
    i.   Choose at using Epsilon-Greedy Policy - $\pi(s_t, \epsilon, rand)$
   ii.   Take action $a_t$, observe reward $r_t$ and next state $s_{t+1}$
  iii.   Update Q($s_t$, $a_t$) using $\alpha$, $\gamma$, $r_t$ and $s_{t+1}$
   iv.   $s_t$ <- $s_t$+1
    v.   t  <- t+1
```

3.2 Implementation

The implementation focuses on the development of a Connect Four game, integrated with a Q-learning agent that learns to play the game optimally through reinforcement learning techniques. The project is structured into two main components: the game logic and the learning agent.

3.2.1 Game Logic

The connect Four game is represented by a class that manages the game state, player turns, and win conditions. The game board is modelled as a 6x7 grid, initialised to zeros, where each cell can hold a value representing either player 1, player 2 or an empty state. The core functionalities of the game include:

Game Initialization and Reset: The game can be initialised and reset to its starting state, allowing for multiple rounds of play without restarting the program.

Piece Dropping Mechanism: Players take turns dropping their pieces into one of the seven columns. The implementation ensures that pieces stack from the bottom of the column, adhering to the rules of the game.

Win Condition Checking: After each move, the game checks for a winning condition by examining all possible directions (horizontal, vertical and diagonal) for four consecutive pieces belonging to the current player. This is achieved through a systematic search algorithm that evaluates potential winning configurations.

3.2.2 Q-Learning Agent

The Q-learning agent is designed to learn optimal strategies for playing Connect Four through trial and error. It utilises a Q-table to store the expected utility of each action (dropping a piece in a column) for every possible game state (dropping a piece in a column) for every possible game state (board configuration). Key features of the agent include:

Exploration vs. Exploitation: The agent employs an epsilon-greedy strategy, where it explores random actions with a certain probability (epsilon) and exploits known information by selecting the action with the highest Q-value otherwise. This balance allows the agent to discover new strategies while refining its existing knowledge.

Q-value Updates: The agent updates its Q-values based on the rewards received from actions taken during gameplay. A higher reward is assigned for winning moves, while penalties are given for losing or drawing situations. The Q-learning update rule is applied to adjust the Q-values, incorporating both immediate rewards and the estimated future rewards.

Epsilon Decay: Over time, the exploration rate (epsilon) is gradually reduced, encouraging the agent to rely more on its learned strategies as it gains experience.

3.2.3 Training Process

The training process involves simulating multiple episodes of the game, during which the agent interacts with the environment, learns from its actions, and updates its Q-table accordingly. Each episode consists of the following steps:

- The game is reset, and the agent begins with an empty board.
- The agent selects actions based on its current policy, which balances exploration and exploitation.

- After each action, the game state is updated, and the agent receives feedback in the form of rewards.
- The training loop continues until a terminal state is reached (either a win or a draw), at which point the agent's Q-values are updated based on the outcome.
- Metrics such as total rewards, number of moves, and exploration counts are tracked to evaluate the agent's performance over time.

4. Experimental Setup

4.1 Game Environment

The environment used is a simplified Connect Four game played on a standard 6-row by 7-column board. The environment is implemented in Python using the ConnectFour class. The board starts empty, and two players take turns to drop their pieces into one of the seven columns.

4.2 Actions

The action space consists of 7 possible actions, representing the columns in which a player can drop their piece.

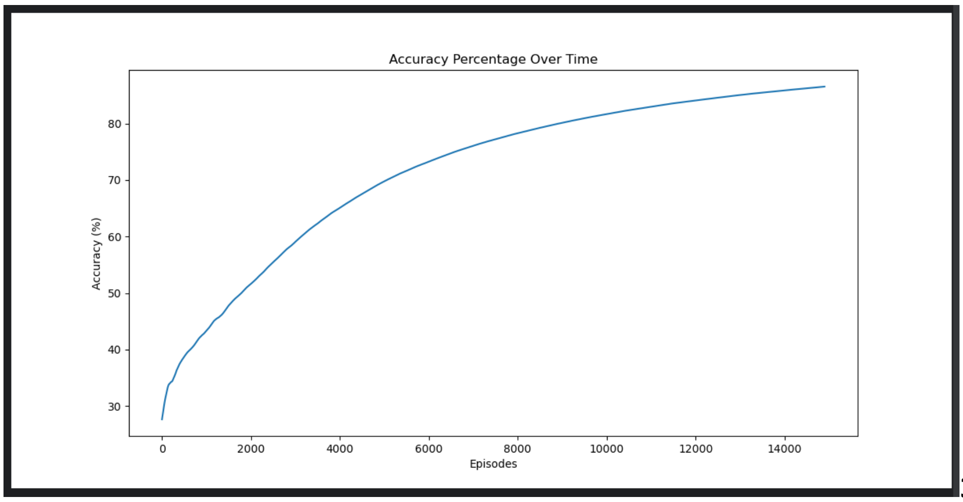
4.3 State Representation

The board state is represented as a 6x7 matrix where:

- 0 indicates an empty position.
- 1 and 2 represent the pieces of Player 1 and Player 2, respectively where Player 2 is making the random moves.

5. Results

The Q-learning agent's performance was assessed based on several metrics, collected over 15,000 training episodes. Following is an analysis of the key metrics that reflect the agent's learning progression and behavioural trends.



333333

Figure 5.1. Accuracy Percentage Over Time

Figure 5.1 represents the percentage of optimal actions taken by the agent during training. Accuracy is defined as the proportion of moves deemed optimal based on the current Q-values in the agent's Q-table. Initially, the accuracy is relatively low (around 30%), indicating a high number of suboptimal decisions. Over time, as the agent learned the state-action value function, the accuracy steadily increased, ultimately exceeding 80%. This increase demonstrates a clear trend of the agent progressively making better decisions, optimising its strategy by effectively utilising past experiences.

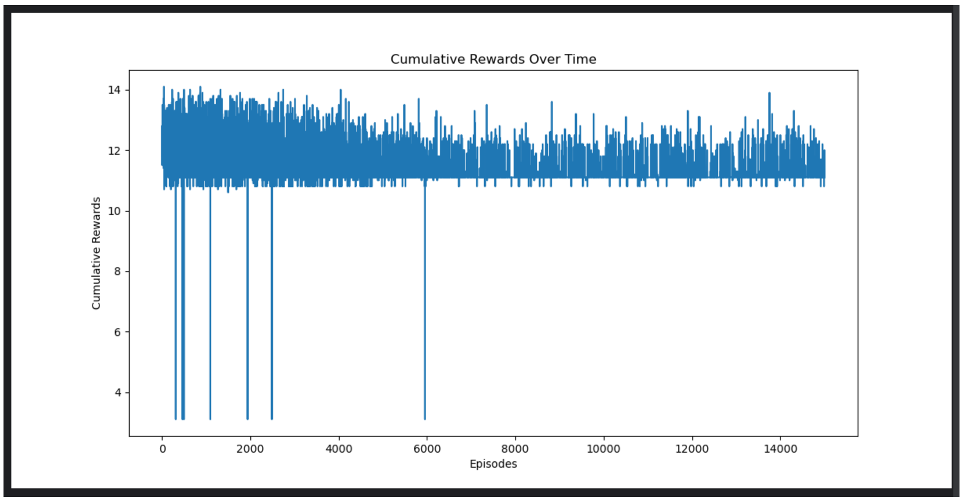


Figure 5.2. Cumulative Rewards Over Time

Figure 5.2 indicates the overall reward obtained by the agent at the end of each episode. Positive rewards are granted for wins, while penalties are applied for losses or draws. The cumulative rewards start with notable fluctuations, including negative values corresponding to losses and suboptimal play. As training progresses, the cumulative rewards stabilise at higher values, indicating that the agent consistently achieves favourable outcomes. This stabilisation highlights an effective learning curve and a marked improvement in overall game performance.

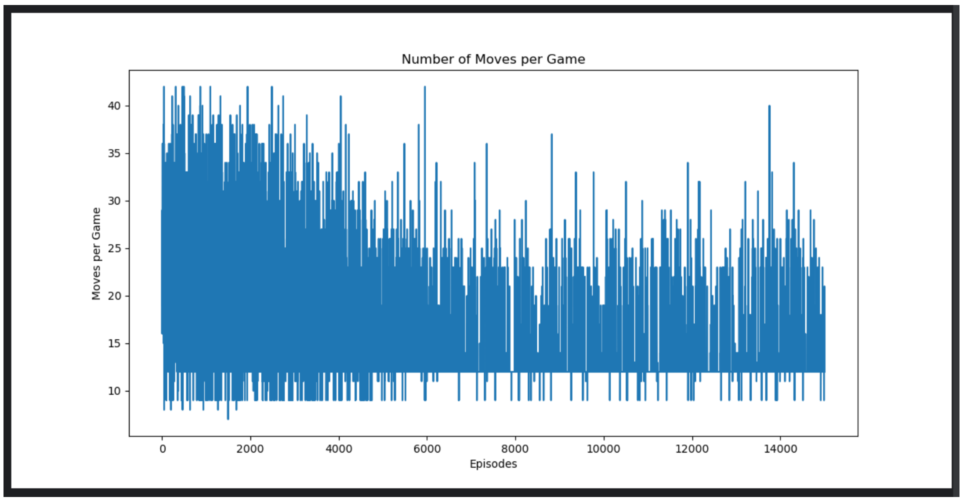


Figure 5.3. Number of Moves per Game

Figure 5.3 displays the number of moves per game as a metric to track the efficiency of the agent's strategy. Shorter games imply either quicker wins or early draws, indicative of effective strategic decisions. The number of moves per game was initially highly variable, with games often lasting longer as the agent explored the action space without clear knowledge of optimal strategies. Over time, the number of moves generally decreased, indicating improved decision-making and an increasingly effective understanding of how to reach terminal states promptly.

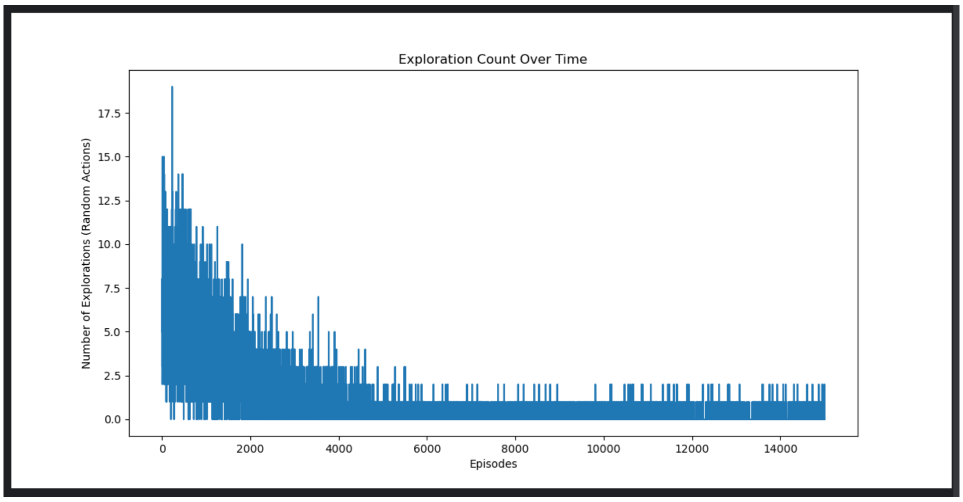


Figure 5.4. Exploration Count Over Time

Figure 5.4 displays the number of exploratory moves taken by the agent during each episode, which is driven by the ε-greedy policy used in Q-learning. The ε value starts high and decays over time, allowing the agent to gradually shift from exploration to exploitation. During the early training phase, exploration counts are high due to the elevated value of ε, which encourages random actions. As training progresses, the exploration count decreases significantly, indicating that the agent shifts its behaviour from exploration towards exploitation. This shift suggests that the agent increasingly relies on its learned policy, signifying confidence in its ability to select effective actions based on prior knowledge.

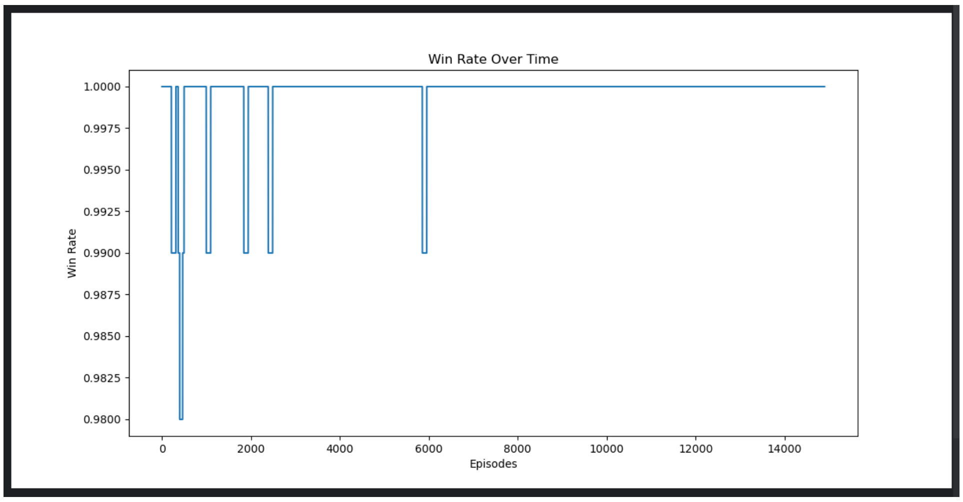


Figure 5.5. Win Rate Over Time

Figure 5.5 displays the win rate as a rolling average over 100 episodes, indicating the proportion of games won by the agent over time. This metric provides insight into the agent's competitive success. Initially, the win rate is below 100%, reflecting the agent's struggle to dominate its opponent consistently. However, as the agent's learning progresses, the win rate approaches and eventually stabilises near 100%, indicating that the agent has developed a sufficiently robust strategy that consistently leads to victories. This trend suggests that the agent has effectively mastered the game.

6. Conclusion

The implementation of a Q-learning agent for the game of Connect Four demonstrates the effectiveness of reinforcement learning for developing competitive game-playing strategies. Through the use of a tabular Q-learning approach, the agent was able to learn and progressively improve its gameplay over 15,000 training episodes.

The training process was guided by an ε-greedy exploration strategy, a reward structure emphasising winning moves, and Q-table updates that allowed the agent to develop a deeper understanding of optimal actions. Over time, the agent demonstrated consistent improvements in its performance.

We compared our implementation against open source kaggle project [11], and observed similar trends in terms of accuracy over time (**Figure 6.1**)

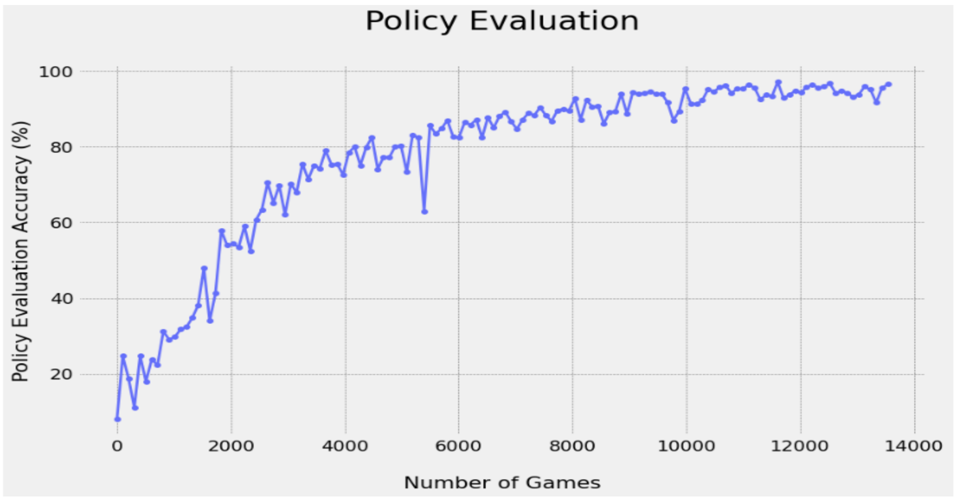


Figure 6.1 Win Rate Over Time

7. Future Work

Implement a **Multi-Agent Reinforcement Learning (MARL)** approach where both Player 1 and Player 2 are treated as independent, yet interacting agents. The focus can be on cooperative or competitive dynamics. Techniques such as Independent Q-learning (IQL) or Centralised Training with Decentralised Execution (CTDE) could be explored to improve convergence and address the non-stationarity introduced by multiple learning agents.

Additionally, explore a **Joint Reward System**, where the rewards are influenced by both players' outcomes to observe how cooperative or competitive dynamics evolve in their behaviour.

References

[1] Connect Four, Wikipedia

[2] Wang Qiang, Zhan Zhongli, et al, Reinforcement learning model, algorithms and its applications. 2011 International Conference on Mechatronic Science, Electric Engineering and Computer Science

[3] Richard S Sutton, D Barto, Reinforcement Learning, 2018, MIT Press

[4] https://towardsdatascience.com/reinforcement-learning-implement-tictactoe-189582bea542, 2019

[5]https://medium.com/@nancy.q.zhou/teaching-an-ai-to-play-the-snake-game-using-reinforcement-learning-6d2a6e8f3b1c, 2023

[6] John's Connect Four Playground". Homepages, 2010

[7] Gilad Wisney, Deep Reinforcement Learning and Monte Carlo Tree Search with Connect 4, Towards Data Science, 2019

[8] Markus Thill, Samineh Bagheri, Patrick Koch and Wolfgang Konen, Temporal Difference Learning with Eligibility Traces for the Game Connect-4, International Conference on Computational Intelligence in Games, 2014

[9] Alderton, Wopat, Koffman, Reinforcement Learning for Connect Four, Project Report, Stanford University, 2019

[10] Henry Taylor and Leonardo Stella, An Evolutionary Framework for Connect-4 as Test-Bed for Comparison of Advanced Minimax, Q-Learning and MCTS, arXiv, 2024

[11] https://www.kaggle.com/code/auxeno/alphazero-connect-4-rl