Solving a game of PopOut Connect 4 with Reinforcement Learning

Candidate Number(1): 46755

Candidate Number(2): 42193

Candidate Number(3): 41705

Candidate Number(4): 50175

Abstract

Training an AI model for complex board games such as Connect 4 is a challenging task that requires sophisticated algorithms and strategies. In this paper, we present a novel variant of the classic board game Connect 4 called "Pop-Out Connect 4," which allows players to "pop" checkers from the bottom. This introduces additional challenges for artificial intelligence (AI) agents. To address these challenges, the conceptual algorithms created by Dabas et al. (2022) for the original game serve as the foundation of our approach. Additionally, we deviate from the combination of deep reinforcement learning, Monte Carlo Tree Search, and alpha-beta pruning proposed in Pekkala (2014) to introduce reinforcement learning approaches in the discussion. We provide a detailed description of our algorithm and evaluate its performance on the new rule setting of Pop-Out Connect 4. Our results demonstrate the efficacy of our approach and highlight the potential of AI to solve complex board games. This work has implications for the design and development of AI-based decision-making systems in board games scenarios.

Keywords: *Connect 4, Reinforcement Learning, Q-Learning, Sarsa*

1. Introduction

1.1 Introduction to the game

Connect 4 is a two-player strategic board game that involves placing colored discs on a 7x6 grid. The objective of the game is to get four of one's own colored discs in a row, either horizontally, vertically, or diagonally. In Connect 4, there are 4.5 trillion potential board arrangements (Dabas et al. (2022)). Players take turns dropping their colored discs into the grid, attempting to create a "four-in-a-row" or "connecting four" configuration in order to win (as illustrated in Figure 1).

The Pop-Out Connect 4 variant is designed with an extra slide bar at the bottom, officially introduced in 2009 by Hasbro (Hasbro Games (2009)). Each turn, a player can choose either to drop a disc into the grid or pop one out of the bottom row. A player must be aware that they can only drop in or pop out counters of their own colour. Players can employ pop-out slots to obstruct their opponents' actions or create their own winning combinations, which adds an additional degree of strategy to the game.

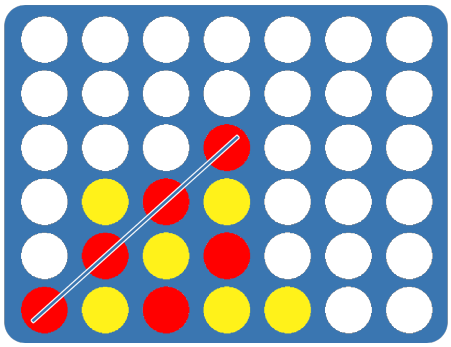

Figure 1: Winning state for player 1

1.2 Related work

The use of artificial intelligence (AI) to solve computer games has a long history and can provide valuable insights into real-world scenarios. In fact, techniques tested through games can often be transferable to more complex problems in actual situations, making this branch of study valuable (Schaeffer (2001)). The game of Connect 4 was first solved using artificial intelligence (AI) in 1988 (Allis (1988)). It was the second two-player zero-sum game to be solved using AI, preceded only by Qubic (Van den Herik et al. (2002)). In Pekkala (2014) solved the variation of the game called Pop Out, providing valuable insights into the addition of new rules to the existing literature on the subject.
In recent years, the advent of supervised learning and reinforcement learning has led to a number of studies on the implementation of this technique in the classic version of Connect 4 (Alderton et al. (2019); Wäldchen et al. (2022); Schneider and Garcia Rosa (2002); Thill et al. (2012)).
By using multilayer perceptron architecture on the traditional game and building an artificial neural network (ANN), Schneider and Garcia Rosa (2002) enhanced the understanding of the environment for applying supervised learning in this research area. The primary goal of the article was to make use of the game's vast data availability and assess the likelihoods of picking columns by using seven neurons, one for each column. The results demonstrated that artificial neural networks are significantly capable of learning Connect 4.
Moreover, the playground for implementing self-play (DDQN) in this branch of literature was finalized by Thill et al. (2012), which represented an important advancement in our knowledge of the classic game. The primary concept of the paper was to combine Temporal Difference Learning (TDL) and an n-tuple network together to expand the board's dimension and provide a mighty feature space. By using this method, the agent was able to create an ideal value function and learn by reducing the TD through self-play. Subsequently, Dabas et al. (2022) assessed a general preference for Montecarlo Tree Search in solving the traditional game, comparing the approach with DDQN and Alpha-Beta Prunning.
Our paper aims to address the gap in the literature by exploring the question of whether it is possible to solve Pop Out Connect 4 using reinforcement learning, and if so, what implications are presented by this technique.

2. Problem Formulation

2.1 Action Space

The action space is defined as the unique and limited set of moves that an agent can take in a specific environment. In the instance of Pop-Out Connect 4, each agent (player) has at most 14 possible moves they can consider at any given state. We can define an agent's action as dropping/popping a piece in/from one column. However, it must be taken into account that the amount of possible actions is dependent on the amount of columns that are full or empty. i.e. if it is the case that one column is empty, the agent is averted from popping a piece from that specific column and the same is applied if a column happens to be full.
In addition to this, each player has the possibility of popping a piece only if it it was previously placed by the same player.
It is possible to compute the amount of feasible actions in the following way:

$$Possible\,actions = 14 - F - E - O \tag{1}$$

Where F = amount of full columns, E = amount of columns that are empty and O = amount of opponent's pieces in the first row (from below).

2.2 State Space

Pop-Out Connect 4 has a well defined state at all times. A state is used to refer to the elements of the problem space which are required to describe the game. The state in Pop-Out Connect 4 includes the arrangement of the discs along with the information required to describe the state. The state space (usually denominated with 'S') is the set of all the states in which is possible for the agent to transition to. In Pop-Out Connect 4 the state space is fairly bigger than the action space mentioned above: we can define it as the board with all the pieces played up until that point that each player sees plus one state for each piece the player can pop. To be noted that the state space for each player differs: the board that the first player sees will always contain an even amount of disks whilst the second player will always encounter an odd number of pieces.

3. Proposed Solution

3.1 Alpha-Beta Pruning in Minimax

Alpha-beta pruning combined with Minimax is the general technique which has been used to solve the game of Connect 4 in the past. This technique has been implemented for Pop-Out Connect 4, the variant of Connect 4 under study. (Nasa et al. (2018)). This serves as a foundational starting point of our study.
The Minimax algorithm is a depth-first search algorithm used for determining the optimal move in a two-player, zero-sum game. It does this by recursively evaluating all possible moves and countermoves, assigning a value to each terminal node that indicates the outcome of the game at that point, and choosing the move that leads to the most favorable value for the player whose turn it is. The Minimax algorithm is typically implemented with two players, called Max and Min, with Max trying to maximize the value of the terminal nodes

and Min trying to minimize them.
The Alpha-beta algorithm is a variant of Minimax that reduces the number of nodes that need to be evaluated by cutting off entire subtrees that cannot affect the value of the root node. It does this by passing two values, alpha and beta, to each recursive function call, representing the scores that Max and Min are guaranteed to get, respectively. When alpha and beta become equal, a cutoff can be made because the line of play cannot be optimal. Alpha-beta is known for its ability to make deep cutoffs in addition to shallow cutoffs, and when it works optimally, it produces a minimal game tree.

3.2 Q-learning

One approach that we have implemented with the aim of solving the game is Q-learning: a strategy used in reinforcement learning that aims to find the best action to perform given the current state. The core principle behind Q-Learning is to generate a map of the full observation space and then record the agent's behaviors inside that table. As a result, each time the agent sees the same observation, it will progressively alter the prior action it made based on whether the action garnered a good or negative reward. The Q-table will be the space where the prior actions for each observation in the observation space are stored. The number of columns of the table is equal to the number of actions and the number of rows is equal to the number of states.
Every time an action is performed, the Q values is updated based on the following Bellman's equation:

$$\underbrace{New\,Q(s,a)}_{\substack{\text{new}\\ \text{Q-value}}} = \underbrace{Q(s,a)}_{\substack{\text{current}\\ \text{Q-value}}} + \underbrace{\alpha}_{\substack{\text{learning}\\ \text{rate}}} [\underbrace{R(s,a)}_{\text{reward}} + \underbrace{\gamma}_{\substack{\text{discount}\\ \text{rate}}} \underbrace{maxQ'(s',a')}_{\substack{\text{max expected}\\ \text{future reward}}} - \underbrace{Q(s,a)}_{\substack{\text{current}\\ \text{Q-value}}}] \tag{2}$$

The reward will be '1' for winning, '-1' for losing, '0.5' for a tie and '0' otherwise. The optimal state for an agent to be in will result in the highest optimal value.

3.3 Sarsa

The Sarsa strategy is a variation of the Q-Learning algorithm that we have tried to execute in our program. Similarly to Q-learning, Sarsa stores the value of each state-action pair in a Q-table. However, contrary to Q-learning, Sarsa does not maximises over all possible set of actions, instead it makes use of the current action taken to update the value of Q. Another difference that we encounter between Q-learning and Sarsa is that the former is an off-policy algorithm whilst the latter is an on-policy algorithm.
The values are updated with the following equation:

$$\underbrace{New\,Q(s,a)}_{\substack{\text{new}\\ \text{Q-value}}} = \underbrace{Q(s,a)}_{\substack{\text{current}\\ \text{Q-value}}} + \underbrace{\alpha}_{\substack{\text{learning}\\ \text{rate}}} [\underbrace{R(s,a)}_{\text{reward}} + \underbrace{\gamma}_{\substack{\text{discount}\\ \text{rate}}} \underbrace{Q'(s',a')}_{\substack{\text{value of next state}\\ \text{next action pair}}} - \underbrace{Q(s,a)}_{\substack{\text{current}\\ \text{Q-value}}}] \tag{3}$$

4. Numerical Experiments

The Pop-Out Connect 4 environment was established for the implementation of the Q-Learning and Sarsa algorithms using Python (SphericalSilver (2019)). This process involved

the creation of functions to initialize the game board, facilitate player switching, and define the winning state.
The implementation of the Q-learning and Sarsa algorithms involved the execution of functions that are similar in nature. Specifically, functions were created to establish the reward, calculate the Q-values for state-action pairs, and update the Q-values.
The sole distinction between the implementation of Q-learning and Sarsa lies in the update function, which follows either Equation 2 for Q-learning or Equation 3 for Sarsa.
Subsequently, to check the effectiveness of our code, we decided to run the game 5000 times for each strategy. Table 1 and Table 2 display the victory results for each player. Figure 2, on the other hand, displays a density plot of the number of moves per game (the blue line is Sarsa while the orange line is Q-learning).

	Victories	Defeats	Ties
Player 1	2671	2329	0
Player 2	2329	2671	0

Table 1: Q-Learning results for 5000 games.

	Victories	Defeats	Ties
Player 1	2800	2200	0
Player 2	2200	2800	0

Table 2: Sarsa results for 5000 games.

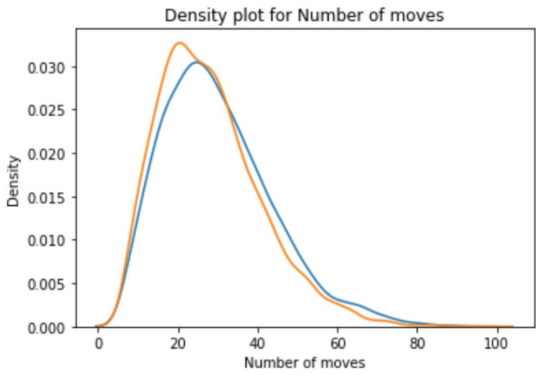

Figure 2: Number of moves per game

It is noteworthy that there are no ties in any of the trials. This is due to the fact that a tie happens in the eventuality of a full board with no winner. However, the Pop Out strategy prevents the board from becoming filled with discs as the player always has the possibility to pop. Therefore, there is potential for the game to continue indefinitely but as we observed, a player ultimately emerges as the victor in every game.

5. Results

5.1 Comparison between Q-learning and Sarsa in solving a game of PopOut Connect 4

In order to establish the difference between the two Reinforcement Learning approaches, we conducted 5000 games in which one player was adopting the Q-learning strategy and the opponent was adopting the Sarsa strategy.
Table 3 displays the victory results for the instance where: Player 1= Q-Learning and Player 2 = Sarsa.
Table 4 displays the victory results for the instance where: Player 1= Sarsa and Player 2 = Q-Learning.

	Victories	Defeats	Ties
Player 1	2870	2130	0
Player 2	2130	2870	0

Table 3: Q-Learning vs Sarsa results for 5000 games.

	Victories	Defeats	Ties
Player 1	2684	2316	0
Player 2	2316	2684	0

Table 4: Q-Learning vs Sarsa results for 5000 games.

6. Conclusions

In this study, two reinforcement learning techniques, Q-learning and Sarsa have been applied to solve the game of Pop-Out Connect 4. The Q-learning agent was able to achieve a win percentage of 57.40% when starting as player 1 against the Sarsa agent. The Sarsa agent was able to achieve a win percentage of 53.68% when starting as player 1 against the Q-learning agent. There has not been any significant difference observed between the winning percentages of the Q-learning agent and Sarsa. It has been shown by Allis (1988) that the first player can always win a game of Connect 4 with perfect play. However, this is not the case for Pop-Out Connect 4. The starting position does not affect the outcome of the game as much as it does for a normal game of Connect 4. The game may continue indefinitely until one player emerges as a winner.

7. Discussion

This is the first time reinforcement learning has been used to solve this variation of Connect 4. It would be interesting to see how other reinforcement learning techniques such as Deep Q Networks could be used to solve this problem. Possible extensions to this study include using reinforcement learning to solve other variations of Connect 4. A board having a relatively

small size has been utilised in our paper due to memory restrictions. Future research could focus on techniques to partition the state space in order to incorporate larger board sizes or higher dimensional boards. Further extensions of the study could improve the performance of the agents by varying the exploration rate or using eligibility traces.

References

E. Alderton, E. Wopat, and J. Koffman. Reinforcement learning for connect four - Stanford University. Technical report, 2019. (`https://web.stanford.edu/class/aa228/reports/2019/final106.pdf`).

V. Allis. A knowledge-based approach of connect-four. Technical report, 1988. (`https://www.informatik.uni-trier.de/~fernau/DSL0607/Masterthesis-Viergewinnt.pdf`).

M. Dabas, N. Dahiya, and P. Pushparaj. Solving connect 4 using artificial intelligence. *International Conference on Innovative Computing and Communications. Advances in Intelligent Systems and Computing*, 1387:727—735, 2022.

Hasbro Games. The original game of connect 4 - guide. Technical report, 2009. (`https://www.hasbro.com/common/documents/dad2614d1c4311ddbd0b0800200c9a66/1EF6874419B9F36910222EB9858E8CB8.pdf`).

R. Nasa, R. Didwania, S. Maji, and V. Kumar. Alpha-beta pruning in mini-max algorithm –an optimized approach for a connect-4 game. *International Research Journal of Engineering and Technology (IRJET)*, 5(4):1637–1641, 2018.

J. Pekkala. Computer analysis of connect-4 popout. Technical report, 2014. (`http://jultika.oulu.fi/files/nbnfioulu-201405281532.pdf`).

J. Schaeffer. A gamut of games. *AI Magazine*, 22(3):29, 2001.

M.O. Schneider and J.L. Garcia Rosa. Neural connect 4 - a connectionist approach to the game. *VII Brazilian Symposium on Neural Networks, 2002. SBRN 2002. Proceedings.*, pages 236–241, 2002.

SphericalSilver. Connect-4-board-game. Technical report, 2019. (`https://github.com/SphericalSilver/Connect-4-Board-Game`).

M. Thill, P. Koch, and W. Konen. Reinforcement learning with n-tuples on the game connect-4. *Parallel Problem Solving from Nature - PPSN XII*, 7491:184–194, 2012.

H. J. Van den Herik, J. W. H. M. Uiterwijk, and J. van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1-2):277—311, 2002.

S. Wäldchen, F. Huber, and S. Pokutta. Training characteristic functions with reinforcement learning: Xai-methods play connect four. Technical report, 2022. (`https://arxiv.org/pdf/2202.11797.pdf`).