Reinforcement Learning on Connect Four

Yi Zhe Zhao
Faculty of Computer Science
McGill University

Emmanuel Ma
Faculty of Computer Science
McGill University

Abstract

In recent years, Connect 4 has emerged as a popular benchmark for evaluating the performance of various AI algorithms. This paper investigates six different RL algorithms for playing Connect 4, namely Expected SARSA, Q-Learning, MiniMax, Monte Carlo Tree Search, Actor Critic, and Deep Q-Learning. Our experiments revealed that Expected SARSA and Q-Learning had limited performance due to their inability to handle the vast number of board states in a tabular and efficient manner. In contrast, MiniMax and Monte Carlo Tree Search were more adept at exploring different board states, albeit at the cost of higher computational overheads. The Actor Critic and Deep Q-Learning algorithms, on the other hand, provided a balanced approach by leveraging both value-based and policy-based methods. We evaluated the performance of these models in terms of their win rate and average game length against a random player and against each other. The results demonstrated that Deep Q-Learning performed extremely well with a win rate of over 90% against a random player, while MiniMax outperformed everyone else with best result. In summary, this study provides insights into the strengths and weaknesses of different AI models for playing Connect 4. Our research highlights the potential for further investigation in this domain, particularly in developing more efficient and effective AI models for playing board games.

1 Introduction

The game of connect 4 is a turned based 2 player game, played on a vertical game board with 6 rows and 7 columns. Each player takes turns dropping a game piece in a column, with the objective of constructing a formation of 4 pieces in a row in any direction. The game of connect 4 has been fully solved, having been proven that a player that plays first can always win within 41 moves if perfectly played. The intention of this paper however, is to compare and analyse different RL algorithms that could potentially solve the game.

1.1 Environment Set Up

We simulate the Connect Four environment using `https://github.com/Danielhp95/gym-connect4/blob/master/gym_connect4/envs/connect4_env.py` and we added some extra functions to facilitate agent-environment interaction. The 7 columns of the board form the action space, and the observation space consists of any combination of player 1 piece, player 2 piece, and empty cell on 42 positions of the board. Our reward system gives a +1 when the model wins, and a -1 when the model loses.

1.2 Contributions

Yi Zhe Zhao has contributed in building Q-Learning, MCTS, Actor-Critic models.
Emmanuel Ma has contributed in building Expected-SARSA, MiniMax, DQL models.

1.3 Background

The two tree searches are beyond course materials.

1.4 Methodology

Our agents first go through a training stage against a random action policy playing 1000 games. Then, we compared the two tree search algorithms since they do not require any training and we try to regulate the time taken per action so it is fair for both sides. The winner then plays against the best trained agent.

2 MiniMax and Monte Carlo Tree Search

2.1 MiniMax

Our first algorithm choice was minimax, a recursive algorithm that chooses an optimal move, assuming the opponent player is playing optimally. The algorithm consists of constructing a game state tree, which we set a maximum depth of 4. Then, for each layer of the tree, the algorithm alternates beteen playing as a maximizing agent i.e playing the move that reaches a game state of maximal value, and minimizing agent i.e playing the move that reaches a minimal value game state. The minimax algorithm performed very well, winning all games against the random player. However, with the number of game states to consider increasing exponentially with increasing tree depth, the time the algorithm takes to decide a move is exponential, thus while its performance is quite high, the running time can easily explode.

2.1.1 Monte Carlo Tree Search

Another tree search approach is Monte Carlo Tree Search (MCTS). We build a tree of possible game states starting from a given current game state and after series of update - each update consists of four phases : selection, expansion. roll-out, and back-propagation - we output the best immediate child node of the starting game state root node. During selection, we traverse down the tree by the UCB score of each node until we reach a leaf node. In expansion, we add children to this leaf node for each valid action from that state. Roll-out uses a random policy to play out a game simulation from a random child node of the expanded children. Finally, we back-propagate the number of visits and the reward to all nodes along the path up to the root node. The flexibility of MCTS comes from the fact that we can easily control the time spent in choosing an action by increasing or decreasing the number of updates ; more updates lead to better actions and vice-versa.

3 Expected SARSA and Q-Learning

We implemented Expected SARSA and tabular Q-Learning although the results were predictably disappointing. Due to the large number of different board states : 4 531 985 219 092[1], each Q value rarely gets updated and the models almost never learn.

3.0.1 Q-Learning

The idea was to set up a Q table for each state action pair and update the Q values according to the following rule:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma max_{a'} Q(s^{'}, a^{'}) - Q(s,a))$$

However, due to the vast number of possible state action pairs, we were not able to evaluate and update all the Q values in a way that facilitates the learning of our model. The consequence is that our model does not learn thus performs potentially worse than a random policy.

[1]Number of legal 7 X 6 Connect-Four positions : `https://oeis.org/A212693`

3.1 Expected SARSA

The expected SARSA learning algorithm is similar to Q-learning, except the update policy takes the expected value of the next state-action pair instead of the max value. Thus it uses the update rule:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1})|S_{t+1}] - Q(S, A)]$$

While it is computationally more expensive compared to Q-learning, it reduces variance that comes from random selection of the action at time t+1. On connect 4 however, the model was unable to converge as it shares the same weakness with Q learning of too large a state space.

4 Deep Q-Learning and Actor Critic

4.1 Deep Q-Learning

To overcome the weakness encountered with traditional Q learning, we chose to implement Deep Q learning. Instead of using a Q-table, we use a neural network to approximate the Q-values, and choose action based on decaying epsilon-greedy policy. Experience replay was implemented to save time by training in batches rather than after every step, and the Q-values are still updated using the same Bellman equation as in Q learning. The model performed quite well, achieving a winrate of 90% vs the random player after training on only 1000 games.

4.1.1 Actor Critic

Another way to fully take advantage of deep neural networks is implementing an Actor Critic model where we build an Actor network and a Critic Network. The Actor outputs the probability of each action for a state, while the critic outputs an estimated value for the state action that serves as feedback. We minimize the critic loss and the actor loss by the following formulas :

$$\delta_t = r + \gamma Q_\pi(s_{t+1}) - Q_\pi(s_t)$$

$$loss_{critic} = \delta_t^2$$

$$loss_{actor} = (-1) * log(\Pi(a_t, s_t)) * \delta_t$$

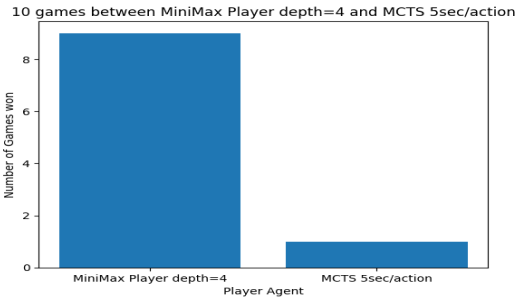
In order to account for illegal moves, when a column is full for instance, we give a reward of -10. The performance of the Actor Critic model stays questionable and further investigation can be conducted as it achieves a winrate of only 50% against a random policy.

5 Results

We observe that the two tree searches perform best, winning all games played against a DQN player. In fact, tree search methods seem to be performing better than deep learning algorithms in decision making problems.

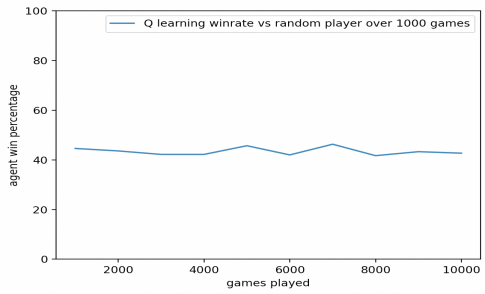
Agent	Average Game Length
MiniMax depth = 2	1.37754
MiniMax depth = 4	42.6822
MCTS 1sec per action	8.71567
MCTS 5sec per action	36.6694

(a) Runtime table for MCTS and MiniMax Agents in seconds.

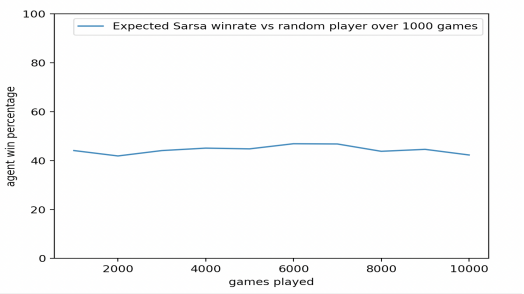


(b) 10 Games between MCTS and MiniMax.

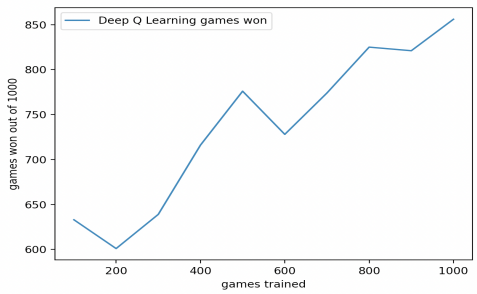
Figure 1: MCTS and MiniMax



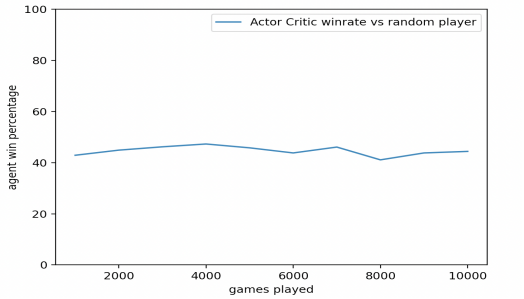
(a) Q-Learning Training Curve



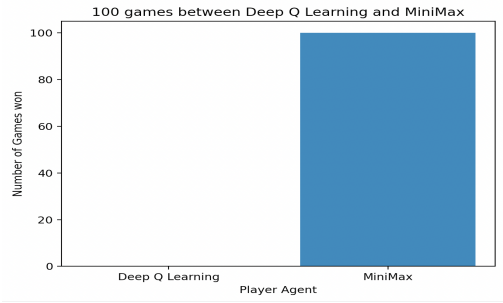
(b) Expected SARSA Training Curve.



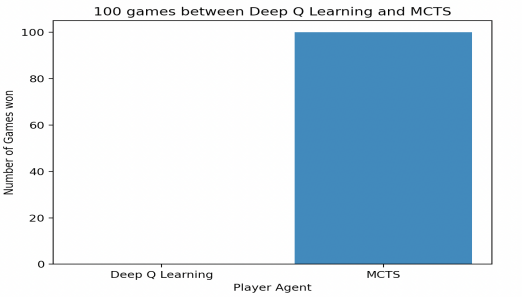
(c) DQN Training Curve.



(d) Actor Critic Training Curve.



(e) DQN vs MiniMax.



(f) DQN vs MCTS.

Figure 2: Results

6 Conclusion and future works

In conclusion, our study examined six different AI algorithms for playing Connect 4, including Expected SARSA, Q-Learning, MiniMax, Monte Carlo Tree Search, Actor Critic, and Deep Q-Learning. We evaluated the performance of these models in terms of win rate and average game length against a random player.

Our findings demonstrated that Deep Q-Learning has a great performance achieving a win rate of over 90%. However, MiniMax yields best results while MCTS has most flexible game length, making it a potentially suitable option for scenarios with time constraints. Expected SARSA and Q-Learning showed limited performance, and MiniMax and Monte Carlo Tree Search were more adept at exploring different board states, but at the cost of higher computational overheads. Actor Critic and Deep Q-Learning algorithms provided a more balanced approach by incorporating both value-based and policy-based methods.

It is worth noting our reward system is straightforward and more testing could be done by varying the reward distribution. Also, we can extend the use of a time limit as a stopping condition to MiniMax similar in what we did with MCTS. There are also papers that combine tree search in MCTS with deep neural network to form more complex models such as AlphaGo and is something worth exploring.

This study highlights the strengths and weaknesses of different AI models for playing Connect 4, providing insights for future researchers to develop more efficient and effective AI models for playing board games. Ultimately, these results can be extended to other domains with similar characteristics, such as other games or decision-making problems with complex state spaces.

7 References

[1] Pranav, Agarval (2020) *Game AI: Learning to play Connect 4 using Monte Carlo Tree Search*. Medium. https://pranav-agarwal-2109.medium.com/game-ai-learning-to-play-connect-4-using-monte-carlo-tree-search-f083d7da451e.

[2] Qi Wang (2022) *Connect 4 with Monte Carlo Tree Search*. Harry's Blog. https://www.harrycodes.com/blog/monte-carlo-tree-search.

[3] Richard, S.Sutton & Andrew, G.Barto (2018) Reinforcement Learning: An Introduction. Second Edition. Cambridge, MA: MIT Press.

[4] Daniel, Hernandez (2021) gym-connect4. Github. https://github.com/Danielhp95/gym-connect4/blob/master/gym_connect4/envs/connect4_env.py.

[5] Cheng, Xi Tsou (2021) *Actor-Critic: Implementing Actor-Critic Methods*. Medium. https://medium.com/geekculture/actor-critic-implementing-actor-critic-methods-82efb998c273.

[6] PyTorch Documentation. https://pytorch.org/docs/stable/index.html.

[7] Phil, Tabor. (2021) Reinforcement Learning Course: Intro to Advanced Actor Critic Methods. Youtube. https://www.youtube.com/watch?v=K2qjAixgLqk&t=13532s.

[8] Mnih et al. (2013) *Playing Atari with Deep Reinforcement Learning, Algorithm.* arXiv:1312.5602

[9] Chakravorty, S., Hyland, D. (2003). Minimax reinforcement learning. AIAA Guidance, Navigation, and Control Conference and Exhibit. https://doi.org/10.2514/6.2003-5718