Documentation of the project: Connect Four with Alpha Zero

University of Applied Science Düsseldorf
Professor: Dr. Dennis Müller
Module: Advances in AI (WiSe 2024 / 25)
Student: Lucas Müller
Date: 16.02.2025

Table of Contents

1. How Connect Four works

Connect Four is a strategic two-player game played on a vertical grid with 7 columns and 6 rows. Players take turns dropping one of their discs (red and yellow) into a column. The disc falls to the lowest available position in the chosen column. The goal of the game is to be the first to connect four of one's own pieces in a row—either horizontally, vertically, or diagonally. If the grid is completely filled without either player achieving four in a row, the game ends in a draw.

2. State of the Art

The state-of-the-art in AI agents for board games like chess and Connect Four primarily relies on three key techniques. First, deep neural networks, specifically convolutional residual networks, are used to evaluate game positions and suggest moves (Sandi). These networks are trained through self-play reinforcement learning, allowing the AI to improve its understanding of the game without human input. Second, *Monte Carlo Tree Search* (MCTS) is employed to explore the game tree efficiently, balancing between exploiting known good moves and exploring new possibilities. MCTS works in tandem with the neural network, using its evaluations to guide the search. Finally, these techniques are often combined with traditional alpha-beta pruning algorithms in hybrid systems, leveraging the strengths of both neural-network-based and classical approaches to create highly effective game-playing agents. These core techniques, when implemented with GPU acceleration and parallel processing, result in AI systems capable of superhuman performance in complex board games.

Because these residual networks and the MCTS are so strong, Alpha Zero is a great and modern technique for creating AI agents.

3. a. History of Alpha Zero

Alpha Zero, developed by DeepMind, represents a significant milestone in artificial intelligence for board games. In December 2017, DeepMind introduced Alpha Zero

in a preprint on arXiv (Silver et al., "A General Reinforcement Learning Algorithm"). This generalized version of the AlphaGo Zero algorithm achieved superhuman performance in chess, shogi, and Go within just 24 hours of training (Silver et al., "A General Reinforcement Learning Algorithm"). Starting from random play and given only the game rules, it improved itself through self-play reinforcement learning (Silver et al., "A General Reinforcement Learning Algorithm"). The algorithm combines sophisticated search techniques with a deep neural network, trained solely by self-play reinforcement learning. This approach allowed it to surpass the strength of previous versions like AlphaGo Lee in just three days, and exceed all previous versions in 40 days (Silver et al., "A General Reinforcement Learning Algorithm"). Alpha Zero's performance was particularly notable in chess. Within its first nine hours of existence, it played 44 million games of chess against itself. After just two hours, it surpassed human-level play, and after four hours, it was beating the best chess program in the world ("AlphaZero: The Story of the Self-Taught AI"). The style of play from the algorithm was described as more human-like and aggressive compared to traditional chess engines ("AlphaZero: The Story of the Self-Taught AI"). It prioritized piece activity over material advantage, often taking positions that appeared risky to human observers ("AlphaZero: The Story of the Self-Taught AI"). A very remarkable thing about the approach of Alpha Zero is that it does not rely on human knowledge. It makes the approach very useful for domains where export data might be unavailable.

2. b. How Alpha Zero works

AlphaZero utilizes a deep convolutional residual neural network that takes the game state as input and outputs two key components: a policy vector and a value estimate.

The policy vector $p(s)$ represents a probability distribution over all possible moves in a given state s. Each element corresponds to a legal move, and the values sum to 1, reflecting move probabilities. At the start of a Connect Four game, this vector contains seven elements, one for each column. The length of the vector is dynamically determined by the number of valid actions. The policy vector guides Monte Carlo Tree Search (MCTS) by prioritizing promising moves during the

selection phase (Silver et al., Mastering Chess and Shogi). Over time, it improves through self-play, capturing increasingly sophisticated strategic patterns.

To make optimal decisions, the AI balances exploration and exploitation. Exploration involves selecting less-frequented moves to gather more information, while exploitation favors moves that maximize the expected reward. In Connect Four, the agent receives a reward of +1 for a win, 0 for a draw, and -1 for a loss. If the agent relies solely on exploitation, it may become over-specialized, repeating the same strategies without adapting to different opponents. For example, if it always plays in the rightmost column and an opponent never blocks it, the agent may assume this move is always winning. However, if faced with a player who does block it, the agent may struggle because it has never encountered this scenario during training.

To mitigate this, a certain degree of exploration is necessary. By occasionally deviating from its most confident choices, the agent can discover alternative strategies that are more resilient against different playing styles. Striking the right balance ensures that the AI learns complex patterns while still prioritizing strong moves when appropriate. Excessive exploitation makes the agent predictable, while excessive exploration leads to erratic behavior, preventing it from developing a consistent winning strategy.

The scalar value function $v(s)$ estimates the expected outcome of the game from a given state s (Silver et al., Mastering Chess and Shogi). This value ranges from -1 to 1, where -1 represents a loss, +1 a certain win, and 0 a draw. The value function provides a quick assessment of the game position without requiring deep search. It also assists MCTS by pruning unpromising branches early, improving search efficiency.

Together, the policy and value functions form the foundation of the neural network's learning process. The policy provides action preferences, guiding move selection, while the value estimates the overall strength of a position. Both are used in the loss function to refine the network, improving both strategic understanding (policy) and positional judgment (value).

The neural network function can be represented as:

$$f_{\theta}(s) = (p(s), v(s))$$

where θ represents the network parameters.

The network is trained to minimize the following loss function:

- $l = (z - v)^2 - \pi^T logp + c\|\theta\|^2$

Each term in the loss function plays a specific role.

- Value Loss $(z - v)^2$

  This term measures the difference between the predicted value v and the actual game outcome z (Silver et al., "Mastering Chess and Shogi"). Minimizing this difference improves the network's ability to evaluate positions accurately.

- Policy Loss $-\pi^T logp$

  Here, π represents the improved policy from MCTS, while p is the policy by the neural network. This term uses cross-entropy loss to align the network's predictions with the MCTS-improved policy, improving move selection (Silver et al., "Mastering Chess and Shogi").

- L2 regularization $c\,||\theta||^2$

  This term helps prevent overfitting by adding a penalty proportional to the square of the network's weights θ. Keeping the weights small ensures the model generalizes better on unseen data (Silver et al., "Mastering Chess and Shogi")

Overfitting is taking place, "*if the model fails to generalize on the newer examples as the model is generalized while training*" (Indura). The L2 regularization term also helps generalization (Indura). By keeping weights small, it helps the model perform better on unseen data.

The self-play process is a key innovation behind Alpha Zero's performance. The neural network starts with random weights, and AlphaZero plays games against itself using MCTS guided by the current network (Silver et al., "A General Reinforcement Learning Algorithm"). During each game, Alpha Zero selects moves by running multiple MCTS simulations.

The number of simulations per move is a key parameter chosen by the user. More simulations lead to stronger gameplay but also increase training time. In my training loop, the number of simulations is 50.

During training, move selection is not always based on the best move but is weighted according to the policy vector. This ensures continued exploration and prevents the agent from prematurely converging on suboptimal strategies (Cineide).

Each game generates training data, consisting of board positions, move probabilities, and game outcomes. AlphaZero typically plays thousands of self-play games to generate sufficient data. In my case, I used around 5,000 games for Connect Four.

Training and self-play run in parallel for efficiency. Once training is complete, the new neural network is tested against the previous version by playing a series of games (Kovansky). If the new network wins more than a set threshold (e.g., 55% of games), it replaces the old network. This iterative process allows AlphaZero to continuously refine its strategies, discovering novel moves and improving without human input beyond the game rules.

4. How the Monte Carlo Tree Search works

The Monte Carlo Tree Search (MCTS) is a powerful algorithm that works in synergy with the neural network in AlphaZero. It efficiently balances exploration (searching for new strategies) and exploitation (favoring known good moves), allowing the AI to make strong decisions in complex game environments.

At the core of MCTS is the Upper Confidence Bound for Trees (UCT) formula, which is used during the selection phase:

$$UCT(s,a) = Q(s,a) + c_{puct} \cdot P(s,a) + b\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

where:

- $Q(s,a)$ is the expected value of action a in state s
- $P(s,a)$ is the prior probability of selection action a in state s and
- c_{puct} is the exploration constant that controls the balance between exploration and exploitation
- $b\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$ is the normalized visit count adjustment. It is an additional exploration term, which adjusts for visit counts across different actions. $N(s, b)$ is the total visit count of all sibling nodes. $N(s, a)$ is the number of times action a has been selected

This formula ensures that actions with a high expected value (exploitation) and low visit count (exploration) are favored, preventing the search from getting stuck in suboptimal strategies.

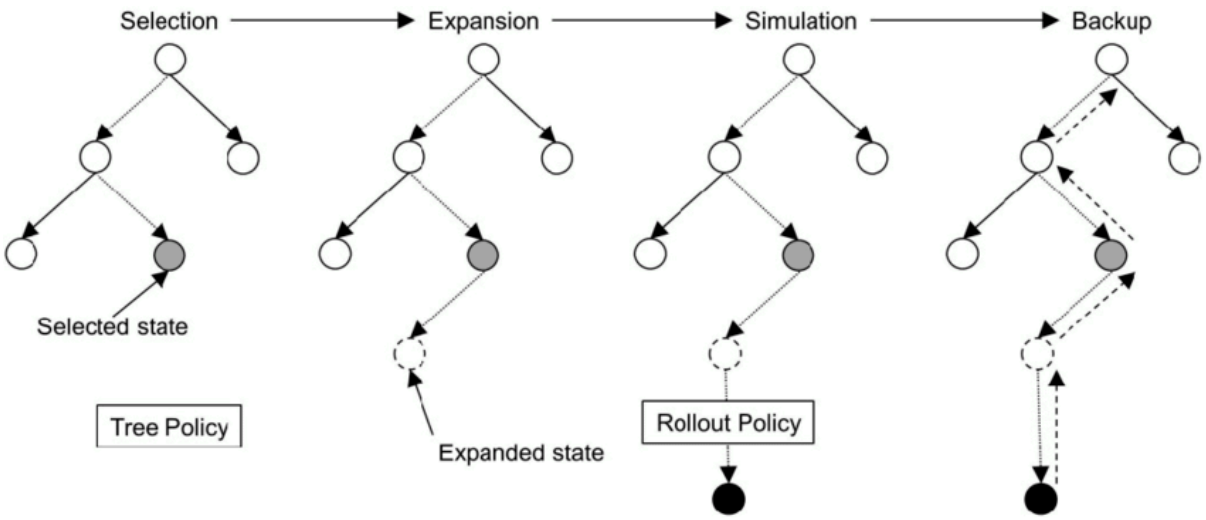

Figure. 1 Steps of the Monte Carlo Tree Search progress (Source: Duarte et al.)

MCTS operates in four key steps:

1. Selection

- The algorithm starts at the root node, which represents the current game state.
- It traverses the tree using the UCT formula, selecting the most promising child node based on both its prior probability and its expected value.

2. Expansion

- When a leaf node (a node with no children) is reached, it is expanded by adding one or more child nodes representing possible future states.

3. Simulation (Playout)

- From the newly expanded node, a random playout is performed until a terminal state (win, loss, or draw) is reached.
- The result of the game is recorded as r = 1 for a win, r = -1 for a loss, and r = 0 for a draw.

4. Backpropagation

- The results from the simulation are propagated back up the tree to update the statistics of all visited nodes.
- The value is updated using:

$$Q\ (s,\ a)\ =\ \frac{Q(s,a)\ *\ N\ (s,a)\ +\ r}{N\ (s,a)\ +1}$$

where:

- r is the simulation result (Jamieson).
- $N\ (s,\ a)\ +\ 1$ is each visited state-action pair along the trajectory
- $Q\ (s, a)$ is updated using an incremental mean

After all simulations are completed, the final move selection is typically based on the action with the highest visit count rather than the highest Q-value. This helps ensure stability in decision-making (Jamieson).

One of the key advantages of MCTS is that it functions effectively with minimal domain knowledge—it only requires knowledge of the game rules and end conditions (Radke). Additionally, MCTS can be halted at any time to return its best current estimate, making it well-suited for time-constrained scenarios.

In my implementation, I used a fixed number of simulations per move, but an alternative approach is to set a time limit instead. In this case, MCTS would run as many simulations as possible within the allowed time.

MCTS also naturally adapts to the search space, focusing on more promising areas while efficiently handling large branching factors (Radke). For example, in Connect Four, the AI often favors placing pieces in the middle columns over the edges, as these positions provide greater flexibility for creating winning sequences.

5. Results

I played multiple games against my agent, and it has reached a solid playing level. Through training, it has learned to identify and execute the fundamental "connect four" strategy, efficiently capitalizing on any obvious mistakes made by the human player. The agent’s performance is directly influenced by the training duration—the longer it trains, the more sophisticated its strategies become.

I played 20 matches against the agent, with the following results:

Player	Wins	Draws	Losses	Win Rate (%)
Human	9	0	11	45
Alpha Zero	11	0	9	55

As the results show, the agent played at a decent level. In order to further improve the model, one could either train the model for a longer period of time or also implement a stronger opponent that plays against the model.

When playing against the agent I also recognized the problems that can come up using this approach. One of them is its training duration. It is hard to know the level of the agent during training. A couple of games against a random opponent or a heuristic approach while training in order to evaluate the level of the agent is highly recommended.
Another problem is biased exploration. The UCT formula might be favoring other moves due to an imbalance in the exploration-exploitation trade-off (Radke).

In summary, Alpha Zero is a solid choice for creating board games agents. It does not require the human to generate test data itself and the combination of MCTS with the neural network achieves good performance. In order to bring it to the next level, a longer training duration is needed. Especially the number of MCTS simulations could be increased for a better move choice.

6. Sources:

"AlphaZero: The Story of the Self-Taught AI That Changed the Game." History of Data Science, www.historyofdatascience.com/alphazero/. Accessed 15 Feb. 2025.

Cinneide, Mel. "What's Inside AlphaZero's Brain?" Chess.com, 18 June 2018, www.chess.com/article/view/whats-inside-alphazeros-brain. Accessed 15 Feb. 2025.
Indura. "How Does L1 and L2 Regularization Prevent Overfitting?" Medium, 22 Feb. 2023, induraj2020.medium.com/how-does-l1-and-l2-regularization-prevent-overfitting-223ef7001042. Accessed 15 Feb. 2025.

Duarte, Fernando & Lau, Nuno & Pereira, Artur & Reis, Luís. (2020). A Survey of Planning and Learning in Games. Applied Sciences. 10. 4529. 10.3390/app10134529

Jamieson, Kevin. "Lecture 19: Monte Carlo Tree Search." CSE599i: Online and Adaptive Machine Learning Winter 2018, scribes: Christopher Mackie, Hunter Schafer, and Nathaniel Yazdani, University of Washington, 2018, courses.cs.washington.edu/courses/cse599i/18wi/resources/lecture19/lecture19.pdf. Accessed 15 Feb. 2025.

Khovanskiy, Maxim. "AlphaZero Chess: How It Works, What Sets It Apart, and What It Can Tell Us." Towards Data Science, 5 May 2022, towardsdatascience.com/alphazero-chess-how-it-works-what-sets-it-apart-and-what-it-can-tell-us-4ab3d2d08867/. Accessed 15 Feb. 2025.

Nair, Surag. "AlphaZero." Surag Nair's Blog, 29 Dec. 2017, suragnair.github.io/posts/alphazero.html. Accessed 15 Feb. 2025.

Radke, Parag. "Monte Carlo Tree Search: The AI Algorithm Powering Your Favorite Games." Built In, 20 July 2023, builtin.com/machine-learning/monte-carlo-tree-search. Accessed 15 Feb. 2025.

Sandi, Francisco. "AI Agents: The End of the Track." Fran Sandi, 13 Jan. 2025, www.fransandi.com/blog/ai-agents-the-end-of-the-track. Accessed 16 Feb. 2025

Silver, David, et al. "A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go Through Self-Play." Science, vol. 362, no. 6419, 2018, pp. 1140-1144, www.davidsilver.uk/wp-content/uploads/2020/03/alphazero.pdf. Accessed 15 Feb. 2025.

Silver, David, et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm." arXiv preprint, arXiv:1712.01815, 2017.

Yang, Angelina. "What Is Exploration vs. Exploitation in Reinforcement Learning?" Medium, 25 July 2022, angelina-yang.medium.com/what-is-exploration-vs-exploitation-in-reinforcement-learning-a3b96dcc9503. Accessed 15 Feb. 2025.