Gomoku with AlphaZero

E6885.2024Fall.Final Project.report.zl3477.jm5876

Authors: ZIYANG LIN, JIAWEI MENG
Columbia University

Abstract

Here we have an abbreviated implementation of the AlphaZero algorithm for a 6x6 Gomoku game. AlphaZero utilizes MCTS along with a Policy-Value Neural Network to train the best strategies by itself without deploying human experts' data. The Policy-Value Network computes move probability (policy) and game value (value), which is used to decide upon MCTS search. The experimental outcomes show that the AlphaZero agent performs significantly better than the Pure MCTS baseline, winning 49 of 50 games. Training loss convergences quickly, and policy entropy estimation stays consistent over self-play cycles. This project demonstrates that AlphaZero can successfully crack strategic games such as Gomoku, even in low-resolution environments, and sets the stage for further studies and optimizations.

1. Introduction

In this project we use a reduced version of the AlphaZero algorithm for the game of Gomoku on a 6x6 board. AlphaZero —is a reinforcement learning algorithm that uses Monte Carlo Tree Search (MCTS) in combination with a deep Policy-Value Network to get superhuman results in a variety of games. Our goal is to develop an optimal implementation for Gomoku based on its success in Go, Chess and Shogi. Gomoku, or Five in a Row, is a two-player grid strategy game in which two people stand on a line, alternating black and white pieces. The game's aim is to get five consecutive stones on the same line (vertically, horizontally or diagonally). The game is easy to play, but it is an especially difficult game for the AI due to the huge search space and optimizations needed to explore and exploit it. This simplified version of AlphaZero incorporates the following components:

- Policy-Value Network: A TensorFlow trained neural network that determines both the probability distribution for legal moves (policy) and the expected result of the game (value) for a given board state.
- Monte Carlo Tree Search (MCTS): A better tree search algorithm that employs network predictions to optimize the search, weighing exploration versus exploitation.
- Self-Play Mechanism: The AI improves by playing against itself, generating high-quality training data for the neural network.
- Pure MCTS for Baseline Comparison: To evaluate the effectiveness of AlphaZero, we compare its performance against a pure MCTS implementation that does not rely on neural network predictions.

Self-play and practice with repetition ensure that the model gradually learns to play Gomoku competitively. We experiment with playing multiple games using AlphaZero and pure MCTS to analyze performance, win rates, and improvement over training. The main goal of this project is to decipher and use the AlphaZero pipeline for a simple board game, learn new distribution algorithms (Dirichlet noise) to optimize MCTS exploration and explore how AlphaZero-Ai works better than traditional search methods. This paper explains the theoretical background, methodology, experiments and findings of the project and how AlphaZero can be used to solve games such as Gomoku.

2. Background and Novel Contribution

2.1 Background and Related Work

Reinforcement Learning has made many significant advances in solving large-scale decision problems — notably in the board games field. One of the most important RL experiments was DeepMind's AlphaGo, which employed MCTS and deep neural networks to perform superhuman Go play [1]. It has since been expanded into AlphaZero, a computer that learns to play board games such as Chess, Shogi and Go autonomously, without needing any human instruction.

AlphaZero was adapted in many ways to other games, including Gomoku, a grid-based strategy game. AlphaZero is extremely fast, but existing implementations are usually highly computationally and resource-consuming. For example, a previous Gomoku AlphaZero implementation includes detailed parallelized MCTS, extensive replay buffers, and large-scale self-play for improved generalization [2]. And these projects typically feature large codebases, complex optimization pipelines, and extensive experimentations to fine-tune performance, making them less accessible for quick prototyping [3].

2.2 Novel Contribution

Our project simplifies and implements the AlphaZero algorithm for Gomoku on a smaller 6×6 grid, with the following key contributions:

1. We're working on a smaller, modular AlphaZero implementation in contrast to other massive implementations. Its codebase keeps the core modules (MCTS, Policy-Value Network, and

Self-Play Training) intact and removes unnecessary complexity like parallelization or replay buffers.

2. We incorporate Dirichlet noise at the root node of MCTS during self-play, enhancing exploration and ensuring the algorithm does not overly exploit early, suboptimal strategies.
3. We implemented a Pure MCTS baseline for performance evaluation. This allows us to compare AlphaZero's learned policy with a traditional search-based approach and measure its effectiveness in decision-making.

As opposed to the massive AlphaZero-based Gomoku projects, our implementation is lightweight and easy to use for prototyping and learning. Moreover, we focus on the main algorithm while achieving high performance using small training rounds.

3. Approach

Gomoku is a two player turn-based game, played over a grid. It aims to place five identical stones vertically, horizontally, or diagonally. The state space expands exponentially with board size, so finding all possible moves by brute-force search is computationally impractical. This drives the adoption of AlphaZero, a machine learning-driven deep learning platform whose MCTS algorithm leverages the efficient use of self-play to train the best strategies.

3.1. Algorithm Development

The neural network predicts two outputs given a board state. First one is policy, a probability distribution over all possible legal moves. Second one is value, a scalar estimation of the probability of the current player winning the game. We designed a convolutional neural network using TensorFlow, structured as input, hidden layers and output layers. Input is a 6×6×2 tensor representing the board state (one channel for each player's stones). Hidden layers are two convolutional layers with ReLU activation and a flattening layer. For the output layers, first one is about a dense layer with a SoftMax activation producing move probabilities. And then A dense layer with a Tanh activation outputting the win probability. The network is optimized using a combined loss function called network loss function shown below [4].

$$L = Lpolicy + Lvalue \quad (1)$$

MCTS serves as the decision-making module, leveraging the neural network predictions to guide the search. At each decision point, the algorithm balances exploration and exploitation. For selection part, traverse the tree using the Upper Confidence Bound for Trees formula is [5]:

$$U(s,a) = Q(s,a) + Cpuct * P(s,a) * \frac{\sqrt{\sum N(s,b)}}{1+N(s,a)} \quad (2)$$

We also use Dirichlet Noise to improve exploration during self-play. This ensures the algorithm explores a broader range of moves early in training. Moreover, the model is trained using self-play to generate game trajectories, which means the agent plays against itself, using MCTS guided by the current policy network and each move generates a training sample. After this, the network is updated by minimizing the combined loss using collected samples and updated network is used in subsequent self-play games.

3.2. Theoretic Results

The AlphaZero algorithm leverages the theoretical strengths of MCTS and deep learning:

1. MCTS Convergence: MCTS, in combination with reliable value estimation, approximates the game's minimax value.
2. Policing Enhancement: By combining the network's predictions with MCTS-enabled policies, the algorithm optimizes the policy over time towards optimal play.
3. Self-Play Convergence: Self-play in MCTS ensures that the agent can learn from stronger opponents (themselves) in order to obtain a stable policy.

3.3. Implementation Simplifications

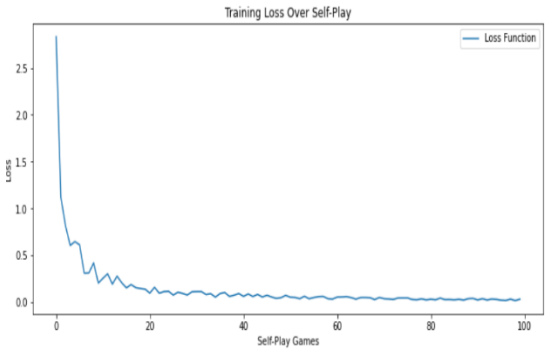
To make the project lightweight and efficient:

- We use a smaller 6×6 board to reduce computational complexity.
- The MCTS implementation is single-threaded, focusing on simplicity.
- Training is limited to 100 iterations for quick prototyping.
- Evaluation is performed against a Pure MCTS baseline, which does not require neural network predictions.

4. Experiment results

4.1 Training Loss Over Self-Play

The below figure [6] shows the training loss over 100 self-play games, we can tell from the figure that the loss

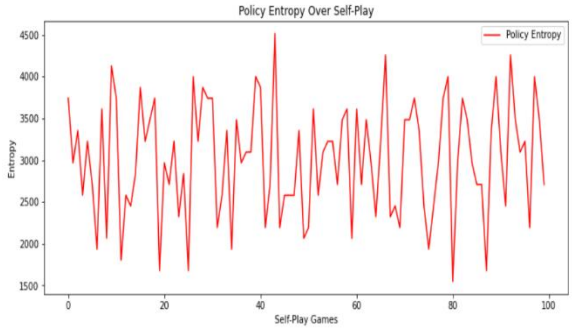


function is the sum of the categorical cross-entropy loss for policy prediction and the MSE loss for value prediction. In the early training, loss starts at a very high level

(approximately 3.0), which means network predictions are initially undetermined. In practice, the loss declines quickly and eventually reaches near to 0.1 after around 40 games. Such behavior indicates that the network is effectively training on estimating both move probabilities (policy) and value estimates with increasing accuracy. This dramatic drop in loss shows that the AlphaZero agent is learning a lot from itself.

4.2 Policy Entropy Over Self-Play

The second graph [6] shows the policy entropy in the self-play games. Policy entropy measures how random the selection of moves is. Higher entropy means an exploratory policy, and lower entropy means a deterministic policy.



It shows that when the policy entropy starts out low, the agent explores more randomly, and so does its search efficiency. As training continues, entropy decreases at some stages, which suggests that the network is finally starting to settle down and making more confident decisions. But later levels remain slightly unstable, which we might chalk up to the exploration of Dirichlet noise. Policy entropy fluctuations are related to the ratio between exploration and exploitation. The resulting downward trend in entropy indicates policy stability over time.

4.3 AlphaZero vs. Pure MCTS

We tested the performance of the trained AlphaZero agent by playing 50 games against a Pure MCTS background. The Pure MCTS picks moves based on random simulations without neural network instructions. The results are summarized below [6]:

```
Game 39/50: Winner - AlphaZero
Game 40/50: Winner - AlphaZero
Game 41/50: Winner - AlphaZero
Game 42/50: Winner - AlphaZero
Game 43/50: Winner - AlphaZero
Game 44/50: Winner - AlphaZero
Game 45/50: Winner - AlphaZero
Game 46/50: Winner - AlphaZero
Game 47/50: Winner - AlphaZero
Game 48/50: Winner - AlphaZero
Game 49/50: Winner - AlphaZero
Game 50/50: Winner - AlphaZero

Evaluation Results:
AlphaZero Wins: 49
Pure MCTS Wins: 1
Draws: 0
```

These are three key observations. First is about the AlphaZero dominance, it shows that AlphaZero achieved a win rate of 98% (49/50 games), demonstrating its superiority over Pure MCTS. And for the pure MCTS performance, it only managed to win a single game, which highlights its limited effectiveness without neural network guidance. Finally, no games ended in a draw, which indicates the decisive nature of the AlphaZero agent's learned strategy. In summary, these results validate the effectiveness of combining Monte Carlo Tree Search with a Policy-Value Network for learning optimal strategies through self-play.

5. Conclusion

Ultimately, this project proved that the AlphaZero algorithm is effective when it comes to solving an aggressive game such as Gomoku. MCTS, deep learning, and self-play enabled the agent to play competitively, outperforming searching techniques. It is a basis for future research into reinforcement learning and its use in board games and beyond.

5.1 Key Findings

The training error fell fast and came close to 0.1 which meant the network had learnt to estimate both policy and value outputs. Policy entropy fluctuated initially, but steadily stabilized as it corresponded to exploration and exploitation. The AlphaZero agent outperformed the Pure MCTS baseline with a 98% win rate (49/50) with no losses. Even on a small 66 board and with 100 games of training, the agent was very competitive, supporting AlphaZero's self-play learning model.

5.2 Contribution

The project achieved the following contributions:

- A lightweight and modular implementation of AlphaZero, making it easier to understand and extend.
- Demonstrated the role of Dirichlet noise in improving MCTS exploration during training.
- Compared AlphaZero's performance with a Pure MCTS baseline, showcasing the advantage of integrating neural network guidance into search algorithms.

5.3 Future Work

While the simplified AlphaZero implementation for the 6x6 Gomoku board demonstrates promising results, there are several directions to extend and improve the current work. These future enhancements can help increase the scalability, efficiency, and robustness of the model, enabling it to tackle more complex scenarios. There are some aspects that can be proposed for future work. Firstly, scaling to larger board sizes which means applying the policy to more large-scale boards (like 1515) would allow the agent to become more competitive and more akin to conventional Gomoku regulations. However, bigger boards bring challenges in terms of search space and computational cost. Optimizations such as parallelized MCTS or experience replay could alleviate these problems. Secondly, in order to test how powerful the trained agent is, subsequent experiments might include competition with

humans or with more sophisticated algorithms playing Gomoku. This would give a more accurate understanding of how the agent performs in the real world by benchmarking against stronger opponents.

6. References

[1] "SmythOS - Reinforcement Learning and AlphaGo: A New Era in AI." Accessed: Dec. 14, 2024. [Online]. Available: https://smythos.com/machine-learning/reinforcement-learning/reinforcement-learning-and-alphago/

[2] "junxiaosong/AlphaZero_Gomoku: An implementation of the AlphaZero algorithm for Gomoku (also called Gobang or Five in a Row)." Accessed: Dec. 14, 2024. [Online]. Available: https://github.com/junxiaosong/AlphaZero_Gomoku

[3] "hijkzzz/alpha-zero-gomoku: A Multi-threaded Implementation of AlphaZero (C++)." Accessed: Dec. 14, 2024. [Online]. Available: https://github.com/hijkzzz/alpha-zero-gomoku

[4] "AlphaZero" Accessed: Dec. 14, 2024. [Online]. Available: https://zhuanlan.zhihu.com/p/32089487

[5] "Monte Carlo Tree Search — how it's done!! | by Karthik Josyula | Medium." Accessed: Dec. 14, 2024. [Online]. Available: https://medium.com/@karthikjsr619/monte-carlo-tree-search-how-its-done-32aa769ed0b1

[6] "Leonard027/E6885_Final_Project." Accessed: Dec. 14, 2024. [Online]. Available: https://github.com/Leonard027/E6885_Final_Project

7. Contributions of each member of the team

	zl3477	jm6876
Name	ZIYANG LIN	JIAWEI MENG
Fraction of (useful) total contribution	1/2	1/2
Coding part	50%	50%
Report part	60%	40%