AlphaGomoku: Applying AlphaZero to the Game of Gomoku

1st Matthew Shumway
CS 401R - Reinforcement Learning
Brigham Young University
Provo, UT, USA
mwshumway7@gmail.com

***Abstract*—This project explores the application of the AlphaZero algorithm to the game of Gomoku, a two-player strategy game played on a 15x15 board. AlphaZero combines deep reinforcement learning with Monte Carlo Tree Search (MCTS) to train agents through self-play, achieving superhuman performance in games like Go, Chess, and Shogi. We adapt AlphaZero to Gomoku by designing a policy-value neural network that incorporates convolutional layers to capture the spatial structure of the board and employing MCTS for intelligent action selection. The training process involves iterative self-play, where the agent refines its strategy by balancing exploration and exploitation. Despite the computational limitations compared to other implementations, the results demonstrate the effectiveness of AlphaZero in solving nontrivial games like Gomoku and provide insights into the algorithm's adaptability to new domains.**

***Index Terms*—Reinforcement Learning, Alpha Zero, Monte Carlo Tree Search, Policy Value Network**

I. Introduction

Reinforcement learning (RL) is a powerful paradigm for training agents to interact with environments in ways that maximize performance based on pre-defined reward signals or costs. Over the past decade, advancements in innovative algorithms and computational capabilities have revolutionized RL, enabling efficient exploration of vast state and action spaces and the accurate approximation of value functions and policies. One notable breakthrough was the development of an algorithm designed for the ancient Chinese game of Go. This algorithm, known as AlphaGo Zero, achieved superhuman performance without relying on human knowledge [1]. The methodology was later generalized into the AlphaZero algorithm, capable of mastering a variety of environments. AlphaZero has excelled in turn-based, two-player games with clear rules and outcomes, while also demonstrating its utility in diverse applications, such as discovering efficient matrix multiplication algorithms [2].

This paper provides an in-depth description of the AlphaZero algorithm and adapts it to the game of Gomoku, a two-player strategy game typically played on a 15x15 board. The objective of Gomoku is deceptively simple: align five stones of the same color in a row—horizontally, vertically, or diagonally. However, the game's strategic depth arises from the complexity of its state space and the nuanced decision-making required. Compared to games like Go or Chess, Gomoku's strategies are more approachable, making it an ideal candidate for exploring novel RL approaches while remaining nontrivial. Notably, Gomoku is a theoretically solved game, with the starting player guaranteed to win under perfect play [3], further underscoring its value as a benchmark for RL algorithms.

II. Related Work

After choosing this topic for my project, I quickly realized that many others have pursued similar ideas [4]–[6]. These projects often benefit from access to far greater computational resources, such as larger clusters of GPUs or CPUs, enabling them to develop more optimized and sophisticated agents. Thus, while the novelty of this project may be questionable, it was nevertheless a highly informative and enjoyable experience for me.

III. AlphaZero Algorithm

The AlphaZero algorithm consists of three main components: (1) a policy-value neural network, (2) Monte Carlo Tree Search (MCTS) for intelligent action selection, and (3) self-play, where the agent repeatedly plays against itself. These components operate in a cyclical process, continuously improving the agent's performance through iterative refinement.

A. Policy-Value Network

Consider a neural network parameterized by weights θ, defined as $f_\theta(s) = (p_s, v_s)$. Here, $s \in S$ represents a state in the state space, and the outputs p_s and v_s are interpreted as follows:

- $p_s \in \mathbb{R}^{|A|}$ is a probability distribution over the action space A. It is optimized to approximate the probability distribution over actions chosen by an optimal player, such that $p_s(a) \approx P(a \text{ is optimal} \mid s)$. Note that our definition of p assumes a finite action space.
- $v_s \in \mathbb{R}$ is a scalar representing an approximation of the expected reward for being in state s, such that $v_s \approx \mathbb{E}[\text{reward} \mid s]$.

In Gomoku, states are typically represented as two-dimensional matrices that model the game board. Each element of the matrix corresponds to a square on the board, where a value of 0 indicates an empty square, 1 represents a square occupied by player 1's stone, and -1 represents a square occupied by player 2's stone.

In Gomoku, the spatial arrangement of stones is crucial for determining optimal moves, as patterns like rows, blocks, or threats dictate the game's strategy. To accommodate for such, we include several convolutional layers in the neural network architecture. The general architecture that was used is displayed below:

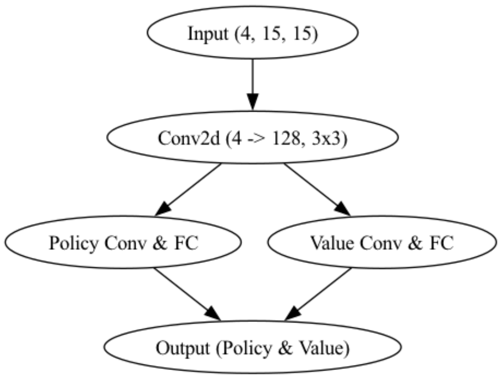

Fig. 1. The general architecture used for the Policy-Value Network.

B. AlphaZero Monte Carlo Tree Search

MCTS is an intelligent algorithm used to simulate multiple actions while focusing on the most promising ones, as determined by game outcomes and prior probabilities. In AlphaZero, MCTS combines traditional search methods with a policy-value network f_θ. The algorithm is summarized below:

Algorithm 1 AlphaZero Monte Carlo Tree Search

```
Input: state $s$, policy-value network $f_\theta$, number of simulations $N$
Output: probability distribution of optimal actions $\pi$
Initialize a tree with the root node corresponding to state $s$

for simulation $i = 1$ to $N$ do
    Selection: Traverse the tree from the root node using the PUCT formula:
        $a^* = \arg\max_a \left( Q(s,a) + c \cdot P(s,a) \cdot \frac{\sqrt{N(s)}}{1 + N(s,a)} \right)$
    Continue until a leaf node $s_{\text{leaf}}$ is reached
    Expansion: Expand $s_{\text{leaf}}$ by adding child nodes for all possible actions
    Use the policy-value network $f_\theta$ to initialize $P(s_{\text{leaf}}, a)$ and the value $v_s$
    Simulation: Use $v_s$ (from $f_\theta$) as the evaluation of $s_{\text{leaf}}$
    Backpropagation: Propagate $v_s$ up the tree, updating:
        $Q(s,a) \leftarrow$ updated mean value
        $N(s,a) \leftarrow$ increment visit count
end for
Return: $\pi(a) \propto N(s,a)$ (visit counts normalized to form a probability distribution)
```

The PUCT formula plays a critical role in balancing exploration and exploitation during the search. Specifically:

- $Q(s,a)$ is the current estimate of the expected value of taking action a in state s.
- $P(s,a)$ is the prior probability of action a, provided by the policy network f_θ.
- $N(s)$ is the total visit count of the parent node (state s), and $N(s,a)$ is the visit count of the child node (action a).
- c is a hyperparameter controlling the balance between exploration and exploitation.

Unlike traditional MCTS, which relies on random rollouts to evaluate states, AlphaZero uses the value v_s, predicted by the policy-value network, as a direct evaluation of the leaf node. During the **Expansion** step, newly added nodes are initialized with $P(s,a)$ from the network, ensuring that prior probabilities guide the search. In the **Backpropagation** step, $Q(s,a)$ is updated to reflect the mean value of the visits, while $N(s,a)$ tracks the visit counts for each action.

Finally, after all simulations are complete, the normalized visit counts

$$\pi(a) \propto N(s,a)$$

form the output policy distribution. This distribution balances exploration and exploitation and is used during training to sample actions or directly select the most promising move.

For further details on the interplay between the policy-value network and MCTS, see [1].

C. Self-Play

The self-play phase generates training data by having the agent play against itself using the current neural network parameters. A predefined number of games are played, each game continuing until completion. At each step, the state s_k, the probabilities of action of MCTS p_k, and the reward v_k are recorded. These data are saved, to be used after self-play to train the neural network, which minimizes the loss function

$$\ell = (z_k - v_k)^2 - \pi_k^T \log p_k,$$

where recall v_k and p_k are the outputs of the neural network. The process of self-play and training is repeated until a specified stopping condition is reached, such as a set number of self-play/train episodes.

IV. Methodology

The game state in Gomoku is represented as a square matrix of a fixed size. For input to the neural network, this state was encoded into four distinct matrices, each providing different information about the game environment:

1) Player's Stones: A binary matrix where a value of 1 indicates the presence of the current players stones
2) Opponent's Sontes: A similar binary matrix, but representing the opponent's stones
3) Last Action: A binary matrix where a value of 1 marks the position of the last action

4) Player Identifier: A matrix filled with the current player's ID

This representation captures both spatial and temporal features. Alternate encodings could potentially enhance performance, such as incorporating pattern-based encodings that explicitly track threats (e.g. open three-in-a-row) or multi-channel historical data that tracks recent moves. Exploring these different representations could be worthwhile in future work.

Rewards were simply defined as

- Win: $+1$
- Loss: -1
- Draw: 0

A. *Simplification of the Problem*

Training an AlphaZero agent is computationally intensive, with the MCTS as the primary bottleneck. Due to limited computational resources, the game was simplified to a 6×6 board with a win condition of achieving four in a row. This significantly reduced state and action space, enabling faster simulations and training. Despite these simplifications, however, the training process remained resource heavy, often taking more than 6 hours per training session. Even after such training, the agents were not highly competitive, suggesting that further optimization, a deeper neural network, or greater computational resources would be required for improved performance.

To overcome these challenges, I used a pre-trained model from Junxiao Song [7], who had trained an AlphaZero-based agent for a 8×8 board with a win condition of five in a row. His model, developed with more extensive computational resources, provided a stronger starting point for experimentation. By using these pre-trained weights, I could bypass some of the computational bottlenecks and focus on testing the adaptations for smaller-scale games.

V. RESULTS

Subjectively, the AlphaGomoku agent exhibited a high level of play, demonstrating strong defensive strategies and the ability to capitalize on mistakes, making it a formidable opponent.

For a more objective evaluation, the agent's performance was tested against a random player and a rule-based player. The rule-based agent uses heuristics to evaluate the state of the board, prioritizing immediate wins, blocking threats from the opponent, and creating advantageous setups such as sequences of two or three stones. Detailed implementation can be found in the attached code.

The results of AlphaGomoku's performance against these baselines are shown in Table I.

Opponent	Opening Player	Win Rate	Draw Rate	Lose Rate
Random	Random	100	0	0
Random	AlphaGomoku	100	0	0
Rule-Based	Rule-Based	60	20	20
Rule-Based	AlphaGomoku	100	0	0

TABLE I
BASELINE COMPARISONS OF ALPHAGOMOKU PLAYED AGAINST RANDOM AND RULE-BASED PLAYERS. PLAYED FOR 20 GAMES EACH.

As shown in Table I, AlphaGomoku consistently outperformed both random and rule-based players, achieving a 100% win rate when starting as the first player. When playing as the second player, a recognized disadvantage in Gomoku, AlphaGomoku achieved a 60% win rate against the rule-based agent, with losses primarily due to the first-move advantage of its opponent.

These results highlight AlphaGomoku's ability to outperform simpler strategies and demonstrate its robustness against random and rule-based approaches. Future work could involve evaluating AlphaGomoku against other reinforcement learning methods, such as Deep Q-Networks (DQNs) or policy gradient algorithms. Limited availability of comparable models for 8×8 boards and time constraints prevented such comparisons in this study, but this represents a promising direction for further research.

VI. DISCUSSION

A. *Performance*

The AlphaZero algorithm demonstrated strong performance in 8x8 gameplay, showcasing its capability to learn and master the underlying strategies of the game. Despite this achievement, its relative performance compared to other reinforcement learning (RL) algorithms remains uncertain due to the lack of a comprehensive benchmarking study. Future evaluations should incorporate direct comparisons with other state-of-the-art RL algorithms to assess its relative strengths and weaknesses.

B. *Challenges*

One of the primary challenges encountered during the project was the computational intensity of training. The Monte Carlo Tree Search (MCTS) component, combined with the neural network training, required significant computational resources. To mitigate this, pretrained weights were utilized, which expedited the training process but limited opportunities for experimentation with alternative configurations. This computational bottleneck highlights the importance of access to high-performance hardware in implementing complex RL algorithms like AlphaZero.

C. *Future Directions*

To address the challenges and limitations encountered, several avenues for future work are proposed:

- **Parallelization of MCTS**: Implementing parallelized MCTS could significantly reduce the computational burden and enable faster simulations.
- **Enhanced Computational Resources**: Accessing more computational power, such as through more GPUs or distributed computing, would enable deeper exploration of the algorithm's capabilities.
- **Comparative Analysis**: Systematic comparisons with other AlphaZero implementations and alternative RL algorithms that train faster (e.g., Proximal Policy Optimization or Deep Q-Networks) should be conducted.

- **User Interface Development**: Creating a visually appealing graphical user interface (GUI) for the game would enhance its usability and accessibility for non-technical users.

D. Overall Reflection

Overall, this project was an exciting and enlightening experience. It provided valuable insights into the intricacies of implementing a sophisticated RL algorithm like AlphaZero. The hands-on nature of the project allowed for a deeper understanding of the computational and algorithmic challenges involved, as well as the potential for future improvements. Despite the limitations, the project was highly informative and highlighted the importance of computational resources, algorithm optimization, and comparative benchmarking in advancing the field of reinforcement learning.

E. Code

The code for the project can be found on the github repository here.

References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, “Mastering the game of go without human knowledge,” *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[2] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz *et al.*, “Discovering faster matrix multiplication algorithms with reinforcement learning,” *Nature*, vol. 610, no. 7930, pp. 47–53, 2022.

[3] L. V. Allis, H. J. Herik, and M. P. Huntjens, *Go-moku and threat-space search*. University of Limburg, Department of Computer Science Maastricht, The …, 1993.

[4] W. Liang, C. Yu, B. Whiteaker, I. Huh, H. Shao, and Y. Liang, “Mastering gomoku with alphazero: A study in advanced ai game strategy,” *Sage Science Review of Applied Machine Learning*, vol. 6, no. 11, pp. 32–43, 2023.

[5] M. Naiyanayagam, “Alphazero implementation in python,” https://github.com/michaelnny/alpha_zero, 2023.

[6] hijkzzz, “Alphazero for gomoku,” https://github.com/hijkzzz/alpha-zero-gomoku, 2023.

[7] J. Song, “Alphazero$_gomoku,''2018.[Online].Available:$