Deep reinforcement learning for Playing Caro Game without human feedback

Hoang-Dieu Vu[1,2,3], Quang-Tu Pham[1], To-Hieu Dao[1,2,3], Dinh-Dat Pham[1], Van-Quan Nguyen[1], Duc-Nghia Tran[4], and Duc-Tan Tran[1,2,+]

[1]*Faculty of Electrical and Electronic Engineering, Phenikaa University, Yen Nghia, Hanoi, 12116, Vietnam*
[2]*PHENIKAA Research and Technology Institute (PRATI), A&A Green Phoenix Group JSC, No.167 Hoang Ngan,Trung Hoa, Cau Giay, Hanoi, 11313, Vietnam*
[3]*Graduate University of Sciences and Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam*
[4]*Institute of Information Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam*

***Abstract*— Gomoku, a popular board game, has been explored by scientists aiming to create computer players. However, existing computer players lack the ability to develop their own strategies, relying on pre-programmed skills. This research draws inspiration from Google's AlphaGo Zero and proposes a Deep Reinforcement Learning system for training machine Gomoku players from scratch, without human skills.**

The self-evolving players gradually develop their own skills, starting with no prior knowledge and adopting different strategies over time. The latest agent demonstrates adaptability and discernment, showing a preference for surrounding squares. While not exceptionally strong, these agents learn from previous games, showcasing the autonomous evolution of machine Gomoku players through Deep Reinforcement Learning.

This study conclusively demonstrates that Gomoku players trained using Deep Reinforcement Learning can evolve autonomously without relying on human knowledge.

Keywords: Deep Reinforcement Learning, Game, Neural Network, Gomoku.

I. INTRODUCTION

Deep reinforcement learning (DRL) [1] is a field within machine learning that combines the power of deep neural networks with reinforcement learning techniques. Reinforcement learning [2] involves an agent learning to interact with an environment by trial and error, receiving rewards or penalties for its actions. Deep neural networks, on the other hand, are capable of learning complex patterns from high-dimensional data. By integrating these two approaches, DRL aims to solve complex problems that require both perception and decision-making abilities, such as playing video games, controlling robots, or optimizing business strategies.

DRL is a vibrant and dynamic research area with significant potential for various applications and implications in the field of artificial intelligence. By learning from its own experiences and interacting with complex environments, DRL enables agents to achieve human-like or even superior performance in a wide range of domains.

This research aims to develop an AI Gomoku player with only using Deep Reinforcement Learning technique. Drawing inspiration from AlphaGo Zero, the networks implemented in this study will not rely on expert gameplay as training data. Instead, they will begin with random weights and learn through self-play with as minium human knowledge as possible. By adopting this approach, the resulting policy/value functions will diverge from human expert strategies, leading to the development of a distinct learning strategy for the AI player.

II. RELATED WORK

A. AlphaGo

AlphaGo, developed by DeepMind (a subsidiary of Alphabet Inc.), is an AI program renowned for its breakthroughs in the game of Go [3]. It became the first AI to defeat a professional player, Lee Sedol. AlphaGo combines the Actor-Critic method with the Monte Carlo Tree Search (MCTS) algorithm. The process involves training the policy network through supervised learning on expert moves and the value network through reinforcement learning using self-play and Monte Carlo rollouts. During gameplay, MCTS guides action selection by exploring and exploiting the search tree, utilizing the policy and value networks.

By combining the Actor-Critic method (with the policy and value networks) and the MCTS algorithm, AlphaGo leverages the strengths of both

approaches. The policy network provides a strong prior knowledge about good moves, while the value network helps evaluate the potential of different game states. MCTS enables AlphaGo to explore a vast search space efficiently, allowing it to make informed decisions and excel in the complex game of Go.

B. AlphaGo Zero

AlphaGo Zero, introduced in 2017 as an advanced successor to AlphaGo, differs from its predecessor in several key aspects [4]. AlphaGo relied on supervised learning from human moves, while AlphaGo Zero uses self-play reinforcement learning. It starts from scratch without human knowledge and learns through games against itself.

Unlike AlphaGo's dual-network architecture, AlphaGo Zero simplifies with a single neural network that handles both policy and value functions. In terms of performance, AlphaGo Zero surpasses its predecessor, achieving superhuman-level play and outperforming all previous versions of AlphaGo.

The emergence of AlphaGo Zero showcases the potential of AI systems to achieve exceptional performance through autonomous learning, without relying on human expertise. It paves the way for exploring self-play reinforcement learning algorithms and their application in complex domains.

C. Mastering the game of gomoku without human knowledge

This is a thesis authored by Yuan Wang in 2018 [5], addresses a similar problem to the present research. Notably, the author employs a combination of advanced techniques, including Convolutional Neural Network (CNN), Residual Neural Network (RNN), Monte Carlo Tree Search (MCTS), and Reinforcement Learning (RL), to train the agent. The training methodology follows a similar approach to that of AlphaGo Zero, relying on self-play and rewarding the agent solely for game victories.

III. REINFORCEMENT ALGORITHM

A. Deep Q-learning Network

Deep Q-learning network (DQN) combines Q-learning with deep neural networks [6], enabling learning from raw pixel inputs in challenging domains like Atari games. DQN approximates the Q-value function using a neural network that takes the environment state as input and outputs Q-values for possible actions. The network parameters are updated through interaction with the environment using temporal difference learning. However, challenges like instability, overestimation, and sample inefficiency arise when applying Q-learning with neural networks.

To address these issues, DQN introduces enhancements such as experience replay, target network, double Q-learning, and dueling network. This project utilizes a basic vanilla DQN network with two hidden fully connected layers of 256 and 512 units. For exploration and exploitation, an epsilon value of 80 is used, with training conducted over 200 games. Initially, the agent explores the environment, while in subsequent games, it follows the optimal Q-values. Therefore, the agent's performance may be limited in this report.

B. Policy Gradient

Policy gradient is a popular reinforcement learning algorithm used to train an agent to learn optimal policies for sequential decision-making tasks. It is particularly well-suited for problems where the action space is continuous or large.

The policy gradient algorithm directly optimizes the policy by estimating the gradients of the policy's performance with respect to its parameters and then updating these parameters to improve the policy. This is done by using gradient ascent to maximize the expected cumulative reward, also known as the objective or return. This means that it updates the parameters in the direction that increases the expected return the most. The update rule is:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \qquad (1)$$

where θ_t are the parameters at time t, α is the step size, and $J(\theta_t)$ is the expected return under the policy parameterized by θ_t.

To compute the gradient of $J(\theta_t)$, Policy Gradient methods use samples from the environment and apply the policy gradient theorem, which states that:

$$\nabla J(\theta) = \mathrm{E}_{\pi}\left[\nabla \log \pi(a \mid s, \theta) Q_{\pi}(s, a)\right] \quad (2)$$

where $\pi(a|s,\theta)$ is the probability of taking action a in state s under the policy parameterized by θ, and $Q_{\pi}(s,a)$ is the action-value function under the policy π. These equations were derived from a related research about Policy Gradient [7].

By repeating the process of collecting trajectories, estimating the advantages, and updating the policy parameters, the policy gradually improves and converges towards an optimal policy that maximizes the expected cumulative reward.

In this research, a policy gradient agent will be built with 2 fully connected layers of neural network for predicting with the 128 units for both layers.

C. Actor-Critic

An actor-critic network is a deep reinforcement learning algorithm consisting of two neural networks: an actor and a critic. The actor network determines actions based on the environment state, while the critic network estimates the value function to evaluate the actor's actions. The network learns by updating its parameters using the critic's feedback and temporal difference learning.

Compared to single-network methods like policy gradient or Q-learning, the actor-critic network balances exploration and exploitation more effectively and handles continuous action spaces better. Variants of actor-critic networks include A2C, A3C, TRPO, PPO, DDPG, and SAC, each with differences in network implementation, value function estimation, and parameter updates.

In this report, a PPO network (a type of actor-critic network) was implemented, utilizing "PPO clipping" to stabilize training [8]. The architectural design comprises an actor network and a critic network, each consisting of two fully connected layers with 256 units per layer. PPO optimizes the policy function while constraining deviations from the previous policy within a range. This prevents excessive policy updates that could lead to poor policies and hinder recovery.

The PPO's lost function:

$$L^{CLIP}(\theta) = E_t\left[\min\left(r_t(\theta)A_t, \text{clip}\left(r_t(\theta), 1-\text{ò}, 1+\text{ò}\right)A_t\right)\right] \quad (3)$$

- θ is the policy parameter.
- E_t denotes the empirical expectation over timesteps.
- r_t is the ratio of the probability under the new and old policies, respectively.
- A_t is the estimated advantage at time t.
- ϵ is a hyperparameter, usually 0.1 or 0.2.

D. Actor-Critic A2C

The Actor-Critic A2C algorithm extends the traditional Actor-Critic architecture, combining policy-based and value-based methods for training agents in sequential decision-making tasks [9]. It offers advantages like simultaneous update, efficiency, exploration-exploitation balance, improved sample efficiency, and policy and value estimation consistency. The goal is to learn a policy $\pi(a|s;\theta)$ and state-value function $V(s;w)$ using the actor and critic network parameters θ and w, respectively.

1) Policy update

- The actor's objective is to maximize the expected cumulative reward $J(\theta)$. The policy gradient update is given by:

$$\nabla_\theta J(\theta) \approx \mathrm{E}[\nabla_\theta \log \pi(a \mid s;\theta) A(s,a)] \quad (4)$$

- Here, $A(s,a)$ represents the advantage function, which estimates how much better (or worse) an action a is compared to the average action in state s. It is computed as:

$$\mathrm{A(s,a) = Q(s,a) - V(s;w)} \quad (5)$$

- $Q(s,a)$ is the action-value function, which represents the expected cumulative reward starting from state s, taking action a, and following the current policy $\pi(a|s;\theta)$.

- To promote exploration, an entropy regularization term is included in the objective function:

$$J(\theta) = \mathrm{E}[\log \pi(a \mid s;\theta) A(s,a) - \beta H(\pi(s;\theta))] \quad (6)$$

where $H(\pi(s;\theta))$ represents the entropy of the policy distribution.

2) Value update

- The critic's objective is to minimize the mean squared error between the estimated state-value function $V(s;w)$ and the actual returns G_t. The value function update is given by:

$$L(w) = \mathrm{E}[(G_t - V(s;w))^2] \quad (7)$$

- G_t represents the discounted cumulative rewards from time step t onward:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{T-t} r_T \quad (8)$$

where r_t is the reward received at time step t, γ is the discount factor, and T is the final time step.

3) Simultaneous updates

- The actor and critic networks are updated simultaneously using the collected experiences from the environment. The gradients obtained from equations (4) and (7) are used to update the network parameters θ and w, respectively, through gradient descent or any other suitable optimization algorithm.

The Actor-Critic A2C algorithm improves policy and value updates for enhanced stability, reduced variance, and better exploration-exploitation balance compared to the normal Actor-Critic approach. It achieves improved sample efficiency and consistency in learning. This makes it popular for training agents in reinforcement learning tasks. In this research, a CNN-based actor-critic

A2C model with one layer, kernel size 3, and a fully connected layer of 256 units was utilized.

IV. IMPLEMENT AND TRAINING

Gomoku, also known as Five in a Row, is a strategic two-player board game originating from China. It is played on a 15x15 grid, where each player possesses black or white stones. The objective is to place five stones of one's own color consecutively in a row, either horizontally, vertically, or diagonally, while simultaneously preventing the opponent from achieving the same. Despite its easy-to-learn nature, Gomoku presents a complex challenge, requiring strategic thinking and foresight.

For the purposes of this study, a Gomoku environment was constructed in Python, employing the object-oriented programming (OOP) methodology. The environment supports variable grid sizes, but a 10x10 board was used for training. The board is represented as a 2D NumPy array of size 10x10, with 0 indicating an empty cell, 1 representing a white piece, and -1 denoting a black piece. For agent training, the board is converted into a one-hot format of size 3x10x10, maintaining the original board structure while adding channels to identify moves made by white and black pieces. Since the neural networks used lack a convolutional neural network (CNN) layer, a flattening function is applied. However, the original one-hot board representation (3x10x10) is preserved as input for the A2C method.

A reward system is implemented to evaluate the agent's actions in the environment. Currently, the agent is not restricted to viable moves, leading to occasional repetition of moves. To address this issue, a penalty of -20 is imposed when the agent selects a previously executed action. However, the problem of repeating moves persists, resulting in prolonged deadlock. To overcome this, a temporary variable monitors non-viable moves, and when a threshold is exceeded, alternative predictions from external sources like random variables or a secondary agent are considered. The reward system is continuously adjusted to encourage the agent to explore diverse gameplay strategies.

In the initial experiments, the agent assumes both sides of the game to evaluate the model's performance. Rewards are assigned for successfully concluding a game, while penalties are imposed for repeating moves. Due to extended training times and subpar performance for this particular problem (show in Figure 1), the DQN method is excluded from further training, while other methods continue to exhibit satisfactory performance.

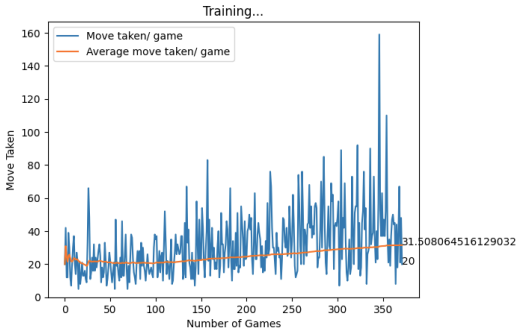

Figure 1. DQN's Performance.

The method's performance is evaluated by training the agent against another agent. The second training round involves exclusively using a random agent as the opponent. To further refine the agent's performance, adjustments are made to the reward system based on its placements of pieces in proximity to other pieces on the board. Rewards of 5, 15, and 25 are granted for connecting 2, 3, or 4 pieces consecutively. Blocking the opponent's pieces consecutively earns rewards of 5, 12, and 40. Winning the game results in a reward of 300, while aimless moves incur a penalty of -5.

After 10,000 training games against the random agent, the policy gradient method is evaluated in 1,000 games. The actor-critic PPO and A2C methods show superior performance and are evaluated in 5,000 games.

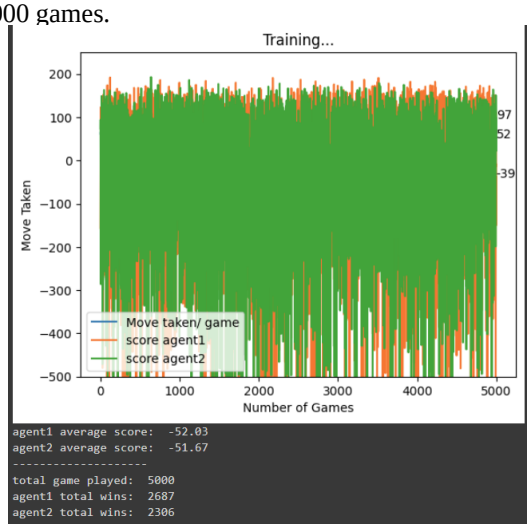

Figure 2. Policy gradient's after train performance.

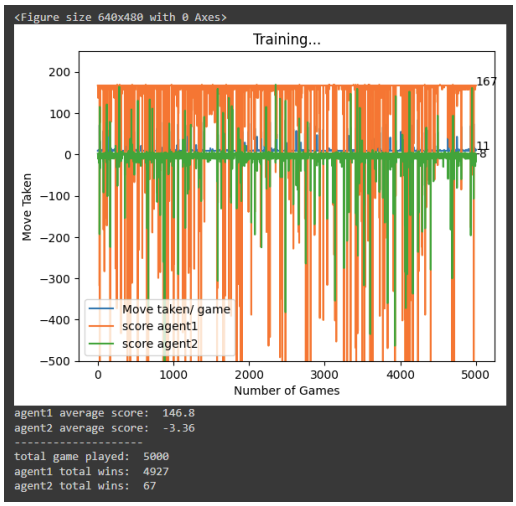

Figure 3. Actor-critic PPO's after train performance.

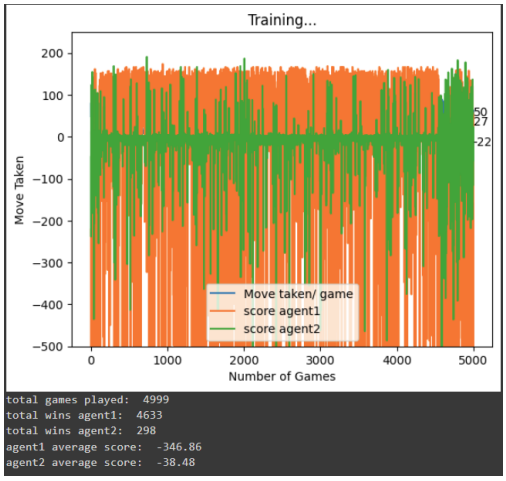

Figure 4. Actor-critic A2C's after train performance.

Agent	***Total wins vs. random agent after train (5000 games)***	***Average reward (last 100 games)***	***Training time (10000 games)***
DQN	X	X	173'53'' (400 games)
PG	2687 (53.74%)	-52.03	122'22''
PPO	4927 (98.54%)	146.8	111'51''
A2C	4633 (92.66%)	-346.86	201'52''

Table 1. Compare between the agents' performance.

Analyzing table 1, the policy gradient (PG) method shows limited learning progress with a 54% win rate after 10,000 games of training against a random agent. Network enhancements could potentially improve the A2C method's performance, but the actor-critic PPO method currently outperforms others, making it the primary focus for future optimization efforts.

The agent can successfully terminate games within five moves but struggles when games exceed this limit, resorting to random moves. To refine its abilities, a self-training approach is implemented by duplicating the agent and transferring learned weights from agent1 to agent2 after a set number of games.

However, there is a flaw in the training process. The agent's performance decline affects the opponent's performance, resulting in a mid-training peak followed by a drop-off. Although the final outcome may not reflect the peak performance, it still demonstrates satisfactory progress for future improvements.

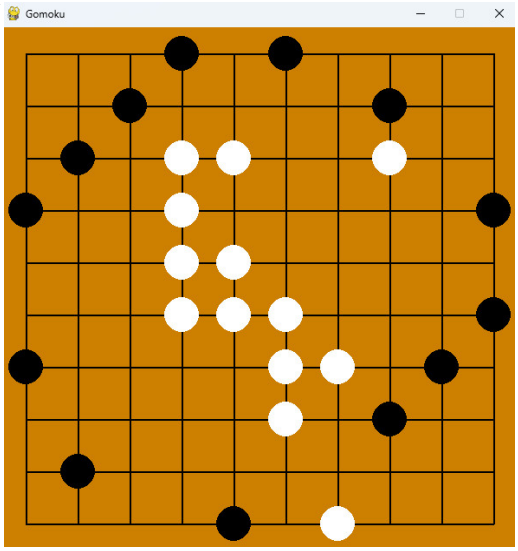

Figure 5. Actor-critic PPO's performance (white) against human (black).

Figure 5 shows the performance comparison between the actor-critic PPO agent (white) and a human opponent (black). The agent still falls short of competing at the level of human players, even when the human makes random moves. However, it has made progress in understanding how to connect pieces, focusing on connecting 3-4 pieces without completing the game. To address this issue, the best agent is established as the default agent2, while agent1 continues training until it achieves a substantial majority of victories against agent2. To improve performance, the neural networks for the actor and critic components are expanded to four fully connected layers with sizes of 256, 256, 512, and 1024 units respectively.

V. RESULT AND CONCLUSION

After several days of training, the agent has reached its peak performance. Although engaging in self-competition during the training phase, the

agent's win rate consistently falls below the 50% mark, despite an overall increase in reward compared to the previous training iteration. This outcome is illustrated in Figure 6. This may be attributed to the agent only receiving rewards for its own actions, without considering penalties for losing the game or allowing the opponent to connect 2, 3, or 4 pieces consecutively.

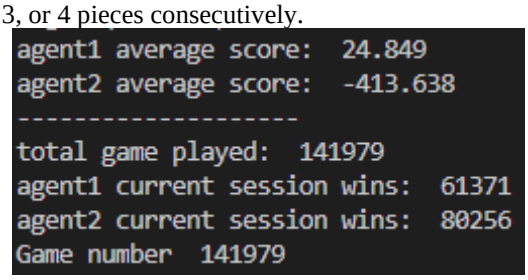

Figure 6. Actor-critic PPO's peak.

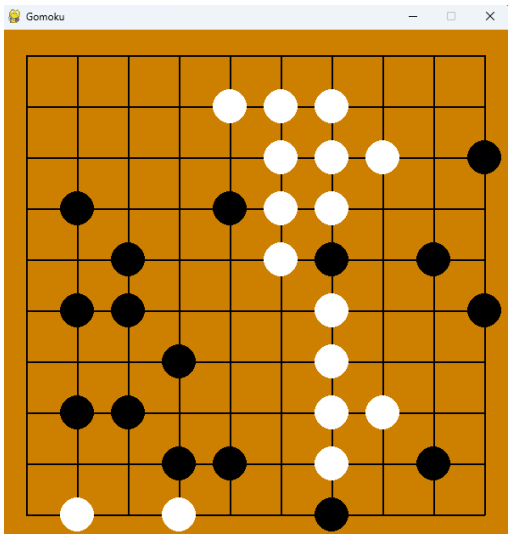

Figure 7. Real time testing against human.

During real-time testing against a human showed in Figure 7, the agent improves in connecting its own white pieces. It focuses on the 7th column but adapts to connect pieces in the 5th and 6th columns if necessary. It also shows improved ability in identifying optimal finishing moves. However, the agent still struggles in effectively blocking the opponent's movements.

In conclusion, the actor-critic PPO method has reached its limits without significant human knowledge. To enhance the agent's performance, we can explore two potential avenues: incorporating convolutional neural networks (CNN) and recurrent neural networks (RNN) or using more complex methods like A2C or A3C. These are our future goals for enhancing the agent's capabilities.

ACKNOWLEDGMENT

We would like to thank the students working at SSA Lab,Phenikaa University, for their support.

REFERENCES

[1] Mnih, Volodymyr and Kavukcuoglu, Koray and Silver, David and Graves, Alex and Antonoglou, Ioannis and Wierstra, Daan and Riedmiller, Martin, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] Sutton, Richard S. and Barto, Andrew G., "Learning from Delayed Rewards," *Machine Learning*, vol. 3, no. 1, pp. 9--44, 1981.

[3] Silver, David and Huang, Aja and Maddison, Chris J and Guez, Arthur and Sifre, Laurent and van den Driessche, George and Schrittwieser, Julian and Antonoglou, Ioannis and Panneershelvam, Veda and Lanctot, Marc and others, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484--489, 2016.

[4] Silver, David and Schrittwieser, Julian and Simonyan, Karen and Antonoglou, Ioannis and Huang, Aja and Guez, Arthur and Hubert, Thomas and Baker, Lucas and Lai, Matthew and Bolton, Adrian and others, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354--359, 2017.

[5] Y. Wang, "Mastering the Game of Gomoku without Human Knowledge," 2018.

[6] Roderick, Melrose and MacGlashan, James and Tellex, Stefanie, "Implementing the Deep Q-Network," *arXiv e-prints*, p. arXiv:1711.07478, 2017.

[7] Sutton, Richard S and McAllester, David and Singh, Satinder and Mansour, Yishay, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Advances in Neural Information Processing Systems*, MIT Press, 1999.

[8] Schulman, John and Wolski, Filip and Dhariwal, Prafulla and Radford, Alec and Klimov, Oleg, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[9] Mnih, Volodymyr and Badia, Adria Puigdomenech and Mirza, Mehdi and Graves, Alex and Lillicrap, Timothy and Harley, Tim and Silver, David and Kavukcuoglu, Koray, "Asynchronous Methods for Deep Reinforcement Learning," in *International Conference on Machine Learning (ICML)*, 2016.