A Step-by-Step Tutorial on AlphaZero and its Implementation in the Game of Gomoku

Zheng Xie *†,XingYu Fu*†,JinYuan Yu*†
*Likelihood Technology
†Sun Yat-sen University

{xiezh25, fuxy28, yujy25}@mail2.sysu.edu.cn

***Abstract*—This paper serves as a step-by-step tutorial on AlphaZero algorithm and demonstrates our implementation in the game of Gomoku, which achieves human-level performance in free-style eight by eight Gomoku.**

***Index Terms*—AlphaZero; Reinforcement Learning; Deep Learning; Monte Carlo Tree Search; Gomoku;**

I. ABOUT THIS PAPER

Google DeepMinds recent AlphaZero algorithm [1][2], accepting no domain-specific knowledges, mastered the game of Go, chess and shogi through self-played reinforcement learning from scratch, showing the potential of general AI, which is the holy grail of the entire AI community.

In this paper, we will summarize the key points of AlphaZero algorithm and give a step-by-step tutorial on it. Further, we implement an AlphaZero-version Gomoku AI program and have opened the source code of it in Github[1].

For more comprehensive explanations, you are strongly advised to read the original paper of AlphaGo Zero [1].

II. COMPONENTS OF ALPHAZERO

A. *Bionics' Explanation of AlphaZero's Components*

The AlphaZero's decision-making system is comprised of two units i.e. the policy-value network and the Monte Carlo Tree Search(MCTS) [6], both of which have some intuitive explanations in bionics.

Policy-value network functions like human's brain, which observes the current board and generates prior judgments, analogous to the human intuition, of the current situation. MCTS is similar to humans meditation or contemplation, which simulates multiple possible outcomes starting from the current state.

Like human's meditation process guided by intuition, the simulation procedure of MCTS is also controlled by the prior judgements generated from the policy-value network. Conversely, the policy-value network is also trained according to the simulation results from MCTS, an analogy that human's intellect is enhanced through deep meditation.

The final playing decision is made by MCTS's simulation results, not the prior judgments from neural network, an analogy that rational person makes decision through deep meditation, not instant intuition.

In the following subsections, we will make the above discussions more precise.

B. *Policy-Value Network*

The policy-value network $f_{\boldsymbol{\theta}}$, with $\boldsymbol{\theta}$ as the parameter, receives a tensor $\boldsymbol{s}$ characterizing the current board and outputs a prior policy probability distribution $\vec{\boldsymbol{p}}$ and value scaler v. Formally, we can write as:

$$(\vec{\boldsymbol{p}}, v) = f_{\boldsymbol{\theta}}(\boldsymbol{s}) \tag{1}$$

Specifically, in our Gomoku AI program, $\boldsymbol{s} = [\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{L}, \boldsymbol{C}]$ is a 4 by n by n tensor, where n is the width and length of the board and 4 is the number of channels. $\boldsymbol{X}$ and $\boldsymbol{Y}$ are consisted of binary values representing the presence of the current player's stones and opponent's stones respectively(X_i equals to one if the i^{th} location is occupied by the current color stone; X_i equals to zero if the i^{th} location is either empty or occupied by the other color; Y_i's assignment is analogous to X_i). $\boldsymbol{L}$ is the last-move channel whose values are binary in a way that $L_i = 1$ if and only the last move the opponent takes is at the i^{th} location. $\boldsymbol{C}$ is the color channel whose values are either constantly one or zero if the current player is black or white respectively. $\vec{\boldsymbol{p}} = (p_1, p_2, ..., p_{n^2})$ is a vector whose i^{th} component p_i represents the prior probability of placing the current stone to the i^{th} location of the board. $v \in [-1, 1]$ represents the winning value of the current player, who is about to place his(her) stone, at the current playing stage. The bigger v is, the more network believes the current player is winning.

The network we used here is a deep convolutional neural network [9] equiped with residual blocks [8] since it can solve the degradation problem caused by the depth of neural network, which is essential for the learning capability. Other mechanisms like Batch-Normolization [10] are used to further improve the performance of our policy-value net. The architecture of the net is consisted of three parts:

- *Residual Tower*: Receives the raw board tensor and conducts high-level feature extraction. The output of residual tower is passed to policy head and value head separately.
- *Policy Head*: Generates the prior policy probability distribution vector $\vec{\boldsymbol{p}}$.

[1] https://github.com/PolyKen/AlphaRenju_Zero

- *Value Head*: Generates the winning value scalar v.

And a detailed topology is shown in Table 1, Table 2, and Table 3:

Layer
(1) Convolution of 32 filters size 3 with stride 1
(2) Batch-Normalization
(3) Relu Activation
(4) Convolution of 32 filters size 3 with stride 1
(5) Batch-Normalization
(6) Relu Activation
(7) Convolution of 32 filters size 3 with stride 1
(8) Batch-Normalization
(9) Shortcut
(10) Relu Activation
(11) Convolution of 32 filters size 3 with stride 1
(12) Batch-Normalization
(13) Relu Activation
(14) Convolution of 32 filters size 3 with stride 1
(15) Batch-Normalization
(16) Shortcut
(17) Relu Activation

TABLE I
RESIDUAL TOWER

Layer
(1) Convolution of 2 filters size 1 with stride 1
(2) Batch-Normalization
(3) Relu Activation
(4) Flatten
(5) Dense Layer with n^2 dim vector output
(6) Softmax Activation

TABLE II
POLICY HEAD

Layer
(1) Convolution of 1 filter size 1 with stride 1
(2) Batch-Normalization
(3) Relu Activation
(4) Flatten
(5) Dense Layer with 32 dim vector output
(6) Relu Activation
(7) Dense Layer with scaler output
(8) Tanh Activation

TABLE III
VALUE HEAD

C. Monte Carlo Tree Search

MCTS $\alpha_{\boldsymbol{\theta}}$, instructed by policy-value net $f_{\boldsymbol{\theta}}$, receives the tensor $\boldsymbol{s}$ of current board and outputs a policy probability distribution vector $\vec{\boldsymbol{\pi}} = (\pi_1, \pi_2, ..., \pi_{n^2})$ generated from multiple simulations. Formally, we can write as:

$$\vec{\boldsymbol{\pi}} = \alpha_{\boldsymbol{\theta}}(\boldsymbol{s}) \tag{2}$$

Compared with those of $\vec{\boldsymbol{p}}$, the informations of $\vec{\boldsymbol{\pi}}$ are more well-thought since MCTS chews the cud of current situation deliberately. Hence, $\vec{\boldsymbol{\pi}}$ serves as the ultimate guidance for decision-making in AlphaZero algorithm. There are three types of policies used in playing:

- *Stochastic Policy*: AI chooses action randomly with respect to $\vec{\boldsymbol{\pi}}$, i.e.

$$a \sim \vec{\boldsymbol{\pi}} \tag{3}$$

- *Deterministic Policy*: AI plays the current optimal move, i.e.

$$a \in \arg\max_i \vec{\boldsymbol{\pi}} \tag{4}$$

- *Semi-Stochastic Policy*: At the every beginning of the game, the AI will play stochastically with respect to policy distribution $\vec{\boldsymbol{\pi}}$. And as the game continunes, after a user-specified stage, the AI will adopt deterministic policy.

and they are used in different contexts which we shall cover later.

Unlike the black-box property of policy-value network, we know exactly the logics inside MCTS algorithm. For a search tree, each node represents a board situation and the edges from that node represents all possible moves the player can take in this situation. Each edge stores the following statistics:

- *Visit Count,* N: the number of visits of the edge. Larger N implies MCTS is more interested in this move. Indeed, the policy probability distribution $\vec{\boldsymbol{\pi}}$ of state $\boldsymbol{s}$ is derived from the visit counts $N(\boldsymbol{s}, k)$ of all edges from $\boldsymbol{s}$ in a way that:

$$\pi_i = N(\boldsymbol{s}, i)^{\frac{1}{\tau}} / \sum_{k=1}^{n^2} N(\boldsymbol{s}, k)^{\frac{1}{\tau}} \tag{5}$$

  where τ is the temperature parameter controlling the trade-off between exploration and exploitation. If τ is rather small, then the exponential operation will magnify the differences between components of $\vec{\boldsymbol{\pi}}$ and therefore reduces the level of exploration.
- *Prior Probability,* P: the prior policy probability generated by the network after network evaluates the edge's root state $\boldsymbol{s}$. Larger P indicates the network prefers this move and hence may guide MCTS to exam it carefully. While note that large P does not guarantee a nice move since it is merely a prior judgment, an analogy to human's instant intuition.
- *Mean Action Value,* Q: the mean of the values of all nodes, generated by neural net, of the subtree under this edge and it represents the average wining value of this move. We can write as:

$$Q(\boldsymbol{s}, i) = \frac{1}{N(\boldsymbol{s}, i)} \sum_{v \in A} \pm v \tag{6}$$

  , where $A = \{v | (\_, v) = f_{\boldsymbol{\theta}}(\hat{\boldsymbol{s}})$, for $\hat{\boldsymbol{s}}$ in the subtree of $\boldsymbol{s}\}$. The tricky part here is the plus-minus sign "$\pm$". v is the wining value of the current player of $\hat{\boldsymbol{s}}$, not necessarily $\boldsymbol{s}$'s current player, while Q here is to evaluate the wining chance of the current player of $\boldsymbol{s}$ if he(her) chooses this move and therefore we need to adjust v's sign accordingly.
- *Total Action Value,* W: the total sum of the values of all nodes of the subtree under this edge. W serves as an

intermediate variable when we try to update Q since we have the following relation: $Q = W/N$.

Before each action is played, MCTS will simulate possible outcomes starting from the current board situation $\boldsymbol{s}$ for multiple times. Each simulation is made up of the following three steps:

- *Selection*: Starting from the root board $\boldsymbol{s}$, MCTS iteratively select edge j such that:

$$j \in \arg\max_k \{Q(\hat{\boldsymbol{s}}, k) + U(\hat{\boldsymbol{s}}, k)\} \tag{7}$$

  at each board situation $\hat{\boldsymbol{s}}$ under $\boldsymbol{s}$ till $\hat{\boldsymbol{s}}$ is a leaf node. The expression $Q(\hat{\boldsymbol{s}}, j) + U(\hat{\boldsymbol{s}}, j)$ is called the upper confidence bound, where:

$$U(\hat{\boldsymbol{s}}, j) = c_{puct} P(\hat{\boldsymbol{s}}, j) \frac{\sqrt{\sum_k N(\hat{\boldsymbol{s}}, k)}}{1 + N(\hat{\boldsymbol{s}}, j)} \tag{8}$$

  and c_{puct} is a constant controlling the level of exploration. The design of upper confidence bound is to balance relations among mean action value Q, prior probability P and visit count N. MCTS tends to select the moves with large mean action value, large prior probability and small visit count.
- *Evaluation and Expansion*: Once MCTS encounters a leaf node, say $\tilde{s}$, which haven't been evaluated by network before, we let $f_{\boldsymbol{\theta}}$ outputs its priori policy probability distribution $\vec{\boldsymbol{p}}$ and wining value v. Then, we create $\tilde{s}$'s edges and initialize their statistics as: $N(\tilde{s}, i) = 0, Q(\tilde{s}, i) = 0, W(\tilde{s}, i) = 0, P(\tilde{s}, i) = p_i$ for the i^{th} edge.
- *Backup*: Once we finish the evaluation and expansion procedure, we traverse reversely along the path to the root and update statistics of all the edges we pass in the backup procedure in a way that $N = N + 1, W = W \pm v, Q = W/N$. The plus-minus sign "$\pm$" here is to implement Equ 6.

Fig. 1. demonstrates the pipeline of decision-making process of AlphaZero algorithm.

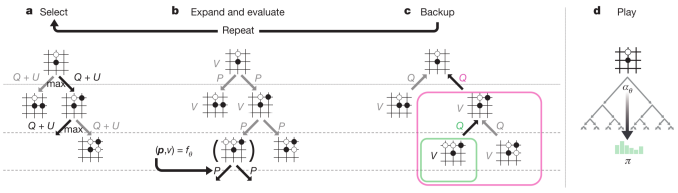

Fig. 1. Decision-Making of AlphaZero Algorithm [1]

One tricky issue to note is the problem of *end node*, e.g. draw or win or lose, which doesn't have legal children nodes. If end node is encountered in simulation, instead of executing the evaluation and expansion procedure, we directly run the backup mechanism, where the backup value is -1 if it is a win(lose) node or 0 if it is a draw node.

D. Training Target of Policy-Value Network

The key to the success of AlphaZero algorithm is that the prior policy probability distribution $\vec{\boldsymbol{p}}$ and wining value v provided by policy-value network narrow MCTS to a much smaller search space with more promising moves. The guidance of $\vec{\boldsymbol{p}}$ and v directly influences the quality of moves chose by MCTS and therefore we need to train our network in a wise way to improve its predictive intellect. There are three criteria set to a powerful policy-value network:

- It can predict the *winner* correctly.
- The *policy distribution* it provides is similar to a deliberate one, say the one simulated by MCTS.
- It can *generalize* nicely.

To these ends, we apply Stochastic Gradient Descend algorithm with Momentum [11] to minimize the following loss function:

$$L(\boldsymbol{\theta}) = (z - v)^2 - \vec{\boldsymbol{\pi}}^T log\vec{\boldsymbol{p}} + c\|\boldsymbol{\theta}\|^2 \tag{9}$$

where c is a parameter controlling the L_2 penalty and z is the result of the whole game, i.e.

$$z = \begin{cases} 1 & \text{if current player wins.} \\ 0 & \text{if game draws.} \\ -1 & \text{if current player loses.} \end{cases} \tag{10}$$

And we can see the three terms in the loss function reflecting the three criteria that we mentioned above.

III. Training Methods

A. Self-Play Reinforcement Learning

The most intriguing part of AlphaZero algorithm lies in the fact that it can master games like Go, chess and shogi without human knowledges. Hence, AlphaZero may mark the dawn of general AI, which is the holy grail of the entire AI community.

To this purpose, we train the network through self-play reinforcement learning, where we maintain a single search tree guided by a single policy-value network and let it compete with itself using semi-stochastic policy. After each move, the search tree alters its root node to the move it takes and discards the remainder of the tree until the game ends. We collect all the data generated in several games and sample uniformly from the collected data to train the network. The structure of each training data is:

$$\{\boldsymbol{s}, \vec{\boldsymbol{\pi}}, z\} \tag{11}$$

Note that Gomoku is a symmetric game, and hence we can augment training dataset by rotating and reflecting the board using seven different ways. After each training, we evaluate the training effect by letting the trained network compete with the currently strongest model using semi-stochastic policy. If the trained model wins, then we set the newly trained model to be the currently strongest and discard the old model. If the trained model loses, then we discard the newly trained model and keep the old one to be the currently strongest. Fig. 2. demonstrates the pipeline of self-play reinforcement learning.

We discuss three important questions that worth special attention in the above paragraph:

- *Why we need the evaluation procedure?* To avoid local optima by discarding poorly performed trained model.

- *Why we adopt semi-stochastic policy in self-play and evaluation?* Firstly, we add randomness into our policy to conduct exploration since more possible moves will be tried. Secondly, we let our model to behave discreetly after a certain stage to avoid bad quality data.

Another interesting observation to note is the variation of time spent in each self-play game. The time will first decay and then extend. The reason for this phenomenon is that as the agent evolves across time, it first grasps the attacking technique, which shortens the game, and then learns the defending technique, which prolongs the game.

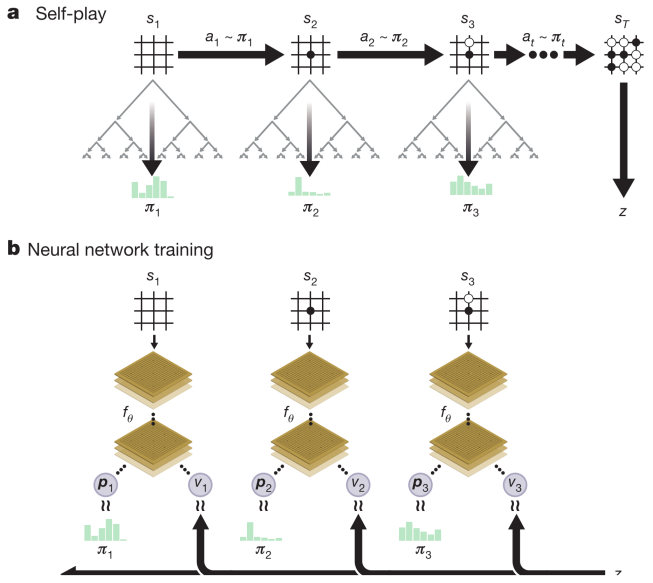

Fig. 2. Pipeline of Self-Play Reinforcement Learning [1]

If we want the trained AlphaZero AI to play with human player, then the root node of the search tree is altered to the move that human player chooses at each stage, which is slightly different from the self-play procedure. Note that we adopt deterministic policy if AI versus human to let AI perform optimally.

B. Training Acceleration through Human's Playing Data

Google DeepMind's paper has shown it is possible to train superhuman-level gaming agents without human guidance while it's computationally intensive if we want the agent to learn the strategy starting *tabula rasa*. Hence, for the sake of accelerating training, we initiate the policy-value net by letting it imitate human's playing strategy through supervised learning.

In our case, we invite two Gomoku players to play the game and record their decision-making at each move as the training dataset by:

$$\{\boldsymbol{s}, \vec{\boldsymbol{\pi}}, z\} \tag{12}$$

, where $\vec{\boldsymbol{\pi}} = (\underbrace{0, ..., 0, 1}_{i}, 0, ..., 0)$ if the i^{th} location is played.

C. Asynchronous Simulation Acceleration

The time bottleneck for the whole training process is the generation of self-play data, which is time-consuming since a large number of simulations needed at each move. Hence, to accelerate training, we need to simulate in a parallel way, to be more specific, in a *asynchronous* way.

There are three points needed to be addressed to implement the asynchronous simulation:

- *Expansion Conflict*: If two coroutines happen to encounter the same leaf node and expand the node simultaneously, then the number of children nodes under this node will be mistaken and a conflict occurs. To address this issue, we maintain an expanding list which is a list of leaf nodes under expansion. When a coroutine encounters a leaf node, instead of expanding the node instantly, the coroutine checks the expanding list to see if the current node is in it. If yes, then the coroutine waits for a short period of time to let the other coroutine expands the node and keeps on selecting after the other coroutine finishes its expansion. If no, then just executes the normal expansion procedure.
- *Virtual Loss*: To let different coroutines try as various paths as possible, after each selection, we discredits the selected edge virtually by increasing its N and deceasing its W temporarily to deceive other coroutines into choosing other edges. And the DeepMind terms the amount of increasing and decreasing as virtual loss. Clearly, each coroutine needs to clear out the virtual loss in the backup procedure.
- *Dilution Problem*: If the virtual loss and the number of coroutines are set too high, then the simulation numbers of the most promising moves may be diluted severely since many coroutines are deceived to search other less promising edges. Therefore, the tunning of related hyperparameters is crucial.

IV. EXPERIMENT

We implement an AlphaZero-version Gomoku AI in the free-style eight by eight Gomoku which achieves human-level performance in our experiment.

A. Human versus AI

Fig. 3. and Fig. 4. demonstrate some games between human and AI adopting deterministic policy with 1600 simulations per move.

We have several observations here:

- The AI has grasped some advanced strategies like "*three and three*", "*three and four*", and "*four and four*".
- We can notice that AI's strategy is *stable* with respect to reflection and rotation, which is credited to the data augmentation process.

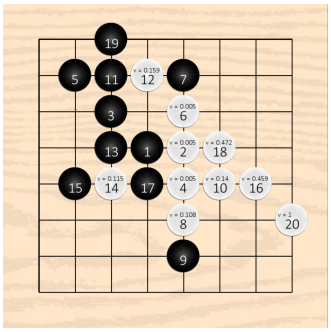

Fig. 3. Human:Black ; AI:White

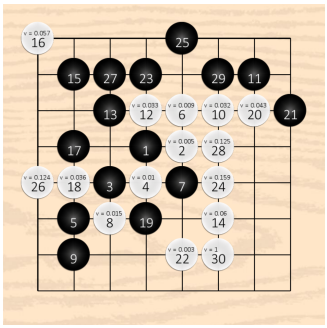

Fig. 4. Human:Black ; AI:White

B. *AI versus AI*

Fig. 5. and Fig. 6. demonstrate some games between AI and AI adopting semi-stochastic policy with 800 simulations per move.

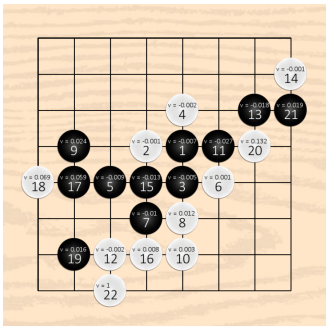

Fig. 5. AI vs AI

C. *Loss Function Graph*

Fig. 7. shows the graph of loss function in the training procedure.

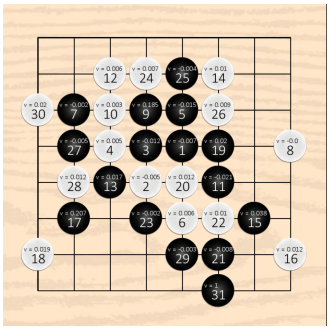

Fig. 6. AI vs AI

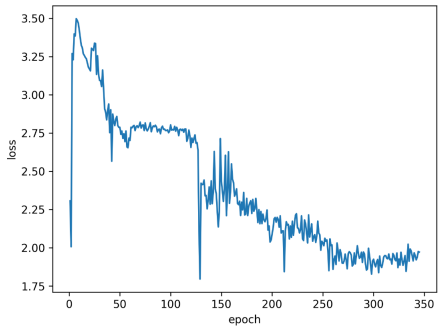

Fig. 7. Loss Function

ACKNOWLEDGMENT

We would like to say thanks to BaiAn Chen from MIT and MingWen Liu from ShiningMidas Private Fund for their generous help throughout the research. We are also grateful to ZhiPeng Liang and Hao Chen from Sun Yat-sen University for their supports of the training process of our Gomoku AI. Without their supports, it's hard for us to finish such a complicated task.

REFERENCES

[1] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of go without human knowledge[J]. Nature, 2017, 550(7676): 354.

[2] Silver D, Hubert T, Schrittwieser J, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm[J]. arXiv preprint arXiv:1712.01815, 2017.

[3] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. nature, 2016, 529(7587): 484.

[4] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529.

[5] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 1998.

[6] Browne C B, Powley E, Whitehouse D , et al. A survey of monte carlo tree search methods[J]. IEEE Transactions on Computational Intelligence and AI in games, 2012, 4(1): 1-43.

[7] Goodfellow I, Bengio Y, Courville A, et al. Deep learning[M]. Cambridge: MIT press, 2016.

[8] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[9] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[10] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.

[11] Ruder S. An overview of gradient descent optimization algorithms[J]. arXiv preprint arXiv:1609.04747, 2016.

[12] Knuth D E, Moore R W. An analysis of alpha-beta pruning[J]. Artificial intelligence, 1975, 6(4): 293-326.