Teaching Deep Neural Network to Play Go and Chinese Chess

Xu Bowen
2020533049
ShanghaiTech University
xubw1@shanghaitech.edu.cn

Ji Kaiyang
2020533124
ShanghaiTech University
jiky@shanghaitech.edu.cn

Li Binglian
2020533125
ShanghaiTech University
libl@shanghaitech.edu.cn

Ni Zhijie
ShanghaiTech University
nizhj1@shanghaitech.edu.cn

Abstract

Our work has three main parts:

1. *The first is network innovation. Based on alphazero, we use different tricks to improve the deep neural network in Monte Carlo tree search, evaluate its effect and make some explanations.*

2. *Secondly, we try scene innovation. Apply our Monte Carlo tree search neural network to different scenarios, including but not limited to Gobang, Chess, and more, and try scenarios not seen before.*

3. *Finally, based on alphazero, we extend an advanced algorithm model : muzero and apply it to different scenarios.*

1. Introduction

Since the rise of AlphaGo, many chess games AI based on deep reinforcement learning have been developed, such as AlphaGo Zero and AlphaZero. In recent years, these developed models not only play a role in chess AI, but also have been widely used in many fields, such as medical and other industries, and have made many important achievements. Therefore, the study of these models based on deep reinforcement learning is of great significance and value for the development of AI industry.

Therefore, the main objective of our project is to continue in-depth research and development on the landmark AlphaZero model. We will apply some deep learning techniques, such as pooling, multi-head model, and so on, to optimize existing networks to improve the performance of the AlphaZero model. And on the basis of AlphaZero continue to study more universal advanced version : Muzero. Compared with AlphaZero, Muzero can adapt to more environments, so it can be widely used in more fields. Moreover, we will use Chinese chess to test the performance of the improved AlphaZero model. We made our own visual interface of chess, and compared our improved AlphaZero model with the baseline model and some ai models on the market. Through experiments, it is found that our improved model through deep learning skills has indeed brought about an improvement in performance and winning rate, which also proves that our work is of great significance.

Our members contributions:

- Li Binglian: Chess logic implementation and baseline network implementation
- Ni Zhijie: Modify network to find deep learning tricks that improve model performance
- Ji Kaiyang: Visualization and training improved model
- Xu Bowen: Implementation and training of muzero model

2. Related Work

We have known the main idea of AlphaZero [7], which makes a combination of Deep Convolutional Neural Networks (CNN) and Monte Caerlo Tree Search (MCTS). Our team have built a basic model on chinese chess so that it can train and collect data by self-playing and make some simple action with MCTS prediction. Our contribution includes find some deep learning tricks on training to build more powerful models and higher efficiency.

2.1. Warm-Start

Since the neural network is trained without expert data and fast rollout policy, it faces a cold-start problem, which

may lead the model lose essentiality of some complex games. [9]. One approach to warm-start search enhancement is to use rollout instead of randomly initialized neural network for the first number of I' iteratoins and then switch it to the regular network. The warm-start iteration number I' can be either fixed or adaptive(which can remove the hyper-parameter through algorithms). Wang's work has shown that the warm-start works better when the games get depper. [9]

2.2. Multiple MCTS

Since its hard to balance the accurate state estimation by DNN and number of simulations by MCTS. One of the recently developed method is the Mulitiple Policy Monte Carlo Tree Search [5], which trians 2 Policy-Value Neural Networks with different sizes. The smaller networks takes a large number of simulations to get state priorities for the large neural network. Then the large neural network can make less simulations with better accuracy, which shows a notable improvement compared to AlphaZero.

Another method is the Dual Monte Carlo Tree Search [4], which makes 2 different tree simulations. The author uses a single neural network to simulate both trees. Also the author reduce the number of value updates by a combinition of PUCT, sliding window and ϵ-geedy algorithms.

2.3. Multihead DNN

The AlphaZero [7] trains one 2-head neural network to estimate policy and value od the inpuet game state. However, trianing such a model need a large amount of computing resource. Many researchers are looking for more powerful and efficient methodes to make AlphaZero more producable.A third action-value head has been designed for the net work.Such structure perform better than 2 head network at the zero-style iterative learning. [1] In addition, Nogozero+, change the network to be 4 heads,which include domination,score, and initial 2 heads. It also replace half number of the residual blocks by global attention residual blocks, each in the interval of two residual block.Such structure increases the training speed to six times of initial alphazero and aims to decrease the cost of training of similar areas. [2] These innovation give us idea to improve the network performance in a significiantly lower computility than the alphazero use.

2.4. Hyperparameters and Loss Function

There has been some researches on the design choices for hyperparameter values and loss functions [8]. It is difficult because of the computational cost to explore the parameter space. In general, the higher values of all hyperparameters lead to higher playing strength, while the outer iteration are more promising than inner iterations. Meanwhile, the best setting of loss functions usually depends on the game (with rules and size) rether than always the sum of policy and value loss. While if no essential information of the game is present, it can be a good default compromise.

3. Method

3.1. Monte Carlo Tree Search

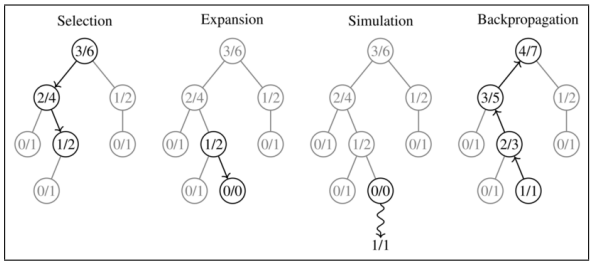

Figure 1. MCTS Procedures

The focus of MCTS is on the analysis of the most promising moves, expanding the search tree based on random sampling of the search space. The application of Monte Carlo tree search in games is based on many simulations, also called roll-outs. In each simulation, the game is played out to the very end by selecting moves at random. The final game result of each simulation is then used to weight the nodes in the game tree so that better nodes are more likely to be chosen in future simulations.

Each round of Monte Carlo tree search consists of four steps:

- Selection: Start from root R and select successive child nodes until a leaf node L is reached. The root is the current game state and a leaf is any node that has a potential child from which no simulation has yet been initiated. The section below says more about a way of biasing choice of child nodes that lets the game tree expand towards the most promising moves, which is the essence of Monte Carlo tree search.

- Expansion: Unless L ends the game decisively (e.g. win/loss/draw) for either player, create one (or more) child nodes and choose node C from one of them. Child nodes are any valid moves from the game position defined by L.

- Simulation: Complete one random simulation from node C. This step is sometimes also called playout or rollout. A simulation may be as simple as choosing uniform random moves until the game is decided (for example in chess, the game is won, lost, or drawn).

- Backpropagation: Use the result of the simulation to update information in the nodes on the path from C to R.

The main difficulty in selecting child nodes is maintaining some balance between the exploitation of deep variants after moves with high average win rate and the exploration of moves with few simulations.Researches have found that the formula $\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$ is good to measure the value of different moves,where the parameters mean:

- w_i stands for the number of wins for the node considered after the i-th move.
- n_i stands for the number of simulations for the node considered after the i-th move.
- N_i stands for the total number of simulations after the i-th move run by the parent node of the one considered.
- c is the exploration parameter—theoretically equal to $\sqrt{2}$.

But the method is based on simulations while each simulation may need a long time to end,especially when the game rules is complicated or the game has a high degree of freedom.So it's hard to simulate the total game process for many times in each move.

However,it's even harder to model a function manually which can correctly analyse and evaluate the current game state.To deal with this issue,we introduce neural networks to help analyse the current game state so that we don't need to simulate the whole game process,thus applying MCTS faster and more accurate.

3.2. Network Structure

We will use a CNN to analyse the current Chess Board and predict all possible drops and their corresponding probabilities.

Firstly,there's 7 kinds of chess pieces in the China Chess game with a color of red or black indicating different player.So there's 14 kinds of chess pieces in a game board in total.We use an indicator vector of length 14 to encode each type of chess piece.Then we have to record the coordinate of a specific chess piece on the chess board.So we will need a matrix of shape $14 \cdot 10 \cdot 9$ to represent the current chess board,where the last 2 dimensions encode the board coordinate of a certain chess piece.

So far we have encode the spatial information of the current chess board into the matrix.We also try to encode the time domain information into our matrix representation.We use an indicator matrix of shape $10 \cdot 9$ to represent the last move of the opponent player,where the position of the certain chess piece before move and the position after move is 1.What's more,we add an indicator matrix of the same shape to encode the current player.If the current is the first player,the matrix is all 1.The matrix is all -1 else.

Here's the network structure.

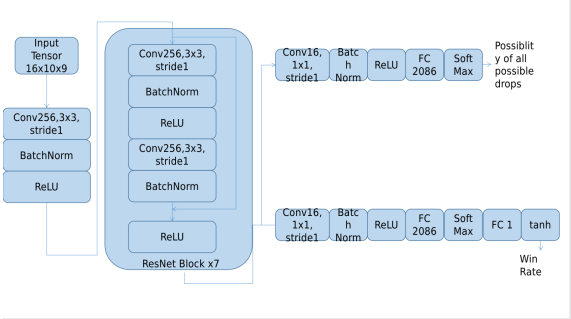

Figure 2. Structure of Networks

The net part which predicts the probability of each possible move is called the PolicyHead and the part which predicts the win-rate is called the ValueHead.

3.3. Train Pipeline

At the very beginning,the net is totally untrained.So we will use a Pure MCTS Player to play with itself and record the game process.Then We will then train the neural network with the collected data.After the net has been trained for a while, we will use the trained network to start self-play and record the games.The recorded games will then be used as the train data.And we will repeat the iteration and strengthen the network during this process.

We will start parallel sessions to accelerate collecting game records,and use one session to train the model iteratively based on self-play game records.The algorithms are shown below.

Algorithm 1: Collect

```
repeat
    game ← new Game()
    records ← previous records
    policy ← trained model
    if policy is None then
        player ← PureMCTSPlayer()
    else
        player ← MCTSPlayer(policy)
    end
    sample = game.selfplay()
    records.append(sample)
    Save records
until;
```

Algorithm 2: Train

repeat
  policy ← trained model
  repeat
    records ← previous records
  until len(records) ≥ batchsize;
  states, mcts probs, winners = records
  probs, value = policy(states)
  value loss = $||\text{value}, \text{winners}||_2^2$
  policy loss = $\frac{\text{sum(mcts probs}\cdot\text{probs,dim=1)}}{\text{batchsize}}$
  Take gradient descent step on
    value loss + policy loss
until;

4. Experiments

4.1. Squeeze-and-Excitation block

In this part,we put Squeeze-and-Excitation block after the residual blocks sequence in the origin network structure, and add drop out layers before each fully-connected layer in both policy head and value head. Squeeze-and-Excitation block compress the N * C * H* W input to N * C * 1 * 1, and scale the origin channels after several fully-connected layer, ReLU and sigmoid, in order that the network learn the importance of different channels. [3] Experiments show that the AI with SE block beat the baseline.

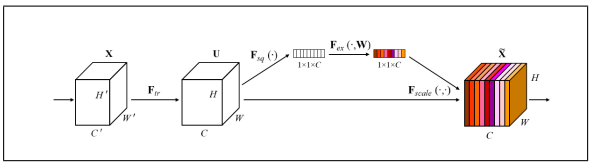

Figure 3. Squeeze-and-Excitation block [3]

4.2. Implementation Details

Training acceleration In order to speed up the training, we adopt the left-right symmetric transformation of the original model in the collect step to expand the training data set, so that the data samples in each training are twice as large as the original ones. Chinese chess is different from Go, which has better symmetry. Go can be performed four times of symmetry in horizontal, vertical and diagonal, meanwhile, three times of rotation in 90 degrees, 180 degrees and 270 degrees, so as to reach 8 times of data scale. Chess, on the other hand, has less symmetry, so we only use twice as much data amplification.

4.3. Comparison and Results

After thousand times of chess training, our model has reached the amateur level of Chinese chess players. With the same number of playouts, our model has a significantly higher winning rate than the model using pure Monte Carlo tree Search (baseling). Meanwhile, after adding Squeeze-and-Excitation blocks, our model achieves a much higher winning rate. (higher playout means deeper and larger Monte Carlo tree in the search step).

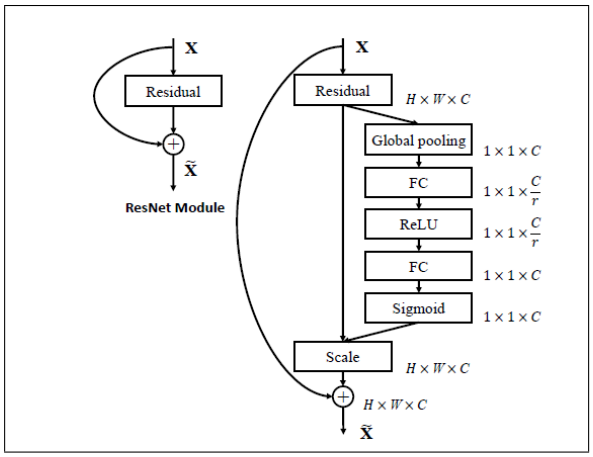

Figure 4. The original Residual module (left) and the SE-ResNet module (right) [3]

The table below shows our winning rate against pure MCTS players, sampled from 20 plays:

Player-Playout	Pure-10	Pure-100	Pure-1000
CNN-10	55%	35%	10%
CNN-100	80%	75%	30%
CNN-1000	95%	85%	45%

Table 1. Baisic Deep MCTS v.s. Pure MCTS

Player-Playout	Pure-10	Pure-100	Pure-1000
SE-10	55%	40%	25%
SE-100	85%	75%	35%
SE-1000	100%	95%	55%

Table 2. SE-Deep MCTS v.s. Pure MCTS

The results show that we introduce some innovative techniques of deep learning to optimize the network can indeed make the performance of the model better, indicating that our work is full of meaning and value.

5. Conclusion

We choose Pure MCTS with 600 simulation times as the baseline.

After training for 400 batches, AI with CNN has learnt to predict to be checkmated and take action to avoid it.Also,it has learnt to use Shi and Xiang to defend and use Pao to attack.But it still can't give a good prediction of move.

After training for 1000 batches,AI with CNN can beat baseline in most cases.In an extreme case,AI checkmates the MCTS player within 6 steps.But AI still can not beat Human player and is short-sighted when giving some predictions.

AI with SE block also beat the baseline, which proves that training make the AI stronger in Chinese chess. The theoretical optimization lacks the proof of a comparison between a origin network and new network with SE(Both should be trained for the strictly same time).

The development of AI is fast at the very beginning.But after training for about 1500 batches, the development of AI becomes slow due to lack of self-play samples.It may need more collectors to collect samples for it to continue developing.

6. Advancement: Muzero

6.1. Introduction

However, AlphaZero still has its limitations: it still needs to enter the rules of a particular game, which means that it is deficient in the general availability of different games. So again, we introduced new neural networks to learn the rules of a game so that our AI could learn the game without knowing the rules of a game, which is the basic idea of muzero. As you can see, this improvement of muzero greatly enhances the universality of different games, thus becoming a new milestone. Next, we'll introduce muzero and some of our work on muzero. [6]

6.2. Algorithm Review

Unlike AlphaZero, Muzero's biggest difference is not inputting the rules of the game. The rules referred to here include:

- terminal state and judgment of rewards
- State transition mechanism: $(s_t, a_t) \rightarrow s_{t+1}$

Therefore, in order to learn the rules of the game, Muzero introduces two additional neural networks: representation network and dynamics network.

In AlphaZero, a single prediction network is used to estimate the policy and value. While in Muzero, the role of prediction network is similar to that of prediction network in AlphaZero, which is used to estimate policy and value in the next step. On the other hand, the other two networks: representation network and dynamics network are used to learn the rules of the game.

So a summary of Muzero's deep neural networks:

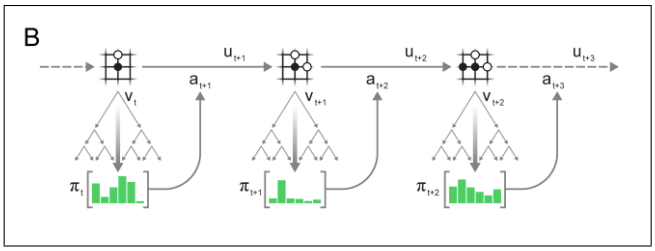

Figure 5. How Muzero acts in environment

- Representation Network h: input real game observations o, output hidden state s;
- Dynamics Network g: input hidden state s and action a, output next hidden state s' and reward r
- Prediction Network f: input hidden state s, output policy p and value v

Next, we will explain how muzero works in a real environment. The schematic is shown in Figure 5.

First, when inputting a real game situation, representation network (h) converts it into a hidden state s. After that, Muzero will use Monte Carlo tree search to expand the current situation (expressed as hidden state s). Select the next action a of the search tree based on the policy and value estimated by prediction network (f). Then, the current hidden state s and the selected action a are input into the dynamics network (g) to obtain the transferred state s' and the action reward r. Then expand the current state s' and continue the Monte Carlo tree search. After an episode of MCTS search (Figure 6), the Monte Carlo tree will give the next action a_{t+1}.

6.3. How to Train a Muzero Model

We then describe how Muzero trained its model. First, we store the observation records obtained by muzero interacting with the environment in replay buffer. Then, during the training process, we will take some records from replay buffer for training.

From previous records in replay buffer, we can get the search policy π, the value of state:z, and the observed reward r. The state value z in MCTS is calculated by the following bootstrapping method:

$$z_t = u_{t+1} + \gamma u_{t+2} + \cdots + \gamma^{n-1} u_{t+n} + \gamma^n \nu_{t+n}$$

where ν is the estimated value of MCTS.

When it comes to how to train the muzero model, we will use the records in replay buffer as a benchmark to train the model. Therefore, we will get the following minimization goals:

- Minimize the gap between the policy estimate p in the predicted network and the search policy π.

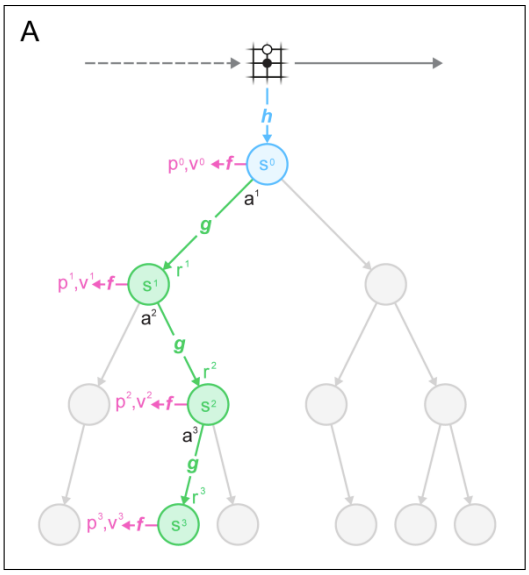

Figure 6. MCTS in Muzero

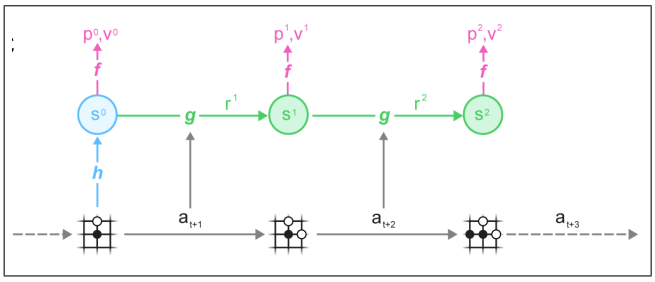

Figure 7. How to Train a Muzero Model

- Minimize the gap between the estimated state value v and the actual state value z in the predict network.
- Minimize the gap between the predicted reward r of the dynamics network and the actual observed reward u.

In order to improve efficiency and accuracy in training, we will jointly train the above three training objectives at the same time and add a L2 regularization. Therefore, we can get the final network loss $l(\theta)$, and through this loss, we use the backward propagation method to update the network parameter θ:

$$l_t(\theta) = \sum_{k=0}^{K} l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) + l^p(\pi_{t+k}, \mathbf{p}_t^k) + c||\theta||^2 \quad (1)$$

where l^r, l^v and l^p are loss functions for reward, value, and policy respectively. The schematic of training is shown in Figure 7.

```
(pid=9343) Tree depth: 302
(pid=9343) Root value for player 1: 0.60
(pid=9343) Player 1 turn. MuZero suggests GE
(pid=9343) Enter an action:
GE
(pid=9343) Played action: GE
(pid=9343)   A B C D E F G H I J K
(pid=9343) A . . . . . X . . . .
(pid=9343) B . . . . . O . X . .
(pid=9343) C . . . . . O . . . .
(pid=9343) D . . . X . O O . . . X
(pid=9343) E . X . O O . . . . . .
(pid=9343) F . . . . . X
(pid=9343) G . . . . O . . . . . .
(pid=9343) H . . . . . X .
(pid=9343) I . . . . . X .
(pid=9343) J . . . . O . . . . . .
(pid=9343) K . . . . . .
(pid=9343) Press enter to take a step
```

Figure 8. Muzero Demo

6.4. Innovations from AlphaZero

Compared to the previous AlphaZero, the most important feature of Muzero is that it no longer needs to enter the rules of the game to learn the rules and the game. To achieve this, muzero introduced two new deep neural networks based on the prediction network: representation network and dynamics network to learn rules.

This innovation greatly enhances the generality of Muzero, allowing Muzero to be used in more applications, which is its advantage over Alphazero.

6.5. Experiments of Muzero

Due to limited computing resources, we used Gomoku to experiment with Muzero rather than other games which have large state space.

The following is an example of simulation result in figure 8:

Due to limited computing resources and insufficient training time due to time constraints, our muzero has not yet been trained to a certain level. However, after testing, it can be known that the trained muzero is able to know the rules of the game, that is, it can learn certain chess principles and make some rules-compliant responses. This proves the effectiveness of the muzero model.

References

[1] Chao Gao, Martin Mueller, Ryan Hayward, Hengshuai Yao, and Shangling Jui. Three-head neural network architecture for alphazero learning, 2020. 2

[2] Yifan Gao and Lezhou Wu. Efficiently mastering the game of nogo with deep reinforcement learning supported by domain knowledge. *Electronics*, 10(13), 2021. 2

[3] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017. 4

[4] Prashank Kadam, Ruiyang Xu, and Karl Lieberherr. Dual monte carlo tree search, 2021. 2

[5] Li-Cheng Lan, Wei Li, Ting-Han Wei, and I-Chen Wu. Multiple policy value monte carlo tree search, 2019. 2

[6] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, and et al. Mastering atari, go, chess and shogi by planning with a learned model. abs/1709.01507, 2019. 5

[7] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 1, 2

[8] Hui Wang, Michael Emmerich, Mike Preuss, and Aske Plaat. Analysis of hyper-parameters for small games: Iterations or epochs in self-play? *CoRR*, abs/2003.05988, 2020. 2

[9] Hui Wang, Mike Preuss, and Aske Plaat. Adaptive warm-start mcts in alphazero-like deep reinforcement learning. *Lecture Notes in Computer Science*, page 60–71, 2021. 2