Multiplayer Gobang Based on Reinforcement Learning

Zhewei Ye
`yezhewei@umich.edu`

Yilin Li
`yilinliz@umich.edu`

Ruitao Long
`ruitao@umich.edu`

Yechen Shi
`yechensh@umich.edu`

Abstract

We proposed a multiplayer (more than two players) Gobang agent based on modified Alpha MCTS consists of transformer blocks implemented with Attention mechanism. We also implemented our agent with simplified form of other architectures such as pure MCTS and the original Alpha MCTS and compared the performance of each architecture in terms of the overall winning rate and training time. We also expanded the possibility of ordinary Gobang games by increasing the number of players that are playing on the board simultaneously. The source code of this project can be found in `https://github.com/Flanker-E/Alpha_Attention_Multiplayer_Gobang`.

1 Introduction

Due to the complex nature of games and the fun they bring, they have long been one of the hottest areas of artificial intelligence research. And as a subset of artificial intelligence, the most publicly known application of machine learning in games is probably the use of deep learning agents that compete with professional human players in complex strategy games.

In this project, we choose Gobang, aka. 5-in-a-row as our research topic. Much progress has been made using supervised learning (examples given in Section 2). However, expert data sets are often hard to collect, and are not covering all situation. By contrast, reinforcement learning systems are trained from their own experience, in principle allowing them to exceed human capabilities, and to operate in domains where human expertise is lacking. Recently, these systems have been proved to outperformed humans in computer games. And our goal is to train a Gobang agent that acquires the skill by self learning.

When designing a gobang agent, there are currently several ways to choose from, such as Minimax, pure Monte Carlo Tree Search algorithm (MCTS), MCTS aided by neural network (We will use Alpha MCTS as representative in the rest of report for it is a AlphaGo style[6] algorithm).The performance of traditional algorithm is certain since it uses hand-crafted strategy and can not be changed. However, neural network based methods can keep evolving.

We aim to incorporate Attention mechanism into our Alpha MCTS agent in order to achieve a better training time and better performance. This improvement comes from an intuitive modeling: when a human is playing gobang, one may focus on the most recent drops and some rival strategy model. Using Attention mechanism helps capture these information and will aid the decision.

Another concept we will explore in this article is multiplayer games where the number of players could be more than two, which requires a different search tree structure and reward function to train our model. At present, there are a large number of 2-player versions of games, but there is no program

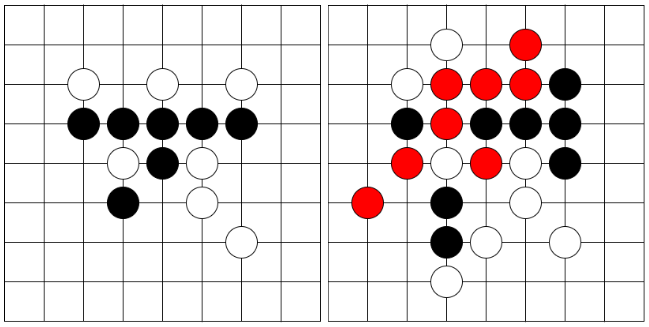
(a) 2-player **(b)** 3-player

Figure 1: Gobang example

that supports multiplayer games. Our goal is to implement a three-player gobang game so that the number of gobang players is not limited to two people.

2 Related Work

2.1 AlphaGo/AlphaGo Zero/AlphaZero

The AlphaGo family is a series of programs developed by Google DeepMind that uses a neural network to guide the search of Monte Carlo Tree Search algorithm (MCTS)[5]. AlphaGo Zero is the successor of AlphaGo, which is trained completely without any data from existing human games.[6] AlphaGo Zero performed better than standard reinforcement deep learning models (such as DQN implementations) due to its integration of Monte Carlo tree search. AlphaZero is a more generalized variation of AlphaGo Zero with the capability to play chess and shogi in addition to go. AlphaZero compensates for the lower number of evaluations by using its deep neural network to focus much more selectively on the most promising variation [9].

2.2 Vision Transformer

Vision Transformer (ViT) was proposed in 2020 [4] to show that the reliance on CNN in the field of image classification is not necessary and pure Attention applied directly to the image patch could also perform very well. This proves that transformer could achieve good results on tasks not limited to NLP, and more studies such as [2] and [1] applied transformer further on fields such as object detection and video classifications.

2.3 SegFormer

SegFormer is a semantic segmentation framework proposed in 2020 which unifies Transformers with MLP decoders.[11] SegFormer proposed a novel architeture which eliminates the need of position encoding and thus avoiding the interpolation of positional codes which worsens performance when the testing resolution differs from training. SegFormer also avoids complex decoders by proposing an MLP decoder that combines information from different layers, and thus merges both local and global Attention.

The pixel-wise prediction proposed by SegFormer has inspired us to apply Attention mechanism to board slot-wise policy prediction.

3 Proposed Method

We proposed a Gobang game software that supports human, pure MCTS agent, Alpha MCTS agent to play with each other, and in addition trains various Alpha-MCTS agents. The software is organized in the following Object Oriented Design (Fig. 2) to make it clear and extendable.

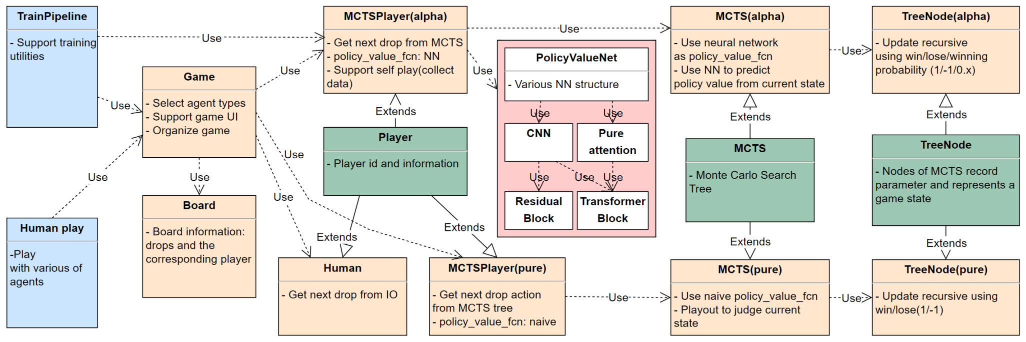
TrainPipeline
- Support training utilities
Human play
-Play with various of agents
Game
- Select agent types
- Support game UI
- Organize game
Board
- Board information: drops and the corresponding player
MCTSPlayer(alpha)
- Get next drop from MCTS
- policy_value_fcn: NN
- Support self play(collect data)
Player
- Player id and information
Human
- Get next drop from IO
MCTSPlayer(pure)
- Get next drop action from MCTS tree
- policy_value_fcn: naive
PolicyValueNet
- Various NN structure
CNN
Pure attention
Residual Block
Transformer Block
MCTS(alpha)
- Use neural network as policy_value_fcn
- Use NN to predict policy value from current state
MCTS
- Monte Carlo Search Tree
MCTS(pure)
- Use naive policy_value_fcn
- Playout to judge current state
TreeNode(alpha)
- Update recursive using win/lose/winning probability (1/-1/0.x)
TreeNode
- Nodes of MCTS record parameter and represents a game state
TreeNode(pure)
- Update recursive using win/lose(1/-1)
Use
Extends

Figure 2: Overall UML class diagram which depicts the inheritance and call relationship. The blue blocks are human interact interfaces, the green blocks are parent classes, the orange blocks are child classes, and the red block is neural network class.

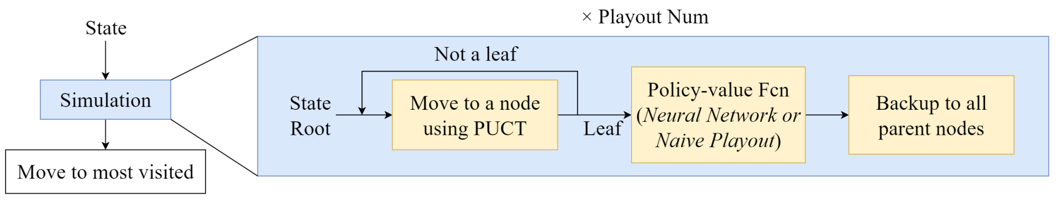
State
Simulation
Move to most visited
× Playout Num
Not a leaf
State Root
Move to a node using PUCT
Leaf
Policy-value Fcn (Neural Network or Naive Playout)
Backup to all parent nodes

Figure 3: Flow of MCTS algorithm which explains how the MCTS performs simulation on current state

3.1 Monte Carlo Tree Search (MCTS)

All of the AI agents implemented in this project are based on MCTS, and the procedure that an MCTS agent determines its next drop is shown in Fig. 3. In an MCTS tree, its root node represents the current state of game. MCTS uses nodes to store possible future actions, and uses large number active sampling search (simulation) to estimate each action's winning possibility.

In each simulation, we firstly move from root to a node before performing an evaluation. In order to form a good perspective of current state within certain amount of searches, the simulation search should balance between exploring new branches and exploiting current branch. This balance is kept by applying Polynomial Upper Confidence Trees (PUCT) algorithm which is calculated as:

$$a_{next} = a_{max(Q+U)} \tag{1}$$

where Q is the mean action-value used to evaluate the action, and U is UCB (Upper Confidence Bound). Q and U can be calculated using Eq.2, 3.

$$Q = \frac{1}{N}\sum_{i=1}^{N} V_i \tag{2}$$

$$U = c_{puct} P \frac{\sqrt{\sum N'}}{1+N} \tag{3}$$

In Eq.2, V_i is the value of former simulations, N represents the number that a node is visited. This equation encourages the exploitation of a branch with larger winning rate. In Eq.3, P is the prior probability of selecting this node. This equation encourages the exploration of other node when the number of exploitation of current branch is too big. With this method, MCTS searches more cleverly than Breadth First Search. [7][3].

Then the algorithm uses policy-value function to estimate the policy and value of this action. In pure MCTS, the value estimation is a naive rollout guided by randomly selecting the next drop until the game is over. The policy probability estimation is calculated as the reciprocal of the number of available slots on the board. After each rollout, the algorithm backups the outcome, updates all parent nodes, and creates new nodes with policy probability estimation. Thus each simulation would help the MCTS form a probabilistic perspective of possible moves, affecting root node's choice (Eq.1).

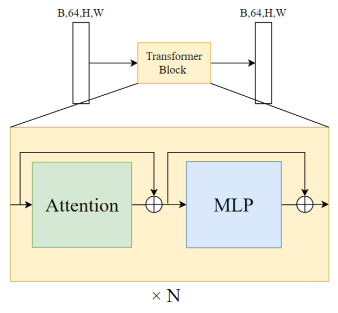

Figure 4: The topology of a single transformer block, where the Attention mechanism is utilized.

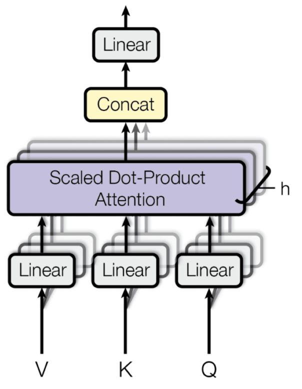

Figure 5: The topology of multi-head Attention. Figure taken from [10].

We used MCTS as our base algorithm because pure MCTS uses random selection as its policy-value function, and its performance is linearly related to its playout number (i.e. number of simulations), that is, the higher the playout number, the better the performance .

3.2 Alpha MCTS

An Alpha MCTS agent is inherited from a pure MCTS agent, and the difference between pure MCTS and Alpha MCTS is their policy-value function. Alpha MCTS introduces the neural network[6][9] to substitute the job of naive evaluation of policy-value. The output of the NN is the probability p of each feasible action in the current situation and the score v of the current situation. Value estimate is used to backup and update all parent nodes, and policy probability estimates are used to create new nodes.

Self-play is used to generate training data for itself. Each state (node) s and the corresponding police's probability π (number of visits of its child) are already recorded. When a self play ends, we have data entries (s, π, z_{empty}). Then we use the outcome (win, lose) as reward to fill all parent nodes' data z[6]. Hence we have determined the values of (s, π, z), which can be used to train the model. The goal of our training is to make the action probability p and state value v output by the strategy value network as close to the probability π and action value Q outputted by the MCTS as possible, so that the network can guide the MCTS efficiently (i.e. requires less simulations during each drop).

3.3 Attention

We tried to use Attention mechanism to improve the neural network. Our motivation of using Attention in this project is to monitor global information simultaneously and take advantage of this performance benefit.

Vaswani et al. (2017) had defined the Attention module as "mapping a query and a set of key-value pairs to an output, where the query(Q), keys(K), values(V), and output are all vectors. "[10] Here in the proposed architecture, we are using multi-head Attention (Fig. 5). Multi-head Attention allows the model to extract information from different representation subspaces at different positions, which is otherwise not achievable with single-head Attention.[10] The multi-head Attention can be calculated using:

$$\begin{gathered} MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O \\ \text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V), \\ W_i^Q \in \mathbb{R}^{d_{in} \times d_k}, W_i^K \in \mathbb{R}^{d_{in} \times d_k}, W_i^V \in \mathbb{R}^{d_{in} \times d_v}, \text{ and } W^O \in \mathbb{R}^{hd_v \times d_{in}} \end{gathered} \tag{4}$$

Unlike convolution, Attention does not record any relative or absolute position information in its structure. Instead, it acquires location representations by adding those representations to its inputs to make sure it learns the positional feature. [8] In this project we use convolution layer as position encoding.

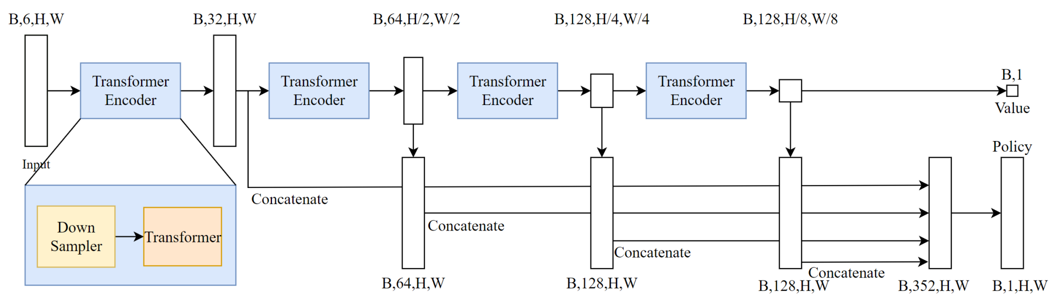

Figure 6: Topology of the Cascaded Attention Network (4 blocks) to generate value and policy

Transformer Block: The most fundamental unit of Attention mechanism implementation is a Transformer Block, which is composed of two components: a multi-head self-Attention module, and a Multi-Layer Perceptron (MLP). (Depicted in Fig.4) Since the input and output channel of a Transformer block is the same, the easiest way of adding Attention mechanism is to add a Transformer Block into a traditional CNN network, similar to plugging in a Residual Block.

Cascaded Attention (Pure Attention): To exploit the potential of Attention mechanism, we proposed a Cascaded Attention structure as encoder to replace convolution layers in our network (Fig. 6). Each transformer encoder block consists of a down sampler, which is essentially a convolutional layer, and a transformer block (See Fig. 4), which is inspired by [11]. We hope that each transformer block can capture information at different scales, therefore the down sampler is necessary to compress the dimensions. Here we used convolution with stride value of 2 (except for the leftmost module which uses stride value of 1) so that after each transformer operation the number of channels doubles and that of the dimensions halves. The output from the last transformer encoder is further converted to get the value $\in \mathbb{R}^1$, and the outputs from each of the transformer encoders are up-sampled to same width, concatenated together, and decoded to get the policy $\in \mathbb{R}^{1\times H\times W}$. Then the policy and the value is used by the Monte Carlo Tree to guide the simulation.

3.4 Multiplayer Gobang Game

Design: We have deduced how MCTS guides the play. To extend it to a 3-player version, the only thing we need is to get and pass action-value in MCTS in a "3-player way". For 2-player, we can flip a number to determine a node belongs to which player, while for 3-player, a circular modulo is needed: when forwarding, $player_{next} = (player_{last} + 1)\%3$, and when backwarding, $player_{last} = (player_{next} + 2)\%3$. The policy value of a rollout should be changed to satisfy Eq.1.

$$Q_a = N_a * 1 + N_b * (-1)$$

When there are two players a and b, assuming their chess skill is matched, the number a wins is N_a, that for b is N_b, with +1 and -1 representing winning and losing, the accumulated Q is 0.

$$Q_a = N_a * 2 * z + N_b * (-1) * z + N_b * (-1) * z \tag{5}$$

If the reward assigned to winning and losing is unchanged (i.e. +1 and -1), 3 equally skilled players won't get balanced Q. So, winning value should be 2 times of absolute value of losing (Eq. 5). We had set $z = 1$ in the above equation for simplicity and got good result.

Similar to pure MCTS implementation, in Alpha MCTS, we include a 3-layer cycle search tree model. Then we adjust the neural network's structure to meet the demand. Firstly, we take the current state, and the nearest drops of players. Then indicate who is the first to play the game. Different from pure MCTS, the backup value would be $z = v_{output}$.

Evaluation: For 2-player games, we can let 2 players play for $2n$ games (where 2 represents the possible sequence of the two players). But for 3 players, we not only should play $3n$ games, but will also face a situation of 1 A agent vs. 2 B agents, which is hard to evaluate from the number of winning and losing. So we designed a method that arranges a symmetric game with 2 A agent vs. 1 B agents, playing the same 3n games. Then, we can judge the score agent A earns in origin case and agent B in symmetric *scene*, the higher one is regarded as having higher performance. For the

Results \ Playouts	8 × 8					11 × 11				
	2000	3000	4000	5000	6000	3000	4000	5000	6000	7000
Win	3	2	2	1	0	5	5	4	3	3
Tie	0	0	0	0	0	0	0	0	0	0
Lose	3	4	4	5	6	1	1	2	3	3

Table 1: Game results of a human vs. a pure MCTS agent with corresponding number of playouts (human win or lose). The left half is the result from games played on a 8×8 board, while the right half is the result on a 11×11 board.

score design, we assign win with 3 point, tie with 1 point, lose with 0 point. This is based on the assumption that if A and B have equal performance, then after 3 games each player would win 1 game or all of them get a tie. So another criterion to determine if Agent A is more skilled than Agent B is whether A earns more than 3 point in a 1 A vs. 2 B's game.

3.5 Multiprocessing

The MCTS plus neural network structure relies on both CPU and GPU resources. By running only one process we are not utilizing all of the CPU resource. Within one self-play game, every simulation search of MCTS is based on its searching history and every search will update the tree, so this can hardly be parallelized. So we initialized two independent MCTS's and perform two self-play games simultaneously. After both self-play ended, we got two self-play data and double the amount of training data updated.

3.6 Dynamic Playout Number

From the training pattern it is worth noting that there is a circumstance in which the model is stuck at a certain playout number and stops evolving. We believe this situation may happen when the self-play data is not good enough to train the model to improve its performance. So we proposed a dynamic playout number mechanism to adjust the playout number according to the training situation. When the model is stuck at a certain pure MCTS playout number, the MCTS playout number is increased while generating self-play data. This can gradually increase the quality of self-play dataset, and consequently prevent the model from being stuck at the local optimum.

4 Experimental Results

4.1 Baseline

In the training process, we used pure MCTS with simulation number (mentioned in Sec. 3.1.) as baseline. But it is not an intuitive baseline. So we play with pure MCTS with different simulation number by ourselves to make sure where a normal human's performance is. From Tab. 1, we determined that a normal human's performance is around 2000∼4000 in 8*8 board, and around 5000∼8000 in 11*11 board.

4.2 Experiment

To fully test the potential of Attention mechanism, we set up 4 groups of networks being trained, which are basic CNN, CNN with residual blocks, CNN with transformer blocks and pure (cascaded) Attention network. We also chose 8*8 and 11*11 board size to test how the models fit game with different complexity in Tab. 2.

In addition to the possible fastest training speed, we also care about the training stability of introducing Attention mechanism, so we set up several experiments with same parameters and test the average and standard deviation of learning pace (i.e. batches of training for a model to beat a pure MCTS with playout num incrementing by 1000) in Tab. 3. We also introduced the Dynamic Playout Number (introduced in Sec. 3.6.) to reduce the possibility of not learning.

In order to balance the efficiency of validating different hyperparameters & network structures and limitation of computing resource, our experiments are firstly held on an 8×8 board, then on 11×11.

Playouts Network	2000	3000	4000	5000	6000	7000	8000	Parameter Size
Base CNN	250	550	850	950	-	-	-	0.68M
Res Blk 1	450	500	550	600	800	1000	1200	1.86M
Res Blk 2	300	**450**	**500**	**550**	**650**	**700**	**850**	3.04M
Transformer Blk 1	600	700	750	800	1050	1250	1450	1.04M
Transformer Blk 2	**200**	**450**	550	600	700	800	**850**	1.41M
Cascaded Attn (3 blks)	350	**450**	**500**	600	**650**	750	900	1.39M
Cascaded Attn (4 blks)	450	500	550	750	800	1000	1100	2.38M

(a) 11 × 11 board.

Playouts Network	2000	3000	4000	5000	6000	Parameter Size
Base CNN	350	1150	1450	1950	-	0.48M
Res Blk 1	600	1250	1500	1800	-	1.66M
Res Blk 2	**200**	**300**	**400**	**450**	**600**	2.72M
Transformer Blk 1	350	450	900	1100	1500	0.84M
Transformer Blk 2	300	400	450	600	950	1.08M
Cascaded Attn (3 blks)	550	700	900	1000	1500	1.19M
Cascaded Attn (4 blks)	350	900	1400	1800	-	2.18M

(b) 8 × 8 board.

Table 2: Number of batches required to train each model to beat a Pure MCTS model with certain amount of playouts. A dash line indicated that we are not able to train our model to beat the MCTS model in a preset threshold (2000 batches in 8*8, 1500 in 11*11).

Playouts Network	2000	3000	4000	5000	6000	Trained Rate‡
Base CNN	500/200	450/350	525/225	-	-	67% / – %
Res Blk 1	775/225	750/100	-	-	-	86% / – %
Res Blk 2	217/94	267/125	183/84	-	-	90% / – %
Transformer Blk 1	650/229	450/350	-	-	-	63% / – %
Transformer Blk 2	433/143	83/24	150/82	150/41	383/403	86% / – %
Cascaded Attn (3 blks)	511/209	367/194	308/151	238/108	-	57% / 71%
Cascaded Attn (4 blks)	827/364	300/170	575/75	-	-	15% / 43%

(a) 8 × 8 board. Here we define "trained" as showing the trend of becoming more skilled to beat a more powerful pure MCTS agent. (i.e. able to beat a pure MCTS agent with 2000 playouts within 800 batches of training) ‡ - This column is recorded with data from both 8 × 8 board and 11 × 11 board.

Playouts Network	2000	3000	4000	5000	6000	7000	8000
Base CNN	-	-	-	-	-	-	-
Res Blk 1	283/103	50/0	83/47	50/0	133/47	133/62	133/62
Res Blk 2	200/41	83/47	133/118	167/85	67/24	117/47	117/24
Transformer Blk 1	350/200	175/75	75/25	50/0	175/75	250/150	150/100
Transformer Blk 2	350/200	225/25	75/25	100/50	75/25	150/50	125/75
Cascaded Attn (3 blks)	-	-	-	-	-	-	-
Cascaded Attn (4 blks)	-	-	-	-	-	-	-

(b) 11 × 11 board.

Table 3: Average (before the slash) and standard deviation (after the slash) of the learning pace (i.e. batches of training for a model to beat a pure MCTS with 1000 playout num increment). A dash line indicates there is not enough data to calculate the corresponding value (i.e. only one group or less of data available in this category). We round all of the data to 1 in this table. The experiments are not sufficient in terms of both length and number of times, and learning has uncertainty. Hence there are some missing slots and strange numbers in the table.

4.3 Analysis

The Attention series (including Transformer blocks and Cascaded Attention) use a smaller parameter size to achieve a similar performance as CNNs (including base and Res-blocks), as shown in Tab.2. In the 11×11 case, Attention series network is more efficient. However, for a chess board of size 8×8, CNNs with Res-Blocks are better than Attention series. We believe the size of receptive field matters. The advantage of Attention is the ability to model globally distributed information (much larger receptive field!), this feature is magnified on the 11×11 board. We choose kernel size of 3 for conv layers, which is suitable for CNNs to learn on the 8×8 board but is relatively small for a 11×11 board. This slows down the learning speed of CNN, and the size of board which is relatively bigger makes the drawback of CNN, whose receptive field is fixed and smaller, become more obvious.

Attention series is capable of handling more complex modeling. The complexity of a board game can be represented as the total search space, in an 8×8 game, the search space is as large as $8!$, and that for a 11×11 board is at most $11!$, which is $\frac{11!}{8!}$ larger than the 8×8 one, while the model size increment is quite linear(Tab. 2), far less than the exponential increment of search space.

Unfortunately, Attention series doesn't show advantage in training stability and speed over CNNs in 8×8 case, as shown in Tab.2b. Cascaded Attention is harder to train than a CNN-style network. CNN naturally has the location modeling for a kernel will encode regional information, while Attention lacks the inductive bias of location modeling. Though it can interpret the input globally, the lack of positional modeling weakens its potential (position is critical in board game!). Cascaded Attention network is harder to train when it becomes deeper (more blocks cascaded in encoder) since the drawback is magnified. This is supported by the fact that the combination of CNN and Attention has an outstanding performance in both 8×8 and 11×11 case.

There are many cases that the network stops learning and ends up being stuck on a certain playout number, and we proposed Dynamic Playout Number in Sec.3.6 to alleviate this problem. We believe self-play won't guarantee the training data to be perfect, so if the network stops learning for a long time, the data in the data-buffer used to train the network will be filled with self-play data that can't help network learn, leaving the network trapped in a vicious circle.

5 Conclusion

We found that Attention mechanism have optimistic potential in board game reinforcement learning. Though it is not outstanding in a smaller challenge (8*8), it is more competitive when facing a more complex situation (11*11). We believe that Attention mechanism will be more powerful when being applied to a larger Gobang board size and other chess board games. Attention can achieve a promising performance with smaller parameter size, as shown in Tab.2, which prove its efficiency. Transformer blocks introduced into neural network can greatly increase the training speed and performance, while this mechanism may lead to some uncertainty in training. The choice of hyperparameter may also influence the success rate of training a good model (For example, we found that the learning rate for Cascaded Attention structure should be smaller than CNNs). This drawback is magnified in pure Attention network, leading to a model that consumes us loads of time to train.

6 Future Milestone

- Find a more stable way to train pure Attention model.
- Use more time to train more powerful agents, satisfying various kinds of user demand.
- Apply our method on a larger size board, further test the potential of Attention mechanism.

Author Contributions

All of the authors contributed equally to this paper.

References

[1] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid. Vivit: A video vision transformer, 2021.

[2] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers, 2020.

[3] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217, 2008.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[5] M. C. Fu. Alphago and monte carlo tree search: the simulation optimization perspective. In *2016 Winter Simulation Conference (WSC)*, pages 659–670. IEEE, 2016.

[6] S. Knapton. Alphago zero: Google deepmind supercomputer learns 3,000 years of human knowledge in 40 days. *The Telegraph*, 2017.

[7] M. P. Schadd, M. H. Winands, H. Herik, G. M.-B. Chaslot, and J. W. Uiterwijk. Single-player monte-carlo tree search. In *International Conference on Computers and Games*, pages 1–12. Springer, 2008.

[8] P. Shaw, J. Uszkoreit, and A. Vaswani. Self-attention with relative position representations, 2018.

[9] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

[11] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo. Segformer: Simple and efficient design for semantic segmentation with transformers, 2021.