Proceedings of the 3rd International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/6/20230845

Machine learning algorithm and training in Go—Take three influential program as example

Qiao Zhang

University of Science and Technology of China, 96 Jinzhai Road, Hefei City, Anhui Province, China

zqa529981932@mail.ustc.edu.cn

Abstract. In 2017, AlphaGo, an artificial intelligence in Go, beat KeJie----the No.1 Go player in 3-0, which have surprised the world, and artificial intelligence came to the attention of the public again. In this article, we take three influential artificial intelligence Go----AlphaGo, AlphaGo Zero and KataGo, as example to discuss how artificial intelligence Go work. We discuss them about their structures and training methods one by one in chronological order, which can also show the process of their development. In addition, some of the structures and training methods are enlightening to us, and we expect them can work in other fields.

Keywords: machine learning, neural network, monte Carlo tree search, AlphaGo.

1. Introduction

At the end of the 20th century, with the development of internet technology and hardware equipment, the innovative research of artificial intelligence (AI) is accelerated, and the artificial intelligence technology is further promoted to be practical, including in chess game, one of the landmark of its progress is that in chess, IBM's Deep Blue supercomputer beat world champion Garry Kasparov in 1997 [1]. After that, artificial intelligence kept going forward in chess game and continuously defeated human best professional chess players in various chess games, except Go.

Go, a strategic two-player board game, use a rectangular checkerboard and black and white dichroic round pieces to play, the two sides alternate. A regular checkerboard has 19 line segments and 361 intersections, and the pieces must walk on the intersections where the spaces are not forbidden.

There are about 10^170 legal variations in Go [2], much more than number of atoms in the universe (about 10^80), so the search tree is too much to Go through all the cases [3-4] and even the judgement of the present situation is hard. So for a long time, artificial intelligence in Go still cannot reach the level to even the best amateur player. In 2012, Zen, an Artificial Intelligence Go (AIGo) from Japan, won Masaki takemiya (one of the best professional Go player of Japan) in five and four handicap game, which meant it has come up with the level of Amateur master-hand. Nevertheless, the development stopped again, and an "optimistic estimate" from the developers of AIGo also sees it would take 15 to 20 years for AIGo to reach the level to human best professional Go player.

However, the turning point happened. In October 2015, AlphaGo made history when it defeated Fan Hui [5], becoming the first AIGo to beat a professional Go player on a 19-way board without handicap. In March 2016, AlphaGo beat Lee Sedol, a leading professional Go player, 4-1 in a five-

DOI: 10.54254/2755-2721/6/20230845

game match, becoming the first AIGo to defeat a professional Go player of nine stages. From the end of 2016 to the beginning of 2017, AlphaGo was strengthened again under the name of "Master", under the condition of not disclosing its real identity, through the informal network hayago battle (both players must make the decision in a short time) for testing, challenged the first-class Master of China, South Korea and Japan, and won all 60 games. In May 2017, an enhanced version of AlphaGo Master won 3-0 against Ke Jie, the world's No. 1 player. After that, Google Deepmind, the developer team of AlphaGo, reported a new version of its program AlphaGo Zero [6]. It learns Go by teaching itself, and beat AlphaGo Lee, the version which beat Lee Sedol in 2016, with the score 100:0 only after 3 days of training, beat Master with 40 days of training.

For other kinds of chess games, AlphaZero (with the similar network and training progress as AlphaGo Zero) all had reached the level of human best professional player within 24 hours' training in Chess, Go and Shogi. For Go, it has changed a lot the way that people think about Go both in amateur games and professional competitions. In addition, many people came to this field to find more powerful AIGo. But even for now, there is still a far way to reach the ultimate stage in Go.

For now the variations in Go is too more to traversal, if we can find some ways to simplify Go, or new way to understand it, it may be meaningful. So the research in Go will not only help us to find better choice in Go, but can also expand to other fields in picture processing, mechanical engineering and so on.

This passage will help you know about some of the influential AIGo, and discuss the potential room for improvement. Furthermore, some new improvements used in AIGo might be expanded to other fields.

2. Literature review

2.1. AlphaGo

Before AlphaGo comes out [5], the experts in AI area believes there is still a long way for Artificial Intelligence Go to reach the level of human experts. However, the emergence of AlphaGo in 2015 surprises us for beating Fan hui—a human professional Go player, and launches a blast of upsurge in the study of AI. It is the first time for a computer program to defeat a human professional player in the full-sized Go game. And it continues updating and beats leading professional Lee Sedol in 2016, the world's No. 1 player Ke Jie in 2017.

After defeated Fan hui, the team of the developers of AlphaGo published the article to explain how AlphaGo works and how did it come to be [5].

2.1.1. Basic architecture and parameters. AlphaGo consists of four main parts: policy networks, value networks, rollout policy, and Monte Carlo tree search (MCTS) [7-8].

Policy networks' input is a simple representation of the board state, which contains the understanding by human like ladder and liberties, and its output is all legal moves' probability distribution at present situation.

The input to value networks is almost the same as which to policy networks, and we use it to predict the outcome played by policy p for both players from the position of games [8].

Rollout policy's input and output are the same as policy networks. It is used to make simulations to the end of the game in only about 2μs—1/1000 of policy networks.

MCTS selects actions by lookahead search with the help of policy and value networks and playouts.

2.1.2. The way of AlphaGo running. The input to policy networks is a simple representation of the board sta.

The policy networks, value networks and rollout policy trained already are combined to an MCTS algorithm, which called AlphaGo.

DOI: 10.54254/2755-2721/6/20230845

AlphaGo traverse the tree by simulation from the root node to the terminal state by rollout policy, and each node stores the data: prior probability P(s,a), action value Q(s,a), Monte Carlo estimation of total action $W_v(s,a)$ and $W_r(s,a)$ accumulated over $N_v(s,a)$ leaf evaluations and $N_r(s,a)$ rollout rewards.

There are four steps at each simulation.

1. Selection

First, from root node, it traverses the tree until reach a leaf node s_L at time L by selecting the edge with maximum value Q(s,a) plus u(s,a),

$$a_t = \text{argmax}_a\big(Q(s_t,a) + u(s_t,a)\big) \tag{1}$$

where $u(s,a) = c_{puct}P(s,a)\frac{\sqrt{N(s)}}{1+N(s,a)}$ and c_{puct} is a constant that determines the exploration tendency (If the node has not been created, Q equals to 0). The strategy of search is to select moves with low visit count and high prior probability initially, and gradually prefer moves with high action value.

2. Expansion

If s_L is not the end state of the game, s_L will be expanded, and the probabilities p_σ for each action computed by the policy networks are stored as prior probabilities. Else, the result will be obtained and go on to the fourth step.

3. Evaluation

We add the new node gained above to a queue waiting for the value networks' evaluation, only if it has been doing so previously. In addition, this game will be simulated to the end by rollout policy, $a_t \sim p_\pi(\cdot\,|s_t)$.

4. Backup

The statistics of sL gained by the value networks and rollout policy are Backpropagated:

$$N_v(s_t,a_t) \leftarrow N_v(s_t,a_t) + 1 \tag{2}$$

$$W_v(s_t,a_t) \leftarrow W_v(s_t,a_t) + v_\theta(s_L) \tag{3}$$

$$N_r(s_t,a_t) \leftarrow N_r(s_t,a_t) - n_{vl} + 1 \tag{4}$$

$$W_r(s_t,a_t) \leftarrow W_r(s_t,a_t) + n_{vl} + z_t \tag{5}$$

$$Q(s,a) = (1-\lambda)\frac{W_v(s,a)}{N_v(s,a)} + \lambda\frac{W_r(s,a)}{N_r(s,a)} \tag{6}$$

We weight the results of value and rollout simulation, and add them up with weighting parameter λ to get Q.

After every simulations are completed, AlphaGo will choose the most visited move as the determination:

$$a = \text{argmax}N(s,a) \tag{7}$$

2.1.3. The training of policy networks, value networks and rollout policy supervised learning of policy networks. At the training pipeline's first step, AlphaGo try to learn to predict human experts' moves in the Go games by supervised learning (SL) [9-13]. The policy networks are trained on a data set of 30 million state-action pairs (s,a) from human experts' games, with the use of stochastic gradient descent to maximize the likelihood of move a at states s made by the human experts.

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma} \tag{8}$$

As a result, using all input features, it has the accuracy of 57.0%, and 55.7% with only raw board position and move history as inputs.

2.1.4. Reinforcement learning of policy networks. The training pipeline's second step focuses on improve the policy networks which we gained at the first stage by policy gradient reinforcement learning (RL) [14,15]. With the same structure and initialization as SL policy networks, the RL policy

DOI: 10.54254/2755-2721/6/20230845

networks are trained by play games between the newest policy networks and a previous randomly selected iteration of the policy networks, which is expected to strengthen the networks and prevent overfitting to the networks meanwhile. There is a reward function r(s), which equals to 0 for all non-terminal time steps t < T.

The result $z_t = \pm r(s_T)$ equals to +1 for winning, and -1 for losing at each time step t, whereafter weights are undated by stochastic gradient ascent to maximize expected outcome.

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t \tag{9}$$

2.1.5. Reinforcement learning of value networks. At the training pipeline's final step, AlphaGo centers around evaluation to the position, which is used to predict the outcome of position s with the game-playing method of policy p made by policy networks [16-18].

$$V^p(s) = E[z_t|s_t = s, a_{t\ldots T}\sim p] \tag{10}$$

And we approximate the value function with weight θ to $v^p(s)$ and the perfect(ideally) value function $v^*(s)$:

$$V_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s) \tag{11}$$

The value function is trained with state-outcome pairs (s,z), with the use of stochastic gradient descent to minimize the mean squared error(MSE) between $v_\theta(s)$ and relevant outcome z.

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial \theta}\big(z - v_\theta(s)\big) \tag{12}$$

In addition, to avoid overfitting, 30 million distinct positions are sampled from different games played by RL policy networks and itself until the game ended.

2.1.6. Rollout policy. Similar to the policy networks, the rollout policy is trained from 8 million state-action pairs (s,a) from human experts' games to maximize likelihood with the use of stochastic gradient descent.

Finally, rollout policy achieves the accuracy of 24.2%.

In the competition, the parameter of policy networks comes from SL policy networks for its test result. It might because the policy network learns from human experts tend to find more possible move than RL policy networks.

2.2. AlphaGo zero

The emergence of AlphaGo has made the history as it is the first time that a computer program had beaten a human professional player successfully in Go game with full-sized broad, but there are still some problems with AlphaGo. In 2017, the developer team reported a new program AlphaGo Zero in their new article: <*Mastering the game of Go without human knowledge*>[6], which is the new version of AlphaGo with the similar architecture but totally different training methods.

As its name, AlphaGo Zero studies Go completely without human's understanding and experience. Specifically in two aspects:

1. Unlikely to AlphaGo, the inputs to AlphaGo Zero only contain komi and history features, in order not to break the rules as repetitions are forbidden, but no other features to represent ladder, liberties and so on as AlphaGo has.

2. No human's experience is used throughout the training process, AlphaGo Zero studies Go only by itself.

2.2.1. Basic architecture and parameters. Similar to AlphaGo, AlphaGo Zero consists of policy networks, value networks, and MCTS, but value networks and playout policy are replaced by only value networks.

DOI: 10.54254/2755-2721/6/20230845

Another difference is that the inputs to the policy and value network do not contain human understanding but only history features and komi.

2.2.2. The way of AlphaGo zero running. AlphaGo Zero runs similarly to AlphaGo, but with better value and policy networks, it replaces rollout policy with only value networks.

The value of the Q is only depended on the assessment of value networks. Its traversing obeys $a_t = \mathrm{argmax}_a(Q(s_t, a) + u(s_t, a))$, and finally choose the most visited move as the determination: $a = \mathrm{argmax}N(s, a)$, the same as before.

2.2.3. The training of policy networks and value networks. As the same to its name of paper of AlphaGo Zero, it was trained by playing with itself but not from human expert's game.

In the beginning, we the initialize neural networks with random weights θ_0. At each subsequent iteration, it generated game by self-playing, which executed the previous iteration of neural networks and played a move by sampling the search probabilities. When both players pass or the game exceeds the maximum length allowed, the game terminates at step T, and then it is provided with a reward of $r_T \in \{+1, -1\}$, which means winning or losing. Each t step is stored as the data of the form of (s_t, π_t, z_t). At the same time, the neural network is trained from the data (s, π, z)sampled uniformly from we got above.

To be specific, the parameters θ of the networks are updated by gradient descent to minimize the loss function, which is the sum of mean-squared error and cross-entropy losses:

$$L = (z - v)^2 - \pi^T \log p + c\|\theta\|^2 \tag{13}$$

where $c\|\theta\|^2$can help prevent overfitting.

AlphaGo Zero surpasses AlphaGo Lee only after 36 hours and defeats it by 100 to 0 with worse hardware.

2.3. KataGo

KataGo [19] is a open-source Go engine with many improvements to accelerate learning, trained by people all over the world providing resource to let it play with itself. Besides basic function of playing Go games, it can also predict score and territory, play handicap games reasonably, and play at various board sizes and rules with the same neural network.

KataGo's overall architecture resembles AlphaGo Zero, consists of policy networks, value networks, and MCTS, but there are some improvement measures in training and new modules added to KataGo to add new features and accelerate learning to a large extent.

2.3.1. Before KataGo. After AlphaGo Zero, there are some new ways mentioned to help improve the strength of Artificial Intelligence Go.

To help discover the unexpected moves, noise is added to the policy prior at the root in Artificial Intelligence Go, and in KataGo:

$$P(c) = 0.75P_{raw}(c) + 0.25\eta \tag{14}$$

where P_{raw} (c) is the initial probability calculated by policy networks, and η follows Dirichlet distribution with parameter α = 0.03*192/N(c) on legal moves and N is the total number of legal moves.

The neural networks which guide search are a convolutional residual net with a preactivation architecture [20], which means this kind of AIGo began with small net and progressively increased its size, concurrently training the larger size on the data same as the smaller size version had, and switch when its average loss caught up to the small size.

DOI: 10.54254/2755-2721/6/20230845

2.3.2. Major general improvements. One of the main contributions of KataGo is present various domain-independent improvements that might directly be transplanted to other AlphaZero-like [21] learning or to reinforcement learning more generally.

1. Considering that there is strong possibility to be efficient for training to have more games although their quality is slightly lower, playout cap randomization is introduced to improve the way of training. On a small proportion p of moves are decided by full searches, stopping after having reached N nodes, with all other search with a much smaller cap of n < N. Only moves performing full searches are added to the training data. And fast searches are disabled Dirichlet noise and other methods in exploration to strengthen the quality.

2. There is no reason to expect the optimal level of playout dispersion in MCTS to also be optimal in real value estimation or just after longer search. So forced playouts is introduced to KataGo to ensure each child c of the root receives a minimum number of searches:

$$N_{forced}(c) = (kP(c)\sum_{c'} N(c'))^{\frac{1}{2}} \tag{15}$$

3. Global pooling adding to the neural networks enables the convolutional layers to work at the condition on global context [22], which is impossible for convolutional layers with limited perceptual radius.

4. A new channel output from the policy head is added to predict the opponent's reply on the following turn [23]. We add a term to the loss function:

$$-\omega_{opp}\sum_{m\in moves}\pi_{opp}(m)\log\left(\hat{\pi}_{opp}(m)\right) \tag{16}$$

where π_{opp} will record the the turn after the current turn as the policy target, and $\hat{\pi}_{opp}$ is the prediction of π_{opp} made by neural network.

2.3.3. Major domain-specific improvements. Some domain-specific methods are found to have nontrivial further gains.

1. Auxiliary Ownership and Score Prediction Targets

These new components are joined to KataGo [24]. To be specific, the output from decomposing the result of the game into some finer variables and three additional terms are added to KataGo:

- Ownership loss:

$$-w_o\sum_{l\in board}\sum_{p\in players} o(l,p)\log(\hat{o}(l,p)) \tag{17}$$

where o(l,p)∈{0, 0.5, 1} indicates whether l finally belongs to p, or is shared, $\hat{o}$ is the prediction of o, and b∈[9,19] is board's width, w_o=1.5/b2.

- Score belief loss("pdf"):

$$-w_{spdf}\sum_{x\in possible\ scores} p_s(x)\log\left(\hat{p}_s(x)\right) \tag{18}$$

where ps is the final score difference as the form of one-hot encoding, $\hat{p}_s$ is the prediction of ps, and $w_{spdf} = 0.02$.

- Score belief loss("cdf"):

$$W_{scdf}\sum_{x\in possible\ scores}(\sum_{y<x} p_s(y) - \hat{p}_s(y))^2 \tag{19}$$

where $w_{scdf} = 0.02$. While "pdf" loss rewards predicting the score accurately, "cdf" loss pushes the quality to be closed to the final score.

2. Go-specific Features

Besides raw features showing the state of the board, the rules, the history and komi, some game-specific higher-level features are also input to KataGo's network, including liberties, pass-alive regions, ladders.

DOI: 10.54254/2755-2721/6/20230845

Furthermore, KataGo uses two minor Go-specific ways to optimize the networks. One is to forbid moves in pass-alive territory after a certain number of consecutive passes. The other one is to add a tiny bias to favor passing when passing and continuing play might give arise to identical scores. These two methods are to reduce the time required.

According to the benefit that domain-specific Improvements brings, it suggests a general meta-learning heuristic: adding subcomponents to predict of desired targets could improve training significantly. In addition, game-specific input features will also improve training greatly.

2.3.4. Achievement. To compare the impact of the different techniques mentioned above, there is a small experiment in which shorter training runs with various components removed. The summary below shows the comparison (See Table 1):

Table 1. Factors are based on shorter runs.

Removed Component	Elo	Factor
Main (baseline model)	1329	1.00x
Playout Cap Randomization	1242	1.37x
Forced Playouts and Policy Target Pruning	1276	1.25x
Global Pooling	1153	1.60x
Auxiliary Policy Targets	1255	1.30x
Auxiliary Ownership and Score Prediction Targets	1139	1.65x
Go-specifific Features	1168	1.55x

3. Discussion

1. The accuracy of neural network is not high enough, which means Artificial Intelligence Go need to search for the subsequent moves to find the better choice. But the search breadth and depth are too large to use the traditional search methods like max-min search used in other AI chess, therefore Monte Carlo tree search is here to solve it.

2. In AlphaGo, the input to the neural network contains not only the state of the board and history, but also with human's understanding like ladder and liberties. But in AlphaGo Zero, the input only includes the board and history, and it learned Go completely by itself. It interprets its title 'without human knowledge' perfectly. However, as <*Accelerating Self-Play Learning in Go*>[19] mentioned, from a practical perspective, the go-specific higher-level features input to neural networks will improve the training greatly.

3. Though in self-played training, whether the move is good or bad only depends on the result, it can be solved with a mass of games for symmetry.

4. The neural network trained as a whole rather than trained one after another like AlphaGo will improve training and increase the ceiling. Convolutional residual net with a preactivation architecture will help achieve this outcome also.

4. Conclusion

As computer science and hardware technology are developing rapidly, AI plays an increasingly significant role in various fields. Specifically, in Go, the level of AI keeps upgrading and finally outperform humans in 2017. But it is not the ultimate form of Go and there is still a lot of room for AIGo to improve. This paper has presented the details of three influential AIGo about their structure and training methods. Their Basic Architecture consists of three main parts: policy networks, value networks and Monte Carlo tree search. Policy networks are in charge of outputting prior probability distribution over all legal moves. Value networks predict the outcome from the position of games. And MCTS performs lookahead search, which can compensate for the inaccuracy of neural network. AlphaGo Zero prove that this neural network inserting in MCTS is enough for AI to learn by itself to reach a level beyond that of human beings. In KataGo, many improvements are applied, and from the comparison we can see these improvements have exactly help a lot in training. Furthermore, some of

DOI: 10.54254/2755-2721/6/20230845

the improvement ways, like playout cap randomization, global pooling and auxiliary ownership and score targets, are worth being doing further research for their innovation and significant improvements bringing for program performance. Nevertheless, the variations in Go are too much for the present AI's structure to get well know all of them, and it seems impossible for it to reach the ultimate Go level by keeping going in this way. Though AI has reached a high level in Go with the help of MCTS, in other more complicated games with the characteristics of larger search space, like Real-Time Strategy Game (RTS), it cannot work as well as we expect, so it needs upgrading in order to adapt other fields with different characteristics. In the future, the ways that KataGo uses to accelerate learning have the value of being studied, and we expect them to be enlightening to developers in other fields.

References

[1] Persson C G A, Erjefält J S, Korsgren M, et al. 1997 The mouse trap[J]. *Trends in pharmacological sciences*, 18(12): 465-467.
[2] Allis L V. 1994 Searching for solutions in games and artificial intelligence[M]. Wageningen: Ponsen & Looijen.
[3] Van Den Herik H J, Uiterwijk J W H M, Van Rijswijck J. 2002 Games solved: Now and in the future[J]. *Artificial Intelligence*, 134(1-2): 277-311.
[4] Schaeffer J. 2000 The games computers (and people) play[M]//Advances in computers. *Elsevier*, 52: 189-266.
[5] Silver D, Huang A, Maddison C J, et al. 2016 ing the game of Go with deep neural networks and tree search[J]. *nature*, 529(7587): 484-489.
[6] Silver D, Schrittwieser J, Simonyan K, et al. 2017 ring the game of go without human knowledge[J]. *nature*, 550(7676): 354-359.
[7] Coulom R. 2006 ient selectivity and backup operators in Monte-Carlo tree search[C]//International conference on computers and games. *Springer, Berlin, Heidelberg*, PP72-83.
[8] Kocsis L, Szepesvári C. 2006 Bandit based monte-carlo planning[C]//European conference on machine learning. *Springer, Berlin, Heidelberg*, PP282-293.
[9] Coulom R. 2007 Computing "elo ratings" of move patterns in the game of go[J]. *ICGA journal*, 30(4): 198-208.
[10] Stern D, Herbrich R, Graepel T.2006 Bayesian pattern ranking for move prediction in the game of Go[C]//Proceedings of the 23rd international conference on Machine learning. PP873-880.
[11] Sutskever I, Nair V. 2008 Mimicking go experts with convolutional neural networks[C]//International Conference on Artificial Neural Networks. *Springer, Berlin, Heidelberg*, PP101-110.
[12] Maddison C J, Huang A, Sutskever I, et al. 2014 Move evaluation in Go using deep convolutional neural networks[J]. arXiv preprint arXiv:1412.6564.
[13] Clark C, Storkey A. 2015 Training deep convolutional neural networks to play go[C]//International conference on machine learning. PMLR, PP1766-1774.
[14] Williams R J. 1992 Simple statistical gradient-following algorithms for connectionist reinforcement learning[J]. *Machine learning*, 8(3): 229-256.
[15] Sutton R S, McAllester D, Singh S, et al. 1999 Policy gradient methods for reinforcement learning with function approximation[J]. Advances in neural information processing systems, P12.
[16] Schraudolph N, Dayan P, Sejnowski T J.1993 Temporal difference learning of position evaluation in the game of Go[J]. Advances in neural information processing systems, P6.
[17] Enzenberger M. 2004 Evaluation in Go by a neural network using soft segmentation[M]//Advances in Computer Games. Springer, Boston, MA, PP97-108.
[18] Silver D, Sutton R S, Müller M. 2012 Temporal-difference search in computer Go[J]. Machine

DOI: 10.54254/2755-2721/6/20230845

learning, 87(2): 183-219.
[19] Wu D J. 2019 Accelerating self-play learning in go[J]. arXiv preprint arXiv:1902.10565.
[20] He K, Zhang X, Ren S, et al. 2016 Identity mappings in deep residual networks[C]//European conference on computer vision. Springer, Cham, PP630-645.
[21] Silver D, Hubert T, Schrittwieser J, et al. 2018 A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play[J]. *Science*, 362(6419): 1140-1144.
[22] Hu J, Shen L, Sun G. 2018 Squeeze-and-excitation networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 7132-7141.
[23] Tian Y, Zhu Y. 2015 Better computer go player with neural network and long-term prediction[J]. arXiv preprint arXiv:1511.06410.
[24] Wu T R, Wu I C, Chen G W, et al. 2018 Multilabeled value networks for computer Go[J]. *IEEE Transactions on Games*, 10(4): 378-389.