바둑학연구, 제18권 제2호, pp. 162
Journal of Go Studies
2024. Vol.18, No.2. pp. 17-50

Combinatorial Game Theory Meets Deep Learning:Efficient Endgame Analysis in Go

Stanisław Frejlak
(University of Warsaw, Poland)

Abstract: The endgame stage of Go presents a unique challenge for scientific research. Contrary to previous stages, in the endgame the key to a successful analysis is board decomposition into smaller, independent local positions. Go players typically analyze these positions separately and prioritize moves based on their value. In this paper, I introduce a novel program that automates this decomposition-based analysis for the endgame stage of Go.

AlphaZero has revolutionized Computer Go, by applying a generic move-selection mechanism, based on neural network judgments and the MCTS search algorithm. However, it does not specifically address the complexity of endgame in the aforementioned manner. On the other hand, by leveraging the decomposition-based analysis, my program reaches decisions in the endgame with relatively little computation. Additionally, it offers insights for Go practitioners by providing accurate move value evaluations.

Notable prior work on automated endgame analysis was done by Martin

Müller (1995). His program Explorer checked all possible variations in every undecided position and aggregated the results based on an algorithm inspired by the Combinatorial Game Theory (CGT). However, due to the exponential growth of the number of variations, Explorer's application was limited to small, tightly bounded local positions.

In contrast, my program leverages a neural network to predict optimal local moves, dramatically reducing the number of variations that need to be explored. Provided that the neural network's predictions are correct, the program can accurately evaluate move values by considering relatively few variations, just like human Go experts do. Thanks to this approach, it is the first program capable of analyzing large, unbounded local positions, which are commonly encountered in real games.

The neural network was fine-tuned from a pre-trained AlphaZero reimplementation on the task of optimal local move prediction. Training data was gathered from KataGo self-play games, utilizing KataGo's network to perform board decomposition.

Keywords: AlphaZero, Fine-Tuning, Combinatorial Game Theory, Temperature, Move Values

I. Introduction

1. Mathematical approach to endgame

Positional analysis in the endgame stage of Go differs from the previous stages of the game. In earlier stages, a Go player's judgment about which moves are the largest on the board mainly depends on a player's intuition which might be difficult to formalize. On the other hand, in endgame, Go players assess move values using a principled approach which leverages certain arithmetical calculations. Prerequisite of this method is understanding what variations could be expected in a given local position. A player would read such variations until the territory borders are fixed, calculate the final local score at the end of each variation, and aggregate these results to arrive at a precise number denoting the value of the first move in these variations. The method is explained e.g. in the book "Rational Endgame" (Törmänen, 2019). While a move with the highest value is not guaranteed to be the optimal play, choosing moves based on their values is a common heuristic used by Go players.

This kind of analysis is possible in endgame because at this stage, the board is already mostly split between White's and Black's territories. Territory borders are not fixed only in a few local regions of the board. In general, situations in each such area could be analyzed independently of each other, because no matter which variation is played out in one region, it does not affect which moves are correct in other regions1). Moreover, these undecided areas of the board have limited size which allows an experienced player to easily find all correct local variations.

1) A notable exception for this general rule are ko fights that introduce interactions between different parts of the board.

These properties of Go endgames have inspired mathematicians to develop a new branch of mathematics (Conway, 1976, Prologue), called Combinatorial Game Theory (CGT). It is an abstract theory applicable to various games (e.g. Nim, Hackenbush) that could be viewed as sums of simpler games, just like a whole-board endgame position can be viewed as a sum of local positions. When it comes to Go, the most important tool provided by CGT is the notion of temperature. Temperature can be described as a measure of urgency of playing in a specific position. The temperature of a local position is a number equal to the value of the best move available in that position. The mathematical theory arrives at the same numbers as the classical method used by Go players, despite using a different algorithm for calculating them. CGT also provides a lot of new results that were not known to Go players, such as a detailed treatment of infinitesimal values (Berlekamp and Wolfe, 1994), or the Orthodox Accounting Theorem (Siegel, 2013, ch. VII, Theorem 2.9) which finds the maximal loss that a player might incur under orthodox play, i.e. when basing their decisions in endgame on move values. Another result (Wolfe, 2002) shows that finding an optimal line of play for a Go endgame position (and proving it optimal) is a problem of infeasible complexity. These findings prove that orthodox play is a good heuristic - a fact which Go players had understood intuitively.

The existence of a solid mathematical theory describing exact algorithms that could aid endgame calculations inspires a natural question whether these algorithms can be implemented in a computer program. Can a program like AlphaGo, or any other, tell us the value of a move in the endgame? Unfortunately, solutions existing to date are not well suited for providing such information.

2. AlphaZero mode of operation

In 2016, for the first time a world champion in Go was beaten by a computer program, AlphaGo. A follow-up paper of the AlphaGo creators described AlphaZero (Silver et al., 2017) - a neat machine learning solution allowing to build an agent that could master Go, or any other game like chess or shogi. By now, many programs inspired by that paper have emerged and have been made publicly available, presenting the Go community with invaluable teaching tools.

While AlphaZero can point out what moves are best on the board in any given whole-board position and tell which player is ahead, it cannot provide all information which a Go player might be interested in during a game analysis. Several programs inspired by AlphaZero offer also other clues. Notably, KataGo (Wu, 2020) predicts the ownership for each intersection of the board and the lead of a player measured in points. Attempts were made (Frejlak, 2020) to leverage this information to the goal of estimating move values in the endgame. However, these estimates could not be made accurate. Generally, local temperature calculation cannot be boiled down to comparison of global scores in a given whole-board position.

AlphaZero-based programs cannot be successfully used for calculating move values because their analysis is inherently global. AlphaZero is trained to predict best moves on the whole board, maximizing its winning chances. This characteristic contributes to AlphaZero's extremely high level of play. At the same time, however, it makes it difficult to perform any analyses focused on specific parts of the board.

3. AI estimating move values

A program that could estimate move values in endgame would be valuable for the Go community. It could provide more explainable clues for Go practitioners than current solutions do. Moreover, if the program is very good at its task, it could potentially achieve higher playing strength in endgame than AlphaZero. An interesting open question is whether a perfect agent following the orthodox play heuristic would perform better than AlphaZero.

The current work continues on my master's thesis (Frejlak 2024) and improves on its results. Better results are achieved thanks to more sophisticated training data construction which I discuss in Section IV.

II. Related work

There is one famous work done on a similar topic as the current paper. Martin Müller (1995) in his PhD dissertation presented Explorer, a program which leveraged Combinatorial Game Theory to search for optimal play. Explorer splits the board into sure territories and undecided positions, checks all possible variations in every position, and aggregates the results to find the best move on the board.

Presented approach had two major limitations:

The number of possible local variations grows exponentially with the size of the position. Because of that, analysis could be performed only for positions with no more than ten empty intersections. Larger positions would require too much computation.

Board segmentation was performed with an algorithm proving that stones

surrounding territories are impossible to kill, and territory areas are too small for a successful invasion. Therefore, only board positions with very solid shapes could be analyzed.

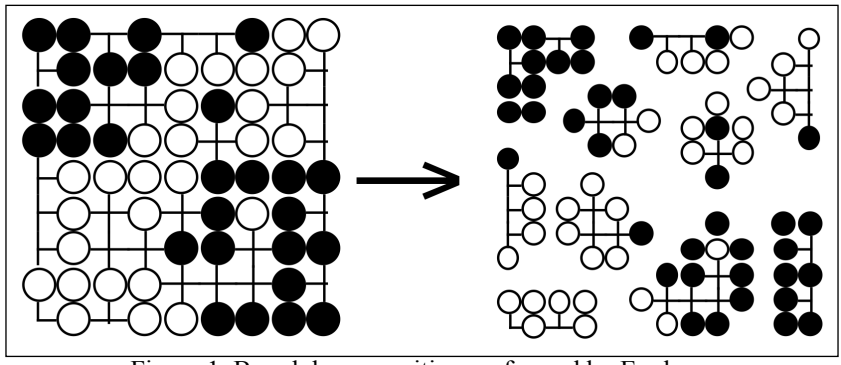

Figure 1. Board decomposition performed by Explorer

Figure 1 is an example taken from the original work of a position that could be analyzed by Explorer. It looks quite artificial with strong walls of stones and each alive group having clear two eyes. Undecided positions between these walls have clear boundaries and are of limited size.

While Explorer was very efficient in finding optimal play when compared with a brute-force approach, its applicability could not be extended to positions taken from actual games.

III. Goal of this work

The goal of Explorer was finding an optimal line of play in the endgame. On the other hand, the goal of this work is facilitating orthodox play. Below, I explain the difference between the two, giving a rationale for my choice. I also take this opportunity to clarify an important technical aspect of the cur-

rent work which is related to the notion of forcing moves (sente).

1. Canonical forms vs. temperature theory

Analysis of a local endgame position could leverage Combinatorial Game Theory in two ways.

Once all possible variations were checked, the resultant variation tree can be simplified by getting rid of all moves which are surely no better than other plays. An example is shown on Figure 2. In no game can Black's move at B be better than the move at A. When we remove all tree branches containing such bad moves, the resultant tree is called a canonical form.

In Explorer, a canonical form was found for the variation tree of every local position, which simplified further analysis without the risk that we miss any move belonging to the optimal line of play.

Another CGT perspective is provided by the temperature theory. From this point of view, in a canonical form there are still many variations not worth considering. Many moves might theoretically be good in unusual circumstances, but in most cases will be worse than choosing another option.

Figure 3 shows a position in which Black's move at A in most games will be the correct one. It gains at least 7 points and leaves a continuation worth another 10 points. The move value is 12 points. On the other hand, Black's move at B is worth 8 points. It directly seizes 8 points but leaves no continuation.

The canonical form will retain variations starting with B. It is possible for this move to be correct, for example, if this is the only undecided position present on the board. However, this is unlikely in an actual game. Temperature theory provides a formalism that allows to tell that under normal cir-

cumstances the move at B does not need to be considered.

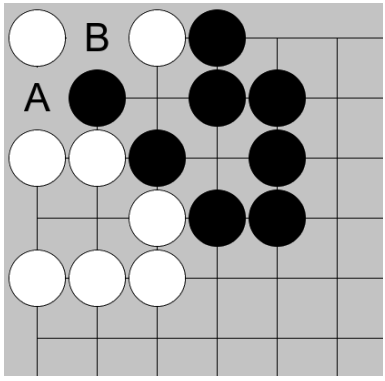

Figure 2. Black's move at B is always worse than A

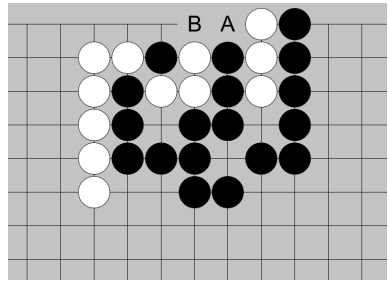

Figure 3. Black's move at B is worse than A in most situations

2. Forcing moves in light of CGT

In practice, temperature theory proves much more useful for Go endgames than analysis based on canonical forms. The reason is related to the notion of forcing moves.

In CGT, a move is called forcing if it raises the local temperature. In Go terms, one would say that a move is sente if its continuation is worth at least twice more than what the move gains for sure. An orthodox play strategy (Berlekamp 1996) advises to always answer, if the opponent has just played a forcing move. In other cases, one should play in the local position with the highest temperature.

From the temperature theory's point of view, one does not need to consid-

er variations that could happen if the opponent gets the continuation of their sente move. Such a situation will never happen under orthodox play. However, continuations of sente moves need to be retained in a canonical form. It might happen that the optimal line of play involves not answering to an opponent's sente move but e.g. playing one's own sente move in another part of the board.

In actual games, forcing moves are ubiquitous. This is because an efficient way of building territories most often comes with leaving little holes in one's walls. Building solid walls with all stones connected is usually too slow a way of development on the go board.

Figure 4 shows a position which might happen after a common joseki, in which Black stones surrounding the territory are not yet connected. White might try to enter Black's area with a move at A. If then White gets to continue with B, Black's territory will be destroyed. The threat is very big, so the move at A counts as sente. In most cases, Black is going to answer, consolidating the territory border.

Checking only two variations: the aforementioned one, and the one in which Black plays first, putting a stone at A, is enough for a successful thermographic analysis of the position. However, to find a canonical form, one needs to consider variations after White's continuation at B. As White enters Black's large territory, there are suddenly myriads of new variations to consider, and the whole analysis becomes infeasible. Noteworthy, on the presented example, there is also another undecided local position which is marked with crosses. If we start to consider variations after White's intrusion into Black's territory, we will not be allowed to neglect them when analyzing this other local position. It means that the whole area on the top side will need to be considered as one huge local position.

The example presented above shows that analysis leveraging canonical forms is impractical for positions taken from actual games. Because of holes which players tend to leave in their walls, board decomposition becomes impossible. Most often, the whole board would need to be considered as one local position, which completely defeats the purpose of deploying CGT, not allowing to anyhow simplify the problem of finding correct moves in the endgame.

In contrast, endgame analysis from the perspective of temperature theory allows decomposing the board into small local positions, even though their borders are not clearly marked. Most often, moves that could lead to a dramatic growth of variations number can be easily recognized by a Go player as sente.

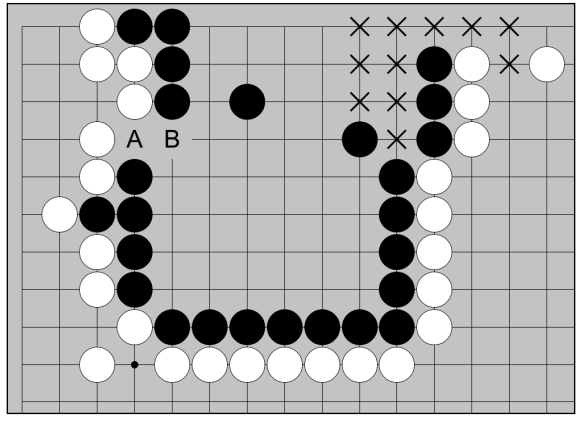

Figure 4. Two local positions which can be considered independent in the light of temperature theory but not from the perspective of canonical forms

3. Chosen approach

The goal of this work is designing a program that facilitates analysis grounded in temperature theory. My approach could be compared with the one used in Explorer. Just like Explorer, my program constructs a variation tree for a given local position. However, only moves relevant for temperature calculations are added to the tree. This makes the solution less computationally expensive, allows to analyze larger local positions, and eliminates the necessity of the position to be bounded by solid walls.

To find relevant moves in each tree node, I will use a neural network. Another network is going to assess local scores in terminal positions. The results will be then aggregated by an algorithm grounded in CGT.

The program's mode of operation will resemble an analysis performed by an experienced player. Player's intuition about which moves are worth considering will be mimicked by a neural network.

IV. Methods

1. Information to be predicted by the network

To allow construction of a local variation tree, the network needs to predict Black's and White's correct moves in the given position. In many cases, we will add both a move of Black and a move of White to the tree. However, if a forcing move has just been played, we should only add the opponent's answer to the tree. There are several ways in which a network can provide us with such information.

The approach which I took is making the network output a single tensor of 2 * 19 * 19 + 1 = 723 numbers. These numbers represent probabilities predicted by the network of each move being correct. The first 361 numbers represent probabilities of White's moves, the next 361 numbers - probabilities of Black's moves. At the end, I include one more number which represents the probability that the position is terminal, and no one will play there anymore.

This design choice stands in contrast with the output of AlphaZero network. In my case, 723 numbers are predicted, whereas in AlphaZero it was only 361 + 1 = 362 numbers (the last number representing pass). AlphaZero considers options only for the player at turn. Because of that, it is impossible to use AlphaZero's output to tell sente from gote. On the other hand, if my network predicts high probabilities only for one of the colors, it will mean that the last move was sente.

The probabilities should be predicted on the basis of not only the current board position, but also information of a few previous moves played in the current local position. Sometimes, whether a move should be treated as sente or not, depends on the whole local sequence. An example on Figure 5 shows a situation in which Black's move at 3 saves their six stones, while threatening to capture three White's stones. If we only take into

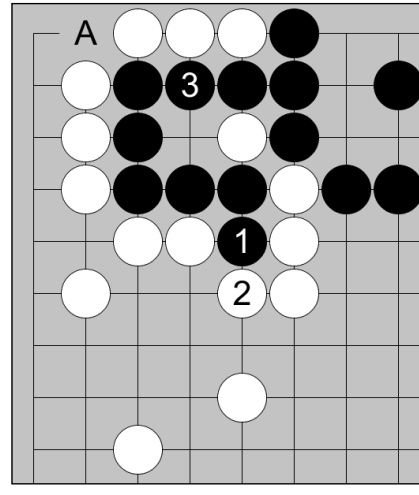

Figure 5. Whether Black's move at 3 shall be considered sente depends on the full local sequence

account Black's last move, it should be considered gote, as the continuation is significantly smaller than what the move gains. However, in the context of the previous 1 – 2 exchange, we realize that in fact, Black's move is sente.

2. Model architecture

Learning Go is a difficult task, not only for humans but also for machine learning models. AlphaZero network needs to be trained for many days on a strong computer to achieve a super-human level. As I did not have sufficient resources to train such a model myself, I needed to resort to a technique called fine-tuning.

Fine-tuning is a common technique in machine learning in which one takes a network which was trained for a long period of time on a huge data-set (probably by a rich company) and tweaks it for a specific task at hand. As the pre-trained network has seen a lot of data, it has been able to acquire a profound understanding of a given domain. It is therefore much cheaper to fine-tune to another task related to that domain than training a network from scratch.

I decided to make use of a pre-trained model based on an AlphaZero architecture found on GitHub (Nguyen, 2022). The model is a clean reimplementation of the solution described in the original paper, despite being smaller (having fewer residual blocks).

The model's input is eight most recent board positions encoded as 19x19 matrices of 1's, 0's and -1's. An additional, 9th matrix is filled with 1's if Black is at turn, and with -1's otherwise. Originally, in AlphaZero these should be the most recent whole-board positions, taken from the game. In my fine-tuned network, these are positions which appeared in a local se-

quence. The sequence I choose might have less than eight moves, in which case I replicate the earliest position in this sequence to match the required size of the input tensor.

The input is processed by a backbone consisting of five residual blocks, using 3x3 convolution and 128 hidden channels. In AlphaZero, the output of the last residual block is fed into two network heads which comprise of a 1x1 convolution, followed by one or two fully-connected layers. One head predicts best moves on the board, and the other - winning chances.

My model does not need these two heads, and so I replaced them with a new head which predicts correct local moves. The new head follows the architecture of the original policy head. As an input, the head accepts the output of the backbone's last residual block and a matrix representing an undecided local position. The matrix has 1's at cells corresponding to the local position, and 0's everywhere else. As the output, it yields 723 numbers summing to 1, which represent probabilities of Black and White moves being correct plays in the local position.

3. Training data construction

Preparation of training data for the network was the largest part of the project. There are no big datasets of local positions with marked correct moves. Creating such a dataset by hand would be too much work, given that neural networks require many thousands of examples for training. Luckily, preparing a good dataset is possible with publicly available tools.

I depict the data collection procedure on Figures from 6 to 9. As a source of endgame positions, I took self-play games of KataGo2). I navigated to a

2) They are available at https://katagoarchive.org/g170/selfplay/index.html

late stage of the game when roughly 80% of all moves were played. Then, I segmented a board into secure territories of Black's and White's and undecided local positions using KataGo network.

One of KataGo's outputs is ownership prediction for each intersection. Here, -1 denotes certainty of territory being White's, and 1 - of being Black's. I used thresholds of -0.9 and 0.9. Everything between these two numbers was interpreted as not full certainty and judged as an intersection of undecided ownership. Next, I grouped such intersections into connected components, and labeled each component as a distinct undecided local position.

Such a procedure does not guarantee that the detected local positions are independent of each other. A closer look at two pairs of local positions in the upper left quarter of the board on Figure 6 shows far distance dependencies between them. A move in one such position might make a move in another one sente. The presence of such situations in selected data introduces noise to training. However, one might expect that the effect will not be very strong, and should not bias the network too much towards any specific type of incorrect predictions.

Additionally, I check the ownership map for the final position in the game, and blacklist all intersections which in the endgame seemed to belong to a secure territory of one color, but ended up seized by the other color. This blacklisting serves later to avoid teaching the network about moves which most probably did not follow orthodox play and were played because of far-distant relationships on the board, such as ko fights.

Furthermore, I find a complete segmentation of the board into regions around the labeled local positions (and the blacklisted area of changing ownership). For every intersection, I calculate its distance to the nearest labeled

local position using the BFS algorithm.

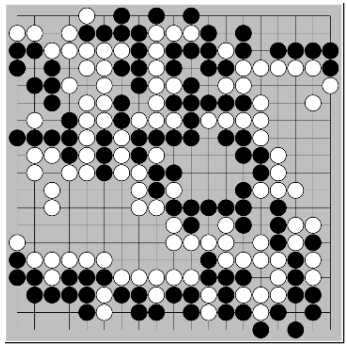

Figure 6. Whole-board endgame position taken from a KataGo self-play game

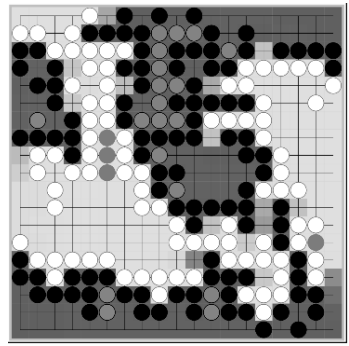

Figure 7. Ownership map predicted by KataGo

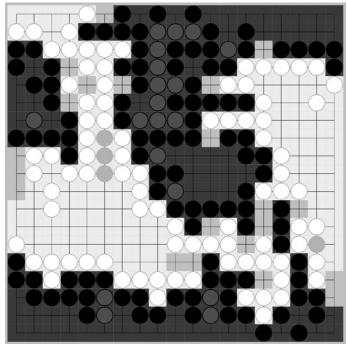

Figure 8. Discretized ownership map

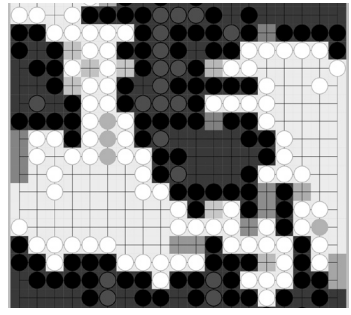

Figure 9. Connected components of undecided ownership labeled as different local positions

Next, I look at all moves starting from the chosen endgame position until the end of the game, and assign each move to the region of the complete segmentation in which it was played. This way, for every region I obtain a local sequence of moves. Commonly, all moves played within such a region

should be related to the corresponding labeled local position. In these sequences, moves are not necessarily played alternately by Black and White.

Finally, for each move of a local sequence, I set the positions which appeared in the sequence until this move as an input to the network and the next move (or lack thereof) - as the target. An example datapoint is presented on Figure 10. The position on the rest of the board looks quite random - I give an explanation for this in the next chapter.

Importantly, such construction of training data should help the network learn about sente moves. In case the previous move in the local sequence was sente, the target to the network would almost always be the opponent's answer to that move. It might seem worrisome that in case of gote moves also only a move of one color will be presented as a target. However, in such situations, the network has no way to guess which player got the next move, so to minimize the loss function the network will try to predict roughly 50% probability for a move of Black, and 50% for a move of White.

In my master's thesis (Frejlak 2024), I trained a neural network using a bit different training data. Most important difference is that in the original approach, I looked for only one future move for a given local position, and not for a full local variation played out in the endgame. This way, I was not able to present the network with a local context of a move except for the single local position in which the previous move was played.

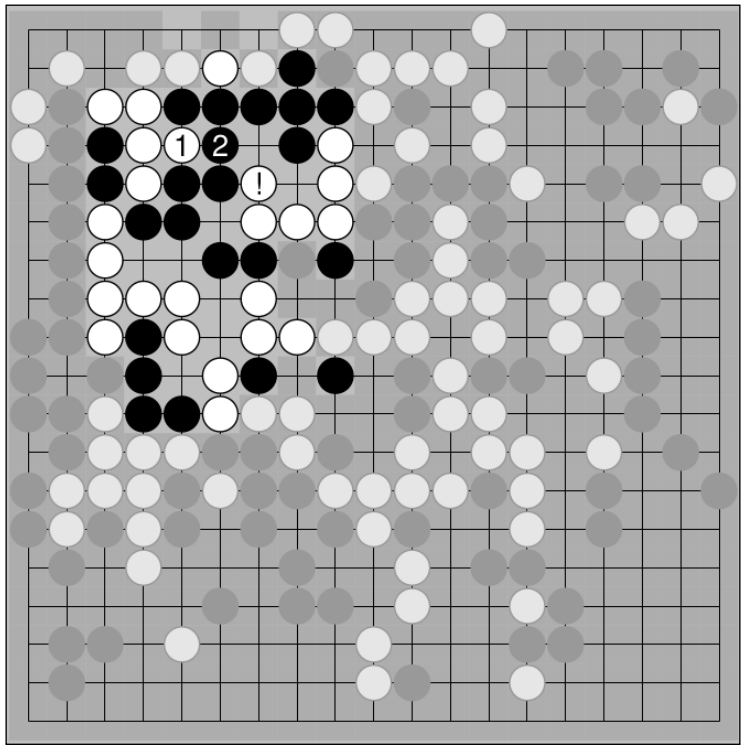

Figure 10. Example training datapoint. Whole-board position with a local sequence of length 2 is presented to the network. The move with an exclamation mark is the target.

4. Data augmentation and sampling

Data augmentation is a common procedure in deep learning aimed at making the model more robust. Oftentimes, a training dataset might be biased in one way or another. For example, a dataset of photos might consist mostly of photos which were shot straight, without rotating the camera. Training a network on such a dataset might make the network perform poorly on tilted photos. Therefore, a common augmentation technique is intro-

ducing random rotations to the input pictures.

Experimenting with my network, I also noticed that it becomes biased toward certain properties of its inputs. I initially trained the network using whole-board endgame positions, and the network ended up giving strange results for a local position laid on an empty board. To bypass this problem, during training I randomly remove stones from the go board outside of the current region of interest. I randomly sample the proportion of stones which should be removed, and then multiply the input matrices by a binary mask sampled randomly according to that proportion.

I apply augmentation also for the local position masks. I randomly choose how broad the neighborhood of the undecided local position should be included in the mask, and randomly exclude some of the intersections on the mask's border to introduce some irregularity in data.

Finally, I calculate some statistics for the created dataset. I noticed that most of the undecided positions are small, with only 1, 2, or 3 intersections. Also, in most positions the local sequences are short. Many of these positions are quite uninteresting having e.g. only one neutral point to be taken by one of the colors. To not flood my network with such simple tasks, I randomly remove from the dataset a chosen proportion of positions of small size and sequence lengths.

5. Training procedure

I train the network on a dataset of 4.7 million endgame positions. The training ran for around 10 epochs which took around 2 days on a laptop with NVIDIA RTX3080 graphic card. I start with the learning rate of 1e-3 and half it every 10.000 steps. The batch size is 256, so one step corresponds

to 256 endgame positions. I reset the training several times, going back to a higher learning rate.

As a loss function, I use a cross entropy applied to predicted move probabilities. Preliminary results showed me that the network struggles when it comes to predicting that the position is terminal. Therefore, in positions where no move should be predicted, I introduce an additional penalty for predicting anything else than no move in such positions. I sum up softmaxed predictions for all moves, multiply this term by parameter lambda which I set to 6, and add it to the cross-entropy loss.

An important change compared to my master's thesis is not penalizing the network for predicting moves outside of the masked region. I found out that the network anyway quickly learns to play moves in the given local position. Moreover, this way I acquire more freedom in choosing a mask of a local position. The mask could be as small as one intersection and still the network realizes which local position this intersection points to. It makes the program more convenient to use, as the user does not need to necessarily mark all intersections which might change ownership.

6. Calculating temperature

Having a model which predicts probabilities for local moves, I calculate a temperature of a given local position in the following way.

First, a variation tree is built. The root node corresponds to the initial endgame position. Then, for each tree node, child nodes are added based on predicted probabilities:

- If the probability of no move exceeds 30%, no child nodes are added.
- Otherwise, if the sum of probabilities for *moves* of one color exceeds

85%, it is judged that we are dealing with sente, and only moves for that color are added.

- Otherwise, moves for both colors are added.
- For a given color, moves are added if their probability exceeds 30% of the total probability for that color. If no move's probability exceeds this threshold, then a single move with the highest probability is added.
- As an exception, in the root node always moves for both colors are added, no matter the relative sum probabilities of Black's and White's moves.

Thresholds used during the tree expansion were chosen a posteriori for a trained neural network.

With this tree expansion strategy, the program should find all variations following optimal play, provided that the model's predictions are close enough to perfect.

The nodes with no children are judged as terminal, and the final local result in them is assessed using KataGo's ownership map. The ownership predictions for each intersection are rounded to integers: -1 for White, 0 for no ownership, e.g. in seki, 1 for Black. These integers are summed within the region marked as the local position, yielding a local score under area scoring.

Having evaluated all leaf nodes in the tree, the results are aggregated, using an algorithm developed by Łukasz Lew (Lew and Frejlak, 2024). One caveat is that the algorithm requires each node of a tree to either be terminal, or to have children of both colors. A question arises what child node should be added to the tree, in case when the network predicts that almost certainly

the next move will be played by a specific color. Since initially I expected such a situation to appear precisely when the previous move played was sente, a natural idea was to add an artificial child for a continuation of the sente move, with an evaluation very much favoring the player who played that move. This way, we would not consider any additional variation on a Go board.

However, this approach fails because the high probability of a specific color getting the next move is predicted by the network also in case the next move is going to be sente for one of the players. I concluded that there is no clean way to establish what artificial child should be added to the tree, as it is impossible to distinguish the situation of the next move being sente and the previous move being sente by mere looking at the network predictions.

This led me to another solution. Apart from using thresholds for a usual tree expansion, I also ensure that for every position, at least one Black's and one White's move is considered. However, in case the probability of a given color getting the next move is below the chosen threshold, I treat a node after such a move as terminal and do not consider any further variations after it. Instead, I evaluate the expected local score in that node using the KataGo ownership map. This solution is not ideal but works in most cases. I offer further discussion on this issue in Section VI.

V. Results

While the neural network is trained on a large dataset to predict next local moves, a real test for my program is how it performs on a harder task: expanding a variation tree to calculate the local temperature. To yield the cor-

rect results, the neural network predictions need to be quite accurate for every position appearing in every local variation. I tested the program on a set of actual endgame problems, designed for players learning endgame theory. To my best knowledge, my program is the first one in the world designed in a way which allows for solving such tasks.

I took the test set of 100 problems from the GoMagic course called “Endgame for Nerds” (Frejlak 2022). The problems are of varying difficulty, featuring a lot of common endgame positions and guiding the student through concepts such as moves with continuation, sente and ko.

My program finds the correct local temperature in 65 out of 100 problems.

1. Comparison with baseline

To date, there are no other programs which could be tested on the same problem set. To still somehow quantify how good the achieved results are, I design two other approaches.

The first approach does not use the network I trained, and deploys the KataGo network instead. As explained before, KataGo is not designed for this specific task, but still one can try to get the most out of it by looking for moves with highest probability within a local region of the board. There is also a natural way of telling a position terminal. In case KataGo predicts every intersection to be either Black’s or White’s with high certainty, territory borders are most probably fixed, and the tree expansion can stop.

As KataGo does not predict move probabilities for both colors in a joint manner, there is no easy way to tell that certain moves are sente. As explained in Section III. this leads to a dramatic growth of the tree size, in

case the position was not clearly bounded by walls of stones from the very beginning. To ensure that calculations end in real time, I limit the length of considered variations to 12. Positions after this many moves are treated as terminal and evaluated using the ownership map. Then, the local scores are aggregated using the same algorithm as in my original program.

With this approach, the program correctly solves 14 out of 100 problems.

For the sake of another comparison, I test one more approach. In each of the problems, I put by hand all correct local variations. This way, I am simulating how an ideal trained neural network should guide the construction of the variation tree. Local scores in terminal positions are calculated from KataGo's ownership maps, and aggregated with the same algorithm grounded in CGT. Correct answers are obtained for 97 out of 100 problems.

This experiment shows that the program design is not flawless. Even if the neural network worked perfectly, the program would still fail to always find the correct local temperature. On the other hand, 97% accuracy is very high, showing that there is still a lot of room for the network to grow.

Mistakes in the three problems come from two different sources. In case of one problem KataGo's evaluation for a non-terminal position is at fault. I evaluate a non-terminal position if the network assigned a very low probability to moves of one color. In case this was a sente move, the continuation of that move should lead to a position significantly better for the player - this should be then reflected in KataGo's evaluation, and lead to a correct calculation of the temperature-finding algorithm, which will disregard the node corresponding to that move. However, in case of the problem shown on Figure 11, not only White's move at 1 is sente but also their continuation is sente - with a much larger continuation as it threatens to kill Black's corner. This leads KataGo network to predict that Black will surely respond to

White's 2, and this certainty is also reflected in its ownership map, yielding too high local score for Black, which hinders correct temperature calculations. While this situation appears only in one of the 100 problems, it shows an inherent limitation to the approach I chose in my program.

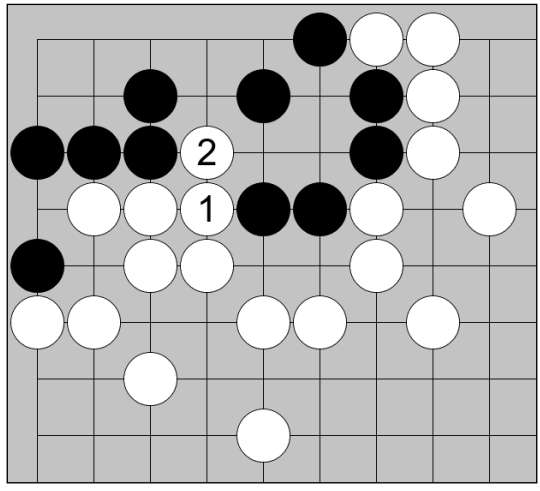

Figure 11. Failure of the procedure of evaluating continuations of sente moves.

Two more mistakes come from incorrect assessments of final scores in terminal positions. Usually, KataGo's assessments of intersection ownerships for terminal positions are highly accurate. However, in the presented problems, secure territories are not marked very clearly. Apparently, KataGo sometimes still sees a weakness in a player's shape and does not judge some intersection as secure territory, subject to chosen threshold.

2. Qualitative analysis

I inspected variation trees built by my program both in problems which it solved correctly and in which it made mistakes. The program learned very well to recognize sente moves. Also, the presence of various types of kos in the problems did not pose difficulties to the program3).

Many of the program's mistakes come from the network not knowing certain endgame techniques. For example, in the problem shown on Figure 12, the network does not realize that White's best move is a monkey jump. Only variations after White's turn and White's knight's move are added to the tree, and consequently the local temperature is estimated as 7 points, and not 9 points which would be the correct answer.

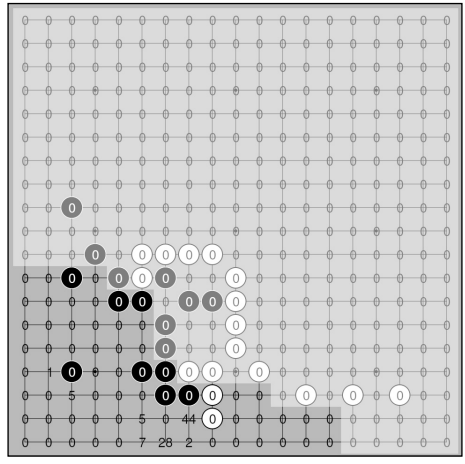

Figure 12. Predictions for best local moves of White in percentages. The network assigns only a 7% probability to the monkey jump.

3) In CGT analysis, kos are generally quite problematic. There are lots of unusual types of kos, some of which are difficult to mathematically formalize. However, most kos appearing in practice in Go endgames, fall into the category of placid kos, which can be successfully analyzed using classical CGT tools.

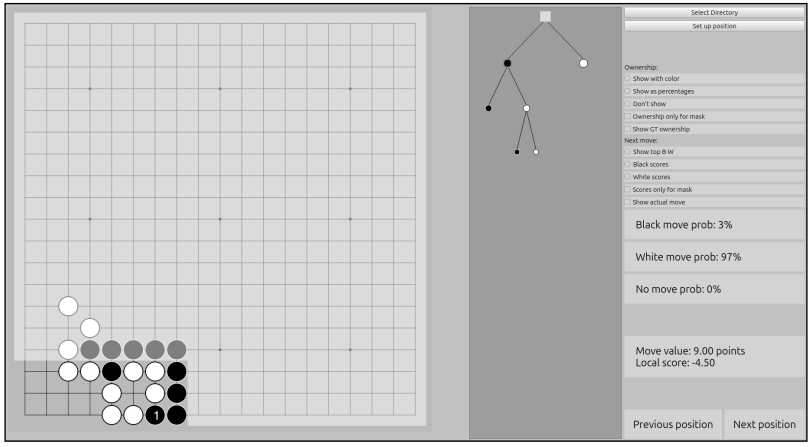

Figure 13. A position in which the network for an unknown reason predicts much higher probability for White's moves. (Screenshot from a GUI I developed for my program)

Another type of mistakes appears in situations where no more moves of value higher than 0 points are left, but the position still needs to be fixed by playing moves threatening a large continuation. An example position is shown on Figure 13. In such positions the network should ideally predict 50% chances of playing for each of the colors. However, I observe that in practice the network predicts a much higher probability for the player who can threaten saving their dead stones. In practice, the program treats the position as if it contained a sente move, expanding the variation after the other player's move only to the depth of 1. This sometimes leads to incorrect calculations, if the KataGo's evaluation of a non-terminal position is inaccurate.

It is difficult for me to understand why the network learned to assess such a high probability for one of the colors. Intuitively, I would rather expect that in the training data, analogous positions should have both continuations by Black and by White.

VI. Future work

There are several areas in which one could seek to improve my program. First of all, taking a stronger pre-trained model for fine-tuning might lead to better performance. The network which I used was pre-trained only on a 9x9 board. In consequence, it might have not learned about some common tesujis met in 19x19 endgames, and so it might be difficult to master them during fine-tuning. There are multiple open-source programs which could be used, such as LeelaZero, ELF OpenGo, or KataGo.

Another direction might be trying to improve the quality of the training data. Visual inspection of training data currently fed into the network leads me to suspicion, that the proportion of interesting endgame positions, including tesujis such as a monkey jump, is too low, with the majority of positions featuring quite obvious play such as 1-point moves or filling neutral points. I tried to overcome this issue by keeping only a small proportion of positions in which the number of undecided intersections or the number of moves left in the local variation was low. However, one could still think of many other heuristics for detecting which training examples might be more educative for the network. For example, one could try to estimate the local temperatures in sampled positions, e.g. reusing the already trained network (but probably without expanding a variation tree too much, as it is computationally expensive).

Finally, one can try to alter the architectural choices taken in this work. One idea which is certainly worth trying is teaching the network the distinction between sente moves and answers to sente moves. In both of these

situations a probability of getting the next move should be higher for one of the colors. However, as discussed in Section IV, it is problematic for the temperature calculation algorithm if these two cases are not distinguished by the network. To tackle this problem, one could make the network predict another piece of information, namely whether the next local move is likely to be played immediately after the previous one. This should hold for answers to sente moves, but generally should not hold for sente moves themselves.

Having such additional information about sente moves predicted by the network, one could simplify the tree expansion process. If the next move is judged to be an answer to a sente move, then one could add an artificial node to the variation tree with evaluation strongly favoring the player who has just played. There is no need to consider which specific move is the correct continuation and get a KataGo evaluation for it, despite the position not being terminal. On the other hand, if the next move is judged to be sente, then there is no harm in normally expanding the tree for the opponent's reverse sente move. The tree will not grow too much as we should not run into the trap of entering a secure territory described in Section III. Instead, such an expansion can help to confirm or refute the initial assumption of the move being a reverse sente.

Conclusion

The solution developed in my master's thesis and improved for this work is the first program in the world that facilitates CGT analysis of local endgame positions taken from real games. The program correctly estimates the local

temperature in 65 out of 100 endgame problems designed for Go practitioners. While this number is much higher than what one could get by using existing Go-playing programs such as KataGo, there is still a lot of space for improvement. Importantly, the program should still be improved before it can be used as a teaching tool for Go players.

References

Berlekamp, E. (1996) The economist's view of combinatorial games, Games of No Chance: 365-405

Berlekamp, E. and Wolfe, D. (1994) Mathematical Go: Chilling Gets the Last Point.

Conway, J. (1976) On numbers and games

Frejlak, S. (2020) Katago's yose: Go endgame theory and deep residual networks, https://github.com/siasio/EndgameBot/blob/main/1000-LIC-MAT-297313.pdf

Frejlak, S. (2022) Video course "endgame for nerds", Go Magic, https://gomagic.org/courses/endgame-for-nerds/

Frejlak, S. (2024) Deep learning and combinatorial game theory; evaluating temperatures in the game of go, https://github.com/siasio/EndgameBot/blob/main/Frejlak-masters_thesis-final.pdf

Müller, M. (1995) Computer go as a sum of local games: an application of combinatorial game theory

Nguyen, T. (2022) Alphazero in jax using deepmind mctx library, https://github.com/NTT123/a0-jax

Lew, Ł. and Frejlak, S. (2024) Finding temperatures through iterative pseudo-stop search

Siegel, A. (2013) Combinatorial Game Theory

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. et al. (2017) Mastering the game of go without human knowledge, nature 550(7676): 354–359

Törmänen, A. (2019), Rational Endgame

Wolfe, D. (2002) Go endgames are pspace-hard

Wu, D. (2020) Accelerating self-play learning in go

Received: 15, November, 2024
Accepted: 20, November, 2024