University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Stanisław Frejlak
Student no. 371283

Deep learning and combinatorial game theory; evaluating temperatures in the game of go

Master's thesis
in MACHINE LEARNING

Supervisor:
Dr. Jacek Cyranka
Dr. Francisco Criado Gallart

Warsaw, July 2024

Abstract

In the game of Go, the endgame phase is best described formally using Combinatorial Game Theory. During this phase, the board can be decomposed into a sum of local positions, which can be analyzed separately using a method called thermography. Master-level Go programs, based on the AlphaGoZero algorithm [SSS+17], have become invaluable tools for players, providing optimal move suggestions. However, their game analysis does not rely on board decomposition. They cannot estimate temperatures of local positions, which would be a useful and understandable insight for players.

In my work, I developed a program to estimate temperatures in local endgame positions. The program constructs a partial game tree, evaluates scores at terminal nodes, and integrates these evaluations using an algorithm grounded in Combinatorial Game Theory. Tree expansion is powered by a deep neural network. I created the network by fine-tuning a pre-trained AlphaZero-based model on a task of predicting optimal local moves for both players. Prior research [Mül95] only allowed for temperature calculation in relatively small, fully enclosed positions. In contrast, my program achieves 55% accuracy on a set of real endgame problems, marking a significant advancement in the field.

Keywords

AlphaZero, Combinatorial Game Theory, Go, Temperature, Fine-tuning

Thesis domain (Socrates-Erasmus subject area codes)

11.4 Artificial Intelligence

Subject classification

I. Computing Methodologies
I.2. Artificial Intelligence
I.2.8. Problem Solving, Control Methods, and Search

Tytuł pracy w języku polskim

Uczenie głębokie i kombinatoryczna teoria gier; szacowanie temperatur w grze go

Contents

Introduction

Go is one of the oldest logical games in the world. Around fifty years ago, the endgame phase of the game of Go became an inspiration for mathematicians to develop a new branch of mathematics, the Combinatorial Game Theory [Con76, Prologue]. Findings of the theory brought astonishing results to Go, as well as other types of games.

A central notion of the theory when applied to Go is *temperature*, a measure of urgency of playing in a given part of the board. Even before the CGT was invented, Go players had been making decisions in endgame based on temperature calculations. The novelty introduced by mathematicians were theorems about relationship between perfect play and play based on temperature values, as well as simple algorithms for certain types of positions which proved more effective than the traditional way.

The game of Go played also an important role in the development of artificial intelligence. Research connected to Go drove such developments in the field of reinforcement learning as the Monte Carlo Tree Search [Cou07]. Eventually, use of deep neural networks led to designing a program which could beat top Go players [SHM+16].

In 2017, AlphaZero [SSS+17] provided a general purpose reinforcement learning algorithm which nowadays serves as a foundation for all state-of-the-art programs in computer Go, as well as in other logical games. The AlphaZero-based programs choose moves in a way which mimics decision process of human experts, leveraging intuition about reasonable moves to decide which variations to consider.

However, in the endgame phase that general move selection method deviates from the analysis performed by a human expert, which follows methodology described by the Combinatorial Game Theory. It could be viewed as a weakness of the AlphaZero approach.

The move selection method used by AlphaZero fails to capture an important feature of endgame phase, a possibility to decompose the board into a sum of independent local positions. In contrast, analysis based on Combinatorial Game Theory offers a dramatic reduction of the search space, facilitating more accurate distinctions which could be achieved with less computation.

In the current work, I present a program which emulates the CGT-based analysis of endgame positions. It sets a new direction in deep learning based research related to Go. A potential benefit of this new approach is a higher explainability of the model's decision. With a program evaluating local temperatures, players could understand better why certain moves are better than other. On the contrary, AlphaZero-based programs are of limited use for players looking for hints about endgame[1].

[1] As shown by the analysis [TMG+19] performed by the authors of ELF OpenGo, a significant growth in go players level has been observed in the opening phase since the introduction of AlphaZero. On the other hand, level in middle game and in endgame phases has remained similar.

Related work

To date, there has been no effort to deploy deep learning in the context of the Combinatorial Game Theory. However, two works are especially relevant for the current thesis.

In 1995, Martin Müller [Mül95] leveraged the Combinatorial Game Theory to create an exact solver of go endgames. As he showed later in [Mü01], use of the CGT tools facilitated an analysis of positions more complex by orders of magnitudes than positions that could be analyzed with the classical approach (the minimax search with alpha-beta pruning). However, the computational complexity of the proposed algorithm was still prohibitively large when applied to most real-game scenarios.

On a different note, in my bachelor these [Fre20], I investigated whether the AlphaZero-based programs could be used for estimating temperatures. To this goal, I utilized KataGo, a program providing a richer information about board positions than vanilla AlphaZero. The investigation showed that the output of KataGo network roughly approximates assessments of the Combinatorial Game Theory. However, the approximations are not accurate enough to provide good temperature estimates. In particular, such temperature estimates could not be used to make fine distinctions between urgency of local positions typically met in endgames.

Main contributions

The program developed in this work evaluates temperatures based on neural network predictions and the algorithm [LF24] for performing CGT calculations based on a game tree. The task of the neural network is to predict best local moves for both colors and find the local score in terminal positions. I developed a pipeline for creating training data for such a network using publicly available resources.

Moreover, I curated a test set of 100 endgame problems for evaluating performance of a program estimating local temperatures. The program developed by me obtains 55% accuracy on the test set.

Work's structure

In the first three chapters, I explain preliminaries of the current work. The first chapter presents the game of Go, and elaborates on the two branches of research connected to it. The second chapter explains the basic concepts of the Combinatorial Game Theory and shows how they apply to Go endgames. In the third chapter, I review the advances in deep-learning-based computer Go.

The fourth chapter comments on the previous research on the thesis topic. It presents Martin Müller's computer implementation of the CGT algorithms and an experiment with KataGo which I performed for my bachelor thesis. The fifth chapter presents the program that I developed, and evaluates its performance on the test set. The sixth chapter offers a discussion about the choices taken while designing the program and potential enhancements that could be a subject of the future work.

Chapter 1

Game of Go

This chapter starts with a brief explanation of the rules of go. It follows with the characteristics of the game which made it a difficult task for artificial intelligence researchers. The last section talks about the endgame phase of Go which has become an inspiration for inventing the Combinatorial Game Theory.

1.1. Rules of Go

The game of Go is played by two players on a square board with black and white stones. In tournaments, the board of size 19×19 is used but friendly matches are played also on boards of other sizes. At the beginning of the game, the board is empty. Then, the two players take turns to put stones at the board, one stone per turn. The player holding black stones is commonly referred to as *Black* and the other player as *White*. Black plays first. The board is a grid of vertical and horizontal lines, and stones are put on their intersections, as shown on Figure 1.1. The goal of the game is to take control over a bigger part of the board than the opponent. The game finishes when the borders between territories controlled by the players have been decided.

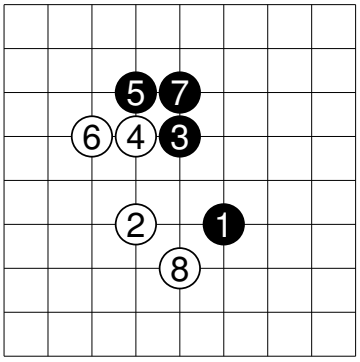


Figure 1.1: Example of first moves in a game.

In go, it is possible to capture opponent's stones and take them off of the board.

Definition 1.1.1. *If stones of the same color lay on neighboring intersections, they belong to a connected* chain of stones.

Definition 1.1.2. *Empty intersections neighboring a chain of stones are called* liberties *of that chain.*

Rule 1.1.1. ***(Capturing rule)*** *To capture an opponent's chain of stones, one needs to occupy all of its liberties with one's own stones.*

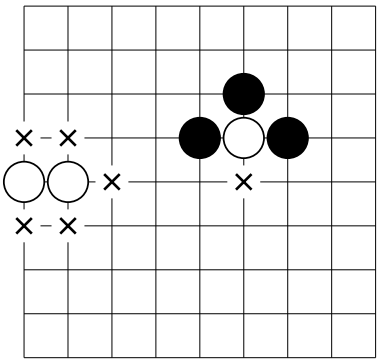


Figure 1.2: Liberties of a stone and a chain of stones.

Examples

On Figure 1.2 White's chain of two stones has five liberties.
On the other hand, White's single stone has only one liberty left so Black can capture it in the next move.

Generally, a player can play a move at any empty intersection of the board. However, there are two types of forbidden moves.

Rule 1.1.2. ***(Suicide ban)*** *It is forbidden to put a stone on the board, if after the move that stone (or the chain to which it belongs) would have no liberties.*

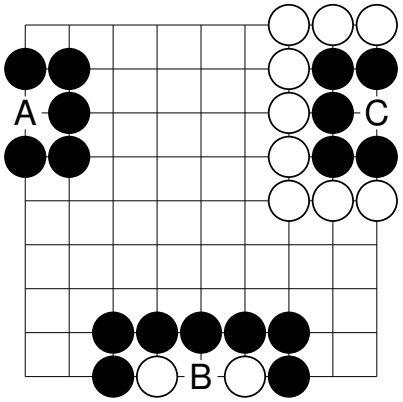


Figure 1.3: White moves A and B are suicidal, C is not.

Examples

Figure 1.3 shows an example of two suicidal moves (A, B) and one move which is not suicidal (C). The move at C takes away the last liberty of Black chain of stones. These stones will be now taken off of the board, and the White stone will be left with three liberties.

After a stone got captured, the intersection under it gets freed. Later, a player can decide to put another stone on the same place. This could potentially lead to a loopy game which would never finish. Such a situation is called *ko*.

Rule 1.1.3. ***(Ko ban)*** *It is forbidden to play a move which would lead to a repetition of the board position which already appeared earlier in the game.*

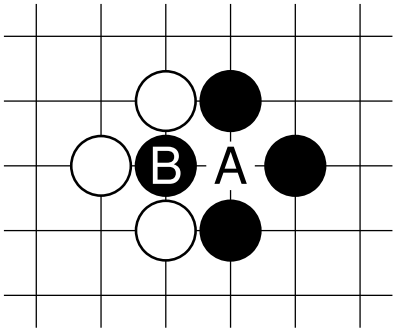


Figure 1.4: A simple example of ko.

Examples

Figure 1.4 presents the most common type of ko situation where the potential loop has a length of two moves. If White plays at A, Black stone at B gets captured and is taken off of the board. Then, Black's move at B could capture White stone at A. However, this is forbidden as it leads to a board position repetition.

It is important to mention that there exist several rulesets of Go. In different rulesets, the exact formulation of the *suicide ban* and the *ko ban* differ, making distinctions between specific types of positions that happen rarely in real games.

The end of the game

As the players continue to build their territories, at one point the territories' borders become precisely marked. Such a moment is depicted on Figure 1.5. There are still legal moves available on the board, but none of them brings any profit. Due to the capturing rule, if a player starts putting stones inside the opponent's area, the opponent will manage to capture them within a few moves. Moreover, if an opponent's stone lies within a player's area, the

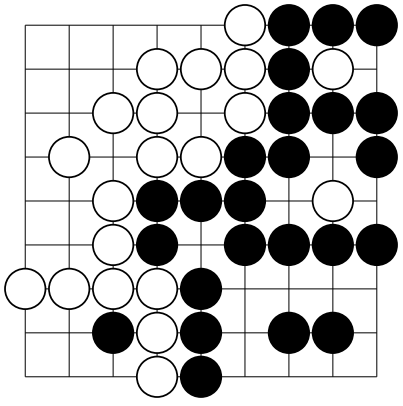


Figure 1.5: The end of the game.

player is not obliged to capture it before the end of the game. It is enough for both players to acknowledge that it can be captured to claim the stone *dead*.

Judging whether the opponent is able to capture stones put into their territory is a question related to the Go tactics. A beginner player is often not sure whether it is possible or not, and cannot tell with certainty that the time has come to finish the game. In such case, they would usually play a few additional unnecessary moves which would not affect the game's result[1].

If a player judges the game as finished, they *pass* instead of putting a stone on the board. Formally, a *pass* is allowed at any moment of the game. In practice, it is reasonable to pass only at the very end of the game. If two players pass one after another, the game finishes, and the players proceed to count the final score. There are two ways of counting the score, adopted in different rulesets.

In both scoring techniques, the final score of the game is defined as a difference between Black's score and White's score. The final score is usually written in the form of $B + 5$ (meaning that Black won by five points) or $W + 1$ (meaning that White won by one point).

Before counting the scores, *dead* stones are taken off of the board.

Rule 1.1.4.
(Area scoring) *For player $X \in \{B, W\}$ (Black or White), their score $\mathcal{A}_X$ is counted as the number E_X of empty intersections surrounded by the player's living stones plus the number L_X of the player's living stones. The final score is $\mathcal{A} = \mathcal{A}_B - \mathcal{A}_W = (E_B + L_B) - (E_W + L_W)$.*
(Territory scoring) *For player $X \in \{B, W\}$, their score $\mathcal{T}_X$ is the number E_X of empty intersections surrounded by the player minus the number of the player's captured and dead stones D_X. The final score is $\mathcal{T} = \mathcal{T}_B - \mathcal{T}_W = (E_B - D_B) - (E_W - D_W)$*

As the players put stones on the board one by one, at the end of the game, the sum of living, dead and captured stones $(L_X + D_X)$ is the same for each player, or it is bigger by 1 for Black if Black played the last move on the board. Therefore, the scores counted by two territory and area scoring generally differ at most by 1 point, $\mathcal{A} = \mathcal{T}$ or $\mathcal{A} = \mathcal{T} + 1$.

In fact, there are possible configurations of stones under which some empty intersections cannot be seized by either of the players at the end of the game. Different rulesets handle such situations in different ways but it does not affect much the considerations of this thesis.

There is one more rule connected to scoring. As Black plays first, they have better chances of winning than White. To make the game balanced, the *komi rule* has been introduced.

Rule 1.1.5. ***(Komi rule)*** *To get the final result of the game, White's score is increased by a fixed* komi *value.*

Commonly, the *komi* value is set to a non-integer number, so that the final score cannot be 0, i.e. the game does not finish with a draw. Nowadays, most widespread rulesets adopt a value of $6,5$ or $7,5$ points.

1.2. Hurdles of Computer Go

The rules of go are very simple. Unlike in chess, all pieces are of the same type. There is a uniform way of making a move. However, there are myriads of possible arrangements of the stones on the board.

[1] In fact, under territory scoring unnecessary moves played when the game could be already finished incur a slight loss for the player.

One of distinct characteristics of go among logical games is a large branching factor. At each moment of the game there is a couple of hundreds allowed moves as there are 361 intersections on a go board. Very often there are a few moves which a human expert would judge as correct. In consequence, already in the opening stage of the game, one could rarely see identical board positions in different games.

For a human player, Go combines strategy and tactics. Both of these require a player to imagine various possible lines of play from a given position which is commonly referred to as *reading variations*. A decision about the next move is motivated by a judgment about the board positions which could be achieved when choosing different moves. Such judgment is mostly based on a player's intuition. Moreover, when Go players read variations, they consider only a very restricted set of possible plays at each moment. Most of available moves just *do not feel right* for a player. Choice of the moves worth considering are to a big extent guided by a sense of aesthetics.

With all the aforementioned features Go posed a big challenge for artificial intelligence researchers. Early tries to create Go playing programs leveraged knowledge bases including common opening patterns known as *joseki*, and local arrangements of stones referred to as *good* and *bad shapes*. The programmers tried to encapsulate a player's intuition used for position judgment in a set of mathematical formulas. However, with all this effort, at the end of 20^{th} century, when in chess there already appeared an engine capable of beating top human players, in Go the strongest programs were still at the beginner's level.

The game of Go continued to stimulate research, bringing to a development of the Monte Carlo Tree Search algorithm [Cou07], and eventually, to the creation of AlphaGo [SHM+16] and AlphaZero [SSS+17] programs. AlphaGo was hailed as a big breakthrough in the field of machine learning, and earned a front page in the *Nature* magazine.

It is worth to mention one feature which exhibits the game's complexity. As explained previously, a mere judgment that the game has come to finish requires some experience from a player. Proving with an algorithm that any stone put into opponent's territory would end up *dead* is a very challenging task. In consequence, a Go playing program might continue to play moves long after the moment when a human player would judge the game as finished. This difficulty in automatic assessment of secure territories bears a special significance for the current work. As we will see in Section 4.1, it set serious restrictions to programs assessing temperatures in Go endgames.

1.3. Mathematical analysis of endgame positions

Conventionally, a game of Go is said to consist of three parts:

- Opening, when the players initially mark their positions, and play out common local sequences called *joseki*
- Middle game with fights between groups of stones, strategic plans, and a lot of global thinking
- Endgame, when majority of the board belongs to secure territories, and the players would only struggle to move the borders in their favor within undecided regions of limited size

Go endgames are especially interesting from a mathematical point of view. At that time, the decision making process greatly differs from the earlier stages of the game. Even though the final goal of the game is to maximize the score, in opening and middle game players would

rarely use precise mathematical calculations to decide about their moves. The board is too large, and there are too many possible variations until the end of the game. However, in the endgame stage, a mathematical analysis becomes possible.

In endgame, parts of the board belonging to secure territories can be excluded from analysis. The rest of the board consists of many local positions which are unrelated to each other. Each of such positions can be analyzed on its own. It is often viable to *read* all possible local variations (or at least all consisting of moves which a player finds reasonable), and draw a conclusion how many points can be gained by playing in a given local position.

Separate analysis of all local positions proves much easier than analyzing the whole board at once. It can be compared with a well-know programming paradigm called *divide and conquer*. However, when a player successfully performs the local analyses, a task still remains to combine these results to reach a global conclusion about the best move. Go players usually approach this problem by finding a single number for each of the positions, called a *value of move*. Choosing the move with the biggest value is a heuristic which not always results in optimal play. Thanks to a mathematical research it turned out that the exact solution to this problem is very difficult.

Chapter 2

Combinatorial Game Theory

In the Combinatorial Game Theory, a *game* is understood in an abstract way. The theory's axiom system resembles the axiom system of the set theory. Under these axioms, the notion of a *number* naturally emerges - integer numbers, as well as all real numbers, turn out to be types of games.

As abstract as it sounds, the theory can be directly applied to Go endgames. Numbers defined by it correspond to a player's score. An addition operation defined on games correspond to the possibility of decomposing a whole-board position into a sum of local positions. Eventually, powerful tools provided by the theory could be used to enrich the classical way in which Go players analyze positions.

Combinatorial Game Theory is a highly technical area of study. This chapter merely sketches its main findings. Interested readers should refer to [Sie13] for a detailed lecture on the topic.

2.1. The notion of game

Definition 2.1.1. *A game G is an ordered pair of sets of games: $\mathcal{L}_G$ (as in* Left*) and $\mathcal{R}_G$ (as in* Right*). We write: $G \equiv \{\mathcal{L}_G|\mathcal{R}_G\}$.*

Definition 2.1.2. *Elements of sets $\mathcal{L}_G = \{G^{L_1}, G^{L_2}, \dots\}$ and $\mathcal{R}_G = \{G^{R_1}, G^{R_2}, \dots\}$ are called* left options *and* right options*, respectively.*

We write: $G \equiv \{\mathcal{L}_G|\mathcal{R}_G\} \equiv \{G^{L_1}, G^{L_2}, \dots|G^{R_1}, G^{R_2}, \dots\}$, skipping the inside curly brackets for brevity. Similarly as in the set theory, games in the Combinatorial Game Theory are defined recursively, starting from an empty set.

Examples

- Game with no left and right options is denoted as $0 :\equiv \{|\}$.
- Games denoted as integers are defined recursively: $\forall_{n\in\mathbb{N}}\ n+1 :\equiv \{n|\}$, $-n-1 :\equiv \{|-n\}$.
- Another simple game is *STAR*, defined as $* :\equiv \{0|0\}$.

One could see that positive integers are games with no right options, and negative integers are games with no left options.

In this work, we will consider only finite games unless specified. The theory also considers games with infinite number of options or games which might be not finished in a finite number

of moves, e.g. $G = \{1, 2, 3, \dots |\}$. It is also possible to define loopy games, such as $G = \{|G\}$. It turns out that loopy games are relevant for Go endgames, so we will come back to this topic in Section 2.5.

At first, it might be not clear what so-defined objects have to do with games as we know them in the real world, and why some of them should be identified as numbers. The following definitions provide a motivation for this nomenclature.

Playing games

A game can be played by two players, called *Left* and *Right*, taking turns one after another. A player's turn consists of moving from the current game to one of the player's options. A player loses when they have no option which to move into.

Examples

- In game 0 whoever starts, loses the game, as neither Left nor Right has any options.
- In game $*$ whoever starts, wins the game, as they move to game 0, where the opponent is left with no options.
- In positive integers $1, 2, \dots$ Left wins, no matter who plays first.
- In negative integers Right always wins.

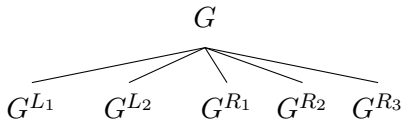


Figure 2.1: A tree representation of a game $G \equiv \{G^{L_1}, G^{L_2} | G^{R_1}, G^{R_2}, G^{R_3}\}$

For a game G we say that a player wins it, if there exists a winning strategy for that player, i.e. we assume that players play optimally.

A common representation for a game is a tree, as shown on Figure 2.1. We draw left options on the left side, and right options on the right side. One could think of playing a game as of going a zigzag path down from the root node until reaching a node where the player at turn has no options available. Figure 2.2 provides a visualization of this process. Red edges represent the consecutive moves. After two moves, Left has no option anymore and the game finishes, despite Right still having an option.

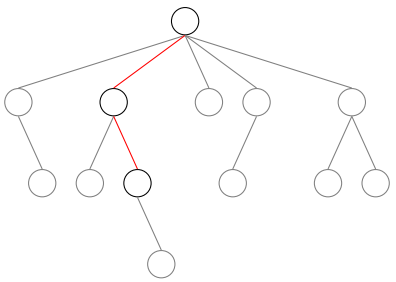


Figure 2.2: An example game being played out.

Definition 2.1.3. *An inverse of a game is defined as a game with left and right options swapped:* $-G = \{-G^{R_1}, -G^{R_2}, \dots | -G^{L_1}, -G^{L_2}, \dots\}$.

A tree representation of $-G$ looks like a mirror reflection of the tree of game G. This notation is in line with the previous definition of negative integers. Indeed, $-n$ is an inverse of n.

Definition 2.1.4. *A sum of two games G and H is another game in which left options are defined as left options from G summed with game H: $G^L + H$, and left options from H summed with game G: $G + H^L$. Right options are defined in analogous way.*

$$G+H :\equiv \{G^{L_1}+H, G^{L_2}+H, \dots, G+H^{L_1}, G+H^{L_2}, \dots | G^{R_1}+H, G^{R_2}+H, \dots, G+H^{R_1}, G+H^{R_2}, \dots\}$$

Although the formal definition is complicated, the tree representation helps to understand it. Usually, we would not draw a single tree of a sum of games, as it might prove a tedious task. Instead, we draw the trees of two games side by side. Then, a Left's move corresponds to picking a left option in one of the games, keeping the other one intact. For example, on Figure 2.3, Left chose to move to game H^{L_2}. Afterwards, Right can pick a right option either in game H^{L_2}, or in G.

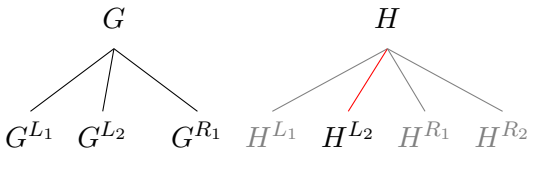


Figure 2.3: A sum of games G and H represented as two trees drawn side by side

As expected, game 0 is a neutral element of addition. Indeed, in this game neither Left nor Right has any options so any game G is not affected by adding the addition of 0: $G+0 \equiv \{G^{L_1}+0, G^{L_2}+0, \dots | G^{R_1}+0, G^{R_2}+0, \dots\} \equiv G$.

A game G might have one of four possible outcomes:

- If Left wins the game, regardless who plays first, we call the game positive, $G > 0$.
- If Right always wins the game, it is negative, and we write $G < 0$.
- If the player who starts, loses in game G, then we write $G = 0$.
- If the players who starts, wins, we call the game fuzzy and write $G \lessgtr 0$.

To compare any two games G, H, we examine the outcome of their difference game defined as $G - H :\equiv G + (-H)$. Then, partial order of games is defined as follows:

- Games are equal $G = H$ if $G - H = 0$
- $G > H$ if $G - H > 0$
- $G < H$ if $G - H < 0$
- Games are incomparable if their difference game is fuzzy: $G - H \lessgtr 0 \Rightarrow G \lessgtr H$

Under this definition of equality, addition of integers works as expected, e.g. $1 + 1 = 2$. Also, an inverse of any game is, as expected, an opposite element of addition with $G - G = 0$. Moreover, in every class of equal games, there exists a uniquely defined simplest representant called *canonical form* [Sie13, ch. II, Theorem 2.7]. For example, game 2 is a canonical form of the sum game $1 + 1$.

As shown in [Sie13, ch. II, Theorem 1.14], the equality classes of games with the addition operation form a partially ordered Abelian group.

A notion developed the Combinatorial Game Theory that bears a special significance for analyzing Go endgames is *temperature*.

Definition 2.1.5. *For game G and number t we say that G is* infinitesimally close *to t if for any positive number*[1] *ϵ we have $-\epsilon < G - t < \epsilon$.*

Definition 2.1.6. *Game G* cooled *by t is defined recursively as $G_t = \{G_t^L - t | G_t^R + t\}$ provided that for no $t' < t$, game cooled by it, $G_{t'}$, is infinitesimally close to a number.*

Definition 2.1.7. Temperature *of game G is the smallest $t \geq -1$ such that G_t is infinitesimally close to a number.*

The next Section shifts the focus to the game of Go. We will learn the notion of the *move value* which is a counterpart of the notion of *temperature* commonly used by Go players.

[1] Numbers in the context of CGT are a sub-group of the games group. It is isomorhpic with a dense sub-group of the group of real numbers with addition.

2.2. Applying CGT to Go endgames

How to interpret go endgames in the light of the introduced theory? While the way in which games are added resembles the nature of board decomposition in endgame, in combinatorial games the goal is to get the last move, while in go it is to take more points.

To frame Go as a combinatorial game, we identify points which players collect in Go with integer numbers, defined in Section 2.1. By convention, Black is identified as the Left player, and White as Right. When we decompose a Go board, the components corresponding to Black and White secure territories are positive and negative integers, respectively. Each undecided local position can be identified with a different combinatorial game.

We will analyze variations consisting of Black and White moves until a moment when the borders of territories in the local position become completely defined. In that moment, the local position would be interpreted as an integer number, equal to the final local score. Importantly, moves in the considered variations do not need to alternate. It is perfectly possible that in a given local position Black would play two times in a row, as in the meantime White can play moves in different parts of the board.

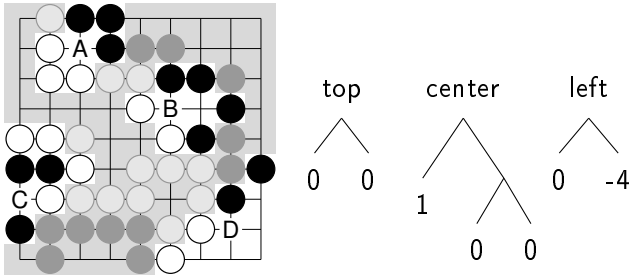


Figure 2.4: Examples of simple local positions and corresponding combinatorial games.

Figure 2.4 shows examples of four undecided local positions and letters A, B, C, D mark the best local moves. Areas of the board corresponding to secure territories are grayed out. Game trees corresponding to three of the positions are drawn next to the board. The last one is not drawn as it is too complex. Following a convention, we count local scores using territory scoring.

- In the topmost position, move A is the only available play. However, it does not surround any point. No matter who plays there, Black or White, the local score will be 0.
- In the center, Black can surround one point by playing at B. After that move, the local score will be 1. If White plays at B instead, the situation will become similar to the topmost position, as no one will be able to surround any point. The final local score will be 0.
- On the left side, if Black connects at C, no one will have any points. However, if White plays at C, two Black stones will be captured. White will surround two intersections and get two additional points for the captured stones. The score will be $\mathcal{T} = -D_B - E_W = -2 - 2 = -4$ (or $W + 4$).

The first three positions were easy to analyze as all possible variations were very short. On the other hand, in the fourth position there are six empty intersections. Therefore, there are almost $3^6 = 729$ possible arrangements of stones in that area (one needs to exclude those which involve stones with no liberties), and the arrangements might be reached in numerous ways. How can a player analyze such a position during the game?

Although the full game tree of the fourth position would be very large, an expert immediately identifies the move at D as the only local play worth consideration. The player might end up considering only six variations.

In fact, in most local positions in endgame, an experienced player would know that only one or two moves need to be checked to perform a proper analysis. Formally, in every subposition the player would consider *orthodox* moves (see [Sie13, ch. VII, Definition 2.1]) at the *ambient temperature* [Sie13, ch. VII, Definition 2.6] and, perhaps, a few other moves if the player cannot immediately recognize them as *non-orthodox*.

Finding reasonable local variations is only the start of the analysis though. In endgame one needs not only to find the right local moves, but more importantly, to decide in which of the local positions it is better to play. This second question is usually much more challenging for players. A common heuristic used by go players is based on counting *values of moves*. We start with an informal definition which aims at conveying the intuitive understanding of the matter shared by Go players.

Definition 2.2.1. *The value of move is a change that the move makes in the expected local score*[2].

This definition depends on the notion of expected local score which we have not defined yet. In case of positions from Figure 2.4 it is simply the mean of possible local scores:

- Playing in the topmost position does not change the local score which will anyway equal 0. The position is considered finished, and the value of the move at A is 0 points.
- In the center, a score could be either 0 or 1, depending on who plays first. The expected score is $\frac{1+0}{2} = 0,5$ and the value of move B equals $1 - 0,5 = 0,5$.
- Similar calculation show that move C is worth 2 points.
- In the bottom right corner, territory borders will become fixed only after a few moves are played. One needs to calculate expected local scores for intermediate positions in order to find the current expected local score. Eventually, the move at D turns out to be worth $2\frac{1}{3}$ points.

It is worth to mention the relationship between area scoring and territory scoring in the context of move values. In the area scoring every intersection controlled by a player (no matter if it is empty, has the player's alive stone, or has the opponent's dead stone) is counted as one point. On the other hand, captured stones are not counted as points. Let us look again at the same examples.

In the topmost position, the move at A changes the local score counted with area scoring. After Black's move the score will equal $\mathcal{A}_S = \mathcal{A}_B - \mathcal{A}_W = 4 - 3 = 1$, and after White's move it will be $\mathcal{A}_S = 3 - 4 = -1$. This means that the value of the move at A equals $1 - \frac{1+(-1)}{2} = 1 - 0 = 1$ point. An interested reader will check that the move at B is worth $1,5$ points, and the move at C has a value of 3 points.

In general, the value counted with area scoring is bigger by 1 than the value counted with territory scoring. Importantly, the order of the values is kept so the same position is going to be considered the most urgent when using both methods.

This relation between two ways of counting move values is important for this thesis because, as we will see in Subsection 5.2.3, we are going to use the area scoring in the program evaluating temperatures. In Combinatorial Game Theory, the relation is expressed as a homomorphism between games and games *cooled* by 1 (see Definition 2.1.6).

[2]This is a convention used e.g. by Antti Törmänen in [TV19] and by all mathematicians doing CGT research about go. Another convention commonly used by go players is defining the value of move as the difference between the expected scores after Black's move and after White's move. Numbers computed this way are generally twice larger than the numbers in this work.

Privilege moves

In the aforementioned examples, we counted the expected local score as the average of the scores in positions achieved by a Black move and by a White move because there is no reason to think that one player has higher chances of playing there first. However, in another type of situations, one player's move can be judged as more urgent than the other player's move.

On Figure 2.5 in the top right corner Black has a chance to surround a territory with the move at A, rendering White's three stones dead and getting $4 + 3 = 7$ points according to territory scoring. On the other hand, White's move at A threatens to save the three stones. If Black answers to White's move, the three stones will be captured, and the local score will be $3 + 3 = 6$. However, if White continues at B, Black will not surround any territory and the score will be 0. Figure 2.6 shows a tree of the corresponding combinatorial game.

If we simple-mindedly averaged scores for all tree nodes, we would calculate the current local score as $\frac{1}{2}(7 + \frac{6+0}{2}) = 5$. However, Black can ensure a better local score (6) by answering to White's move.

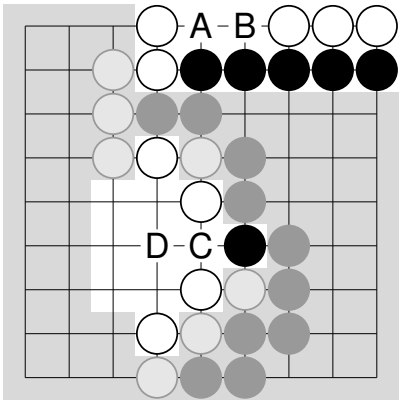


Figure 2.5: Privilege moves

Definition 2.2.2. *If a number of points gained by a move is smaller than the value of its continuation, the move is called a player's privilege, or* sente *in go jargon.*

In practice in most games, a player would play their sente move, and the opponent would immediately answer. It is thanks to the notion of privilege that board decomposition into independent regions becomes possible. Let us look at the center of the board on Figure 2.5.

Left side of the board is controlled by White. However, it is not completely surrounded yet. Black's move at C threatens to enter that region and destroy a lot of points with the continuation at D. It seems that White has no secure territory and the whole left side should be considered a single undecided position.

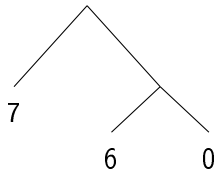


Figure 2.6: Game tree of the top right corner

However, one can easily judge that the continuation of Black's move would be worth more than the gain from playing at C. Thus, the move is considered sente, and all intersections further to the left of D can be considered White's secure territory.

With the notion of *sente* at hand, it becomes possible to formulate a definition of the expected local score which follows the intuition shared by Go players.

Definition 2.2.3. *The expected local score is defined as:*

1. *An average of expected local scores after Black's and after White's move if none of them is sente*

2. *An expected local score after a sente move followed by the opponent's answer*

A local position analysis performed by a Go player consists of finding all reasonable variations (i.e. consisting of orthodox moves at the ambient temperature) and afterwards, recursively applying the expected local score definition, to finally calculate the value of move. In Section 2.3 we will see a formal argumentation why values of moves serve as a good heuristic, as well as examples where the heuristic falls short of achieving perfect play.

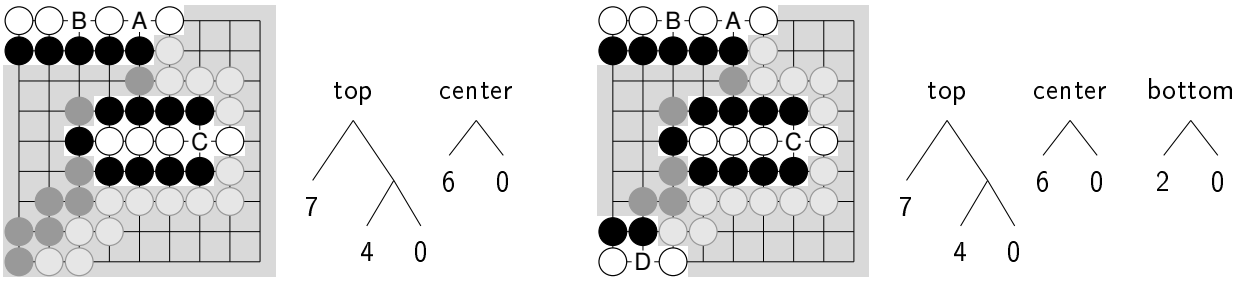


Figure 2.7: Counterexample for value-of-move heuristic. Game trees correspond to top, central and bottom positions respectively.

2.3. Orthodox play

Failure of move value heuristic

Playing a move with a bigger value, i.e. offering a bigger change in an expected score, makes intuitive sense. However, it does not always offer the optimal play.

On Figure 2.7, the moves at A and C have values of 5 and 6 points, respectively. It seems that playing at C should be better. However, on the left board, White can achieve a better result by starting at A (6 instead of 7). Game trees corresponding to the two positions are drawn next to the board. If White (*Right* player) starts by a move to 0 in game labeled as *center*, Black (*Left* player) will play in the other game to 7. White starting in the other game will end up in the final score 6.

On the right board the overall position is the same, except for an undecided region on the bottom. In this environment, White's move at C gives a better outcome than A.

Juxtaposition of these two examples proves that there exists no function mapping local positions to any totally ordered set based on which one could decide in which local position it is better to play. Finding optimal play in a sum of games cannot be reduced to comparing values of moves or any similarly simple algorithm. In fact, Go endgames were proven to be PSPACE-hard [Wol02].

Correspondence between temperatures and move values

In the following text, we are going to use the notions of *temperature* and *move value* interchangeably. However, it is surely not self-evident that the formal and abstract Definition 2.1.7 describes the same mathematical object as the informal Definition 2.2.1 which captures the understanding shared by Go players.

This correspondence is stated as a fact in [Lib]. Unfortunately I did not find any article which would show with mathematical rigor that the calculations performed by Go players provably yield values of moves that are equal to temperatures of the corresponding combinatorial games.

Readers will be able to see the correspondence between the two notions once the technique of *thermography* is introduced in Section 2.4. Also, interested readers can find a further discussion on this topic in [Ber96, p. 385].

Coupon Go

Despite the pessimistic theoretical result about the difficulty of Go endgames which was mentioned above, playing moves with biggest values serves as a good heuristic in positions com-

monly met in Go endgames. Usually, in endgame there are many undecided local positions, and values of moves are roughly uniformly distributed. In Combinatorial Game Theory, sum of such games is referred to as a *rich environment*. In sufficiently rich environments it is optimal to play move with largest values. To facilitate a formal discussion on rich environments, the Combinatorial Game Theory introduces an abstract notion of a *coupon stack*.

Definition 2.3.1. *A* coupon *of value t is a combinatorial game of a form $\{t|-t\}$ where t is a number. Abbreviated notation is $\pm t$.*

Applying Definition 2.1.7 to a so-defined coupon, it is evident that it has temperature t.

Definition 2.3.2. *A* coupon stack $\mathcal{E}_t^\delta$ *of temperature t and granularity δ is a sum of coupons:*

$$\mathcal{E}_t^\delta = \pm\delta \pm 2\delta \pm \cdots \pm t$$

Game G *enriched* by a coupon stack $\mathcal{E}_t^\delta$ is a sum game $G + \mathcal{E}_t^\delta$. Intuitively, we can think of such an enriched game in the following way:

We start from a fixed endgame position in Go, and next to the board there lays a deck of cards. On the topmost card, number t is written; on the next one, number $t - \delta$, and so on. Every time, the player at turn has an option either to put a stone on the board, or to take the card from the top of the deck. The final score of the game will then be decided as the score in the game of Go summed with the numbers on cards that each player took.

In such an enriched game, the information about move values in local positions is enough to find optimal moves. An *orthodox* strategy, called `SENTESTRAT`, is stated as follows.

1. Find optimal moves and calculate temperatures (i.e. values of moves) in all local positions.
2. If the previous opponent's move raised the local temperature, answer in the same local position[3].
3. Otherwise, if the hottest available coupon has temperature higher than the board temperature, play on it.
4. Otherwise, play in the hottest local position.

As shown by [Sie13, Ch. VII, Theorem 2.11], following `SENTESTRAT` a player can ensure getting the optimal score if a coupon stack $\mathcal{E}_t^\delta$ has sufficient temperature and granularity. This optimal score is called the *orthodox forecast*.

`SENTESTRAT` is also commonly used by Go players in the absence of the coupon stack. We already saw on Figure 2.7 that it does not guarantee perfect play. However, the Combinatorial Game Theory shows that a loss which a player might incur is upper-bounded. This result is called the Orthodox Accounting Theorem and here we formulate it in its simple version as stated in [Sie13, ch. VII, Corollary 2.10].

Theorem 2.3.1. *Let game G be a sum of subgames $G_1, G_2, \ldots, G_k$ and let $t = \max_i t(G_i)$ denote the temperature of the hottest subgame. Then, a player following `SENTESTRAT` will achieve a score at most $\frac{t}{2}$ worse than the orthodox forecast.*

[3]This proviso is necessary. Strategy which advises to simply play in the hottest position does not ensure optimal play. A counterexample defined as a combinatorial game can be found in [Ber96, p. 391] and its corresponding Go position is shown in [Fre20, App. A]

2.4. Loop-free thermography

In Section 2.2 we introduced in a bit informal language the way in which Go players count values of moves. The Combinatorial Game Theory provides a method called *thermography* which is a counterpart of the technique used by Go players. *Thermography* calculates game's *mean* and *temperature* using *thermographs*. Thermographs have an understandable visual representation as shown on Figure 2.8.

Definition 2.4.1. *Left Score $L_t(G)$ of game G is the result of the game enriched by a coupon stack of temperature t and sufficient granularity, with half of t subtracted: $G + \mathcal{E}_t - \frac{t}{2}$. Right score $R_t(G)$ is defined likewise, despite that we add half of t instead of subtracting it.*

Left and Right Score are functions of temperature t.

Definition 2.4.2. *Thermograph of game G is an ordered pair $(L_t(G), R_t(G))$.*

By convention, a thermograph is represented as graphs of functions L_t, R_t drawn on the same plot, which is a typical Cartesian plot rotated by 90 degrees anti-clockwise. The vertical axis corresponds to the independent variable t, and score is represented by the horizontal axis with numbers raising from right to left.

For simplicity, with the word *thermograph* we often refer to its graphical representation.

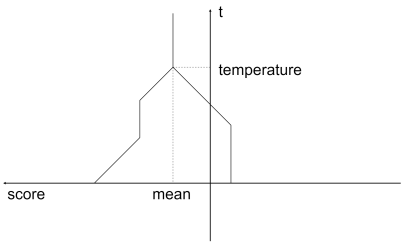


Figure 2.8: Example thermograph of a loop-free game

Definition 2.4.3. *We call functions L_t and R_t a left and a right wall of a thermograph, respectively.*

Properties of thermographs

Thermographs of finite, non-loopy games have a number of elegant properties:

1. Walls of a thermograph are continuous, piece-wise linear functions.
2. Left wall L_t is not smaller than the right wall R_t for every t.
3. A derivative of a wall on the segment on which it is linear is called a *slope*. Left wall has a slope 0 or -1 on each segment. Right wall on each segment has a slope 0 or 1.
4. For large t both walls have slope 0 and have the same value m, called the *mast* value.

The mast value of a thermograph equals the *mean* value of the corresponding game, as defined in [Sie13, Ch. II, Definition 3.25]. The smallest t_0 for which both walls equal the mast value $L_{t_0} = G_{t_0} = m$ is equal to the temperature of the game.

A formal derivation of the thermographic calculus, together with proofs of the listed properties can be found in [Sie13, Ch. II.5].

2.5. Ko fights introducing interactions between positions

All the aforementioned results apply to finite, loop-free games. In Go, there is always a finite number of possible moves. However, a game might be not finite if it involves a ko.

When we build a game tree for a local position, we consider all possible variations, even lifting the requirement that the moves should alternate because between the moves other moves could be played on different parts of the board. In a position with ko, as shown on Figure 2.9, when Black captures White's stone (which used to lay at A) with move 1, Go rules forbid to immediately recapture it with the move at A. However, instead White might for now play somewhere else on the board. Move 2 poses a threat to capture two Black stones so Black is likely to protect at B. After this exchange is made, White might come back to the ko fight, as the whole-board position has changed.

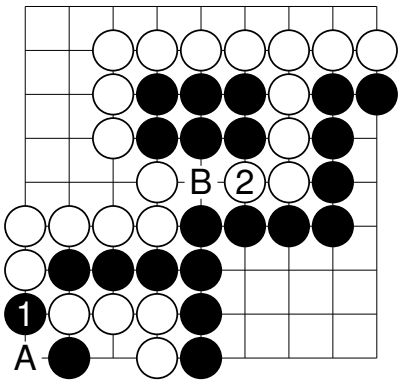


Figure 2.9: Ko fight for the bottom left corner.

When a ko fight starts, players take turns to find threats which the opponent would judge as more urgent than what can be gained by the ko. The line of play restricted to the local position with the ko will be loopy - the same positions would repeat, until one player decides that the opponent's threat is not large enough.

Loopy games are generally much harder to analyze for the Combinatorial Game Theory, but efforts were made [Spi01] to extend the temperature calculations also to such cases. The complications are well visible in the extension of thermography to loopy games. The elegant properties mentioned in Section 2.4 do not hold anymore.

Thermograph walls of loopy games can have different slopes than $\{0, -1\}$ (for Left) or $\{0, 1\}$ (for Right). In particular, the left wall could have a positive slope, and the right wall - a negative slope. As a result of collaboration with prof. Criado, the supervisor of this thesis, we have designed an exotic Go position shown in Appendix C. To our knowledge, it is the first discovered position for which a thermograph has a non-integer slope.

Despite the general theoretical difficulties, for most common positions with kos it is still possible to calculate temperatures. For a certain class of kos called *placid kos*, the upper bound on the possible loss of `SENTESTRAT` still holds. Local temperatures of such positions can be easily calculated by slightly adjusting the technique described in Section 2.2.

Chapter 3

AlphaZero revolution

Computer Go remained a challenge for two decades longer than computer chess. One difficulty in Go was the high branching factor which rendered the classical *minimax* tree expansion algorithm computationally expensive. It also proved difficult to approximate intuitions of human experts with handcrafted formulas.

One improvement was the *Monte Carlo* heuristic, implementation of which did not require expert knowledge. Then, a big breakthrough came with an introduction of the *Monte Carlo Tree Search*[1] [Cou07]. The algorithm greatly reduced a size of an expanded game tree required to provide a valuable insight about the next moves.

In the 2010s deep learning emerged as a state-of-the-art method to model human intuitions across various fields. In 2016, DeepMind researchers caused a real sensation when their program, AlphaGo [SHM+16], beat a world champion. The program was an eclectic solutions, utilizing deep learning alongside with some classical methods, such as Monte Carlo playouts. It used several neural networks, trained in different regimes.

From the scientific perspective, the biggest achievement came one year later when the same team published a paper on AlphaZero [SSS+17]. Presented solution was neat and self-contained. The program used only one neural network, paired with the MCTS algorithm. The most striking was the fact that the AlphaZero network, unlike AlphaGo, was trained with no human knowledge.

3.1. Network architecture

In AlphaZero, the architecture of choice was a deep residual network. This architecture was originally used in computer vision. The choice was well motivated as understanding Go positions shares a significant resemblance with visual recognition, with the geometrical structure playing a key role in the way a human expert views the game.

One of central concepts in Go are good and bad shapes, a term which refers to local configurations of stones. These could be recognized in early layers of the convolutional network. Next layers could be able to extract information about territorial frameworks, a crucial idea for Go strategy.

Two version of network were tested: one with 20 residual blocks, and the other with 40[2].

[1] Words *Monte Carlo* in the name of the algorithm refer to the evaluation heuristic originally used during tree expansion. The algorithm can be used also with other heuristics, e.g. neural network predictions, like in AlphaZero.

[2] Strictly speaking, the first block differed from all other, and it was a one-layer convolutional, not residual, block, which means that no skip connection adding input tensor was used.

Each block applied two convolutional layers. Each convolution had kernel 3×3 and 256 output channels.

3.1.1. Network input

Developers of AlphaZero prioritized not showing the network any information obtained from human expertise about Go. Therefore, in contrast to AlphaGo, the neural network's input is a raw representation of the game state.

The input tensor has a spatial dimension of the Go board (19×19). One of the tensor's channels encodes stone positions of the current player. Another one marks placements of opponent's stones. Analogous channels provide representations of board positions from a few earlier moves. It is necessary because of the *ko rule* which forbids certain moves based on the earlier board positions. Finally, there is one channel which indicates the color to play. It is needed because of the *komi* which makes Black's and White's objectives slightly different.

All the input channels are binary. In case of the last channel, it has a constant value of 1 if the current player is Black, and 0 otherwise.

The spatial dimension of the input tensor is kept through all network's layers.

3.1.2. Network output

AlphaZero was trained to assess two pieces of information: firstly, which moves are the most promising at a given position, and secondly, what chances does the current player have to win the game. To make it possible, the network was designed to have two *heads*.

The *policy head* outputs a $1D$ tensor consisting of 362 numbers: one number per each intersection on a board (a total of $19 \cdot 19 = 361$ numbers), and one for the *pass* move. These numbers are scaled with a softmax function, and called *prior probabilities*.

The *value head* outputs a single number representing a chance of winning. Both heads were light convolutional networks with a fully-connected final layer.

3.2. Move selection

In earlier works [TZ16] deploying deep learning in computer Go, a neural network was usually only predicting the next move. A program would play the move to which the network assigns the highest number. Such networks were trained on positions from games of human expert's and were trying to predict the expert's next move. This solution was also used in AlphaGo [SHM+16].

State-of-the-art programs from before the era of deep learning chose a move based on a tree search. They traversed possible variations and evaluated every position with a heuristic function.

AlphaZero combined these two methods. With a bare output of the policy head, it would be possible to use the first technique. A bare output of the value head could serve as a heuristic function in the second technique. Combining both provided an efficient way to run a tree search guided by the policy head predictions.

The exact algorithm used was a variation of the Monte Carlo Tree Search. The MCTS algorithm builds a tree, focusing on moves which lead to more promising results, i.e. the ones with better statistics generated using the heuristic function. This way, moves which look very bad after one evaluation would not be checked for a long time and the program would avoid unnecessary computations.

In AlphaZero, the MCTS algorithm was enhanced using the *prior probabilities*. From the very beginning of the tree building process, the program would focus on moves which seem more promising for the policy head. This trick greatly saved computation time, since without it in every tree node hundreds of possibilities would need to be checked.

The MCTS algorithm would run for a certain number of iterations, providing a more informed judgment about available moves than prior probabilities. The updated statistics for available moves would produce *output probabilities*, based on which the final decision would be made.

Using both policy head and value head, AlphaZero managed to mimic the decision process of human experts based on reading variations, as described in Section 1.2. The program would consider much less variations than its predecessors. At each node of the tree, it would focus its attention at only a few good looking moves, completely omitting dozens of moves which would look obviously bad[3].

3.3. Training scheme

When training a neural network in a supervised setting on positions taken from actual games, natural candidates for the targets for policy and value head are the next move played in the game, and the game final result respectively. However, AlphaZero did not use any external data. All positions which appeared in its training were taken from the program's selfplay games.

If a network learns to predict its own moves, and results of its own games, one might wonder how it could ever learn anything new. The bootstrapping was facilitated by the use of the MCTS algorithm. The output probabilities obtained with the tree search contain better clues than the prior probabilities, providing a powerful *policy improvement operator*. The output probabilities were then used as the target for the policy head, and the information about the game winner served as the target for the value head.

At the beginning of the training, the network's weights were randomly initialized. Then, a number of selfplay games was produced, and positions from these games were used as training data. After updating the network's weights via stochastic gradient descent, a new version of the network was used to produce more selfplay games. To avoid the repetition of selfplay games, moves were chosen by sampling from the output probabilities, instead of simply taking the move with maximum value. Additional diversity was obtained by adding a Dirichlet noise to the prior probabilities in the root node, before starting the tree search.

Noteworthy, this last technique was recognized five years later in [DGSS22] as not ensuring policy improvement. The new paper, also published by researches from DeepMind, suggested a different way of ensuring diversity, called *planning with Gumbel*, which had stronger statistical grounding. Experiments proved that it leads to a more efficient training.

3.4. Novelties of KataGo

KataGo is currently one of the strongest publicly available Go playing programs. While being to a large extent a reimplementation of AlphaZero, KataGo benefits from a number of enhancements to the vanilla algorithm [Wu20].

[3] To be precise, the MCTS algorithm is going to check all possible moves, when run for a sufficient number of iterations. However, when run for thousands of iterations, some of the possible moves would remain unchecked, and the program's strength would already exceed top human players.

KataGo's author gave up on a pure zero-domain-knowledge approach. For example, to mitigate one well-known shortcoming [TMG+22, p. 6] of the AlphaZero-based programs, he included an information about *ladders* in the network's input[4]. Despite using several domain-specific optimizations, the training procedure still included only data from selfplay games, like in AlphaZero.

Further improvements related to network architecture (e.g. use of *global pooling* layers) and setup of selfplay games (e.g. varying number of MCTS iterations throughout the game). A particularly interesting novelty are auxiliary targets predicted by new network heads. The most relevant from the current work's perspective are heads predicting the *final score* and the *ownership map*.

To facilitate training of these two heads, not only an information about the game result (i.e. who won the game) was kept for selfplay games, but also the point difference (i.e. by how many points a player won), and an information for each intersection to whose area it belonged.

The auxiliary targets are not used in the move selection process, as this is still based on MCTS guided with policy head and value head predictions. However, they force the network to learn more versatile information about board positions. As shown in the ablation study [Wu20, p. 12-13], their adoption in training improves the program's performance.

[4] A ladder is a long sequence which can potentially capture an opponent's stone. It depends on stones located far away, in a different part of a board, whether the technique works or not. Deciding if a ladder works is a straightforward procedure calculated in KataGo with a simple algorithm.

Chapter 4

CGT and Computer Go

Combinatorial Game Theory provided powerful tools to analyze go endgames. For some specific positions, it designed simple algorithms assuring optimal play. Some of these findings were not known for go players and it let mathematicians design a position, known as *9-dan stumping problem* [Ch. 4.9][BW94b], in which they would be able to beat top players.

Although many CGT findings are articulated in algorithmic form, to date there has been limited effort in translating these algorithms into computer implementations. Martin Müller's PhD dissertation [Mül95] from 1995 remains the main work in the field.

4.1. Classical approach

In his PhD dissertation [Mül95], Martin Müller explained how CGT can be applied to Go endgames and presented two algorithms implementing CGT findings. The *approximate algorithm* works for any position but not always finds optimal play. The *exact algorithm* can be successfully applied only to a specific class of positions which can be provably decomposed into independent regions of limited size. The rule of operation is as follows:

- **Decompose the board into independent regions** This is done by an algorithm recognizing safe stones that provably cannot be captured by an opponent under, and safe territories - regions surrounded by safe stones of one color in which every stone the opponent puts in can be provably captured. The rest of the board is split into connected components which correspond to independent local positions.
- **Construct a CGT game for every local position** Within every local position, all possible variations are taken into account. For each variation the final score is computed. Based on these statistics a CGT game tree is built. Afterwards, the game's canonical form is found by reversing reversible options and removing dominated options.
- **Find optimal play in the sum game** Games calculated for all local positions can be summed, and the sum game can be solved with the minimax algorithm. As complexity of this solution grows exponentially, several CGT-based improvements are deployed, such as *pruning moves with dominated incentives* [Mül95, Ch. 3.1].

The algorithm facilitates an exact analysis of many whole-board positions that traditional methods cannot analyze due to computational complexity. However, the class of positions covered by it is still very limited.

The first serious constraint is a need for the positions to be tightly bounded by safe stones. As noted in Section 2.2, this is a rare scenario. In real games, players tend to leave

holes which could theoretically allow an opponent to enter their territory with a few-move sequence. In practice though, they are determined to consolidate their formation on the opponent's attempted incursion. Assumption of such play lets Go players analyze such not tightly bounded positions.

The decomposition algorithm suffers also from difficulties in recognizing secure territories even if they are perfectly bounded with safe stones. In real games, territories often span across large regions, leaving a lot of empty space for potential opponent's invasion. While an experienced Go player is able to easily judge that they would be able to kill any invasion, the proposed algorithm would fail to prove the same thing.

The third issue is the size of a local position which could be analyzed. Since the algorithm checks all possible variations, the solution has an exponential complexity. Local positions with more than 10 legal moves were judged as too complex for analysis.

To address the aforementioned shortcomings, the author presented *Explorer*, a program which implemented an approximate algorithm of choosing moves. In Explorer, a number of heuristics was used to decompose a board into a sum of local positions, even if territories are not tightly bounded with safe stones. The program utilized a few routines for generating move suggestions to avoid checking all local variations.

Explorer was tested in a few different settings. One test was guessing the next move from a game of expert players. The program achieved 28% top-3 accuracy which was a massive improvement over a random choice but did not match the performance of beginner players [Mül95, p. 92].

4.2. AlphaZero, KataGo and CGT

Deep learning led to a breakthrough in computer Go. For the first time, programs were developed that could play Go better than human experts. However, could they be used to analyze endgame positions in the way discussed in this work?

4.2.1. Vanilla AlphaZero algorithm

AlphaZero chooses moves in endgame in the same uniform way as in earlier stages. The MCTS algorithm gathers statistics about winning chances across positions which could be reached with reasonable variations. The tree search is global and takes no advantage of potential board decomposition. Nevertheless, with enough iterations it is able to find strong moves and outplay human masters.

It is not possible to retrieve information about position temperatures or about perfect local play from AlphaZero outputs. Even if one performs the board decomposition by hand, the output of policy head is not likely to contain useful information about moves different than the one (or a few) in which playing is the most urgent.

It is theoretically possible to run the MCTS while restricting the set of expanded moves to those belonging to a selected local position. However, this will not always lead to exploring optimal local variations. The neural network always tries to find moves which maximize the winning chance so if it is forced to choose between moves in not the most urgent local position, it will aim to find a *sente* local move. The opponent would be forced to answer it, leaving the network a possibility to play in the most urgent position. Such *sente* play would very often be an unoptimal move from the local perspective.

4.2.2. Experiments with KataGo's ownership maps

Katago network provides a richer output which facilitates experiments related to various CGT concepts. In my bachelor thesis [Fre20], I tested *ownership maps* produced by KataGo predictions against the mean scores calculated with CGT.

KataGo's *ownership head* is trained to approximate the final ownership of each intersection. As for the network it is impossible to predict the course of the game until its end during a single forward pass, the predictions restricted to a local position can be expected to approximate a mean of possible local outcomes achieved under perfect play in different environments. Consequently, if one sums ownership predictions for all intersection in a given local position, the result should approximate the position's mean score under area scoring[1].

One should remember to not assess with this method temperatures of the most urgent positions on the board. In such case, the network's predictions will be biased by an assumption that the current player will play there the next move. Moreover, it has been demonstrated [MKT+21, Fre20] that the hidden states of AlphaZero-based networks could include expectations about future positions distant by more than one move.

In the described procedure it is also possible to not sum up the predicted numbers, but to check the statistics for one selected intersection. In my work, I opted for this solution as it offered a more detailed insight into the network's assessments. To find *ground truth* ownership of an intersection, one needs to analyze the game tree to find out which variations are optimal, check the intersection's ownership at the end of each such variation, and average these numbers with the same formula which would be used for averaging local scores.

To test the precision of KataGo's estimates, I assembled 24 local positions bounded by safe stones. Afterwards, I embedded them into a middle-game whole-board stones arrangement, so that the local positions were not the most urgent places on the board. To test consistency of network outputs I repeated the process for a thousand of different stone arrangements. The predictions proved relatively consistent with the standard deviation lower than $0,1$ for most tested position.

Predicted *ownership maps* exhibited a rather strong, yet still imperfect, understanding of the expected ownership of intersections. Within safe territories, the network predicted values close to 1, even when proving the security of territories required knowledge of certain tactical techniques. For positions in which territory borders could be fixed with a single move by each of the colors, the network correctly assessed the expected ownership as close to 0.

Situations in which a move has a continuation posed a bigger difficulty to KataGo. According to the theory, if the continuation's value is not more than twice larger than what is surely gained with the first move, then the move does not raise the temperature, and it should not be considered *sente*. Thus, the chance of the players getting that move is the same, and the ground truth ownership of intersections seized by it equals 0. In contrast, KataGo's predictions suggested a belief that the player whose move has a continuation has higher chances of getting it than the opponent.

Further, I examined situations in which a temperature does not drop or even raises after a move, i.e. in which the continuation has a value equal and bigger than twice the sure gain respectively. Ownerships assessed by KataGo proved that the network has a rough understanding of the concept of *sente*. Unlike in the non-sente situations, predictions for intersections that could be seized by the continuation were close to 1 (where White plays the sente move, and Black defends against it), indicating a belief that a player will not let the opponent continue after the sente move. However, predictions for intersections seized by

[1] In area scoring, a player gets point for every controlled intersection, and not only for surrounded empty intersections. Similarly, the ownership map provides a number from -1 to 1 for every intersection.

the first move were unfortunately not really close to -1. It shows that the network is not completely convinced that White will manage to play their sente move. Nevertheless, the predictions were visibly closer to -1 than for non sente moves.

Were the ownership predictions close to ground truth, one could estimate a mean score of a position under area scoring by summing up the values for all intersections. However, the difference between the ground truth and the predicted number is often around $0,1$, or even bigger in scenarios with sente moves. These inaccuracies quickly add up to more than one point for a position with not too few undecided intersections.

One needs to remember that these inaccuracies generally do not influence the playing strength of KataGo. Being an auxiliary target, ownership map is not used in the program's decision making process.

4.3. Towards neural network guided CGT computations

Combinatorial Game Theory provides certain tools which can facilitate finding optimal play lines in Go endgames. Martin Müller implemented them, achieving an exact algorithm for solving Go endgames [Mül95]. In a later work [Mü01] he demonstrated how dramatic improvement of computational complexity this solution offers compared to a brute-force minimax analysis of the whole board.

Unfortunately, usability of the developed program was greatly restricted by difficulties in automatic decomposition of the board and selection of reasonable moves in local positions. Go endgames are not explicitly defined sums of combinatorial games (unlike some other games, e.g. *nim*). The CGT analysis of Go positions can be done by humans only when coupled with an expert judgment facilitating a board decomposition. Furthermore, a human can avoid checking exponentially many local variations by leveraging an intuition about reasonable moves.

Deep learning proved successful in modeling intuitions connected to Go. However, Go playing programs existing to date cannot be directly applied to aid a CGT analysis of Go endgames.

Analysis of ownership maps predicted by KataGo provides a clue on how well the network understands a theory of Go endgames. In my bachelor thesis [Fre20], I presented a simple methodology which could be used to estimate local mean scores leveraging the network's predictions. However, experiment results show that such estimates are not very accurate.

KataGo may aid endgame analysis also in other ways. For example, the ownership maps provide a natural tool to perform a board decomposition. Intersections for which predicted ownership exceeds $0,9$ can be taken as secure territory of Black, and those with ownership below $-0,9$ - as White's secure territory. Then, one could treat connected components of intersections not belonging to secure territories as local positions (although positions found by this heuristic are not always independent, as shown in Section 6.2).

Finally, having local score estimations, it is possible to estimate a temperature by finding a difference between the current local score, and the local score after one player's move. One only needs to make sure that the examined move is not *sente*, in which case the other player's move should be checked. A downside of this method is the need to enter manually the optimal local moves after which the scores should be checked.

To recap, existing programs could provide valuable insights into the CGT applied to Go, but they are not capable of performing a human-like analysis of local positions. This being said, advances in deep learning give hopes that this can be done in principle. Let us examine how this could be done.

AlphaZero training scheme for the first time in history provides a way to obtain informed, high-level representations of Go positions in a computer program. Taking an existing neural network of a Go playing program, or training a new one from scratch, one could design auxiliary targets for it which would be related to the Combinatorial Game Theory. For example, one might think of predicting local temperatures or optimal local moves.

Another idea, which could potentially produce more precise predictions, is adjusting the training scheme of the program so that moves are selected not using the MCTS algorithm, but leveraging a CGT analysis of a game sum instead. This path, however, would require a lot of experimentation. One would need to develop a new procedure of collecting training data and presenting it to the network, instead of using an of-the-shelf algorithm.

Regardless of the training scheme choice, one could afterwards leverage the network's predictions either to directly estimate values such as temperatures or mean scores, or to couple the network with a CGT-based algorithm analogous to the one developed by Martin Müller [Mül95].

In the next chapter, I present the choices I made embarking on this new pursuit.

Chapter 5

Program evaluating temperatures in Go endgames

Deep learning proved the right approach to create a super-human Go playing program. Therefore, it is reasonable to expect that it could be also used to aid CGT analysis of endgame positions in Go. As described in Chapter 4, implementation of CGT computations is a challenging task due to difficulties in assessing sure territory and a high branching factor in local positions appearing in endgame. A neural network could mitigate these issues by providing judgments about intersections ownership, and predicting sensible moves in a local position.

When a Go player analyzes a local position in endgame, they try to understand what is the correct move and what is its value (i.e. what is the local temperature). Only with these two pieces of information at hand, one could make an informed decision about the best move on the whole board. A program that is useful in the endgame phase of Go should also be able to assess these two pieces of information.

The approach used by Martin Müller was to build a full tree of a local position, run a CGT analysis over it and based on this analysis find the temperature and the best move. On the other hand, one could use a deep neural network to find the best local move and/or the local temperature.

In my program, I utilize a network which predicts the best local moves. Based on these predictions, a partial game tree is built. Next, I run an algorithm developed by Łukasz Lew [LF24] over this tree to find the local temperature.

In Section 5.1, I present how the program works. In the next two sections, I describe the architecture and training of the neural network. In Section 5.5, I show how the program performs on a set of 100 real endgame problems.

5.1. How the program works

The input of the program is a whole-board arrangement of stones with a local position marked by a user. Then, a partial game tree for the local position is built by iteratively using predictions of a neural network:

Algorithm 1: Game Tree Expansion Algorithm

Input: Initial arrangement of stones
Output: Expanded game tree with temperatures calculated in all nodes

```
(1)  Function SelectNode()
(2)      return a non-terminal, not expanded node;

(3)  Function ExpandNode(node)
(4)      if IsTerminal(node) then
(5)          assess the final local score in the position;
(6)          return;
(7)      moves ← BestMoves(node);
(8)      foreach move ∈ moves do
(9)          create a new child node for move;

(10) Function BackupScore(node)
(11)     while calculations are finished for all node's children do
(12)         CalculateTemperature(node);
(13)         node ← node.parent;

(14) AddRoot();

(15) while not all nodes are expanded or terminal do
(16)     node ← SelectNode();
(17)     ExpandNode(node);
(18)     if IsTerminal(node) then
(19)         BackupScore(node);
```

Node selection simply visits every node created in the expansion step. Node expansion is performed using neural network predictions. Result backup is done using the algorithm for finding temperatures [LF24].

5.1.1. Node expansion

A deep learning model returns to the program five pieces of information:

- Best moves for Black in the given local situation, expressed as a probability distribution over all intersections of the board
- Best moves for White
- Information whether a move of one player is *sente*, expressed as the probability that the next move in the position will be played by Black
- Information whether the position is terminal, expressed as the probability that no move will be played in the position anymore
- Local score

If the probability that the position is terminal surpasses a given threshold, no new nodes are created. Otherwise, there are a few possible ways in which the output concerning next moves can be used for node expansion. I perform experiments using different strategies:

- **Best move strategy** Assume that the top choices of the model for Black and for White are the best local moves, and always create a node for each of them. In this case, the information about *sente* is not used.

- **Threshold strategy** Add nodes for moves for which a probability predicted by the network is higher than a given threshold. The probability for all moves of a given color is initially multiplied by the probability that this color will play the next move.

Importantly, the algorithm for finding temperatures requires that for every non-terminal node in the game tree there is at least one option for both of the players. Therefore, when using the threshold strategy, a node still needs to be added for a color even if the probability for this color playing the next move is very low. For such a case, I test two different solutions:

- **Threshold strategy with phantom values** If a probability for a given color is very low, artificially add a child for that color which is a terminal node with an extreme local score.

- **Threshold strategy with expansion of depth 1** Instead of adding an artificial child, create a node for the top predicted move, immediately assess the expected local score for it and treat it like a terminal node.

Noteworthy, these two strategies are used only in non-root nodes. In the root node I always add at least one child for each color even if a probability of that color getting the next move is low (because the opponent's move is sente).

Comparison of results achieved with these strategies is presented in Subsection 5.5. In Chapter 6, I discuss which classes of positions could be correctly analyzed with different strategies and what shortcomings they exhibit.

5.1.2. Result backup

The program's goal is to find a CGT game corresponding to the root node, and then to use to calculate the temperature for this game. Finding a CGT game for any node is possible only if CGT games for all its child nodes are known.

If a selected node is judged as terminal, a CGT game is set for this node to a constant value corresponding to the local score. Then, we check if for the parent node all its children have their CGT games set. If it is the case, the CGT game is set by treating games corresponding to positions after Black's moves as the Left options, and games corresponding to positions after White's move as the Right options. The mean and temperature are counted for this game. Then, we repeat the process, checking if for this node's parent's all children have CGT games set.

One addition is needed to the temperature finding algorithm to make it usable on real game positions. The algorithm does not support loopy games. However, in endgame positions, kos appear very often. In vast majority of the cases these are the simplest kos where only a single point is at stake.

In the current work, I do not aim at finding a general algorithm which could handle all different types of kos possible in Go. Instead, I design a simple algorithm which works correctly for simple loopy games (as defined in [Sie13]) in which there is only one exit for Black and one for White.

When a ko situation is detected (because a move suggested by a network leads to a repetition of a previous position), the node for which the expansion is performed and its

parent node are added to a *ko chain*. If it happens that a node that is about to be added to a ko chain already belongs to a ko chain, the chains are merged (this situation corresponds to so-called *multi-stage kos*). In most of the cases, the ko chains have length of 2.

Once CGT games are set for all nodes lying below nodes of a ko chain, the CGT games for the nodes in the ko chain are set with an addition of phantom options in place of the moves that were not added to the tree because the ko ban. CGT games of the phantom nodes are set as constant values computed using the mean values of the games appearing in the exit nodes of the ko chain. Given that the ko chain has length k, mean value of the game in the Left's exit node is l and in the Right's exit node it is r, the phantom Right option added in a node at distance i from the Left's exit node is a game equal to number $l - (i+1)\frac{l-r}{k+1}$.

5.2. Model architecture

To create a model which could predict best local moves, I utilize a pre-trained Go playing neural network. This way, the network can benefit from game understanding obtained during a long pre-training process. In an ablation experiment, I train a network of the same architecture without using pre-trained weights, and its learning curve is much steeper.

5.2.1. Pre-trained model

As the pre-trained model I chose a vanilla implementation [Ngu22] of the Gumbel AlphaZero algorithm [DGSS22] available on GitHub, called shortly A0jax. The repository provides a neat and simple code in `jax` library, following the original architectural choices of the AlphaZero network.

An available model was trained on a 9×9 board, however, being a pure residual network, it can be also run on a bigger board, except for the last layers which are fully-connected and therefore depend on the exact dimensions of a tensor.

The model is very light compared to state-of-the art Go playing programs, comprising of mere 5 residual blocks, each producing a 128-channel tensor. It was trained on samples extracted from over 10 million training games. While no performance tests are published for the program, the web demo version using 512 MCTS iterations is able to beat a professional player, as well as KataGo using 4 MCTS iterations.

5.2.2. Policy head

I added a new head to the network's backbone which predicts the best local moves. As opposed to the original policy head of AlphaZero-based networks, my policy head predicts moves for both colors at once. The output of the head is a tensor of length $723 = 2{*}19{*}19{+}1$. First 361 numbers of the tensor are probabilities for White moves per each intersection of the board, the next 361 numbers correspond to Black's next moves and the last number corresponds to the possibility that no move is going to be played in the given position anymore.

Softmax function is used on the outputs so that they sum up to 1. Then, the last predicted number is interpreted as the probability that the position is terminal. The sum of probabilities for Black moves is divided by sum of probabilities for both Black and White moves is interpreted as the probability that the next move will be played by Black. Softmax function is again used on numbers predicted for each of the colors to obtain best move predictions.

5.2.3. Score assessment

To assess a local score for a terminal position, existing tools can be used. An ownership map predicted by KataGo is one option. For intersections belonging to the local position, a predicted ownership is rounded to the nearest integer[1] and numbers obtained this way are summed. The result corresponds to the score counted with area scoring $\mathcal{A} = \mathcal{A}_B - \mathcal{A}_W = (L_B + E_B) - (L_W + E_W)$.

When using the *threshold strategy with expansion of depth 1*, expected local score needs to be assessed also for non-terminal nodes. This is also possible by using KataGo ownership maps. In such case, the ownership predictions are not rounded to integers before summing up. For non-terminal nodes the predictions do not follow the CGT as closely as in case of secure territories, as shown in [Fre20] but in most cases it should not affect temperature calculations since these assessments are used only in case of nodes which should be not visited in correct variations. Further discussion on this topic can be found in Chapter 6.

To make the solution in this work self-contained, I attach also a new head to the A0jax neural network. The network is trained to approximate KataGo's ownership head's output. This way, these predictions can be used for the score assessment, and KataGo network is not needed for running my program.

5.2.4. Local position mask

In AlphaZero based models, the neural network's input is a 3D tensor with spatial dimensions of a go board. In such setting, the simplest representation of a single local position is a binary mask: an array of the same shape as the board with ones on the intersections belonging to the given region, and zeros everywhere else. There are several ways to incorporate a region mask in a forward pass of a neural network. However, to date no research on similar topics in the field of computer Go has been made. As mentioned earlier in Subsection 3.1, the AlphaZero network has similar architecture to models used in computer vision. Therefore, I examine ways of masking regions used in that field.

Most prominent works in computer vision leveraging the concept of region proposals are two-shot object detectors. For example, R-CNN [GDDM14] uses one network for detecting areas potentially containing an object and another for extracting features from each of the proposed regions. As the feature extractor expects inputs of a fixed square size, rectangle patches corresponding to bounding boxes of proposed regions are cut out of the original picture and resized to match that size. Since the proposed regions do not necessarily have a rectangular shape, background pixels within the region's bounding box are replaced with a constant value.

Newer two-shot object detectors [Gir15, RHGS16] run a feature extractor on the whole picture, and only then include the information about proposed regions. Areas of the feature tensor (the extractor's output) spatially corresponding to detected regions are cut out and fed into a classifier.

The aforementioned approaches could hardly be directly applied to modeling local positions in Go. As explained in Subsection 3.1, in networks analyzing Go positions, the spatial dimension is kept throughout all layers. A resizing operation used on patches cut out of a tensor is a viable solution in computer vision, but in Go it would change the semantics of a local position.

[1] As noted in Section 1.1 in certain rare terminal positions there are intersections not seized by neither color - for them the ownership prediction is going to be close to 0.

Moreover, replacing background area with a constant value is dangerous in the context of Go. The correct play in a local position defined as an area of intersections with uncertain ownership might still depend on the stones placed in the adjacent areas of secure territories.

Taking into account these two assertions, I opted for a simpler method of incorporating an information about the selected local position into the network. Similar to [Now17], I concatenate the binary mask representing the local position with the tensor that I want to mask. This method ensures that the whole context of the position will be included in the network's computations. I add this mask as a new tensor channel only after the forward pass of the network's backbone, before feeding the tensor to the network's heads.

5.3. Training data

To collect big amounts of endgame positions I used a dataset of KataGo selfplay games. For each game, I calculated its length and extracted the board position corresponding to the move of number equal to 85% of length. This constant was found empirically with the aim of obtaining game states facilitating board decomposition while ensuring that the components are not too easy to analyze.

On every extracted position I ran inference of the KataGo network and saved the predicted ownership map. Apart from that, for every position I kept the full game record.

I decomposed each whole-board position by setting a threshold of $0,9$ for the absolute value of ownership predictions. The choice of the threshold was motivated by findings on KataGo assessments about secure territories from [Fre20]. Intersections not belonging to secure territories were then grouped using `connectedComponents` function from `OpenCV`.

Afterwards, for each whole-board position three components were chosen for training data. Each of the components was expanded to include also intersections from neighboring secure territories. My experiments showed that it was crucial to expand the area in a random way, ensuring that the undecided position is not always located at the center of the masked region. Otherwise, the network was learning a simple heuristic of predicting moves in the center of the masked area, failing to generalize.

To make the training data more versatile, in half of the training samples I removed stones from far regions of the board which are most likely not related in any way to the analysis of the picked local position.

Every time, one of the chosen regions was a component neighboring the last played move. If the move did not neighbor any undecided region, an area of random shape was chosen around that move. Introduction of such training samples was aimed at ensuring that the heads correctly recognize a secure territory and predict that no additional move is needed in it.

Including regions neighboring the last played move was crucial for the heads to properly learn the notion of sente. If the last move happened to be sente, the heads will have an incentive to assign high probabilities to opponent's moves, and adjust the ownership predictions according to this expectation.

To facilitate this type of reasoning inside the network, I provide the representation of the board state before the last move in network's input. By default, the input tensor of AlphaZero should contain representations of eight last board states. I randomly chose a number between 2 and 8 (or 1 and 8 for the local positions different than the one in which the last move was played), include this number of last board state representations and simply stack the last of them to ensure the correct size of the input tensor.

For every chosen region, I look through the game record to find the next local move, or

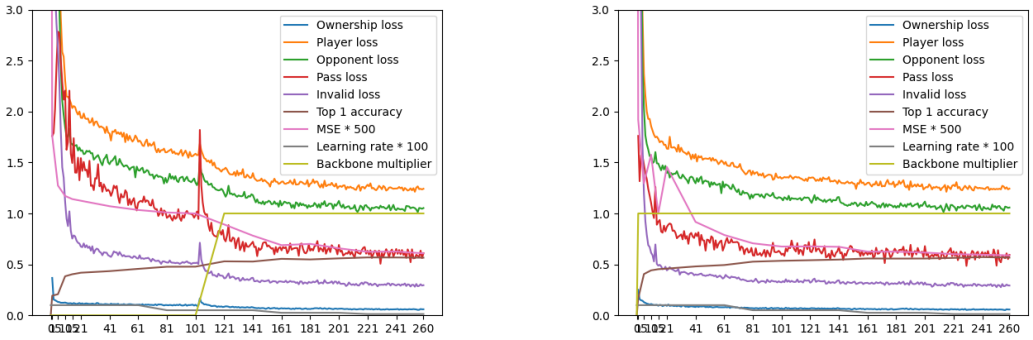


Figure 5.1: Losses and metrics for two runs. In the left one, the backbone was initially frozen. In the right one, I didn't freeze any weights.

to find out that there was none. I save the coordinates and color of the move, and use it as a target for the policy head. As the target for the local ownership head, I chose the ownership maps predicted by KataGo.

There are millions of KataGo selfplay games publicly available, and I processed around 10% of them. This amount proved to be more than enough for a success of the fine-tuning process.

5.4. Training process

For the local ownership head, I use the `L2`-loss. For the local policy head I use `cross entropy` loss. Moreover, I add a penalty term for move predictions outside of the local position or on intersections occupied by stones.

The training runs with batch size of 128. The learning rate is initialized to $0,01$, as in the original implementation. I experimented with lower values, but the convergence was slower, and the achieved metrics were worse. Learning rate is halved after every 10.000 steps. In the original implementation this number was ten times higher, but in my scenario it would not make sense, as the metrics reach plateau much earlier. Also, I keep the weight decay parameter at 10^{-4} as specified in the original code.

I performed two training runs. In one, I started the training process only by modifying the weights of the network's heads. I kept the weights of the backbone froze them only for the first 15.000 steps, which corresponds roughly to 75 iterations. In the second run, I didn't perform any weight freezing. Final metrics achieved in both runs are very similar. Top-1 accuracy for guessing the next move is 57% and the ownership mask is MSE $= 0,0012$, both evaluated on a validation set consisting of samples not seen in training.

It should be noted that in positions in which no player's optimal move is sente and the last played move is not sente, the network has only 50% chance of guessing who played the next move in the game (or even less, if there is more than one optimal move for a player). Only in positions with sente one could hope to observe near 100% accuracy. While a proportion of such positions in the training dataset is unknown, a top-1 accuracy surpassing 50% can already be considered an achievement.

5.5. Results

To evaluate how well the program works, I use authentic endgame problems used for training by Go players. I took a set of 100 problems for counting values of moves prepared for an endgame course at the GoMagic platform [Fre22]. In these problems, the ground truth value of temperature is available. For each of the problems, I marked the intersections belonging to the local positions by hand. Moreover, I added a few stones to the positions in many of the problems to more clearly mark the borders of secure territories, and this way to ensure that the ownership assessments are as expected. Importantly, I do not surround the territories by chains of Black and White stones which are provably alive, contrary to what was needed when using the classical approach.

The program is designed in a way that it can work with different models used. In my tests, I tried a few approaches which differ by how the local score was assessed and how the children nodes were picked for a given node during expansion.

1. KataGo baseline - Use the existing KataGo network for both local score assessment (using the ownership map) and top local moves taken from the policy head output (the best move strategy of node expansion)

2. A0jax using the best move strategy - Choice of best moves and local score assessment are based on policy head and ownership head outputs of A0jax

3. A0jax using the threshold strategy with phantom values

4. A0jax using the threshold strategy with expansion of depth 1

5. A0jax moves + KataGo score - Pick the children nodes based on A0jax output using the threshold strategy with expansion of depth 1; assess the local score in terminal nodes (and in non-terminal nodes expanded in case of sente) using KataGo ownership maps

6. Ground truth moves + KataGo score - use a game tree prepared by a human expert, containing only the correct variations

In case of using only KataGo model, the threshold node expansion strategy cannot be used because KataGo does not provide a relative probability of which color will be the next one to play in a given position. KataGo network in one forward pass predicts moves only for one of the players, indicated in the input tensor, unlike the fine-tuned A0jax network which outputs probabilities for both players.

The final approach which uses ground truth moves is useful for checking how good the CGT algorithm used during backup works, and how reliable the KataGo ownership assessments are.

Model	**Accuracy**
KataGo best moves	14%
A0jax best moves (**ours**)	18%
A0jax phantom values (**ours**)	21%
A0jax depth-1 expansion (**ours**)	29%
A0jax depth-1 expansion (**ours**) + KataGo scores	55%
GT moves + KataGo scores	97%

Figure 5.2: Program performance on the test set of 100 problems from the GoMagic course

The results show that guiding the game tree construction with my fine-tuned A0jax network offers a great improvement compared to extracting the best local moves from KataGo's policy head. The KataGo baseline succeeds to correctly analyze only the simplest problems from the set in which the local position is clearly bounded by chains of Black and White stones and only a few legal moves are available at each moment. Analysis of game trees built using KataGo's policy head reveals that often they contain nonsensical moves which would not be even considered by beginner Go players.

Building the game trees with moves suggested by A0jax offers an improvement even when used coupled with the best move strategy that does not leverage an information about sente predicted by the network. When this information is used, a bigger class of problems can be correctly analyzed. However, the best results are achieved when the local score is assessed using ownership maps predicted by KataGo and not the ones predicted by A0jax. This is not particularly surprising as the A0jax is a much smaller network, trained for much less number of steps to approximate ownership maps predicted by KataGo.

Noteworthy, the program does not achieve 100% accuracy even when used with ground truth variations prepared by a human expert. Reasons are two-fold. In some of the positions, the expected score assessed in a non-leaf node added because of the *depth-1 expansion* is so inaccurate that a sente move ends up not being recognized as sente by the program[2]. This is an inherent problem connected to the expansion strategy used and it can be mitigated only by designing a new strategy or changing the model that assesses local scores.

The other problem comes from an incorrect final score assessment in terminal positions. Even though KataGo's ownership map is highly accurate when it comes to secure territories, the local positions in the test set are not bounded by provably alive stones. As a result, sometimes a position which should be considered terminal from a human perspective still has a potential weakness left which makes KataGo not consider some of the intersections as a secure territory.

Apart from the program's accuracy, another consideration is its speed. Some of the approaches result in bigger game trees which means that the neural network inference needs to be run more times. For this, utilizing the information is sente is particularly useful as it allows to not expand big parts of the full game tree as they cannot be reached under orthodox play and are therefore irrelevant for temperature calculations. Moreover, the approaches using only the A0jax network work 3 times faster than when the KataGo inference is also run on the nodes, as the latter network is much bigger.

[2]The most prominent example of this issue is problem 4.08 (see Appendix B for a hyperlink to the test set). In this problem, one could clearly see the reason of KataGo's erroneous assessment. The continuation of a sente move is by itself a sente move threatening to kill Black's group. The threat is so large that the local position becomes the most urgent part of the board. KataGo's network recognizes it and predicts the local ownership under the assumption that Black will surely play there next.

Chapter 6

Discussion and future work

The program described in the previous chapter facilitates analysis of positions appearing in the game of Go which was not possible using solutions existing before. The novelty is a capability to calculate temperatures of local positions appearing in actual games without a need of manually bounding them by chains of provably alive stones. This being said, the program does not work perfectly, and return correct values only on 55% of problems from the test set.

In the previous chapter, I presented the choices I made while creating the program. In this chapter, I give a rationale for these choices and discuss on whether different solutions might be also possible. Further experimentation might lead to the improvement of model's results.

6.1. Tree construction

6.1.1. Tackling loopy games

The current work adds an enhancement to the temperature finding algorithm which allows for correct temperature assessment in a wide class of positions in kos. The enhancement is designed for simple loopy games as defined in [Sie13], i.e. positions for which in every node of a ko chain each player has at most one move which captures a ko. My algorithm works correctly for positions in which every player has only one exit from the ko chain. However, it might fail if the ko chain is long (what is called *multi-stage ko*) and one player has another exit from it (if capturing one of the kos by the opponent is sente).

A more sophisticated algorithm for dealing with loopy games would improve the program's performance in positions with more rarely seen types of kos.

6.1.2. Node expansion strategies

There are numerous ways in which one could expand a node based on the next move predictions of a model. Within the strategies described in Section 5.1, one could experiments with different thresholds for assessing that a move is sente or that the position is terminal.

If a threshold chosen for sente assessment is too lenient, new nodes would be added to a tree, even though they should be irrelevant for temperature computations. What it means in practice is that the program would start to consider numerous variations of what could happen if a player gets into the opponent's secure territory. Very often in Go a secure territory consists of a large area of empty intersections, so the number of possible variations after the

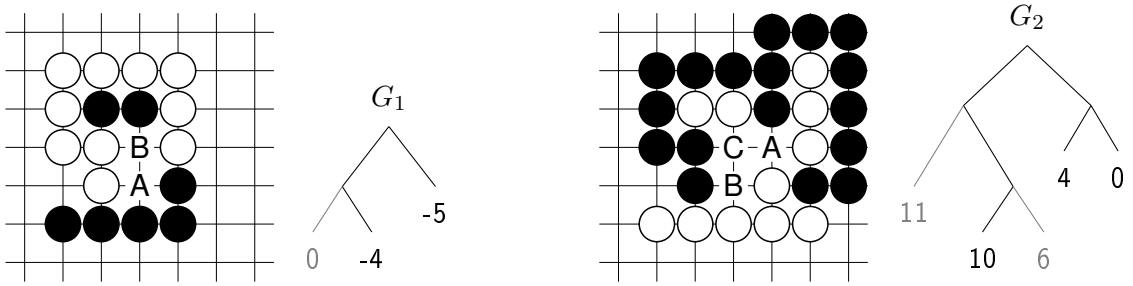


Figure 6.1: Two different types of positions in which a network is going to predict low probability of Black's (Left's) next move.

invasion might be huge. Constructing the part of the tree corresponding to it might be computationally too expensive leading to the program's failure.

On the other hand, if the threshold is too strict, correct moves might be omitted during tree construction and the calculated local temperature might be wrong.

Nodes added in case of sente moves

I experiment with two strategies of adding nodes for a player for whom the probability of getting the next move is below the threshold. Both of these strategies have their downsides.

Adding a phantom node with an extreme value works correct in case that the previous move was sente. In this case, we assume that the player could get *really a lot* by playing the next move so the opponent should not allow that. For temperature calculations it does not matter how big exactly that hypothetical gain is so I set it to value `inf` if Black's move is considered and `-inf` otherwise. This solution also works correctly in case of *reversible options* - situations in which a player's move is counted as sente not because of its continuation being at least twice bigger than the move value, but because of the position before the previous opponent's move being better for the opponent than the position after that move and the player's answer.

However, there is also another class of positions in which one could expect the next move probability for one of the colors being much higher than for the other one. This can happen if the previous move played in the position was not sente, but the next move for one of the players is going to be sente once the whole-board temperature drops below a certain value.

The issue is illustrated on Figure 6.1. Game trees correspond to the local positions shown on the go boards. In both cases, after Black's move, the probability of Black getting the next move assigned by a well-trained network should be low. However, the reason is different.

In game G_1, Black's move at A is sente, and White needs to answer at B. In such case, the grayed out node in the tree can be well replaced by a phantom `inf` value and the temperature calculations will still be correct.

On the other hand, in game G_2 Black's move at A is gote. Only after the whole-board temperature drops below 2, will White play the move at B forcing Black to answer at C. If however, White followed by immediately playing the move at B, Black would not have to answer. The $A - B$ exchange changed the mean value of the game from 6 to 8 so it was profitable for Black.

Replacing the grayed out node representing the next Left's option (11) with a phantom `inf` value would amount to assuming that White needs to immediately answer at B. Then, we would estimate the mean value of the whole game as 8, and the temperature as 6, whereas

in reality the temperature equals 4.

In case if no sente moves appear deeper in the tree after gote moves, the *threshold strategy with phantom values* works correctly. However, in cases like G_2 expanding nodes for both colors is necessary. Unfortunately, with the current shape of the model's output, it is impossible to distinguish these two cases. Therefore, the *threshold strategy with depth-1 expansion* is more robust and gives better results.

The latter strategy also has its downsides though. The local score assessment for non-terminal nodes is in principle not accurate. If the estimation error happens to be large, sente moves might be in turn not recognized as sente in the game tree.

Further discussion on how these issues could be mitigated follows in Section 6.2

6.1.3. More informed ways of constructing the game tree

A common idea of all tree building strategies tried by me is to add child nodes based on a model prediction for that node. Once all nodes are expanded in this way, the CGT game of the root node can be set and then the local temperature can be calculated. In practice, I calculate means and temperatures of CGT games in non-root nodes before the full tree is built, so that I can terminate the whole process and raise an exception if any of the intermediate CGT games is malformed. However, I do not use the results of these computations to aid the tree construction in any way.

A possible enhancement of the tree building algorithm might be to try to incorporate means and temperatures computed for non-root nodes to decide on how to continue expanding the nodes. One idea of how to use such information would be to add more child nodes to a node for which the CGT game happened to be malformed and the temperature calculation failed. A reason for such failure could be that the correct move was not considered for one of the players.

Moreover, one could try to set approximations of CGT games for all nodes after each algorithm iteration, without waiting until all leaf nodes below a given node are judged as terminal. To this goal, one could utilize expected local scores computed from KataGo's ownership maps in non-terminal nodes.

A hypothetical tree building algorithm utilizing such information would be much more sophisticated than the one in the current work. However, the idea to base the tree construction on intermediate results from non-terminal nodes is very natural. Such approach is used in other tree building algorithms, e.g. in MCTS.

6.2. Considerations about the model design

6.2.1. Choice of the base model

Nowadays, there is a number of publicly available strong Go-playing programs based on AlphaZero approach. Making use of one of them is a good starting point for training a model that could predict moves in local positions. Neural networks of programs such as KataGo, LeelaZero, ELF OpenGo were trained for millions of steps on the course of many GPU days, accumulating a deep understanding of the game.

Trying to train a network from scratch for the local move prediction task would require more training time and likely, the network would have hard time learning sophisticated playing techniques.

Fine-tuning is a common technique in deep learning. In several fields, a researcher can make use of an existing model which was pre-trained specifically with the goal of facilitating

fine-tuning on different downstream tasks. Examples include Detectron model for semantic segmentation [HGDG17] and wav2vec for speech recognition [BZMA20].

Unfortunately, there are no Go playing program designed with a similar motivation. There is no API to easily design a new task and create a new network head for it. To fine-tune a Go playing model, one needs to understand the underlying code which might be a tedious task.

When choosing a pre-trained model, my main consideration was the cleanliness of the codebase so that is is easy to modify it. The A0jax network I chose [Ngu22] does not have a state-of-the-art playing strength but it is anyway very strong. Thus, I assumed that its latent states are going to contain meaningful representations of the board positions, facilitating fine-tuning for my downstream task.

The assumption was confirmed by an ablation study in which I trained the same network on the same data but using a random weight initialization instead of pre-trained weights. The learning curve was much steeper and the training metrics never matched those from the official run.

It is possible that one could achieve even better results by fine-tuning a state-of-the-art Go-playing program such as KataGo. Not only was KataGo trained for many more steps than A0jax, but also its author introduced many novelties compared to the vanilla AlphaZero algorithm which gave the network an incentive to learn more versatile information about board positions and aimed at making it more robust.

6.2.2. Model's output

In Section 5.1 I outlined what pieces of information a model should assess so that a game tree expansion can be guided by it. Apart from the next move predictions, the model should also provide an information about sente and whether the position is terminal. Moreover, it is necessary to compute the local score in terminal nodes but this could be done well using an existing KataGo network.

In Section 6.1, it was shown that another piece of information might be helpful: a distinction between nodes in which the previous move was sente and nodes in which the next move is sente.

One could think of multiple ways in which all this information can be represented in model's output. However, the main question is always about constructing training data for a given output shape.

6.2.3. Moves played in games as policy head targets

In my model, the policy head output one tensor jointly assigning probabilities for next Black's moves, next White's moves and of the position being terminal. As the positions were taken from KataGo self-play games, it was possible to use the next move played within the masked region as a target for the policy head.

Technically, we do not want the network to predict just one move for a position but two: the best local move for Black and the best local move for White (or even more, if more than one move is optimal for one of the players). We would like to see the network assigning close to 50% probability to one of them, close to 50% for the second one, and close to 0% for all other moves. The only exceptions would be positions with sente moves where we would expect a high probability only for one of the players.

At first, it might seem concerning that in training we treat only one move for each position as a target. However, in case of positions with no sente moves the network has no way to

guess which of the players gets the next move, so the network's outputs should converge to the desired probability distribution.

To address the issue mentioned in Section 6.1, one could try to incorporate another piece of information in the network's target: whether the previous move was sente. This information can be assessed by checking if the previous move in the game was played in the given local position and the next one is also played in it.

Then, one can change the shape of the network's output tensor, extending it by another 361 numbers which would correspond to immediate local answers. Alternatively, one could keep the shape of the output tensor but design a new output for the network which would be one number denoting the probability that the next move in the game will be played within the given local position. Such output could facilitate a more sophisticated way of constructing game trees.

6.2.4. Designing new training data

Using positions from KataGo self-play games and moves played in them as targets is a pragmatic solution which allows for a quick creation of millions of training samples. However, such training data is not of the best quality.

First, the targets used are not a ground truth which we would like the network to predict. In particular, in positions where the temperature drops very slightly after a move, the chance that the opponent answers to it immediately is quite high, even though the move is formally speaking not sente. Moreover, even if a move is sente, it is not guaranteed that the opponent answers immediately because in practice, the opponent might for example play their own sente move elsewhere on the board.

Moreover, positions that are evaluated during game tree construction often include many moves played in a row in one local position (the network's input consists of 8 previous board states). In self-play games, such situations happen very rarely.

To create training data of better quality, one could try to take positions met during game tree constructions and find ground truth targets for policy head (and also for ownership head) based on CGT calculations. However, creation of such data requires a well-working program evaluating local temperatures.

One could try to use the program created in the current work to produce such training data. The program does not work perfectly, but the produced data could potentially still be of higher quality than the original training data. If as a result a better model could be trained, it can be in turn used to produce training data of even better quality.

This is an analogous idea to the one used in AlphaZero training scheme where data produced by earlier versions of a network were used to create new stronger versions, making use of the MCTS algorithm that provided a policy improvement operator. Such process would require more experimentation than the straightforward approach used in the current project. It can be a subject of future work.

6.2.5. Board decomposition

To perform an analysis of a local position, it needs to be first found within a whole-board position. During the training process, I do it in an automatized way, making use of ownership maps predicted by KataGo. During inference, I require the user to mark the region by hand.

Noteworthy, the heuristic used by me during training is not always correct. I assume that connected components of intersections not belonging to secure territories are independent of each other. In practice, far-distance dependencies between such regions are possible.

An example is shown on Figure 6.2. Positions around intersections on the left and on the right are parsed as independent. However, if White ever gets both A and B moves, Black's three stones will be in danger, as the move at C could capture them. The dependence would not be recognized by the proposed method, as Black is always going to respond to such threat, so the three stones are considered safe by KataGo.

Such situations are another weak point of the training data. To recognize dependencies between far regions, one would need to perform CGT analysis of all undecided regions on the board and see if in some variations the positions are no longer parsed as independent. However, this is not a straightforward task.

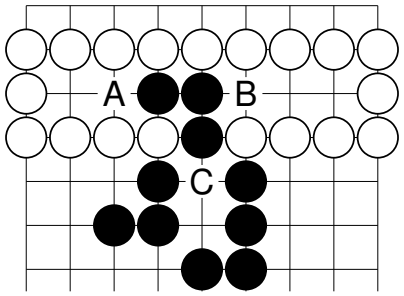


Figure 6.2: *Unconnected group invading multiple corridors* scenario [BW94a, p. 26]

Concerning masking local positions during inference, it would be very helpful to also do it in an automatized way. If the program is to be used as a teaching tool, it would be very convenient for a user to set a local position simply by choosing one intersection belonging to it, without the need of manual selection of all relevant intersections. It could be done using the same heuristic as during training (though, as showed above, the heuristic is not flawless).

However, choosing the right size of the local region is not an obvious task. In particular, it should consist not only of intersections with undecided ownership, since variations in local game trees would often include moves that threaten to take away intersections belonging to the opponent's secure territory. A pragmatic solution would be to dynamically adjust the region of interest based on new nodes added to a game tree.

6.3. Future applications

The program developed in the current work is the first attempt to utilize deep learning techniques in the context of the CGT applied to Go endgames. Results achieved on real endgame problems can serve as a proof of concept that game tree construction guided by neural network predictions might allow for correct analysis of positions which was not possible to achieve with classical techniques.

The program's performance is not perfect and in particular, it is worse than analyses performed by a human expert. In the previous Sections, possible directions of future work were outlined which could lead to higher accuracy.

If a successor of the program created in this work achieves a better performance, it can be used in several applications.

First of all, a correct analysis of local positions in Go endgames might in principle provide a better guidance for choosing moves than the procedure used in AlphaZero-based programs (i.e. MCTS algorithm guided by neural network predictions). Potentially, it could set a new record of playing strength in the endgame phase of Go.

Moreover, decisions made by the program developed in this work provide a much better explainability than decisions of the state-of-the-art Go-playing programs. Therefore, the program (or its better performing successor) can make a good teaching tool for Go players. First of all, it evaluates temperatures in local positions which is also a basis of analysis performed by Go players. Furthermore, these judgments are based on a game tree constructed by the program which might be viewed by a user to understand which local variations the program considers to be optimal.

Bibliography

[Ber96] Elwyn R. Berlekamp. The economist's view of combinatorial games. 1996.

[BW94a] E. Berlekamp and D. Wolfe. *Mathematical Go: Chilling Gets the Last Point*. CRC Press, 1994.

[BW94b] Elwyn Berlekamp and David Wolfe. *Mathematical Go - chilling gets the last point*. 02 1994.

[BZMA20] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *CoRR*, abs/2006.11477, 2020.

[Con76] John H. Conway. On numbers and games. 1976.

[Cou07] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers, editors, *Computers and Games*, pages 72–83, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[DGSS22] Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2022.

[Fre20] Stanisław Frejlak. Katago's yose: Go endgame theory and deep residual networks. 2020. `https://github.com/siasio/EndgameBot/blob/main/1000-LIC-MAT-297313.pdf`.

[Fre22] Stanisław Frejlak. Video course "endgame for nerds". *Go Magic*, 2022. `https://gomagic.org/courses/endgame-for-nerds/`.

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[Gir15] Ross Girshick. Fast r-cnn, 2015.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[LF24] Łukasz Lew and Stanisław Frejlak. Finding temperatures through iterative pseudo-stop search. 2024.

[Lib] Sensei's Library. Temperature (cgt). `https://senseis.xmp.net/?TemperatureCGT`.

[MKT+21] Thomas McGrath, Andrei Kapishnikov, Nenad Tomasev, Adam Pearce, Demis Hassabis, Been Kim, Ulrich Paquet, and Vladimir Kramnik. Acquisition of chess knowledge in alphazero. *CoRR*, abs/2111.09259, 2021.

[Mül95] Martin Müller. Computer go as a sum of local games: an application of combinatorial game theory. 1995.

[Mü01] Martin Müller. Global and local game tree search. *Information Sciences*, 135(3):187–206, 2001. Heuristic Search and Computer Game Playing.

[Ngu22] Thong Nguyen. Alphazero in jax using deepmind mctx library., 2022. `https://github.com/NTT123/a0-jax`.

[Now17] Steve Nowee. Directing attention of convolutional neural networks. Master's thesis, 2017.

[RHGS16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[SHM+16] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[Sie13] Aaron N. Siegel. Combinatorial game theory. 2013.

[Spi01] William L. Spight. Extended thermography for multiple kos in go. *Theoretical Computer Science*, 252(1):23–43, 2001. CG'98.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, L. Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

[TMG+19] Yuandong Tian, Jerry Ma, Qucheng Gong, Shubho Sengupta, Zhuoyuan Chen, James Pinkerton, and Larry Zitnick. A new elf opengo bot and analysis of historical go games. 2019.

[TMG+22] Yuandong Tian, Jerry Ma, Qucheng Gong, Shubho Sengupta, Zhuoyuan Chen, James Pinkerton, and C. Lawrence Zitnick. Elf opengo: An analysis and open reimplementation of alphazero, 2022.

[TV19] A. Törmänen and Hebsacker Verlag. *Rational Endgame*. Hebsacker, Steffi, 2019.

[TZ16] Yuandong Tian and Yan Zhu. Better computer go player with neural network and long-term prediction, 2016.

[Wol02] David Wolfe. Go endgames are pspace-hard. 2002.

[Wu20] David J. Wu. Accelerating self-play learning in go, 2020.

Appendix A

Repository and graphical interface

The codebase of the project presented in this work is available on GitHub: `https://github.com/siasio/EndgameBot/`

As a part of the project, I developed a graphical user interface for the program. A user could set up a local position by putting stones on a board, or by loading a file in the standard format used for Go called `sgf`. After the program analyzes the position, an expanded game tree is shown in a right-hand-side panel. The user can navigate through the tree and check the program's evaluations for every node in the tree. The GUI can be started by running script `visualize.py`.

For the sake of running tests over many local positions, I also developed script `benchmark.py` which has no graphical interface. The results presented in Section 5.5 were collected using this script. Output files produced by the script are available in directory `output/`.

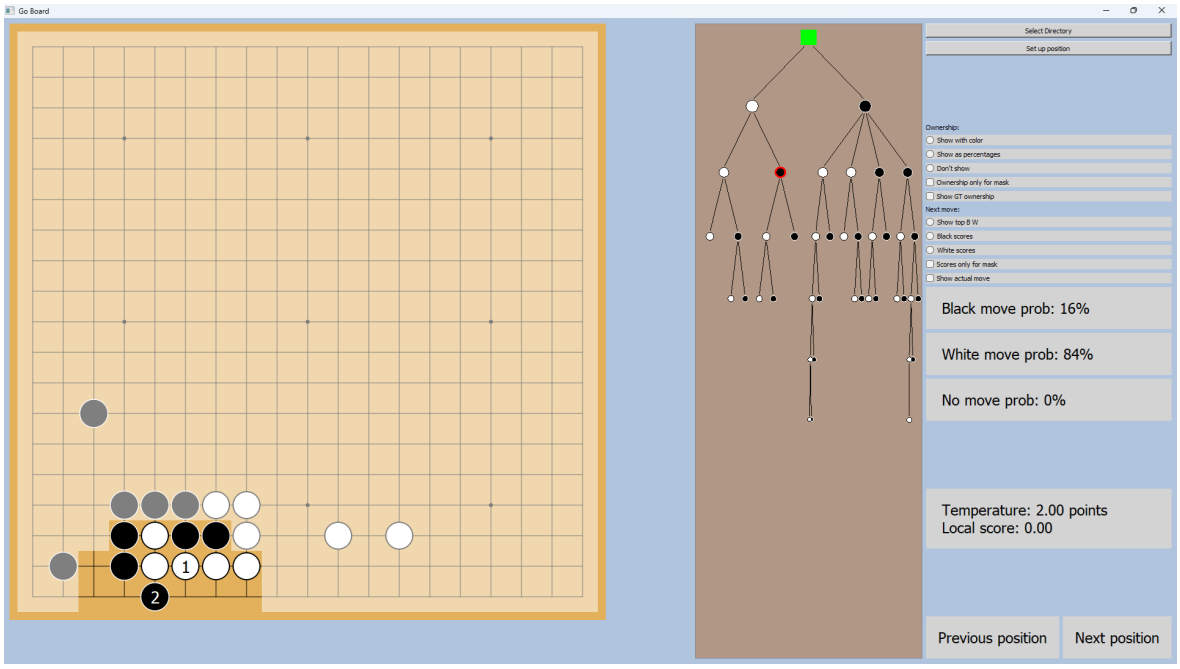


Figure A.1: A screenshot of the graphical interface with the program evaluating temperature in one of the positions from the test set

Appendix B

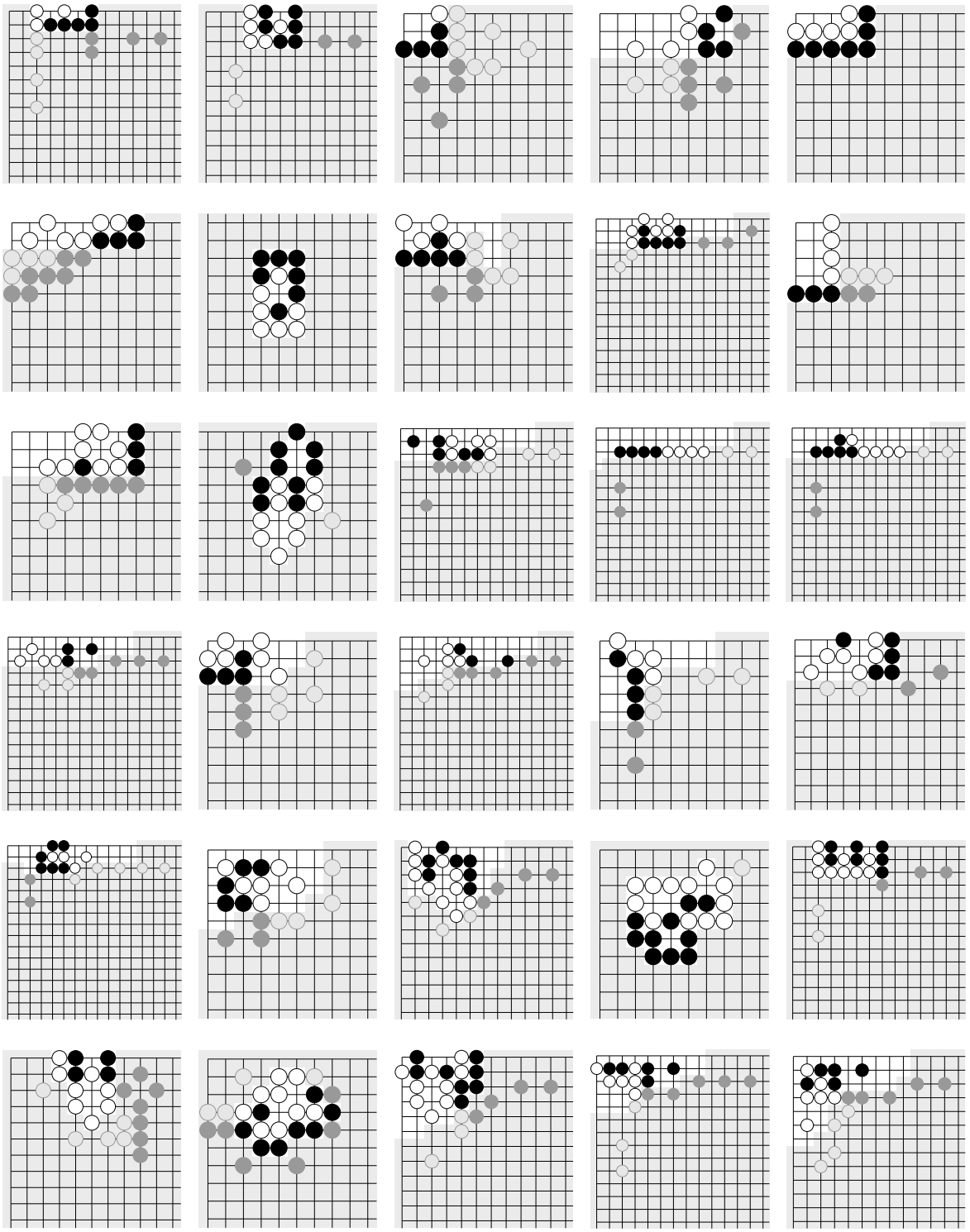
Train and test data

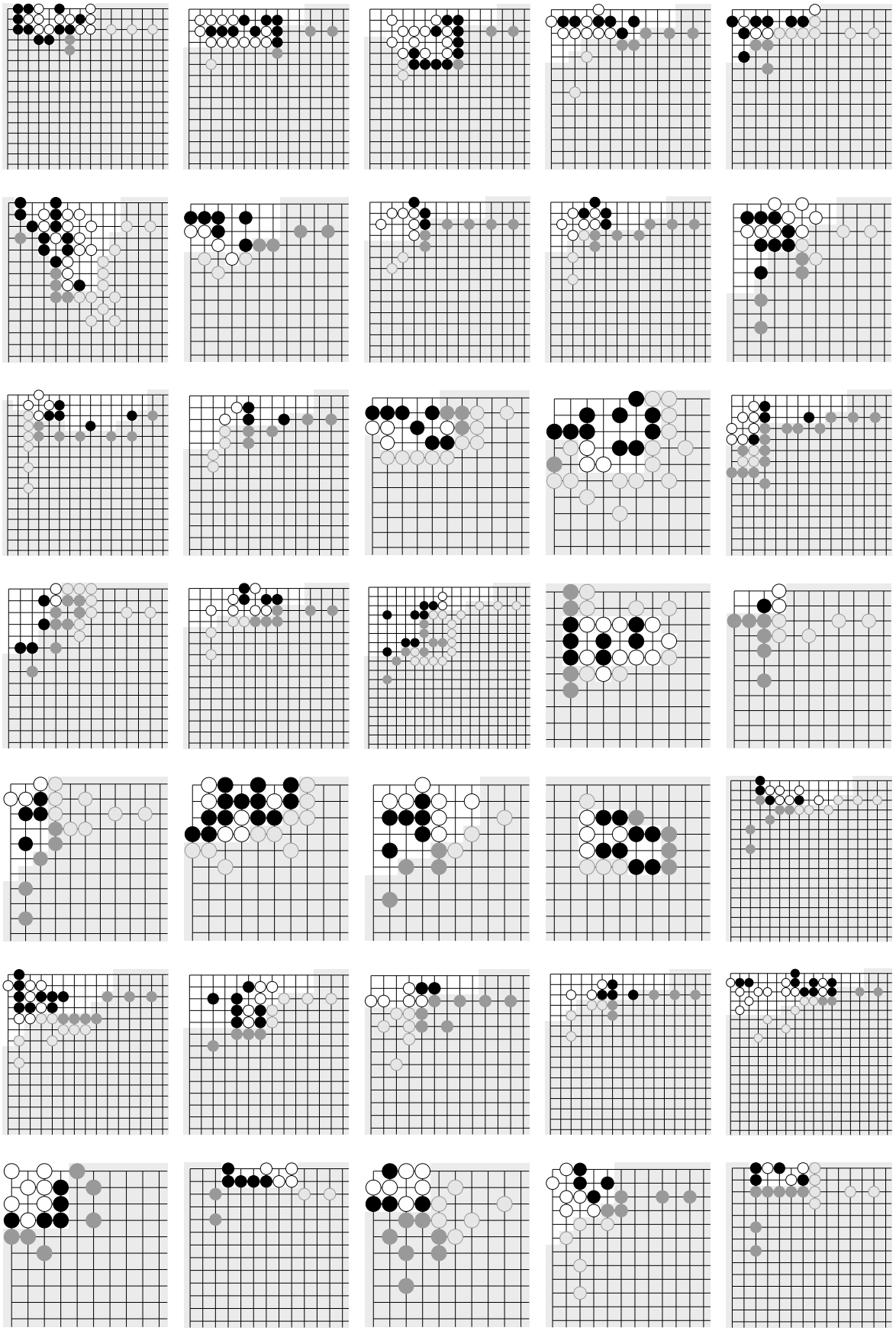
To make the results of this work reproducible, I publish the training data on Google Drive: `https://drive.google.com/drive/folders/14YMGO1dC4FWuEwNj-FWLqWUjB_5es7G8`

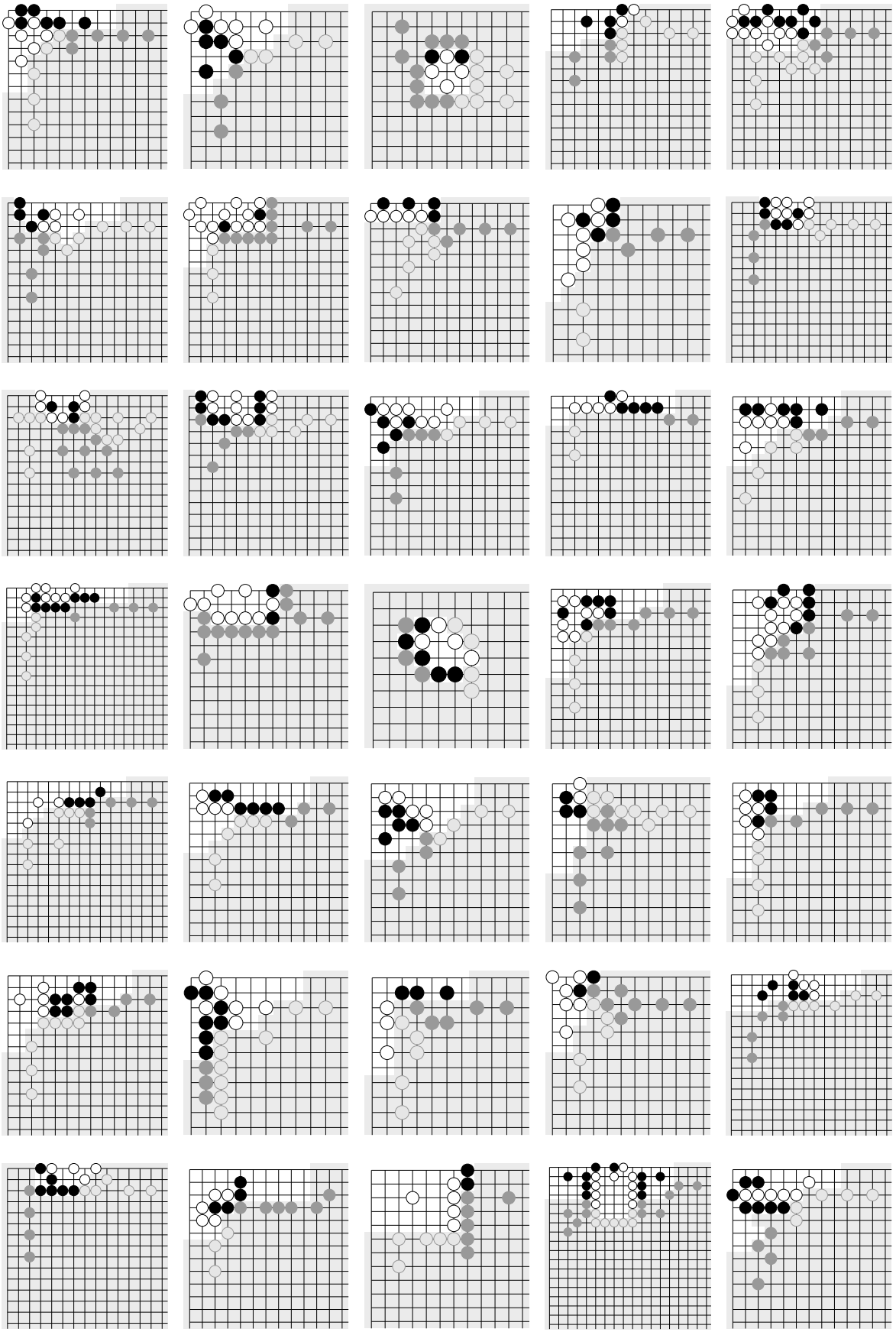
There are two subdirectories: `train_data/` and `eval_data/` with the joint size of 4,8`GiB`. The evaluation data is constructed in the same way as the training data but it is not seen by the network during training. The `train_data/` folder includes `zip` archives with pre-processed positions taken from KataGo selfplay games. Pre-processing consists of picking a single position from a game and running KataGo over it to get the ownership map. To make this data ready for feeding into a network, further processing is needed which is done in `a0-jax/train_agent.py` script. I do not publish post-processed training data as they take nearly 20 times more disk space. In `eval_data/` folder, the post-processed data is published as a `pkl` file.

Below, I show 100 problems from the GoMagic course [Fre22] used as a test set. I also publish it as a `zip` archive containing 100 `sgf` files in the project's GitHub repository in directory `assets/`. They contain the ground truth game trees prepared by an expert.

The test data can be used by the community as a benchmark dataset for testing new solutions for the problem of evaluating temperatures in Go endgames.







Appendix C

Frejlak-Criado beast

In Go jargon, exotic positions which are difficult to analyze from the perspective of endgame calculations or from the perspective of Go rules are dubbed *beasts*. Perfect play in such positions would often include not intuitive moves and might differ depending on the specific ruleset of Go.

The position depicted on Figure C.2 below induces a truly bizarre perfect play. White needs to save their big group from dying and the only way to do it is to capture the ko at A. At that point, Black can (and should) play elsewhere on the board. White is forced to continue locally by capturing the next ko, and Black can (and should) again play elsewhere. Then, White captures the next ko, threatening to kill Black's group. Black is forced to use a ko threat at B to avoid dying. White answers to the threat, and the whole local position reaches a state symmetric to the starting position.

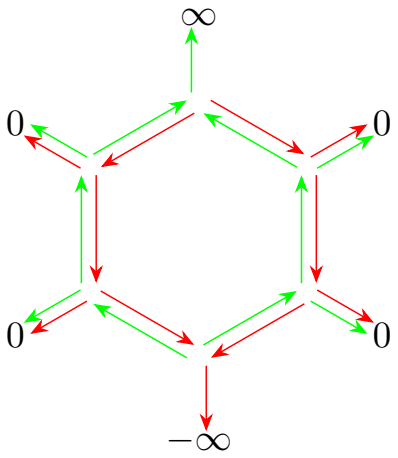


Figure C.1: Abstract game graph designed by Dr. Criado which was an inspiration for constructing the position shown on C.2. Green edges represent Left options, red edges - Right options

As a result, when the Frejlak-Criado beast stands on the board, under perfect play Black will be entitled to play two moves in a row in the environment, then White will play two moves in a row in the environment, and so on. It completely opposes the usual course of Go games with alternating moves of both players. The most striking impact can be observed in positions with sente moves where a player might be not able to answer the opponent's big threat.

From the perspective of the Combinatorial Game Theory, presence of the Frejlak-Criado beast as a subgame of a game G raises the advantage which a player can get from playing in the environment. In consequence, it raises a potential error of the *orthodox forecast* for the game result and the upper bound for the loss which a player might incur under orthodox play. In terms of thermography, left and right walls of Frejlak-Criado beast's thermograph have slopes $-\frac{1}{2}$ and $\frac{1}{2}$ respectively on one segment.

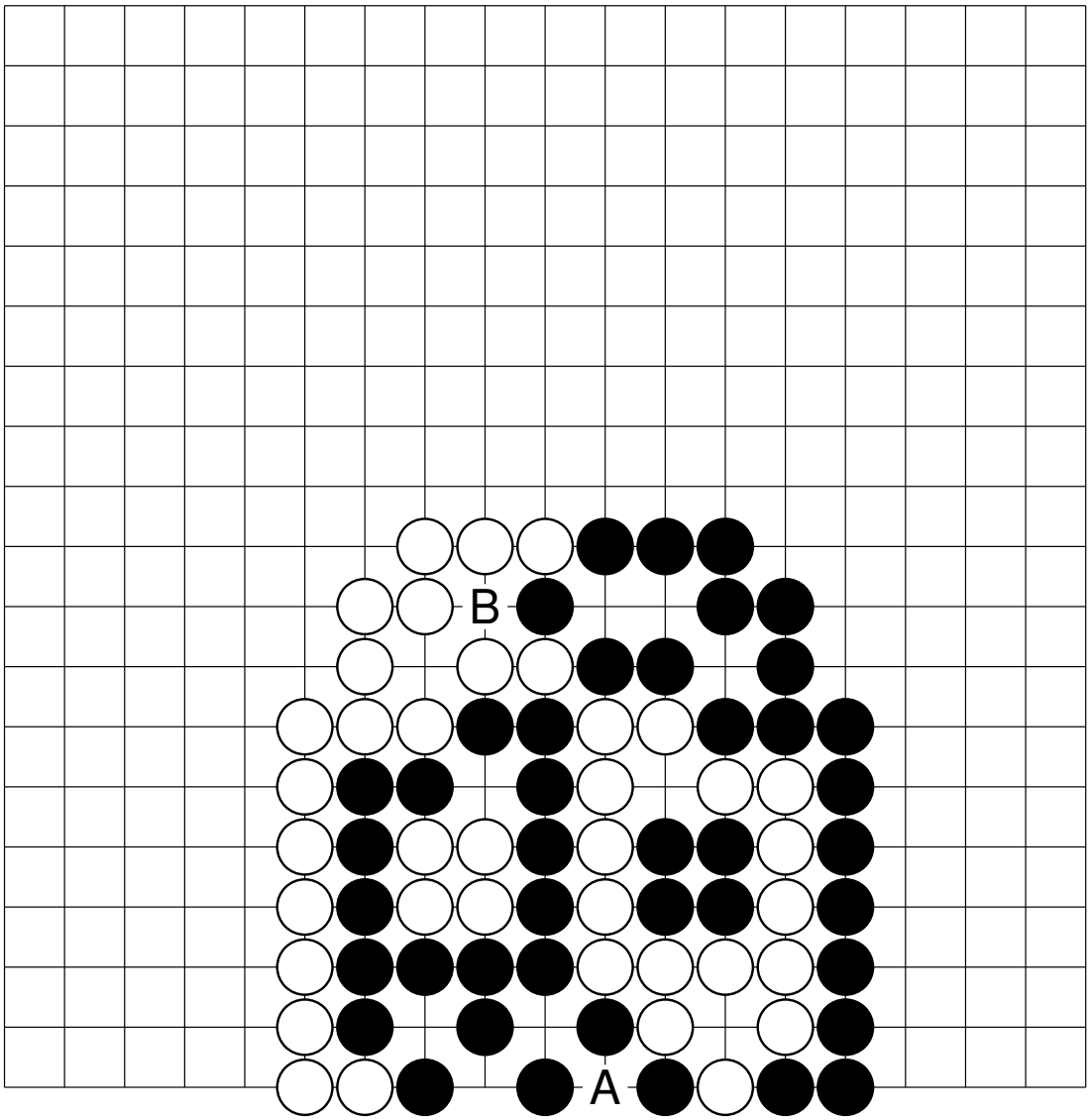


Figure C.2: **Frejlak-Criado beast** 3-stage ko with a swapping local threat