LEELA ZERO SCORE: A STUDY OF A SCORE-BASED ALPHA GO ZERO

A PREPRINT

Luca Pasqualini*
Department of Information Engineering and Mathematical Sciences
University of Siena
Siena, 53100 Italy
pasqualini@diism.unisi.it

Maurizio Parton
Department of Economical Studies
University of Chieti-Pescara
Pescara, 65127 Italy
maurizio.parton@unich.it

Francesco Morandin
Department of Mathematical, Physical and Computer Sciences
University of Parma
Parma, 43124 Italy
francesco.morandin@unipr.it

Gianluca Amato
Department of Economical Studies
University of Chieti-Pescara
Pescara, 65127 Italy
gianluca.amato@unich.it

Rosa Gini
ARS Toscana
Florence, 55144 Italy
rosa.gini@ars.toscana.it

Carlo Metta
KDD Lab
ISTI-CNR Pisa
Pisa, 56127, Italy
carlo.metta@isti.cnr.it

February 1, 2022

ABSTRACT

AlphaGo, AlphaGo Zero, and all of their derivatives can play with superhuman strength because they are able to predict the win-lose outcome with great accuracy. However, Go as a game is decided by a final score difference, and in final positions AlphaGo plays suboptimal moves: this is not surprising, since AlphaGo is completely unaware of the final score difference, all winning final positions being equivalent from the winrate perspective. This can be an issue, for instance when trying to learn the “best” move or to play with an initial handicap. Moreover, there is the theoretical quest of the “perfect game”, that is, the minimax solution. Thus, a natural question arises: is it possible to train a successful Reinforcement Learning agent to predict score differences instead of winrates? No empirical or theoretical evidence can be found in the literature to support the folklore statement that “this does not work”. In this paper we present Leela Zero Score, a software designed to support or disprove the “does not work” statement. Leela Zero Score is designed on the open-source solution known as Leela Zero, and is trained on a 9×9 board to predict score differences instead of winrates. We find that the training produces a rational player, and we analyze its style against a strong amateur human player, to find that it is prone to some mistakes when the outcome is close. We compare its strength against SAI, an AlphaGo Zero-like software working on the 9×9 board, and find that the training of Leela Zero Score has reached a premature convergence to a player weaker than SAI.

1 Introduction

The game of Go has been a landmark challenge for AI research since its very beginning. It is very suited to AI, with the importance of patterns and the need for deep exploration, and very tough to actually solve, with its whole-board features

*Corresponding author: http://sailab.diism.unisi.it/people/luca-pasqualini/
and subtle interdependencies of local situations. Nowadays, AI has reached superhuman level in the game of Go with the well-known DeepMind algorithm AlphaGo Zero [1] (AGZ), a zero-knowledge evolution of AlphaGo [2] (AG). More generally, any perfect information two-player zero-sum game like Go should be tackled efficiently by DeepMind algorithm AlphaZero [3] (AZ).

In perfect information two-player zero-sum games, where the win-lose outcome is given by a final score difference, maximizing this score difference is still an open and important question, see the detailed discussion in [4, Introduction]. In fact, AG derivatives play suboptimal moves in the endgame, see for instance [5, moves 210 and 214, page 252]. The open-source clean room implementation of AGZ known as Leela Zero [6] (LZ) is also known to play suboptimal moves, see Section 4.4 in [7].

This phenomenon is rooted in the win-lose reward in the Reinforcement Learning (RL) pipeline. Giving a reward of 1 (win) or 0 (lose) at the end of the game means that AG derivatives maximize the winrate instead of the actual score difference. It is a folklore statement that replicating the AG pipeline using score instead of the binary outcome as a primary target is unsuccessful. A qualitative argument is that score is unlikely to be a successful reward, because a single point difference may change the winner, thus inducing instability in the training. As a matter of fact, the two most recent and successful RL approaches to score maximization in the game of Go, that is, KataGo [8] and SAI [7], have taken different routes. KataGo does include score estimation, but only as a secondary target: the value to be maximized is a linear combination of winrate and expectation of a nonlinear function of the score difference, not the score difference itself. In SAI, the winrate is modeled as a two-parameters family of sigmoids \( \sigma_{\alpha,\beta} \): while \( \alpha \) can be seen as the final score difference, \( \alpha \) and \( \beta \) are learnt indirectly by training \( \sigma_{\alpha,\beta} \) against the classical binary reward.

Still, humans do use score estimations instead of winrate estimations while playing score-based games. Therefore, the question remains: if a Deep Reinforcement Learning (DRL) agent can learn how to maximize its winrate, why should it not be possible to learn how to win by maximizing the final score? The implications from a DRL training process perspective are far from trivial.

This paper is meant to provide the RL community with a sound and quantified direct evidence of the limitations of training a DRL agent directly on the score difference. To this aim we train an instance of LZ on the 9×9 board, using score as a target. We name this instance Leela Zero Score (LZS). We show that the training is successful, see 6.1, but converges prematurely to a player weaker than a corresponding AGZ-like player, see [0,2].

2 Background: Leela Zero and SAI

Free and open source, Leela Zero (LZ) [6] is a Go program with no human provided knowledge, known as one of the most faithful reimplementations of the system AlphaGo Zero described in [1]. For all intents and purposes, it is considered an open-source AlphaGo Zero (AGZ). The agent plays using MCTS with a deep residual convolutional neural network stack and without Monte Carlo playouts.

LZ was initially released on 25 October 2017. The neural networks powering the agent of LZ were trained by a distributed effort, which was coordinated at the LZ website. Lacking the computational power required to train AlphaGo Zero, members of the community provided computing resources by running the client, which generates self-play games and submits them to the server.

The self-play games were used to train newer networks. Newer networks were then matched against current best in multiple games and promoted according to a process known as gating. These games are known as matches.

The training process of LZ ended on 15 February 2021. LZ is available at [https://zero.sjeng.org/] and [https://github.com/leela-zero/leela-zero].

SAI is a fork of LZ created by the authors of this paper, and described thoroughly in [4,5,7,10]. While Go is usually played on a 19×19 board, and LZ was trained on this size, SAI was trained also on the 9×9 board, of much simpler complexity. For the purposes of this work, SAI 9×9 can be considered an AGZ-like software.

3 Setup of Leela Zero Score

The overall architecture of the open-source engine LZ was replicated in our setting. We implemented LZS on a reduced \( N \times N \) board size, where \( N = 9 \), for efficiency purposes.

The second main difference between LZ and LZS is in the target. To implement a score instead of a binary target during the training and inference phases, the following issues were addressed.
• The outcome: instead of a binary flag, the target became an integer number \( s \in \{-N^2, \ldots, N^2\} \), \( N \) being the Go board size.

• The winrate: the role of a probability between 0.0 and 1.0 in comparing the possible moves according to the MCTS tree was taken by the expected score, which is a number \( s_e \in [-N^2, N^2] \), where \( N \) is the Go board size.

• To minimize the modifications to the existing MCTS code in the LZ software, the outcome \( s \) and the expected score \( s_e \) were both normalised to \( \hat{s} = \frac{s}{N^2} \) and \( \hat{s}_e = \frac{s_e}{N^2} \) in \([-1.0, 1.0]\).

• The heuristics: during the training, LZ employs a set of heuristics during self-plays to avoid useless sets of actions at the end of the match. For example, there is no point for a player to keep playing if it wins by passing. This heuristic doesn’t work anymore in our new setting: to maximize score, the agent should keep playing so long it has a chance of increasing its current score.

• No resigning threshold was used.

Scaling down the board from size \( n = 19 \) to size \( \rho n = 9 \) with \( \rho < 1 \) yields several benefits.

• Average number of legal moves at each position scales down by \( \rho^2 \).

• Average length of games scales down by \( \rho^2 \).

• The number of visits in the MCTS tree scales down by \( \rho^4 \), which can be inferred from the previous two.

• The number of layers in the residual convolutional neural network stack scales by \( \rho \).

• The fully connected layers at the end of the neural network stack are smaller. This grants increased training and inference speed.

To summarize the performance benefits of scaling down the board dimension from \( 19 \times 19 \) to \( 9 \times 9 \), we can estimate a total speed improvement for self-play games in the order of \( \rho^9 \).

4 Training

Our training was composed by multiple phases, inspired from the original LZ training process, as well as general knowledge inferred from SAI training. Specifically, the phases of each training cycle were as follows.

• Self-play. A set of 2,000 self-plays per cycle, where the network plays Go against itself using the modified LZ engine with the following parameters: a variable amount of visits \( v \), increased randomness while playing the first 15 moves, a set of 6 threads, a batch size of 5, policy noise randomization and specific heuristics for passing during the game.

• Training. The network is trained over a window of self-plays. Specifically, the most recent self-plays are downloaded and added to the previously downloaded ones, if available. Then, positions from all the self-plays within a variable window \( w \) are extracted and used as data, see (2), in a supervised learning fashion with a training and test set.

• Gating. A set of 400 matches played between the new trained network and the current best network using the modified LZ engine is played. Parameters are set as follows: a variable amount of visits \( v \), no randomness in the moves, a set of 6 threads, a batch size of 5, no policy noise randomization and no heuristics for passing during the game.

• Promotion. Depending on the results of the gating step, the old best network is maintained or replaced. Specifically, if at least 55% of the 400 matches are won by the candidate network, it is promoted to be the new best network. It’s important to note that while we train the network to maximize its score, through gating we are assessing its capabilities in winning the game. In other words, the most desirable outcome is to obtain a network good at winning through score maximization.

In all games, white had a 7.5 points komi, that is, an additional 7.5 points were added to its score at the end, to compensate the fact that black plays the first move. After the end of each cycle, a new cycle starts. When a network is promoted to be the new best network, the current training generation number increases. At some values of the
generation number $g$, the hyperparameters $v(g)$ and $w(g)$ are updated, where $v(g)$ is the amount of visits in MCTS tree during self-plays, and $w(g)$ is the size of the training window, as follows:

$$
v(g) = \begin{cases} 
100 & \text{if } g \leq 15 \\
150 & \text{if } 15 < g \leq 31 \\
250 & \text{if } 31 < g \leq 47 \\
400 & \text{if } 47 < g \leq 63 \\
600 & \text{if } 63 < g \leq 79 \\
850 & \text{otherwise} 
\end{cases}
$$

(1)

$$
w(g) = \begin{cases} 
4 & \text{if } g \leq 15 \\
8 & \text{if } 15 < g \leq 31 \\
12 & \text{if } 31 < g \leq 47 \\
16 & \text{if } 47 < g \leq 63 \\
20 & \text{if } 63 < g \leq 79 \\
24 & \text{otherwise} 
\end{cases}
$$

(2)

5 Outcome of the training

In Figure 1 we show training results in terms of uncalibrated Elo rating. On the $x$-axis the amount of self-plays are shown. On the $y$-axis the uncalibrated Elo is reported. Like in LZ [6], the blue circle represents matches won by the new trained network, i.e. when a promotion happens, while pink triangles show when such matches fail to promote the last trained network. The Elo estimate is uncalibrated, because it is based on setting to 0 the Elo of the first network, which was chosen arbitrarily as a random network. Moreover, the Elo estimate is uncalibrated because the Elo of new networks was estimated only with the gating matches, and this clearly yields a positive bias. A calibrated estimate of the Elo rating is described in the next sections.

However, for the purposes of the training, the uncalibrated estimate was sufficient, as it allowed to assess when the training process stalled: when the new trained network was not able to beat the previous best network for an empirically-chosen amount of cycles, we scaled up the network size. Starting from a base network with 2 residual convolutional layers of 64 filters, from now on referred to as $2 \times 64$, we scaled to:

- $4 \times 128$ at $g = 1$, after 2,000 self-plays.
- $8 \times 160$ at $g = 25$, after 150,000 self-plays.
- $10 \times 192$ at $g = 43$, after 720,000 self-plays.
- $12 \times 256$ at $g = 50$, after 908,000 self-plays.

Similarly, we changed the learning rate $l_r$ of training at certain times during the entire process, with the goal of reducing stalls. Such times are selected empirically, depending on the size of the network and the current generation. As a simple rule of thumb, we reduced the learning rate when we trained bigger networks at reasonably high generation numbers and we saw very little improvements from one set of 2,000 self-plays to the next. Specifically, we trained with $l_r = 0.02$ for $g < 44$ and with $l_r = 0.002$ for $g \geq 45$.

Based on the expectation from the SAI 9×9 run, a stopping rule for the training was decided a priori, that the run would have been stopped when no promotion was obtained after 40 cycles, i.e., 80,000 self-plays. The rule was met after 1,400,000 self-plays.

6 Evaluation

To assess the strength of LZS, we ran both a qualitative and a quantitative evaluation.
6.1 Qualitative evaluation against a human player

Fifteen games were played between the best LZS network and Carlo Metta, a strong amateur player\(^2\). The SGFs with comments are available as ancillary files with this paper on arXiv. Ten games were played with 400 visits for each move, that is, the same setting as games in the quantitative evaluation, see Section 6.2, while five games were played with 20,000 visits for each move, to test a stronger player.

A thorough analysis of such games shows that training has been successful in producing a consistent player, which, however, exhibits some unusual characteristics when compared to other artificial agents. The match ended with a score of 14–1 in favor of the human player: although LZS found itself in a position of clear advantage several times, it was only able to win one game, one of those with 20000 visits. LZS showed some peculiar and not always desirable features. LZS certainly has a direct and aggressive style. It does not seem to admit sacrificing few stones for a better final result (e.g., SGF\(^2\) Carlo_vs_Leela_Score_400_visit_game_6, move 17), nor to foresee sacrifice on the opponent’s side (e.g., SGF\(^2\) Carlo_vs_Leela_Score_400_visit_game_9, move 16). This is clearly in contrast with the flexibility shown by other artificial agents.

Another striking situation occurred several times (e.g., SGF\(^2\) Carlo_vs_Leela_Score_400_visit_game_8, move 21): when in balanced positions, LZS attempted to further increase the score difference, rather than settling for a narrow victory, in such an aggressive and self-delusional way that it resulted in an inevitable defeat. It may be argued that this phenomenon was a direct effect of the LZS training scheme.

6.2 Quantitative evaluation against SAI

For a quantitative evaluation of the training process of LZS, we needed a calibrated Elo rating for LZS networks promoted during the training. To get a sensible anchoring, we selected a LZS panel of 14 networks, one every four LZS promoted networks, and compared their strength against a calibration panel of 32 SAI networks of similar strength. Elo ratings are computed with a maximum likelihood optimization algorithm, similar to Rémi Coulom’s Bayes Elo \([11]\). All the matches in this evaluation step were played with the following settings.

- LZS: \(v = 400\), no randomness in the moves, a set of 6 threads, a batch size of 5, no policy noise randomization and no heuristics for passing during the game.

---

\(^2\)Player profile on the European Go Database: \([https://www.europeangodatabase.eu/EGD/Player_Card.php?&key=14713996]\).
• SAI: \( v = 400 \), no randomness in the moves, a set of 6 threads, a batch size of 5, no policy noise randomization, resign threshold of 1%, no heuristics for passing during the game, \( \lambda = 0 \) and \( \mu = 0 \).
• LZS and SAI played black at alternate times.
• Komi of 7.5 points.

To select the calibration panel of SAI networks, we first estimated the Elo rating of the best LZS network using the panel from [7, Section 3.7]. LZS lost all games against the second weakest net of the panel, which had a Elo of 3500. We therefore chose a second, weaker panel, namely, SAI
\[2000\), SAI
\[2500\), SAI
\[3000\), and SAI
\[3500\), where SAI
\[x\) is a SAI network (of the principal run in [7]) having Elo rate approximately \(x\).

After 200 games between the best LZS network and the preliminary panel, LZS had the following winrates:

\[
w = \begin{cases} 
0.985 & \text{against SAI}_{2000} \\
0.925 & \text{against SAI}_{2500} \\
0.480 & \text{against SAI}_{3000} \\
0.155 & \text{against SAI}_{3500} 
\end{cases}
\]

Based on these results, we chose as calibration panel a set of 32 SAI networks, whose Elo ranged from 683 to 3501. This calibration panel was eventually used to play matches of 20 games with the panel of 14 LZS networks. For efficiency reasons, only those matches were played between pairs of opponents whose difference of strength made the results informative from a statistical point of view. To make this selection, we calibrated roughly the Elo of the LZS panel based on the Elo range between SAI
\[2000\) and SAI
\[3500\). In Figure 2, the SAI panel and LZS networks are detailed, as well as the uncalibrated and roughly calibrated Elo, and the set of matches that were eventually played.

Figure 2: Match table between LZS networks (columns) and the calibration panel of SAI networks (rows). In both row and columns generations, hashes, and baseline Elo ratings are displayed. In columns, the roughly calibrated Elo rate used to choose which matches would be played is displayed as well. In the cells, the relative Elo differences are displayed. Colors indicate relative strengths based on the comparison between SAI’s Elo and LZS’ roughly calibrated Elo: green for stronger LZS, yellow for similar strength, red for stronger SAI.

The total amount of 20-games matches was 207. The results of the matches were fed into the Elo algorithm. To make this step more robust, we also used all matches available from [7] between the 32 networks of the SAI calibration panel. In Figure 3, we compare LZS with the run of SAI described in [7] as run 1, which converged after a comparable number of self-plays. This comparison shows that, during training, LZS had consistently lower values of Elo. Moreover, this calibrated version of Figure 1 confirms that the last increase in the size of the network, at 900,000 self-plays, had not produced any relevant improvement in the next 400,000 self-plays, thus confirming that LZS was converging prematurely at a weaker player. After 200,000 games the two curves are approximately parallel, with a Elo difference of around 1,500 points out of 5,500. This is a remarkable difference in strength: the interpretation of the Elo formula means that LZS would win against SAI, on average, 1 out of \(10^{1500/400}\) times, i.e., 1 out of 5623 times.
7 Conclusions

We trained an instance of 9×9 LZ with target score, using the same pipeline as the pipeline of 9×9 SAI, a network trained with the traditional binary target. The training was successful, and produced LZS, a player with valid play. A strong human amateur analyzed the style of LZS and found qualitative characteristics that reflected the score-based training, such as the inability to sacrifice points to improve winning chances. The networks of the training run were shown to be consistently weaker than the corresponding networks in the SAI training run. The training converged prematurely, to a Elo level lower by approximately 1,500 points, thus implying that the strongest LZS would win against the strongest SAI less than once out of 5,000 times.

This provides a sound and quantified direct evidence of the limitations of training a DRL agent directly on the score difference.

Figure 3: Calibrated Elo ratings of various LZS networks and associated SAI 9×9 networks in the first run, expressed w.r.t. the amount of self-plays.
References

[1] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676 (2017), pp. 354–359.

[2] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.

[3] David Silver et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144.

[4] Francesco Morandin et al. “SAI a Sensible Artificial Intelligence that plays Go”. In: *International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary*. 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852266

[5] Antti Törmänen. *Invisible: the games of AlphaGo*. Hebsacker Verlag, 2017. ISBN: 978-3937499062.

[6] Gian-Carlo Pascutto and contributors. *Leela Zero*.[http://zero.sjeng.org](http://zero.sjeng.org) [Online; accessed 21-December-2021].

[7] Francesco Morandin et al. “SAI: A Sensible Artificial Intelligence That Plays with Handicap and Targets High Scores in 9×9 Go”. In: *ECAI 2020 - 24th European Conference on Artificial Intelligence, Santiago de Compostela, Spain*. Vol. 325. Frontiers in Artificial Intelligence and Applications. 2020, pp. 403–410. DOI: 10.3233/FAIA200119

[8] David J Wu. “Accelerating Self-Play Learning in Go”. In: *arXiv* (2019). arXiv: [1902.10565](https://arxiv.org/abs/1902.10565)

[9] Francesco Morandin et al. “SAI: a Sensible Artificial Intelligence that plays with handicap and targets high scores in 9x9 Go (extended version)”. In: *arXiv* (2019). arXiv: [1905.10863](https://arxiv.org/abs/1905.10863)

[10] Gianluca Amato and contributors. *SAI: a fork of Leela Zero with variable komi*[https://github.com/sai-dev/sai](https://github.com/sai-dev/sai) [Online; accessed 21-December-2021].

[11] Rémy Coulom. *Bayesian Elo Rating*.[http://www.remi-coulom.fr/Bayesian-Elo](http://www.remi-coulom.fr/Bayesian-Elo) [Online; accessed 21-December-2021].