Alternative loss functions in AlphaZero-like self-play

Wang, H.; Emmerich, M.T.M.; Preuss, M.; Plaat, A.

2019 IEEE Symposium Series on Computational Intelligence (SSCI)
December 6-9 2019, Xiamen, China

Alternative Loss Functions in AlphaZero-like Self-play

Hui Wang, Michael Emmerich, Mike Preuss and Aske Plaat
Leiden Institute of Advanced Computer Science
Leiden University
The Netherlands
Email: h.wang.13@liacs.leidenuniv.nl

***Abstract*—Recently, AlphaZero has achieved outstanding performance in playing Go, Chess, and Shogi. Players in AlphaZero consist of a combination of Monte Carlo Tree Search and a deep neural network, that is trained using self-play. The unified deep neural network has a policy-head and a value-head, and during training, the optimizer minimizes the sum of policy loss and value loss. However, it is not clear if and under which circumstances other formulations of the loss function are better. Therefore, we perform experiments with different combinations of these two minimization targets. In contrast to many recent papers who adopt single run experiments and use the whole history Elo ratings from self-play, we propose to use repeated runs. The results show that this method can describe the training performance quite well within each training run, but there is a high self-play bias, such that it is incomparable among different training runs. Therefore, inspired by the AlphaGo series papers, a self-play bias avoiding performance assessment, final best player Elo rating, is adopted to evaluate the playing strength in a direct competition between the evolved players. For relatively small games, based on this new evaluation method, surprisingly, minimizing only value loss achieves the strongest playing strength in the final best players' round-robin tournament. These results indicate that more research is needed into the relative importance of value function and policy function in small games.**

***keywords*—AlphaZero-like self-play; loss combination; Elo evaluation.**

The AlphaGo series of papers [1]–[3] have sparked enormous interest of researchers and the general public alike into deep reinforcement learning. AlphaGo Zero [2], the successor of AlphaGo, masters the game of Go even without human knowledge. It generates game playing data purely by an elegant form of self-play, training a single unified neural network with a policy head and a value head, in a Monte Carlo Tree Search (MCTS) searcher. AlphaZero [3] uses a single architecture for playing three different games (Go, Chess and Shogi) without human knowledge. Many applications and optimization methods [4], [5] have been published and transformed the research field into one of the most active of current computer science.

Despite the success of AlphaGo and related methods in various application areas, there are unexplored and unsolved puzzles in the design and parameterization of the algorithms. The neural network in AlphaZero is represented as $f_\theta = (\mathbf{p}, v)$ (a unified deep network with a policy head and a value head). Policy $\mathbf{p}$ is a probability distribution of choosing the best move. A lower policy loss ($l_\mathbf{p}$) indicates a more accurate selection of the best move. Value function v is the prediction of the final outcome. A lower value loss (l_v) indicates a more accurate prediction of the final outcome. The use of a double-headed network by Alpha(Go) Zero is innovative, and we know of no in-depth study of how the two losses ($l_\mathbf{p}$ and l_v) contribute to the playing strength of the final player. In Alpha(Go) Zero the sum of the two losses is used. Other studies based on the AlphaGo series algorithms just use it that way. However, the finding in the work of Matsuzaki et al. [6] is different, which reminds us to carefully study alternative evaluation functions. Thus, In order to increase our understanding of the inner workings of the minimization of the double-headed network we study different combinations of policy and value loss in this paper. Therefore, in this work, we investigate:

a) what will happen if we only minimize a single target?
b) is a product combination a good alternative to summation?

We perform our experiments using a light-weight AlphaZero implementation named AlphaZeroGeneral [7] and focus on smaller games, namely 5×5 and 6×6 Othello [8], 5×5 and 6×6 Connect Four games [9]. The smaller size of these games allows us to do more experiments, and they also provide us largely uncharted territory where we hope to find effects that cannot be seen in Go or Chess.

As performance measure we use the Elo rating that can be computed during training time of the self-play system, as a running relative Elo. It can also be computed separately, in a dedicated tournament between different trained players. Our contributions can be summarized as follows:

- Experimental results show that there is a high self-play bias in computing training Elo ratings, such that it is incomparable among different training runs. A full tournament is necessary to compare final best players' Elo ratings to measure the playing strength.
- Based on a full tournament computation, in our smaller games, minimizing the value loss gives better results than the summation of the value and the policy loss, contradicting both the default setting of AlphaZeroGeneral and Alpha(Go) Zero, that use the sum of the two losses.

The paper is structured as follows. Part I presents related work. Part II presents games tested in the experiments. Part III

Hui Wang acknowledges financial support from the China Scholarship Council (CSC), CSC No.201706990015.

introduces the AlphaZero-like self-play algorithm (with important parameters and default loss function) and Bayesian Elo system. Part IV sets up the experiments. Part V presents the experimental results. Part VI discusses future work and concludes the paper.

I. Related Work

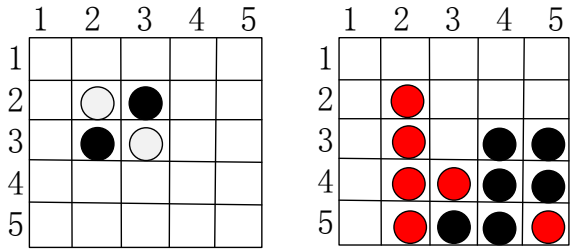
Deep reinforcement learning [10] is currently one of the most active research areas in artificial intelligence, reaching human level performance for difficult games such as Go [11], which was almost unthinkable 10 years ago. Since Mnih et al. reported human-level control for playing Atari 2600 games by means of deep reinforcement learning [12], the performance of Deep Q-networks (DQN) improved dramatically.

We have also observed a shift in DQN from imitating and learning from expert human players [1] to relying more on self-play. This has been advocated in the area of reinforcement learning [13], [14] for quite some time already. Silver et al. [2] turned to self-play to generate training data instead of training from human data (AlphaGo Zero), which not only saves a lot of work of collecting and labeling data from human experts, but also shifts the constraining factor for learning from available data to computing power, and achieves a form of efficient curriculum learning [15]. This approach was generalized to a framework (AlphaZero), showing the same approach that worked in Go, also worked in Shogi and Chess, demonstrating how to transfer the learning process [3].

Reinforcement learning is a very active field. We see a move away from human data to self-play. After many years of active research in MCTS [16], currently most research effort is in improving DQN variants. AlphaGo is a complex system with many tunable hyper-parameters. It is unclear if the many choices concerning parameters and methods that have been made in the AlphaGo series are close to optimal or if they can be improved by, e.g., changing parameters [17]. This includes the choice of minimization tasks (loss functions) used for measuring training success. For instance, [18] studied policy and value network optimization as a multi-task learning problem [19]. Even if the choices were very good for Go and other complex games, this does not necessarily transfer well to less complex tasks. For example, AlphaGo's PUCT achieves better results than a single evaluation function, but the result in [6] is different while playing Othello. Moreover, [20] showed that the value function has more importance than the policy function in the PUCT algorithm for Othello.

II. Test Games: Othello/Connect Four

In our experiments, we use the games Othello and Connect Four, each with 5×5 and 6×6 board sizes. Othello is a two-player game. Players take turns placing their own color pieces. Any opponent's color pieces that are in a straight line and bounded by the piece just placed and another piece of the current player's are flipped to the current player's color. While the last legal position is filled, the player who has most pieces wins the game. Fig. 1(a) is the start configuration for 5×5 Othello. Connect Four is a two-player connection game.



(a) 5×5 Othello (b) 5×5 Connect Four

Fig. 1. Our test games on 5×5 boards

Players take turns dropping their own pieces from the top into a vertically suspended grid. The pieces fall straight down and occupy the lowest position within the column. The player who first forms a horizontal, vertical, or diagonal line of four pieces wins the game. Fig. 1(b) is a game termination example for 5×5 Connect Four where the red player wins the game.

There is a wealth of research on finding playing strategies for these two games by means of different methods. For example, Buro created Logistello [21] to play Othello. Chong et al. described the evolution of neural networks for learning to play Othello [22]. Thill et al. applied temporal difference learning to play Connect Four [23]. Moreover, Banerjee et al. tested knowledge transfer in General Game Playing on small games including 4×4 Othello [24]. Wang et al. assessed the potential of classical Q-learning based on small games including 4×4 Connect Four [25]. Obviously, these two games are commonly tested in game playing.

III. AlphaZero-like Self-play

A. The Base Algorithm

According to [2], [3], the fundamental structure of AlphaZero-like Self-play is an iteration over three different stages (see Algorithm 1).

The first stage is a **self-play** tournament. The computer player performs several games against itself in order to generate data for further training. In each step of a game (episode), the player runs MCTS to obtain an enhanced policy π based on $\mathbf{p}$ provided by f_θ. In MCTS, parameter c is used to balance exploration and exploitation of game tree search. Parameter m is the number of times to run down from the root for building the game tree, where the f_θ provides the value (v) of the states for MCTS. For actual (self-)play, from T' steps on, the player always chooses the best move according to π. Before that, the player always chooses a random move based on the probability distribution from π. After finishing the games, these new examples are normalized as a form of (s_t, π_t, z_t) and stored in D.

The second stage consists of **neural network training**, using data from self-play tournament. During training, there are several epochs. In each epoch (*ep*), training examples are divided into several small batches [26] according to the specific batch size (*bs*). The neural network is trained to minimize [27] the value of the *loss function* which (see (1))

Algorithm 1 AlphaZero-like Self-play Algorithm

function ALPHAZEROGENERAL
  Initialize f_θ with random weights; Initialize retrain buffer D with capacity N
  for iteration=1, ..., I **do**
    for episode=1,..., E **do** ▷ stage 1
      for t=1, ..., $T\prime$, ..., T **do**
        Get an enhanced best move prediction π_t by performing MCTS based on $f_\theta(s_t)$
        Before $T\prime$ step, select an action a_t randomly based on probability π_t, otherwise, select an action $a_t = \arg\max_a(\pi_t)$
        Store example (s_t, π_t, z_t) in D
        Set s_t=excuteAction(s_t, a_t)
      Label reward z_t ($t \in [1, T]$) as z_T in examples
    Randomly sample minibatch of examples (s_j, π_j, z_j) from D ▷ stage 2
    $f_{\theta\prime} \leftarrow$ Train f_θ by performing optimizer to minimize (1) based on sampled examples
    Set $f_\theta = f_{\theta\prime}$ if $f_{\theta\prime}$ is better than f_θ ▷ stage 3
  return f_θ;

sums up the mean-squared error between predicted outcome and real outcome and the cross-entropy losses between **p** and π with a learning rate (*lr*) and dropout (*d*). Dropout is used as probability to randomly ignore some nodes of the hidden layer. This mechanism is used to reduce overfitting [28].

The last stage is **arena comparison**, which is comparing the newly trained neural network model ($f_{\theta\prime}$) with the previous neural network model (f_θ). The player will adopt the better model for the next iteration. In order to achieve this, $f_{\theta\prime}$ and f_θ are compared by playing against each other for n games. If the $f_{\theta\prime}$ wins more than a fraction of u games, it is replacing the previous best f_θ. Otherwise, the $f_{\theta\prime}$ is rejected and the f_θ is kept as current best model. Compared with AlphaGo Zero, AlphaZero does not entail the arena comparison stage anymore. However, we keep this stage for making sure that we can safely recognize improvements.

B. Loss Function

The **training loss function** consists of $l_{\mathbf{p}}$ and l_v. The neural network f_θ is parameterized by θ. f_θ takes the game board state s as input, and provides the value $v_\theta \in [-1, 1]$ of s and a policy probability distribution vector **p** over all legal actions as outputs. $\mathbf{p}_\theta$ is the policy provided by f_θ to guide MCTS for playing games. After performing MCTS, we obtain an improvement estimate policy π. It is an aim of the training to make **p** more similar to π. This can be achieved by minimizing the cross entropy of both distributions. Therefore, $l_{\mathbf{p}}$ can be defined as $-\pi^\top \log \mathbf{p}$. The other aim is to minimize the difference between the output value ($v_\theta(s_t)$) of the s according to f_θ and the real outcome ($z_t \in \{-1, 1\}$) of the game. Therefore, l_v can be defined as a mean squared error $(v - z)^2$. Summarizing, the total loss function of AlphaZero can be defined as (1).

$$l_+ = -\pi^\top \log \mathbf{p} + (v - z)^2 \tag{1}$$

Note that in AlphaZero's loss function, there is an extra regularization term to guarantee the training stability of the neural network. In order to pay more attention to two evaluation function components, instead, we apply simple measures to avoid overfitting such as the **drop out** mechanism.

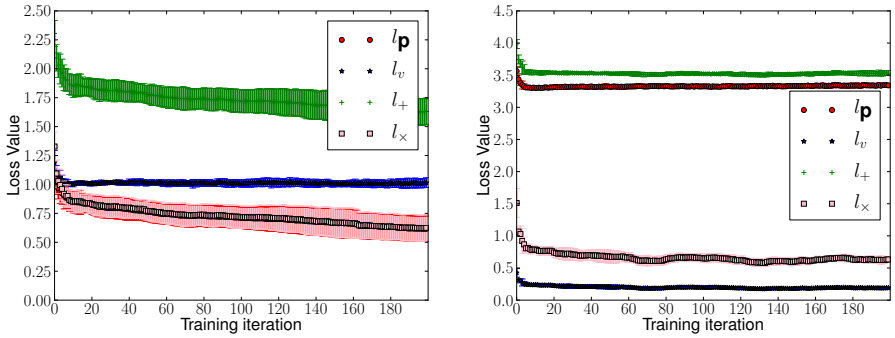
C. Bayesian Elo System

The **Elo rating function** has been developed as a method for calculating the relative skill levels of players in games. Usually, in zero-sum games, there are two players, A and B. If their Elo ratings are R_A and R_B, respectively, then the expectation that player A wins the next game is $E_A = \frac{1}{1+10^{(R_B - R_A)/400}}$. If the real outcome of the next game is S_A, then the updated Elo of player A can be calculated by $R_A = R_A + K(S_A - E_A)$, where K is the factor of the maximum possible adjustment per game. In practice, K should be bigger for weaker players but smaller for stronger players. Following [3], in our design, we adopt the Bayesian Elo system [29] to show the improvement curve of the learning player during self-play process. We furthermore also employ this method to assess the playing strength of the final models.

IV. EXPERIMENTAL SETUP

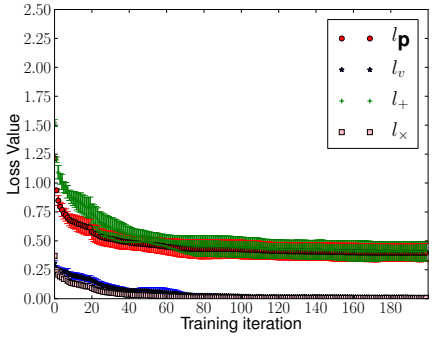
TABLE I
DEFAULT PARAMETER SETTINGS

Parameter	Brief Description	Default Value
I	number of iteration	200
E	number of episode	50
T'	step threshold	15
m	MCTS simulation times	100
c	weight in UCT	1.0
rs	number of retrain iteration	20
ep	number of epoch	10
bs	batch size	64
lr	learning rate	0.005
d	dropout probability	0.3
n	number of comparison games	40
u	update threshold	0.6

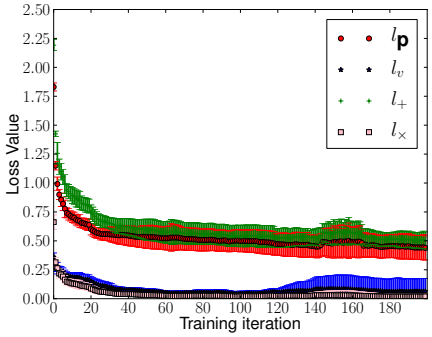


(a) Minimize $l_{\mathbf{p}}$

(b) Minimize l_v



(c) Minimize l_+



(d) Minimize $l_\times$

Fig. 2. Training losses for minimizing different targets in 5×5 Othello, averaged from 8 runs. All measured losses are shown, but only one of these is minimized for. Note the different scaling for subfigure (b). Except for the l_+, the target that is minimized for is also the lowest

Our experiments are performed on a GPU server with 128G RAM, 3TB local storage, 20 Intel Xeon E5-2650v3 CPUs (2.30GHz, 40 threads), 2 NVIDIA Titanium GPUs (each with 12GB memory) and 6 NVIDIA GTX 980 Ti GPUs (each with 6GB memory). On these GPUs, every algorithm training run takes 2∼3 days. In this work, all neural network models share the same structure, which consists of 4 convolutional layers and 2 fully connected layers [7]. The parameter values for Algorithm 1 used in our experiments are given in Table I. In order to enhance reproducibility, we used values based on work reported by [17].

A. Minimization Targets

As we want to assess the effect of minimizing different loss functions, we employ a weighted sum loss function based on (1):

$$l_\lambda = \lambda(-\pi^\top \log \mathbf{p}) + (1-\lambda)(v-z)^2 \tag{2}$$

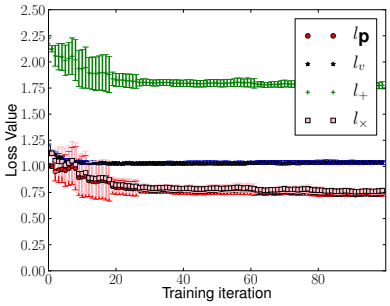
where λ is a weight parameter. This provides some flexibility to gradually change the nature of the function. In our experiments, we first set λ=0 and λ=1 in order to assess $l_{\mathbf{p}}$ or l_v independently. Then we use (1) as training loss function. Furthermore, inspired by that, in the theory of multi-attribute utility functions in multi-criteria optimization [30], a sum tends to prefer extreme solutions, whereas product prefers more balanced solution. We employ a product combination loss function as follows:

$$l_\times = -\pi^\top \log \mathbf{p} \times (v-z)^2 \tag{3}$$

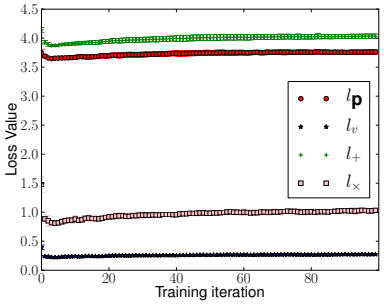
For all experiments, each setting is run 8 times to get statistically significant results (with error bars) using the parameters of Table I as default values. However, in order to save training time, we reduce the iteration number to 100 in the larger games (6×6 Othello and 6×6 Connect Four).

B. Measurements

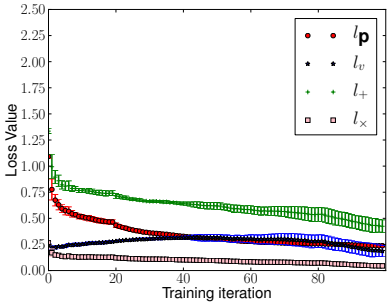
The chosen loss function is used to guide each training process, with the expectation that smaller loss means a stronger model. However, in practise, we have found that this is not always the case and another measure is needed to check. Therefore, following Deep Mind's work, we employ Bayesian Elo ratings [29] to describe the playing strength of the model in every iteration. In addition, for each game, we use all best players trained from the four different targets ($l_{\mathbf{p}}$, l_v, l_+, $l_\times$) and 8 repetitions plus a random player to play the game with each other for 20 times. From this, we calculate the Elo ratings of these 33 players to show the real playing strength of a player, rather than the playing strength only based on its own self-play training.



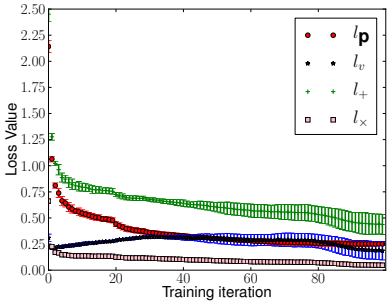
(a) Minimize $l_{\mathbf{p}}$



(b) Minimize l_v

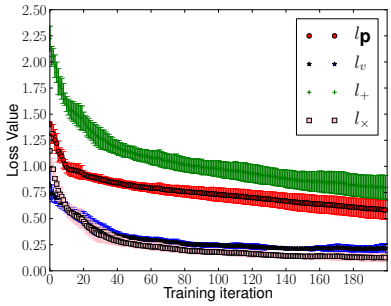


(c) Minimize l_+

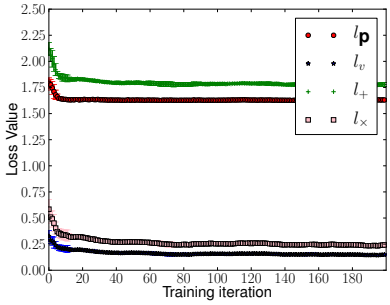


(d) Minimize $l_\times$

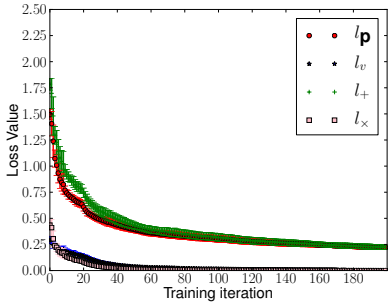
Fig. 3. Training losses for minimizing different targets in 6×6 Othello, averaged from 8 runs. All losses are shown while we minimize only one (similar to Fig 2). Note the different scaling for subfigure (b). Except for l_+, the target that is minimized for is the lowest



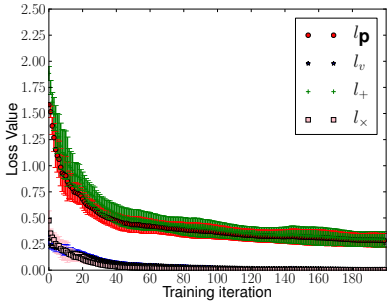
(a) Minimize $l_{\mathbf{p}}$



(b) Minimize l_v



(c) Minimize l_+



(d) Minimize $l_\times$

Fig. 4. Training losses for minimizing the four different targets in 5×5 Connect Four, aggregated from 8 runs. l_v is always the lowest

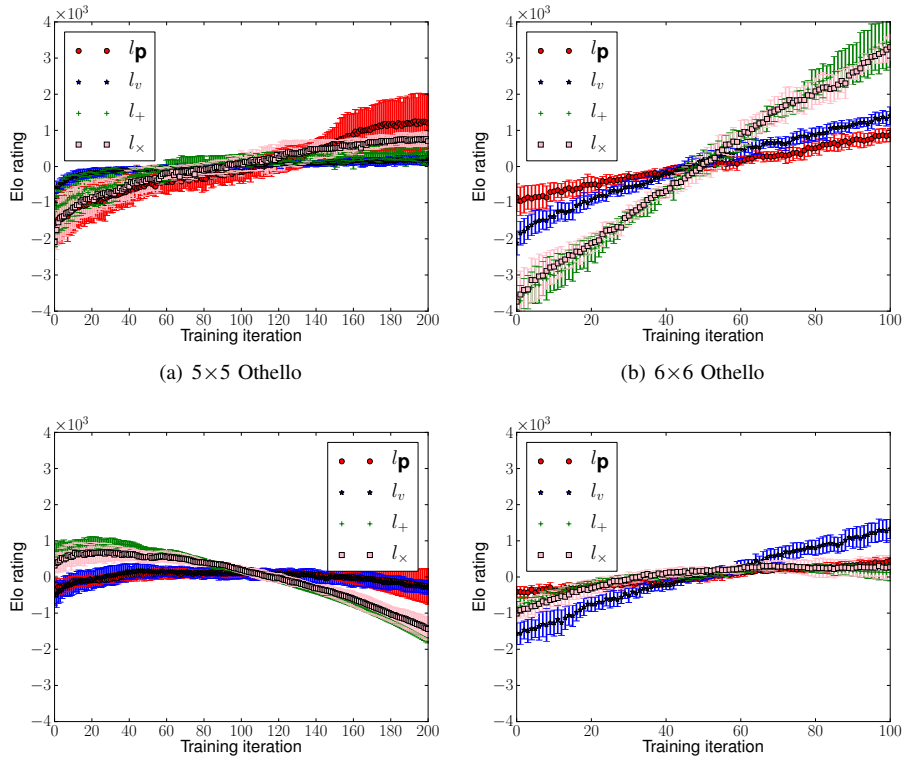
V. Experiment Results

In the following, we present the experimental results from 3 aspects according to the measurements introduced above (i.e. training loss, the whole history training Elo rating and the tournament Elo rating of the final best player). Error bars indicate standard deviation of the 8 runs.

A. Training Loss

We first show the training losses in every iteration during the training phase, with different loss measures, but only one minimization task per diagram, which means we need four of these per game. Therefore, we can see what minimizing for a specific target actually means for the other loss types.

For 5×5 Othello, from Fig. 2(a), we find that when minimizing $l_{\mathbf{p}}$ only, it decreases significantly to about 0.6 at



(a) 5×5 Othello

(b) 6×6 Othello

(c) 5×5 Connect Four

(d) 6×6 Connect Four

Fig. 5. The whole history Elo rating at each iteration during training for different games, aggregated from 8 runs. The training Elo for l_+ and $l_\times$ in panel b and c shows inconsistent results

the end of each training, whereas l_v stagnates at 1.0 after 10 iterations. Minimizing only l_v (Fig. 2(b)) brings it down from 0.5 to 0.2, but $l_{\mathbf{p}}$ remains stable at a high level. In Fig. 2(c), we see that when the l_+ is minimized, both losses are reduced significantly. The $l_{\mathbf{p}}$ decreases from about 1.2 to 0.5, l_v surprisingly decreases to 0. Fig. 2(d), it is similar to Fig. 2(c), while the $l_\times$ is minimized, the $l_{\mathbf{p}}$ and l_v are also reduced. The $l_{\mathbf{p}}$ decreases to 0.5, the l_v also surprisingly decreases to about 0.

For the larger 6×6 Othello, we find that minimizing only $l_{\mathbf{p}}$ reduces it significantly to about 0.75, where l_v is stable again after about 10 iterations (Fig. 3(a)). For minimizing l_v (Fig. 3(b)), the results show that l_v is reduced from more than 0.5 to about 0.25 at the end of each training, but $l_{\mathbf{p}}$ seems to remain almost unchanged. For minimizing the l_+ (Fig. 3(c)), we find in contrast to 5×5 Othello that $l_{\mathbf{p}}$ decreases from about 1.1 to 0.4, whereas l_v increases slightly from about 0.2 and then decreases to about 0.2 again. We also find a similar behavior of l_v when minimizing the $l_\times$ (Fig. 3(d)), with the difference that the final computed loss is much lower as the values are usually smaller than one. However, the similarity of the single losses is striking.

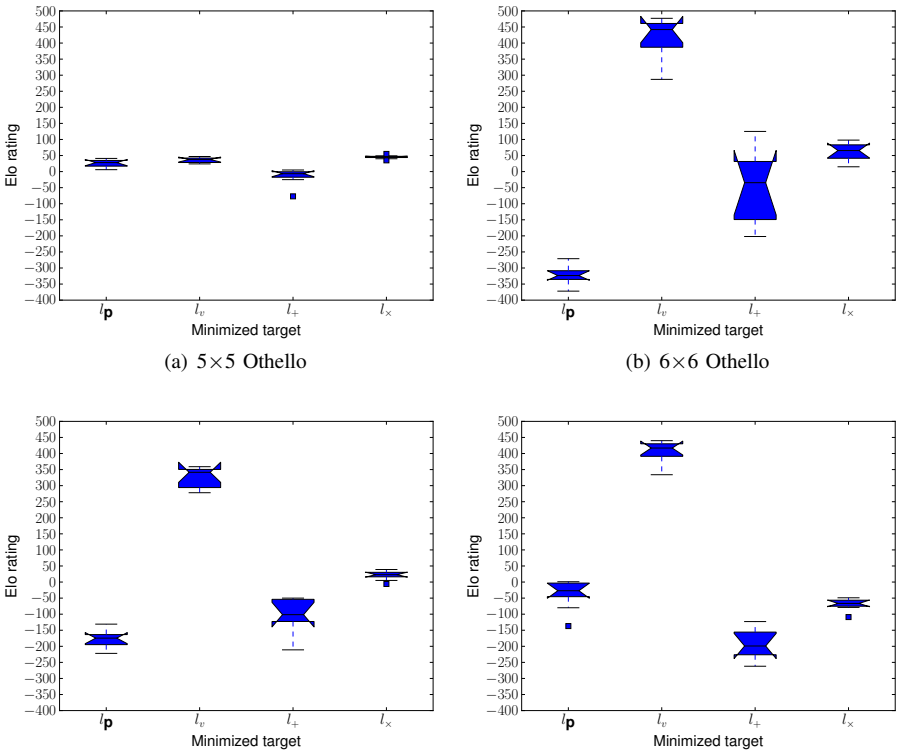
For 5×5 Connect Four (see Fig. 4(a)), we find that when only minimizing $l_{\mathbf{p}}$, it is significantly reduced from 1.4 to about 0.6, whereas l_v is minimized much quicker from 1.0 to about 0.2, where it is almost stationary. Minimizing l_v (Fig. 4(b)) leads to some reduction from more than 0.5 to about 0.15, but $l_{\mathbf{p}}$ is not moving much after an initial slight decrease to about 1.6. For minimizing the l_+ (Fig. 4(c)) and the $l_\times$ (Fig. 4(d)), the behavior of $l_{\mathbf{p}}$ and l_v is very similar, they both decrease steadily, until l_v surprisingly reaches 0. Of course the l_+ and the $l_\times$ arrive at different values, but in terms of both $l_{\mathbf{p}}$ and l_v they are not different. Figures for 6×6 Connect Four (not shown) are very similar to 5×5.

B. Whole History Training Elo Rating

Following the AlphaGo series papers, we also investigate the whole history training Elo rating of every iteration during training. However, these works present results of single training runs, whereas we provide means and variances for 8 runs for each target, categorized by different games in Fig. 5.

From Fig. 5(a) (small 5×5 Othello) we see that for all minimization tasks, Elo values steadily improve, while they raise fastest for $l_{\mathbf{p}}$. In Fig. 5(b), we find that for 6×6 Othello version, Elo values also always improve, but much faster for the l_+ and $l_\times$ target, compared to the single loss targets.

Fig. 5(c) and Fig. 5(d) show the Elo rate progression for training players with the four different targets on the small and larger Connect Four setting. This looks a bit different from the Othello results, as we find stagnation (for 6×6 Connect Four) as well as even degeneration (for 5×5 Connect Four). The latter actually means that for decreasing loss in the training phase, we achieve decreasing Elo rates, such that the players



(a) 5×5 Othello (b) 6×6 Othello

(c) 5×5 Connect Four (d) 6×6 Connect Four

Fig. 6. Round-robin tournament of all final models from minimizing different targets. For each game 8 final models from 4 different targets plus a random player (i.e. 33 in total). In panel (a) the difference is small. In panel b, c, and d, the Elo rating of l_v minimized players clearly dominates.

get weaker and not stronger. In the larger Connect Four setting, we still have a clear improvement, especially if we minimize for l_v. Minimizing for $l_{\mathbf{p}}$ leads to stagnation quickly, or at least a very slow improvement.

Overall, we display the Elo progression obtained from the different minimization targets for one game together. However, one shall be aware that their numbers are not directly comparable due to the high self-play bias (as they stem from players who have never seen each other). Nevertheless, the trends are important, and it is especially interesting to see if Elo values correlate with the progression of losses. Based on the experimental results, we can conclude that the whole history Elo rating is certainly good for assessing if training actually works, whereas the losses alone do not always show that. We may even experience contradicting outcomes as stagnating losses and rising Elo ratings (for the big Othello setting and l_v) or completely counterintuitive results as for the small Connect Four setting where Elo ratings and losses are partly anti-correlated. We seemingly have experimental evidence for the fact that training losses and Elo ratings are by no means exchangeable as they can provide very different impressions of what is actually happening.

C. The Final Best Player Elo Rating

In order to measure which target can achieve better playing strength, we let all final models trained from 8 runs and 4 targets plus a random player pit against each other for 20 times in a full round robin tournament. This enables a direct comparison of the final outcomes of the different training processes with different targets. It is thus more informative than the whole history training Elo due to the self-play bias, but provides no information during the self-play training process. In principle, we could of course do that also during the training at certain iterations, but this is computationally very expensive.

The results are presented in Fig. 6. and show that minimizing l_v achieves the highest Elo rating with small variance for 6×6 Othello, 5×5 Connect Four and 6×6 Connect Four. For 5×5 Othello, with 200 training iterations, the difference between the results is small. We therefore presume that minimizing l_v is the best choice for the games we focus on. This is somewhat surprising because we expected the l_+ to perform best as documented in the literature. However, this may apply to smaller games only, and 5×5 Othello already seems to be a border case where overfitting levels out all differences.

In conclusion, we find that minimizing l_v only is an alternative to the l_+ target for certain cases. We also report exceptions, especially in relation to the Elo rating as calculated during training. The relation between Elo and loss during training is sometimes inconsistent (5×5 Connect Four training shows Elo decreasing while the losses are actually minimized). A combination achieves lowest loss, but l_v achieves the highest

Elo. If we minimize the combination, minimizing the $l_\times$ can result to a higher Elo rating in these games.

VI. Conclusion

Most function approximators in supervised learning and reinforcement learning use a single neural network with a single input and output. In reinforcement learning, this is either a policy or a value network. Alpha(Go) Zero innovatively minimizes *both* policy and value, using a single unified network with two heads, a policy head and a value head. Alpha(Go) Zero and other works minimize the sum of policy and value loss. Here, we study four different loss function combinations: (1) $l_{\mathbf{p}}$, (2) l_v, (3) l_+, (4) $l_\times$. We use the open source AlphaZeroGeneral system for light-weight self-play experiments on two small games, Connect Four and Othello. Surprisingly, we find that l_v achieves the highest tournament Elo rating, in contrast to what AlphaZero uses and in contrast to the defaults of AlphaZeroGeneral. Much research in self-play is going on using the default loss function. More research is needed into the relative importance of value function and policy function in small games. Furthermore, default hyper-parameter settings may be non-optimal, especially for the smaller games we investigate here.

During training, we compute a running Elo rating. We find that the training losses trend and the Elo ratings trend are inconsistent in some games (5×5 Connect Four and 6×6 Othello). Training Elo, while cheap to compute, can be a misleading indicator of playing strength, influenced by self-play training bias [2]. Our results provide the methodological contribution that for comparing playing strength, tournament Elo rating should be used, instead of running training Elo.

This study shows that the choice of the optimal combined loss function can have a huge impact on Elo performance. Unfortunately, our computational resources did not allow us to test the approach on large board sizes, but the results should encourage research of loss functions and alternative Elo computation also for large scale games.

Acknowledgment

Hui Wang acknowledges financial support from the China Scholarship Council (CSC), CSC No.201706990015. We thank the ADA Research Group in LIACS (especially Holger Hoos and Marie Anastacio) for providing computational resources.

References

[1] D. Silver, A. Huang, C. J. Maddison, et al, "Mastering the game of Go with deep neural networks and tree search", *Nature*, vol. 529(7587), pp. 484–489, 2016.

[2] D. Silver, J. Schrittwieser, K. Simonyan, et al, "Mastering the game of go without human knowledge", *Nature*, vol. 550, pp. 354–359, 2017.

[3] D. Silver, T. Hubert, J. Schrittwieser, et al, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play", *Science*, vol. 362(6419), pp. 1140–1144, 2018.

[4] M. C. Fu, "AlphaGo and Monte Carlo tree search: the simulation optimization perspective", *Proceedings of the 2016 Winter Simulation Conference. IEEE Press*, pp. 659–670, 2016.

[5] M. H. S. Segler, M. Preuss and M.P. Waller, "Planning chemical syntheses with deep neural networks and symbolic AI", *Nature*, vol. 555(7698), pp. 604-610, 2018.

[6] K. Matsuzaki and N. Kitamura, "Do evaluation functions really improve Monte-Carlo tree search", *ICGA Journal*, 2018 (Preprint): 1-11.

[7] N. Surag, https://github.com/suragnair/alpha-zero-general, 2018.

[8] S. Iwata and T. Kasai , "The Othello game on an n×n board is PSPACE-complete", *Theoretical Computer Science*, vol. 123(2), pp. 329–340, 1994.

[9] V. Allis, "A knowledge-based approach of Connect-Four-the game is solved: White wins", 1988.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, et al, "Playing atari with deep reinforcement learning", *arXiv preprint*, arXiv:1312.5602, 2013.

[11] C. Clark and A. Storkey, "Training deep convolutional neural networks to play go", *International Conference on Machine Learning*, pp. 1766–1774, 2015.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, et al, "Human-level control through deep reinforcement learning", *Nature*, vol. 518(7540), pp. 529–533, 2015.

[13] M.A. Wiering, "Self-Play and Using an Expert to Learn to Play Backgammon with Temporal Difference Learning", *Journal of Intelligent Learning Systems and Applications*, vol. 2(2), pp. 57–68, 2010.

[14] M. Van Der Ree and M. Wiering, "Reinforcement learning in the game of Othello: Learning against a fixed opponent and learning from self-play", *In Adaptive Dynamic Programming And Reinforcement Learning*, pp. 108–115, 2013.

[15] Y. Bengio, J. Louradour, R. Collobert and J. Weston, "Curriculum learning", *Proceedings of the 26th annual international conference on machine learning, ACM*, pp. 41–48, 2009.

[16] C.B. Browne, E. Powley, D. Whitehouse, et al, "A survey of monte carlo tree search methods", *IEEE Transactions on Computational Intelligence and AI in games* vol. 4(1), pp. 1–43, 2012.

[17] H. Wang, M. Emmerich, M. Preuss and A. Plaat, "Hyper-Parameter Sweep on AlphaZero General", *arXiv preprint*, arXiv:1903.08129, 2019.

[18] Y. Mandai and T. Kaneko, "Alternative Multitask Training for Evaluation Functions in Game of Go", *2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI). IEEE*, pp. 132–135, 2018.

[19] R. Caruana, "Multitask learning", *Machine learning*, 1997, vol. 28(1), pp. 41–75.

[20] K. Matsuzaki, "Empirical Analysis of PUCT Algorithm with Evaluation Functions of Different Quality", *2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI). IEEE*, 142-147, 2018.

[21] M. Buro, "The Othello match of the year: Takeshi Murakami vs. Logistello", *ICGA Journal*, vol. 20(3), pp. 189–193, 1997.

[22] S. Y. Chong, M. K. Tan and J. D. White, "Observing the evolution of neural networks learning to play the game of Othello", *IEEE Transactions on Evolutionary Computation"*, pp. 240–251, 2005.

[23] M. Thill, S. Bagheri, P. Koch and W. Konen, "Temporal difference learning with eligibility traces for the game connect four", *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8, 2014.

[24] B. Banerjee and P. Stone, "General Game Learning Using Knowledge Transfer", *International Joint Conference on Artificial Intelligence*, pp. 672–677, 2007.

[25] H. Wang, M. Emmerich and A. Plaat, "Assessing the Potential of Classical Q-learning in General Game Playing", *arXiv preprint*, arXiv:1810.06078, 2018.

[26] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift", *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, vol 37, pp. 448–456, 2015.

[27] D. P. Kingmaand J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint*, arXiv:1412.6980, 2014.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, et al, "Dropout: a simple way to prevent neural networks from overfitting", *The Journal of Machine Learning Research*, vol. 15(1), pp. 1929–1958, 2014.

[29] R. Coulom, "Whole-history rating: A Bayesian rating system for players of time-varying strength", *International Conference on Computers and Games. Springer, Berlin, Heidelberg*, pp. 113–124, 2008.

[30] M. Emmerich and A. H. Deutz. "A tutorial on multiobjective optimization: fundamentals and evolutionary methods", *Natural computing*, vol. 17(3), pp. 585–609, 2018.