Chapter 4

Loss Functions of AlphaZero-like Self-play

4.1 Introduction

As we introduced in Chapter 3, the AlphaGo series of papers [10, 16, 17] have sparked enormous interest of researchers and the general public alike into deep reinforcement learning [85, 86, 87, 88]. AlphaGo Zero [17], 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 an MCTS searcher. AlphaZero [10] uses a single architecture for playing three different games (Go, Chess and Shogi) without human knowledge. Many applications and optimization methods [89, 90] 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. We have demonstrated the hyper-parameter tuning results of a light-weight AlphaZero-like self-play framework in Chapter 3. Therefore, in this Chapter, we focus on the loss function design of the AlphaZero-like self-play.

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). A policy $\mathbf{p}$ is a probability distribution for choosing the best move. A lower policy loss ($l_\mathbf{p}$) indicates a more accurate selection of the best move. A 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. [91] 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 chapter. 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 [65] and focus on smaller games, namely 5×5 and 6×6 Othello [76], 5×5, 6×6 Connect Four games [92] and 5×5 and 6×6 Gobang [93].

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 and accurately measure the playing strength of different players relative to each other.

- We evaluate 4 alternative loss functions for 3 games and 2 board sizes, and find that the best setting depends on the game and is usually not the sum of policy and value loss. However, the sum might be considered as a good default compromise if no further information about the game is present.

The chapter is structured as follows. Part 4.2 presents related work. Part 4.3 presents games tested in the experiments. Part 4.4 introduces the default loss function of AlphaZero-like self-play and alternative loss functions. Part 4.5 sets up the experiments. Part 4.6 presents the experimental results. Part 4.7 discusses future work and summarizes this chapter.

4.2 Related Work

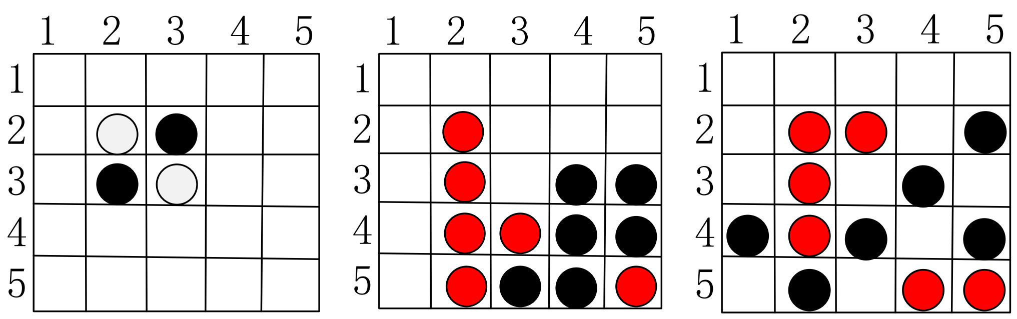
Deep reinforcement learning [94] is currently one of the most active research areas in AI, reaching human level performance for difficult games such as Go [69], which was almost unthinkable 10 years ago. Since Mnih et al. reported human-level control for playing Atari 2600 games by means of deep reinforcement learning [44], 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 [16] to relying more on self-play. This has been advocated in the area of reinforcement learning [72, 73] for quite some time already. Silver et al. [17] 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 [95]. 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 [10].

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 [7], 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 [32]. This includes the choice of minimization tasks (loss functions) used for measuring training success. For instance, [96] studied policy and value network optimization as a multi-task learning problem [97]. 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 [91] is different while playing Othello. Moreover, [98] showed that the value function has more importance than the policy function in the P-UCT algorithm for Othello.

4.3 Test Games

In our experiments, we use the games Othello, Connect Four and Gobang, each with 5×5 and 6×6 board sizes. As described in Chapter 3, Othello is a popular 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



(a) 5×5 Othello (b) 5×5 Connect Four (c) 5×5 Gobang

Figure 4.1: Our test games on 5 × 5 boards

placed and another piece of the current player's are flipped to the current player's color. After the last legal position is filled, the player who has most pieces wins the game. Fig. 4.1(a) is the start configuration for 5×5 Othello. Connect Four is a two-player connection game. 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. 4.1(b) is a game termination example for 5×5 Connect Four where the red player wins the game. Gobang is another connection game that is traditionally played with Go pieces (black and white stones) on a Go board. Players alternate turns, placing a stone of their color on an empty position. The winner is the first player to form an unbroken chain of 4 stones horizontally, vertically, or diagonally. Fig. 4.1(c) is a game termination example for 5×5 Gobang where the black player wins the game.

There is a wealth of research on finding playing strategies for these three games by means of different methods. For example, Buro created Logistello [77] to play Othello. Chong et al. described the evolution of neural networks for learning to play Othello [78]. Thill et al. applied temporal difference learning to play Connect Four [99]. Zhang et al. designed evaluation functions for Gobang [100]. Moreover, Banerjee et al. tested knowledge transfer in GGP on small games including 4×4 Othello [46]. Wang et al. assessed the potential of classical Q-learning based on small games including 4×4 Connect Four [31]. Obviously, these two games are commonly tested in game playing.

4.4 Loss Function

4.4.1 Minimization Targets

As we want to assess the effect of minimizing different loss functions for AlphaZero-like self-play (Algorithm 2), besides the default loss function (Equation: 3.1), we employ a weighted sum loss function based on Equation 3.1:

$$l_\lambda = \lambda(-\pi^\top \log \mathbf{p}) + (1-\lambda)(v-z)^2 \tag{4.1}$$

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 Equation 3.1 as training loss function. Furthermore, inspired by that, in the theory of multi-attribute utility functions in multi-criteria optimization [101], a sum tends to prefer extreme solutions, whereas product prefers more balanced solution in case of coefficient objectives. Thus a product combination loss function is employed as follows:

$$l_\times = -\pi^\top \log \mathbf{p} \times (v-z)^2 \tag{4.2}$$

For all experiments, each setting is run 8 times to get statistically significant results (with error bars) using the parameters of Table 4.1 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).

4.5 Experimental Setup

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 as used in Chapter 3, which consists of 4 convolutional layers and 2 fully connected layers [65]. The parameter values for Algorithm 2 used in our experiments are given in Table 4.1. In order to enhance reproducibility, we used values based on work reported by [32].

Table 4.1: Default Parameter Settings

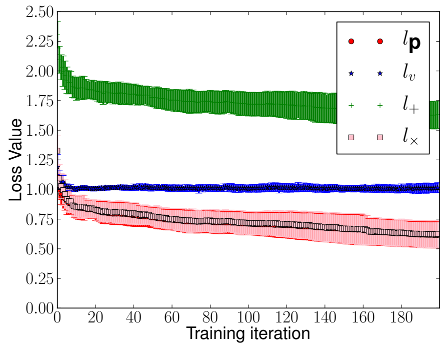
Parameter	Brief Description	Default Value
I	number of iteration	200
E	number of episode	50
T'	step threshold	15
m	MCTS simulation times	100
c	weight in UCT	1.0
rs	number of retrain iteration	20
ep	number of epoch	10
bs	batch size	64
lr	learning rate	0.005
d	dropout probability	0.3
n	number of comparison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

4.5.1 Measurements

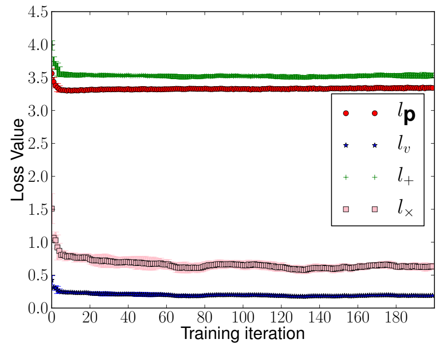
The chosen loss function is used to guide each training process, with the expectation that smaller loss means a stronger model. However, in practise, we have found that this is not always the case and another measure is needed to check based on trained models real playing performance in competitions. Therefore, following Deep Mind's work, we employ Bayesian Elo ratings [82] 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.

4.6 Experiment Results

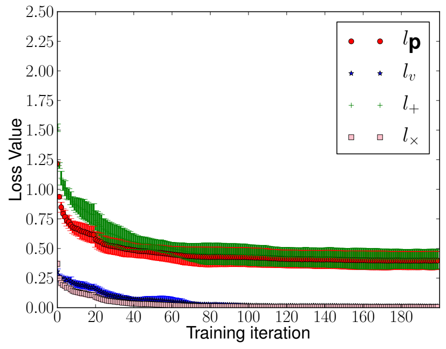
In the following, we present the results of different loss functions. We have measured the individual value loss, the individual policy loss, the sum of the two, and the product of the two, for the three games. We report training loss, the training Elo rating and the tournament Elo rating of the final best players. Error bars indicate standard deviations of 8 runs.



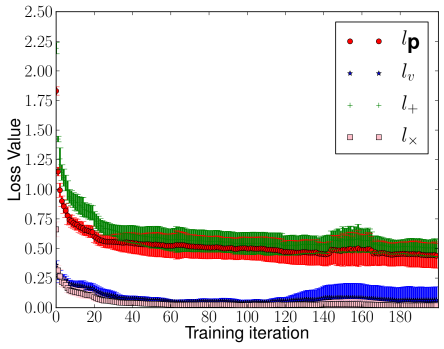
(a) Minimize $l_{\mathbf{p}}$



(b) Minimize l_v



(c) Minimize l_+



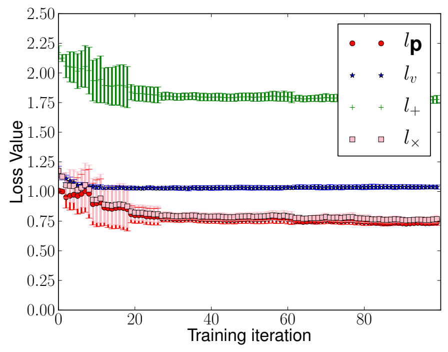
(d) Minimize $l_\times$

Figure 4.2: Training losses for minimizing different targets in 5×5 Othello, averaged over 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

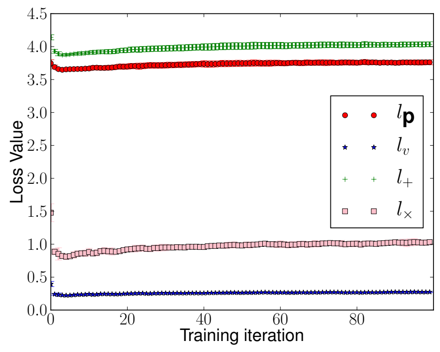
4.6.1 Training Loss

We first show the training losses in every iteration with one minimization task per diagram, hence we need four of these per game. In these graphs we see what minimizing for a specific target actually means for the other loss types.

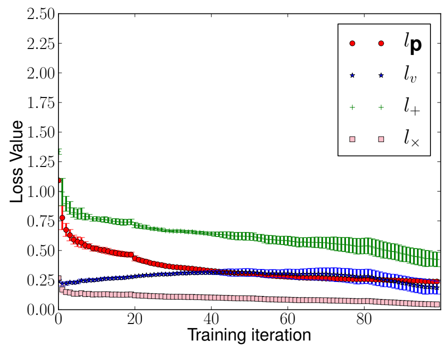
For 5×5 Othello, from Fig. 4.2(a), we find that when minimizing $l_{\mathbf{p}}$ only, the loss decreases significantly to about 0.6 at the end of each training, whereas l_v stagnates at 1.0 after 10 iterations. Minimizing only l_v (Fig. 4.2(b)) brings it down from 0.5 to 0.2, but $l_{\mathbf{p}}$ remains stable at a high level. In Fig. 4.2(c), we see that when the l_+ is minimized, both losses are reduced significantly. The $l_{\mathbf{p}}$



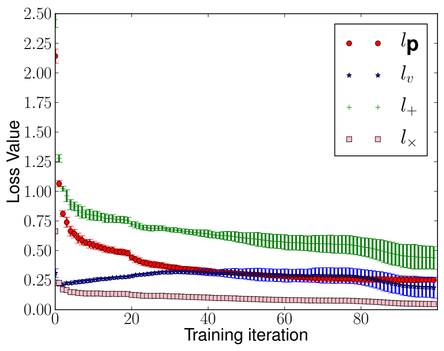
(a) Minimize $l_\mathbf{p}$



(b) Minimize l_v



(c) Minimize l_+

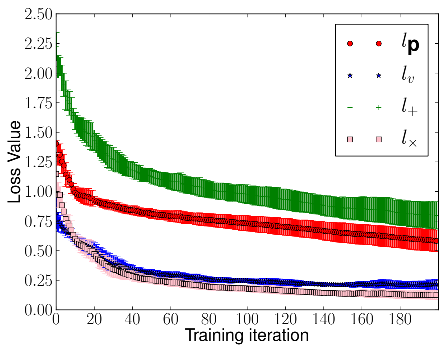


(d) Minimize $l_\times$

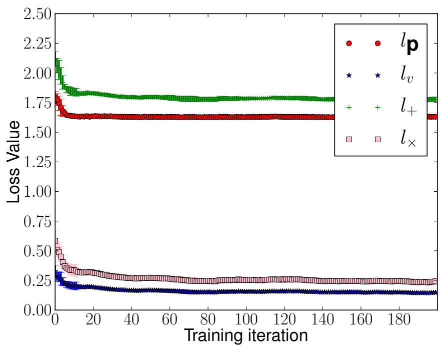
Figure 4.3: Training losses for minimizing different targets in 6×6 Othello, averaged from 8 runs. All losses are shown while we minimize only one (similar to Fig 4.2). Note the different scaling for subfigure (b). Except for l_+, the target that is minimized for is the lowest

decreases from about 1.2 to 0.5, l_v surprisingly decreases to 0. Fig. 4.2(d), it is similar to Fig. 4.2(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.

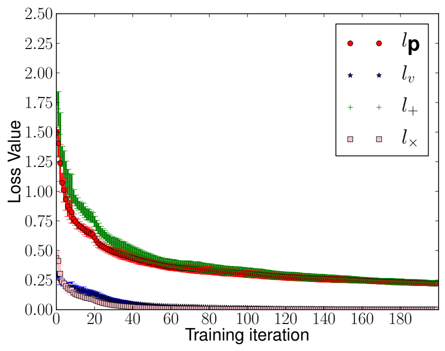
For the larger 6×6 Othello, we find that minimizing only $l_\mathbf{p}$ reduces it significantly to about 0.75, where l_v is stable again after about 10 iterations (Fig. 4.3(a)). For minimizing l_v (Fig. 4.3(b)), the results show that l_v is reduced from more than 0.5 to about 0.25 at the end of each training, but $l_\mathbf{p}$ seems to remain almost unchanged. For minimizing the l_+ (Fig. 4.3(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



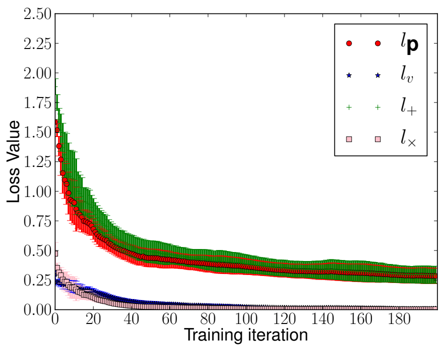
(a) Minimize $l_{\mathbf{p}}$



(b) Minimize l_v



(c) Minimize l_+

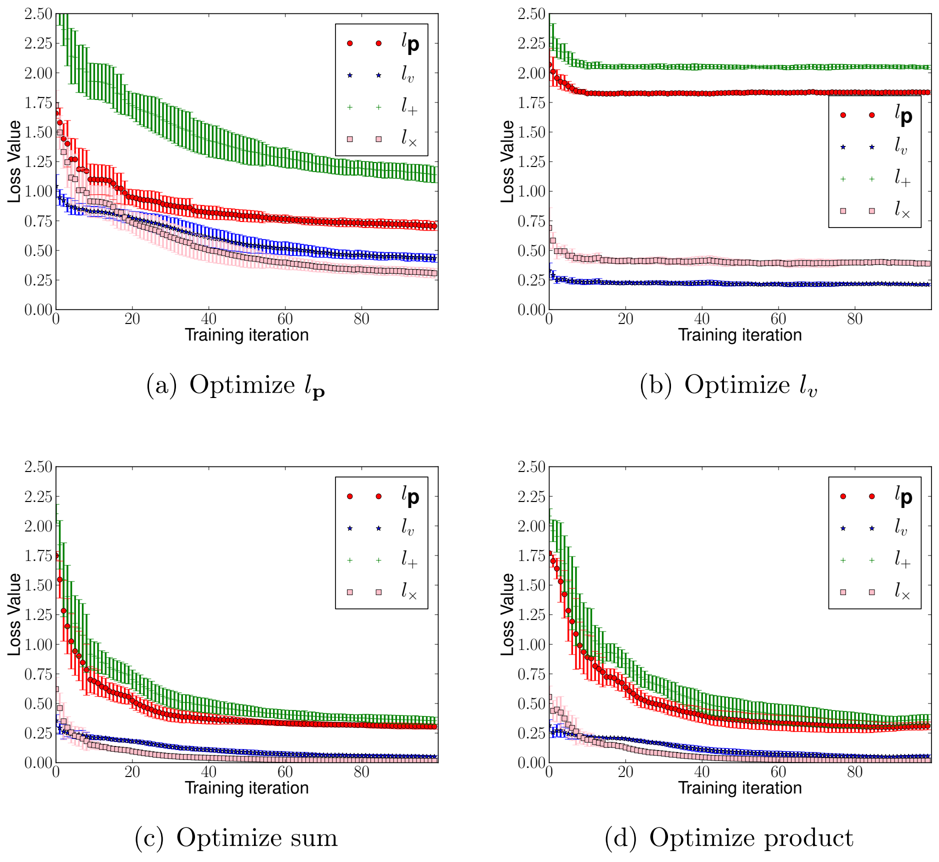


(d) Minimize $l_\times$

Figure 4.4: Training losses for minimizing the four different targets in 5×5 Connect Four, averaged from 8 runs. l_v is always the lowest

when minimizing the $l_\times$ (Fig. 4.3(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. 4.4(a)), we find that when only minimizing $l_{\mathbf{p}}$, it significantly reduces 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. 4.4(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. 4.4(c)) and the $l_\times$ (Fig. 4.4(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.



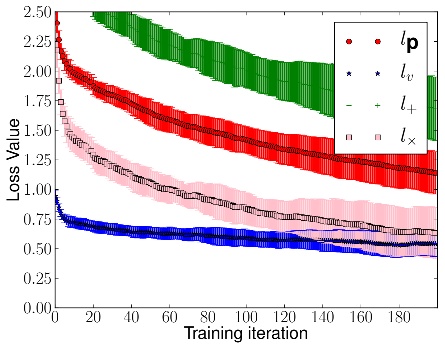
(a) Optimize $l_{\mathbf{p}}$ (b) Optimize l_v

(c) Optimize sum (d) Optimize product

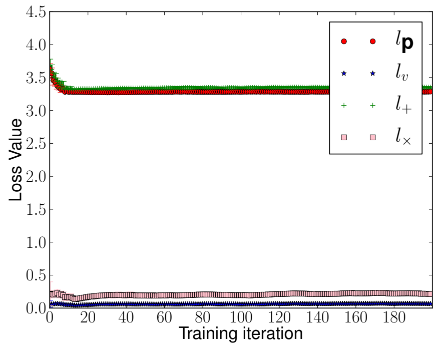
Figure 4.5: Training losses for optimizing different targets in 6×6 Connect Four, averaged from 8 Runs. l_v is the lowest except for the product target

The training process of the larger 6×6 Connect Four is investigated in Fig 4.5(a). We find that optimizing $l_{\mathbf{p}}$ reduces it significantly from 1.7 to about 0.7 at the end of each training, where l_v is minimized from 1.2 to about 0.4. For the scenario with optimizing l_v (Fig 4.5(b)), we find a similar behavior than for the smaller Connect Four. After some initial progress, there is only stagnation. Again, for optimizing the sum and the product, the target value changes, but the single loss values $l_{\mathbf{p}}$ and l_v behave similarly (Figs 4.5(c) and 4.5(d)). Thus we see that both targets lead to very similar training processes.

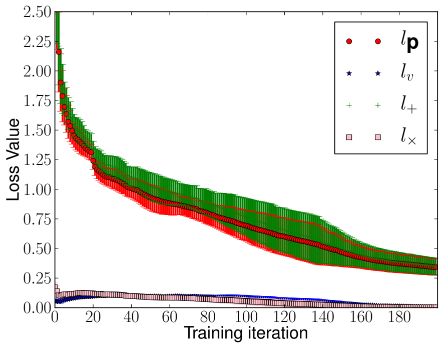
For 5×5 Gobang game, we find that, in Fig. 4.6, when only minimizing $l_{\mathbf{p}}$, the $l_{\mathbf{p}}$ value decreases from around 2.5 to about 1.25 while the l_v value reduces from 1.0 to 0.5 (see Fig. 4.6(a)). When minimizing l_v, l_v value quickly reduces to a very low level which is lower than 0.1 (see Fig. 4.6(b)). Minimizing l_+ and $l_\times$ both



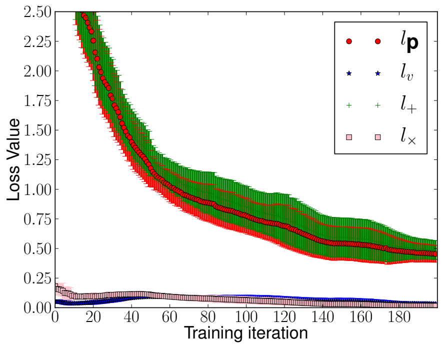
(a) Minimize $l_{\mathbf{p}}$



(b) Minimize l_v



(c) Minimize l_+



(d) Minimize $l_\times$

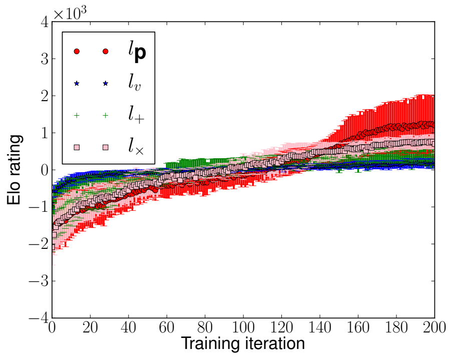
Figure 4.6: Training losses for minimizing the four different targets in 5×5 Gobang, averaged from 8 runs. l_v is always the lowest

lead to stationary low l_v levels from the beginning of training which is different from Othello and Connect Four.

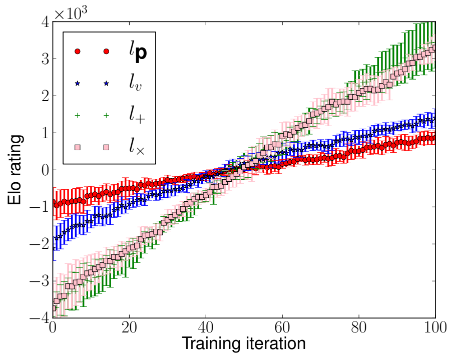
4.6.2 Training Elo Rating

Following the AlphaGo papers, the training Elo rating of every iteration during training is investigated. Instead of showing results from single runs, means and variances for 8 runs for each target are provided, categorized by different games in Fig. 4.7.

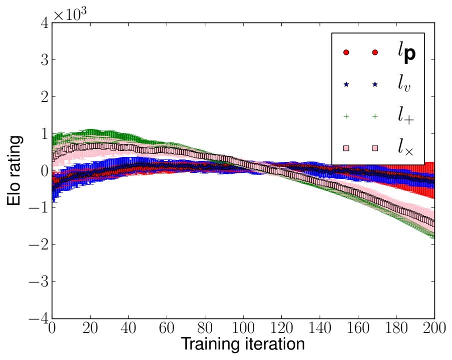
From Fig. 4.7(a) (small 5×5 Othello) we see that for all minimization tasks, Elo values steadily improve, while they raise fastest for $l_{\mathbf{p}}$. In Fig. 4.7(b), we find that for 6×6 Othello version, Elo values also always improve, but much faster for



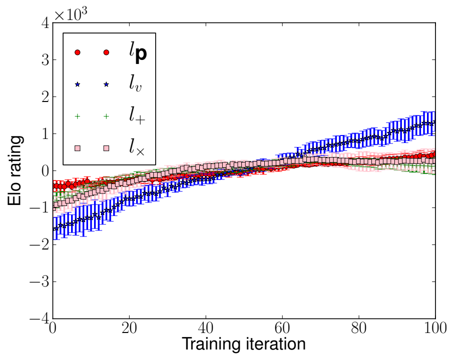
(a) 5×5 Othello



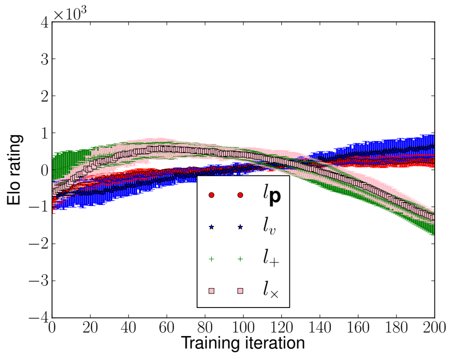
(b) 6×6 Othello



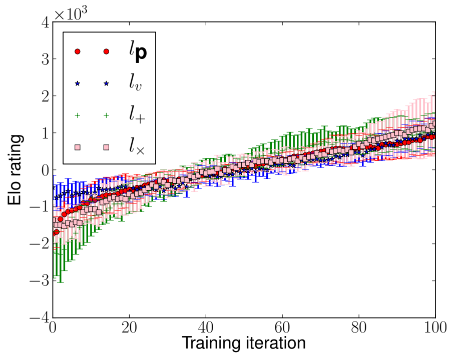
(c) 5×5 Connect Four



(d) 6×6 Connect Four



(e) 5×5 Gobang



(f) 6×6 Gobang

Figure 4.7: 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 for example shows inconsistent results

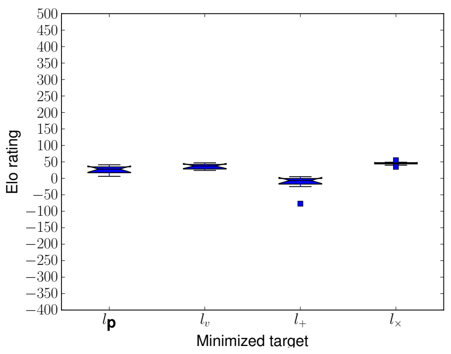
the l_+ and $l_\times$ target, compared to the single loss targets.

Fig. 4.7(c) and Fig. 4.7(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 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 to a very slow improvement.

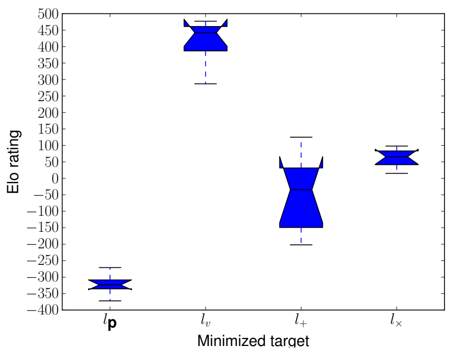
Overall, we display the Elo progression obtained from the different minimization targets for one game together. However, one must be aware that their numbers are not directly comparable due to the high self-play bias (as they stem from players who have never played against each other). Nevertheless, the trends as observed for single self-play are of interest, 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 training 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 have experimental evidence for the fact that training losses and Elo ratings are by no means interchangeable as they can provide very different impressions of what is actually happening.

4.6.3 The Final Best Player Tournament Elo Rating

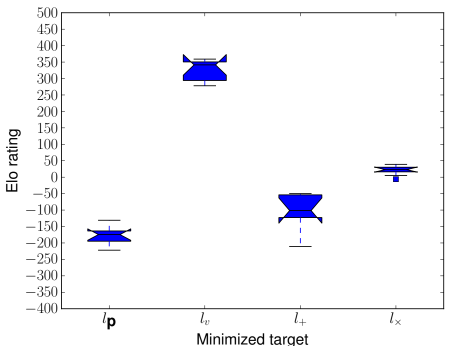
In order to measure which target can achieve better playing strength, we let all final models trained from 8 runs and 4 targets plus a random player pit against each other for 20 times in a full round robin tournament. This enables a direct comparison of the final outcomes of the different training processes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



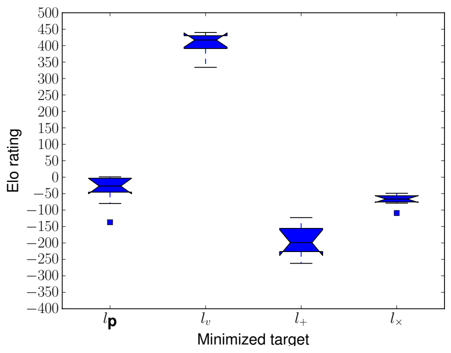
(a) 5×5 Othello



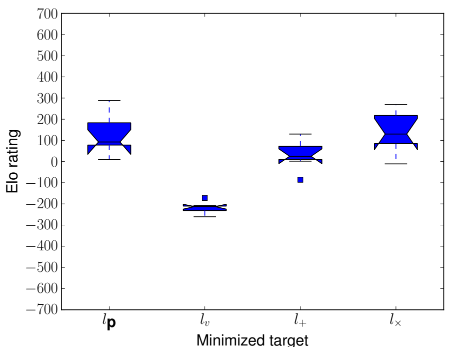
(b) 6×6 Othello



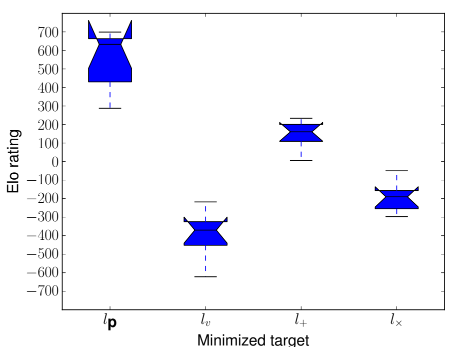
(c) 5×5 Connect Four



(d) 6×6 Connect Four



(e) 5×5 Gobang



(f) 6×6 Gobang

Figure 4.8: 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. However, in panel (f), the Elo rating of $l_{\mathbf{p}}$ minimized players clearly achieve the best performance.

The results are presented in Fig. 4.8. 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 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 solely minimizing l_v is an alternative to the default sum objective l_+ in many 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) due to training bias. And for Gobang game, only minimizing $l_{\mathbf{p}}$ is the best alternative. A combination achieves lowest loss, but l_v achieves the highest training Elo. If we minimize product loss $l_\times$, this can result in higher Elo rating for certain games. More research (such as training bias and in which case which objective function (combination) should be employed) should be studied further.

4.7 Summary

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 in many cases l_v achieves the highest tournament Elo rating, in contrast to the default sum objective in AlphaZero and AlphaZeroGeneral. The obtained experimental results in this chapter however indicate that relying on default setting, major performance gains are likely to be missed out. Much research in self-play is recently going on using the default loss function without questioning this default choice. 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, because it is influenced by self-play training bias [17]. Our results provide the methodological contribution that for comparing playing strength among players, tournament Elo ratings should be used, instead of running training Elo ratings.

This chapter 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 similar research of loss functions and alternative Elo computation also for large scale games.