Adaptive Warm-Start MCTS in AlphaZero-Like Deep Reinforcement Learning

Hui Wang(✉), Mike Preuss, and Aske Plaat

Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands
h.wang.13@liacs.leidenuniv.nl
http://www.cs.leiden.edu

Abstract. AlphaZero has achieved impressive performance in deep reinforcement learning by utilizing an architecture that combines search and training of a neural network in self-play. Many researchers are looking for ways to reproduce and improve results for other games/tasks. However, the architecture is designed to learn from scratch, tabula rasa, accepting a cold-start problem in self-play. Recently, a warm-start enhancement method for Monte Carlo Tree Search was proposed to improve the self-play starting phase. It employs a fixed parameter I' to control the warm-start length. Improved performance was reported in small board games. In this paper we present results with an adaptive switch method. Experiments show that our approach works better than the fixed I', especially for "deep", tactical, games (Othello and Connect Four). We conjecture that the adaptive value for I' is also influenced by the size of the game, and that on average I' will increase with game size. We conclude that AlphaZero-like deep reinforcement learning benefits from adaptive rollout based warm-start, as Rapid Action Value Estimate did for rollout-based reinforcement learning 15 years ago.

Keywords: MCTS · AlphaZero · Deep reinforcement learning

1 Introduction

The combination of online Monte Carlo Tree Search (MCTS) [1] in self-play and offline neural network training has been widely applied as a deep reinforcement learning technique, in particular for solving game-related problems by AlphaGo series programs [9–11]. The approach of this paradigm is to use game playing records from self-play by MCTS as training examples to train the neural network, whereas this trained neural network is used to inform the MCTS value and policy. Note that in contrast to AlphaGo Zero or AlphaZero, the original AlphaGo also uses large amounts of expert data to train the neural network and a fast rollout policy together with the policy provided by neural network to guide the MCTS.

However, although the transition from a combination of using expert data and self-play (AlphaGo) to only using self-play (AlphaGo Zero and AlphaZero) appears to have only positive results, it does raise some questions.

The first question is: 'should all human expert data be abandoned?' In other games we have seen that human knowledge is essential for mastering complex games, such as StarCraft [14]. Then when should expert data be used?

The second question is: 'should the fast rollout policy be abandoned?' Recently, Wang et al. [19] have proposed to use warm-start search enhancements **at the start phase** in AlphaZero-like self-play, which improves performance in 3 small board games. Instead of only using the neural network for value and policy, in the first few iterations, classic rollout (or RAVE etc.) can be used.

In fact, the essence of the warm-start search enhancement is to re-generate expert knowledge in the start phase of self-play training, to reduce the cold-start problem of playing against untrained agents. The method uses rollout (which can be seen as experts) instead of a randomly initialized neural network, up until a number of I' iterations, when it switches to the regular value network. In their experiments, the I' was fixed at 5. Obviously, a fixed I' may not be optimal. Therefore, in this work, we propose an adaptive switch method. The method uses an **arena** in the self-play stage (see Algorithm 2), where the search enhancement and the default MCTS are matched, to judge whether to switch or not. With this mechanism, we can dynamically switch off the enhancement if it is no longer better than the default MCTS player, as the neural network is being trained.

Our main contributions can be summarized as follows:

1. Warm-start method improves the Elo [2] of AlphaZero-like self-play in small games, but it introduces a new hyper-parameter. Adaptive warm-start further improves performance and removes the hyper-parameter.
2. For deep games (with a small branching factor) warm-start works better than for shallow games. This indicates that the effectiveness of warm-start method may increase for larger games.

The rest of paper is designed as follows. An overview of the most relevant literature is given in Sect. 2. Before proposing our adaptive switch method in Sect. 4, we describe the warm-start AlphaZero-like self-play algorithm in Sect. 3. Thereafter, we set up the experiments in Sect. 5 and present their results in Sect. 6. Finally, we conclude our paper and discuss future work.

2 Related Work

There are a lot of early successful works in reinforcement learning [12], e.g. using temporal difference learning with a neural network to play backgammon [13]. MCTS has also been well studied, and many variants/enhancements were designed to solve problems in the domain of sequential decisions, especially on games. For example, enhancements such as Rapid Action Value Estimate (RAVE) and All Moves as First (AMAF) have been conceived to improve MCTS [3,4]. The AlphaGo series algorithms replace the table based model with a deep neural network based model, where the neural network has a policy head (for evaluating of a state) and a value head (for learning a best action) [16], enabled by the GPU hardware development. Thereafter, the structure that combines MCTS with neural network training has become a typical approach for reinforcement learning tasks [8,18] of this kind model-based deep reinforcement learning [6,7]. Comparing AlphaGo with AlphaGo Zero and AlphaZero, the latter did not use any expert data to train neural network, and abandoned the fast rollout policy for improving MCTS on the trained neural network.

Within a general game playing framework, in order to improve training examples efficiency, [15] assessed the potential of classic Q-learning by introducing Monte Carlo Search enhancements. In an AlphaZero-like self-play framework, [20] used domain-specific features and optimizations, starting from random initialization and no preexisting data, to accelerate the training. We also base our work on an open reimplementation of AlphaZero, AlphaZero General [5].

However, AlphaStar, which defeated human professionals at StarCraft [14], went back to utilize human expert data, thereby suggesting that this is still an option at the start phase of training. Apart from this, [19] proposed a warm-start search enhancement method, pointed out the promising potential of utilizing MCTS enhancements to re-generate expert data at the start phase of training. Our approach differs from AlphaStar, as we generate expert data using MCTS enhancements other than collecting it from humans; further, compared to the static warm-start of [19], we propose an adaptive method to control the iteration length of using such enhancements instead of a fixed I'.

3 Warm-Start AlphaZero Self-play

3.1 The Algorithm Framework

Based on [10,16,19], the core of AlphaZero-like self-play (see Algorithm 1) is an iterative loop which consists of three stages (self-play, neural network training and arena comparison) within the single iteration. The detail description of these 3 stages can be found in [19]. Note that in the Algorithm 1, line 5, a fixed I' is employed to control whether to use MCTS or MCTS enhancements, the I' should be set as relatively smaller than I, which is known as warm-start search.

3.2 MCTS

Classic MCTS has shown successful performance to solve complex games, by taking random samples in the search space to evaluate the state value. Basically, the classic MCTS can be divided into 4 stages, which are known as *selection*, *expansion*, *rollout* and *backpropagate* [1]. However, for the default MCTS in AlphaZero-like self-play (e.g. our Baseline), the neural network directly informs the MCTS state policy and value to guide the search instead of running a rollout.

Algorithm 1. Warm-start AlphaZero-like Self-play Algorithm

```
1: Randomly initialize $f_\theta$, assign retrain buffer $D$
2: for iteration=1, ..., $I'$, ..., $I$ do
3:     for episode=1,..., $E$ do                                  ▷ self-play
4:         for t=1, ..., $T'$, ..., $T$ do
5:             if $I \leq I'$ then $\pi_t \leftarrow$ MCTS Enhancement
6:             else $\pi_t \leftarrow$ default MCTS
7:             if $t \leq T'$ then $a_t$ = randomly select on $\pi_t$
8:             else $a_t = \arg\max_a(\pi_t)$
9:             executeAction($s_t$, $a_t$)
10:        $D \leftarrow (s_t, \pi_t, z_t)$ with outcome $z_{t\in[1,T]}$
11:    Sample minibatch ($s_j$, $\pi_j$, $z_j$) from $D$             ▷ training
12:    Train $f_{\theta'} \leftarrow f_\theta$
13:    $f_\theta = f_{\theta'}$ if $f_{\theta'}$ is better, using default MCTS    ▷ arena
14: return $f_\theta$;
```

3.3 MCTS Enhancements

In this paper, we adopt the same two individual enhancements and three combinations to improve neural network training as were used by [19].

Rollout runs a classic MCTS random rollout to get a value that provides more meaningful information than a value from random initialized neural network.

RAVE is a well-studied enhancement to cope with the cold-start of MCTS in games like Go [3], where the playout-sequence can be transposed. The core idea of RAVE is using AMAF to update the state visit count N_{rave} and Q-value Q_{rave}, which are written as: $N_{rave}(s_{t_1}, a_{t_2}) \leftarrow N_{rave}(s_{t_1}, a_{t_2}) + 1$, $Q_{rave}(s_{t_1}, a_{t_2}) \leftarrow \frac{N_{rave}(s_{t_1},a_{t_2})*Q_{rave}(s_{t_1},a_{t_2})+v}{N_{rave}(s_{t_1},a_{t_2})+1}$, where $s_{t_1} \in VisitedPath$, and $a_{t_2} \in A(s_{t_1})$, and for $\forall t < t_2, a_t \neq a_{t_2}$.

RoRa is the combination which adds the random rollout to enhance RAVE.

WRo introduces a weighted sum of rollout value and the neural network value as the return value to guide MCTS [9].

WRoRa also employs a weighted sum to combine the values from the neural network and the RoRa.

Different from [19], since there is no pre-determined I', in our work, *weight* is simply calculated as $1/i, i \in [1, I]$, where i is the current iteration number.

4 Adaptive Warm-Start Switch Method

The fixed I' to control the length of using warm-start search enhancements as suggested by [19] works, but seems to require different parameter values for different games. In consequence, a costly tuning process would be necessary for each game. Thus, an adaptive method would have multiple advantages.

We notice that the core of the warm-start method is re-generating expert data to train the neural network at the start phase of self-training to avoid learning

Algorithm 2. Adaptive Warm-Start Switch Algorithm

```
1: Randomly initialize $f_\theta$; Initialize retrain buffer $D$, Switch←False, $r_{mcts} \leftarrow 0$
2: for iteration=1, ..., $I$ do                                        ▷ no $I'$
3:     if not Switch then                                              ▷ not switch
4:         for episode=1,..., $E$ do                                   ▷ arena with enhancements
5:             for t=1, ..., $T'$, ..., $T$ do
6:                 if $episode \leq E/2$ then
7:                     if t is odd then $\pi_t \leftarrow$ MCTS Enhancement
8:                     else $\pi_t \leftarrow$ default MCTS
9:                 else
10:                    if t is odd then $\pi_t \leftarrow$ default MCTS
11:                    else $\pi_t \leftarrow$ MCTS Enhancement
12:                if $t \leq T'$ then $a_t$ = randomly select on $\pi_t$
13:                else $a_t = \arg\max_a(\pi_t)$
14:                executeAction($s_t$, $a_t$)
15:            $D \leftarrow (s_t, \pi_t, z_t)$ with outcome $z_{t\in[1,T]}$
16:            $r_{mcts}$+= reward of default MCTS in this episode
17:    else                                                            ▷ switch
18:        for episode=1,..., $E$ do                                   ▷ purely self-play
19:            for t=1, ..., $T'$, ..., $T$ do
20:                $\pi_t \leftarrow$ default MCTS
21:                if $t \leq T'$ then $a_t$ = randomly select on $\pi_t$
22:                else $a_t = \arg\max_a(\pi_t)$
23:                executeAction($s_t$, $a_t$)
24:            $D \leftarrow (s_t, \pi_t, z_t)$ with outcome $z_{t\in[1,T]}$
25:    Set Switch←True if $r_{mcts}$ >0, and set $r_{mcts} \leftarrow 0$
26:    Sample minibatch ($s_j$, $\pi_j$, $z_j$) from $D$                ▷ training
27:    Train $f_{\theta'} \leftarrow f_\theta$
28:    $f_\theta = f_{\theta'}$ if $f_{\theta'}$ is better, using default MCTS   ▷ arena
29: return $f_\theta$;
```

from weak (random or near random) self-play. We suggest to stop the warm-start when the neural network is on average playing stronger than the enhancements. Therefore, in the self-play, we employ a tournament to compare the standard AlphaZero-like self-play model (Baseline) and the enhancements (see Algorithm 2). The switch occurs once the Baseline MCTS wins more than 50%. In order to avoid spending too much time on this, these arena game records will directly be used as training examples, indicating that the training data is played by the enhancements and the Baseline. This scheme enables to switch at individual points in time for different games and even different training runs.

5 Experimental Setup

Since [19] only studied the winrate of single rollout and RAVE against a random player, this can be used as a test to check whether rollout and RAVE work.

Table 1. Default parameter setting

Para	Description	Value	Para	Description	Value
I	Number of iterations	100	lr	Learning rate	0.005
rs	Number of retrain iterations	20	m	MCTS simulation times	100
ep	Number of epochs	10	d	Dropout probability	0.3
E	Number of episodes	50	c	Weight in UCT	1.0
bs	Batch size	64	n	Number of comparison games	40
T'	Step threshold	15	u	Update threshold	0.6

However, it does not reveal any information about relative playing strength, which is necessary to explain how good training examples provided by MCTS enhancements actually are. Therefore, at first we let all 5 enhancements and the baseline MCTS play 100 repetitions with each other on the same 3 games (6×6 Connect Four, Othello and Gobang, game description can be found in [19]) in order to investigate the relative playing strength of each pair.

In the second experiment, we tune the fixed I', where $I' \in \{1, 3, 5, 7, 9\}$, for different search enhancements, based on Algorithm 1 to play 6×6 Connect Four.

In our last experiment, we use new adaptive switch method Algorithm 2 to play 6×6 Othello, Connect Four and Gobang. We set parameters values according to Table 1. The parameter choices are based on [17]. The detail introduction of these parameters can be found in [17].

Our experiments are run on a high-performance computing (HPC) server, which is a cluster consisting of 20 CPU nodes (40 TFlops) and 10 GPU nodes (40 GPU, 20 TFlops CPU + 536 TFlops GPU). We use small versions of games (6×6) in order to perform a medium number of repetitions. Each single run is deployed in a single GPU which takes several days for different games.

6 Results

6.1 MCTS Vs MCTS Enhancements

Here, we compare the Baseline player (the neural network is initialized randomly which can be regarded as an arena in the first iteration self-play) to the other 5 MCTS enhancements players on 3 different games. Each pair performs 100 repetitions. In Table 2, for Connect Four, the highest winrate is achieved by WRoRa, the lowest by Rave. Except Rave, others are all higher than 50%, showing that the enhancements (except Rave) are better than the untrained Baseline. In Gobang, it is similar, Rave is the lowest, RoRa is the highest. But the winrates are relatively lower than that in other 2 games. It is interesting that in Othello, all winrates are relatively the highest compared to the 2 other games (nearly 100%), although Rave still achieves the lowest winrate which is higher than 50%.

One reason that enhancements work best in Othello is that the Othello game tree is the longest and narrowest (low branching factor). Enhancements like Rollout can provide relatively accurate estimations for these trees. In contrast,

Table 2. Results of comparing default MCTS with Rollout, Rave, RoRa, WRo and WRoRa, respectively on the three games with random neural network, weight as 1/2, $T' = 0$, win rates in percent (row vs column), 100 repetitions each.

	Default MCTS		
	ConnectFour	Othello	Gobang
Rollout	64	93	65
Rave	27.5	53	43
RoRa	76	98	70
WRo	82	96	57
WRoRa	82.5	99	62

Gobang has the shortest game length and the most legal action options. Enhancements like Rollout do not contribute much to the search in short but wide search tree with limited MCTS simulation. As in shorter games it is more likely to reach a terminal state, both Baseline and enhancements get the true result. Therefore, in comparison to MCTS, enhancements like Rollout work better while it does not terminate too fast. Rave is filling more state action pairs based on information from the neural network, its weaknesses at the beginning are more emphasized. After some iterations of training, the neural network becomes smarter, and Rave can therefore enhance the performance as shown in [19].

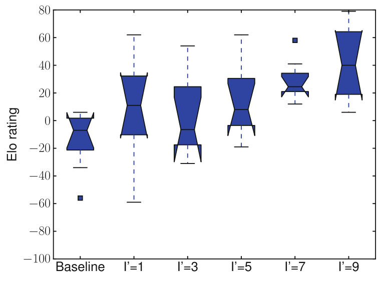
6.2 Fixed I' Tuning

Taking Connect Four as an example, in this experiment we search for an optimal fixed I' value, utilizing the warm-start search method proposed in [19]. We set I' as 1, 3, 5, 7, 9 respectively (the value should be relatively small since the enhancement is only expected to be used at the start phase of training). The Elo ratings of each enhancements using different I' are presented in Fig. 1.

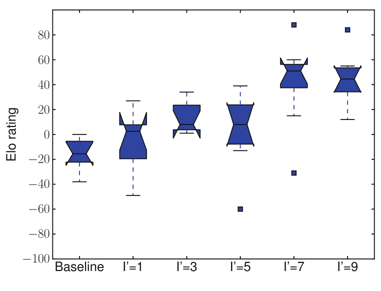
The Elo ratings are calculated based on the tournament results using a Bayesian Elo computation system [2], same for Fig. 2. We can see that for Rave and WRoRa, it turns out that $I' = 7$ is the optimal value for fixed I' warm-start framework, for others, it is still unclear which value is the best, indicating that the tuning is inefficient and costly.

6.3 Adaptive Warm-Start Switch

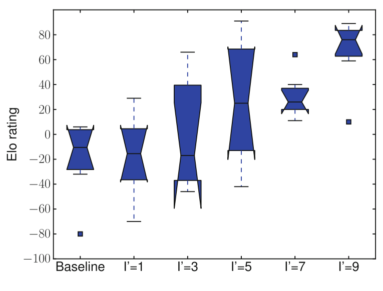
In this final experiment, we train models with the parameters in Table 1 and then let them compete against each other in different games. In addition, we record the specific iteration number where the switch occurs for every training run and the corresponding self-play arena rewards of MCTS before this iteration. A statistic of the iteration number for the 3 games is shown in Table 3. The table shows that, generally, the iteration number is relatively small compared to the total length of the training (100 iterations). Besides, not only for different



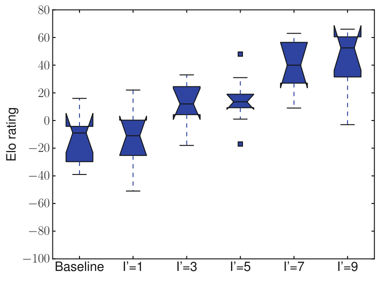
(a) Rollout



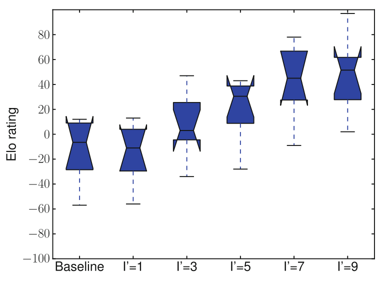
(b) Rave



(c) RoRa



(d) WRo



(e) WRoRa

Fig. 1. Elo ratings for different warm-start phase iterations with different search enhancement on 6×6 Connect Four

games, but also for different training runs on the same game, the switch iteration varies. This is because for different training runs, the neural network training progresses differently. Therefore, a fixed I' can not be used for each specific training. Note that for Gobang, a game with a large branching factor, with the default setting, it always switches at the first iteration. Therefore, we also test with larger $m = 200$, thereby providing more time to the MCTS. With this change, there are several runs keeping the enhancements see Table 3, but it still shows a small influence on this game.

Table 3. Switching iterations for training on different games with different enhancements over 8 repetitions (average iteration number ± standard deviation)

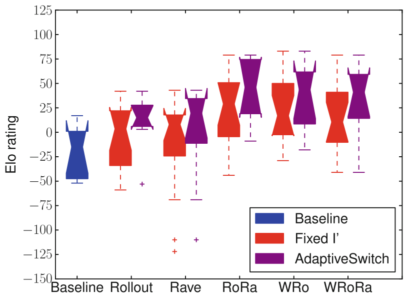
	Connect Four	Othello	Gobang
Rollout	6.625 ± 3.039	5.5 ± 1.732	1.375 ± 0.484
Rave	2.375 ± 1.218	3.125 ± 2.667	1.125 ± 0.331
RoRa	7.75 ± 4.74	5.125 ± 1.364	1.125 ± 0.331
WRo	4.25 ± 1.561	4.375 ± 1.654	1.125 ± 0.331
WRoRa	4.375 ± 1.576	4.0 ± 1.0	1.25 ± 0.433

More importantly, we collect all trained models based on our adaptive method, and let them compete with the models trained using fixed $I' = 5$ in a full round-robin tournament where each 2 players play 20 games.

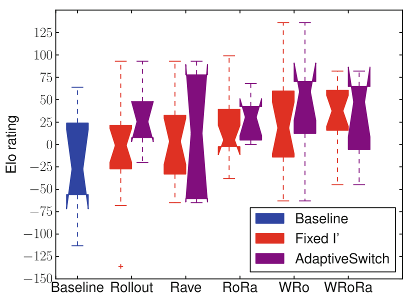
From Fig. 2, we see that, generally, on both Connect Four and Othello, all fixed I' achieve higher Elo ratings than the Baseline, which was also reported in [19]). And all adaptive switch models also perform better than the Baseline. Besides, for each enhancement, the Elo ratings of the adaptive switch models are higher than for the fixed I' method, which suggests that our adaptive switch method leads to better performance than the fixed I' method when controlling the warm-start iteration length. Specifically, we find that for Connect Four, WRo and RoRa achieve the higher Elo Ratings (see Fig. 2(a)) and for Othello, WRoRa performs best (see Fig. 2(b)), which reproduces the consistent conclusion (at least one combination enhancement performs better in different games) as [19]).

In addition, for Connect Four, comparing the tuning results in Fig. 1 and the *switch iterations* in Table 3, we find that our method generally needs a shorter warm-start phase than employing a fixed I'. The reason could be that in our method, there are always 2 different players playing the game, and they provide more diverse training data than a pure self-play player. In consequence, the neural network also improves more quickly, which is highly desired.

Note that while we use the default parameter setting for training in the Gobang game, the *switch* occurs at the first iteration. And even though we enlarge the simulation times for MCTS, only a few training runs shortly keep using the enhancements. We therefore presume that it is meaningless to further perform the tournament comparison for Gobang.



(a) Connect Four



(b) Othello

Fig. 2. Comparison of adaptive switch method versus fixed I' based on a full tournament for 6×6 Connect Four and Othello

7 Discussion and Conclusion

Since AlphaGo Zero' results, self-play has become a default approach for generating training data tabula rasa, disregarding other information for training. However, if there is a way to obtain better training examples from the start, why not use them, as has been done recently in StarCraft (see DeepMind's AlphaStar [14]). In addition [19] investigate the possibility of utilizing MCTS enhancements to improve AlphaZero-like self-play. They embed Rollout, RAVE and combinations as enhancements at the start period of iterative self-play training and tested this on small board games.

Confirming [19], we find that finding an optimal value of fixed I' is difficult, therefore, we propose an adaptive method for deciding when to switch. We also use Rollout, RAVE, and combinations with network values to quickly improve MCTS tree statistics (using RAVE) with meaningful information (using Rollout) before we switch to Baseline-like self-play training. We employed the same games, namely the 6×6 versions of Gobang, Connect Four, and Othello. In these experiments, we find that, for different games, and even different training runs for the same game, the new adaptive method generally switches at different iterations. This indicates the noise in the neural network training progress for different runs. After 100 self-play iterations, we still see the effects of the warm-start enhancements as playing strength has improved in many cases, and for all enhancements, our method performs better than the method proposed in [19] with I' set to 5. In addition, some conclusions are consistent to [19], for example, there is also at least one combination that performs better.

The new adaptive method works especially well on Othello and Connect Four, "deep" games with a moderate branching factor, and less well on Gobang, which has a larger branching factor. In the self-play arena, the default MCTS is already quite strong, and for games with a short and wide episode, the MCTS enhancements do not benefit much. Short game lengths reach terminal states early, and MCTS can use the true reward information more often, resulting in a higher chance of winning. Since, Rollout still needs to simulate, with a limited simulation count it is likely to not choose a winning terminal state but a state that has the same average value as the terminal state. In this situation, in a short game episodes, MCTS works better than the enhancement with $T' = 15$. With ongoing training of the neural network, both players become stronger, and as the game length becomes longer, $I' = 5$ works better than the the Baseline.

Our experiments are with small games. Adaptive warm-start works best in deeper games, suggesting a larger benefit for bigger games with deeper lines. Future work includes larger games with deeper lines, and using different but stronger enhancements to generate training examples.

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

References

1. Browne, C.B., et al.: A survey of Monte Carlo tree search methods. IEEE Trans. Comput. Intell. AI Games **4**(1), 1–43 (2012)
2. Coulom, R.: Whole-history rating: a Bayesian rating system for players of time-varying strength. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) CG 2008. LNCS, vol. 5131, pp. 113–124. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87608-3_11
3. Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: Proceedings of the 24th International Conference on Machine Learning, pp. 273–280 (2007)
4. Gelly, S., Silver, D.: Monte-Carlo tree search and rapid action value estimation in computer go. Artif. Intell. **175**(11), 1856–1875 (2011)
5. Nair, S.: Alphazero general (2018). https://github.com/suragnair/alpha-zero-general. Accessed May 2018
6. Plaat, A.: Learning to Play: Reinforcement Learning and Games. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59238-7
7. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
8. Segler, M.H., Preuss, M., Waller, M.P.: Planning chemical syntheses with deep neural networks and symbolic AI. Nature **555**(7698), 604–610 (2018)
9. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
10. Silver, D.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
11. Silver, D., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (2017)
12. Sutton, R.S., Barto, A.G.: Reinforcement learning: An Introduction. MIT Press, Cambridge (2018)
13. Tesauro, G.: Temporal difference learning and TD-Gammon. Commun. ACM **38**(3), 58–68 (1995)
14. Vinyals, O.: Grandmaster level in starcraft II using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)

15. Wang, H., Emmerich, M., Plaat, A.: Assessing the potential of classical Q-learning in general game playing. In: Atzmueller, M., Duivesteijn, W. (eds.) BNAIC 2018. CCIS, vol. 1021, pp. 138–150. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31978-6_11
16. Wang, H., Emmerich, M., Preuss, M., Plaat, A.: Alternative loss functions in alphazero-like self-play. In: 2019 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 155–162. IEEE (2019)
17. Wang, H., Emmerich, M., Preuss, M., Plaat, A.: Analysis of hyper-parameters for small games: Iterations or epochs in self-play? arXiv preprint arXiv:2003.05988 (2020)
18. Wang, H., Preuss, M., Emmerich, M., Plaat, A.: Tackling morpion solitaire with alphazero-like ranked reward reinforcement learning. arXiv preprint arXiv:2006.07970 (2020)
19. Wang, H., Preuss, M., Plaat, A.: Warm-start alphazero self-play search enhancements. In: Proceedings of the Parallel Problem Solving from Nature - PPSN XVI, pp. 528–542 (2020)
20. Wu, D.J.: Accelerating self-play learning in go. arXiv preprint arXiv:1902.10565 (2019)