AlphaGo Zero & AlphaZero
Mastering Go, Chess and Shogi wihtout human knowledge

Seminar Report: Artificial Intelligence for Games SS19

Philipp Wimmer

August 30, 2019

Contents

1 Motivation

Our goal in the seminar "Artificial Intelligence for Games" was to gain insight into the exciting field of Artificial Intelligence. The application to Games is obvious when considering the complexity of the real world. Testing concepts in the context of games where rules and objectives are clearly defined present a much more manageable problem.
Much of the work preceding the presented paper did not focus on creating real intelligence but instead relied on brute-force methods to achieve superhuman performance. It mostly focuses on search algorithms that try to efficiently evaluate as much of the decision tree as possible rather than relying on human traits like knowledge or intuition. Go was a long standing challenge in the field of Artificial Intelligence because of its considerably higher complexity than chess. In the papers "Mastering the game of Go without human knowledge" and "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm" Silver et al. achieved a major breakthrough by introducing AlphaGo Zero and AlphaZero. They were able to surpass state-of-the-art performance without relying on any human knowledge. They use a novel self-play algorithm to train a neural network to augment a Monte Carlo Tree Search (MCTS). Their program is able to extract an understanding of the game that is much more "human-like" than any previous work. Additionally, the fact that they do not use any domain knowledge widens the applicability of their work considerably.

2 Related Work and Predecessors

2.1 Pachi

Pachi is an open source Go program that was introduced to the public by Petr Baudiš et al. in 2011 [2]. It implements a variant of the Monte Carlo Tree search algorithm. They use a simplified version of the RAVE (Rapid Action-Value Estimates) algorithm [3] to choose the next action during descent. It focuses on a compromise between UCT and additional heuristic information. Their heuristics are very specific to Go and also include a dictionary of known moves. They reported that their final performance was achieved after extensive tuning of 80 different hyperparameters. Pachi was able to win multiple KGS Go Bot Tournaments and is ranked as amateur 3-dan (this is comparable to advanced amateur play) [12].

2.2 Crazy Stone

Crazy Stone is a proprietary Go program, developed by Rémi Coulom [4] which was originally published in 2012. It utilizes MCTS with UCT. In addition, the author leverages pattern-based heuristics to determine potentially advantageous moves during playouts thus narrowing down the search tree. In contrast to the purely handcrafted features of Pachi, Crazy Stone originally used an unspecified supervised machine learning approach to learn these heuristics. The machine learning approach used a combination of several predefined pattern-based features. Crazy stone consistently outperformed Pachi in KGS Go Bot Tournaments and was awarded an amateur 6-dan rating [5]. Additionally, it was the first Go algorithm to play at a professional level. In 2016, Rémi Coulom published a new version which replaced the former pattern-based approach with deep-learning. This new version significantly improved Crazy Stone's playing strength.

2.3 AlphaGo

> *This is the first time that a computer program has defeated a human professional player in the full sized game of Go, a feat previously thought to be at least a decade away.*
>
> – Silver et. al, 2015

AlphaGo is a Go Program that was developed by Google DeepMind [10]. In 2015, it was the first computer program to beat a human professional Go player. In March 2016, it was able to beat Lee Sedol, making it the first program to beat a professional

9-dan level player without handicap. In contrast to prior work which were based on MCTS and enhanced by shallow policies, AlphaGo uses a different approach to improve MCTS. It uses two neural networks, a value and a policy network to reduce the depth, and breadth of the search tree. These networks are trained using a pipeline consisting of several stages of machine learning:

Supervised learning of policy networks The goal of using a policy network is to reduce the breadth of the search tree by focusing more attention on moves that the policy networks deems as advantageous. The SL policy network $p_\sigma(a|s)$ is a convolutional neural network with alternating convolutional layers with weights σ and rectifier nonlinearities. This network gets trained to predict human expert moves using gradient descent:

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma} \tag{2.1}$$

In addition to p_σ, they trained a simpler and faster rollout policy $p_\pi(a|s)$. This policy replaces the CNN with a simple linear softmax classifier. This classifier uses small handcrafted local features which are similar to the features that the original implementation of Crazy Stone used. Using this fast rollout policy p_π only takes $2\mu s$ instead of $3ms$ for the policy network p_σ.

Reinforcement learning of policy networks This stage tries to improve the policy network p_σ by self play. They define a new network p_ρ that is initialized as a copy of p_σ. This network is trained by policy gradient reinforcement learning. They define a reward function $r(t)$ which equals zero for all non-terminal time steps. The reward for a terminal time step is $+1$ for winning and -1 for loosing. This means that they only use terminal rewards $z_t = r(s_T)$ for gradient descent:

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial\sigma} z_t \tag{2.2}$$

Reinforcement learning of value networks The use of a value network enables the MCTS to truncate the MCTS rollout at specific depth and compute the expected outcome with the value function $v^p(s)$.

$$v^p(s) = \mathbb{E}[z_t|s_t = s, a_{t...T} \sim p] \tag{2.3}$$

Given that both players use the same policy p. Given that Go is a two player zero sum game with perfect information, there exists an optimal value function under perfect play $v^*(s)$. The authors approximate $v^*(s)$ by estimating the value function for p_ρ with the value network $v_\theta(s)$ ($v^*(s) \approx v^{p_\rho}(s) \approx v^\theta(s)$). Instead of predicting game outcomes from complete games, which leads to overfitting, they generate a self-play dataset consisting of 30 million distinct positions sampled from distinct games. A single evaluation of v_θ significantly outperforms a policy guided Monte Carlo rollout using p_ρ, while taking 15000 times less computation time. The value

network gets trained by minimizing the mean squared error between the predicted value and the observed outcome z by gradient descent:

$$\delta\theta \propto \frac{\partial v_\theta}{\partial \theta}(z - v_\theta(s)) \tag{2.4}$$

Searching with policy and value networks The authors combine the policy and value networks with an MCTS algorithm. The tree saves an action value $Q(s,a)$, visit count $N(s,a)$, and prior probability $P(s,a)$ for each of its edges (s,a). At each non-terminal timestep t, an action a_t is selected from state s_t:

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t,a) + u(s_t,a)) \tag{2.5}$$

The bonus $u(s_t,a)$ is selected using a variant of the PUCT algorithm [8]. The bonus is given by:

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)} \tag{2.6}$$

where the prior probability is calculated with by SL-trained policy network: $P(s,a) = p_\sigma(a|s)$. When reaching a leaf node at timestep L, the win probability is calculated by the evaluation of the value network $v_\theta(S_L)$ and the outcome of a rollout using the fast rollout policy p_π. The results are combined using a mixing parameter λ.

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L \tag{2.7}$$

After the simulation is complete, the visit count and the action value get updated:

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i) \tag{2.8}$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} (s,a,i) V(s_L^i) \tag{2.9}$$

3 AlphaGo Zero

Maybe it can show humans something we've never discovered. Maybe it's beautiful.

– Fan Hui, 2016

AlphaGo Zero is the Google Deepmind's successor to AlphaGo [11]. It falls into the category of deep learning augmented MCTS algorithms. In contrast to AlphaGo, it is trained completely unsupervised and no domain knowledge other than the rules of the game is implemented. Thus, the main contribution of the paper is to demonstrate that superhuman performance can be achieved without relying on human knowledge. Furthermore, the value and policy networks get replaced by a single network that combines both tasks. This single CNN only uses the raw board as input.

3.1 Search algorithm

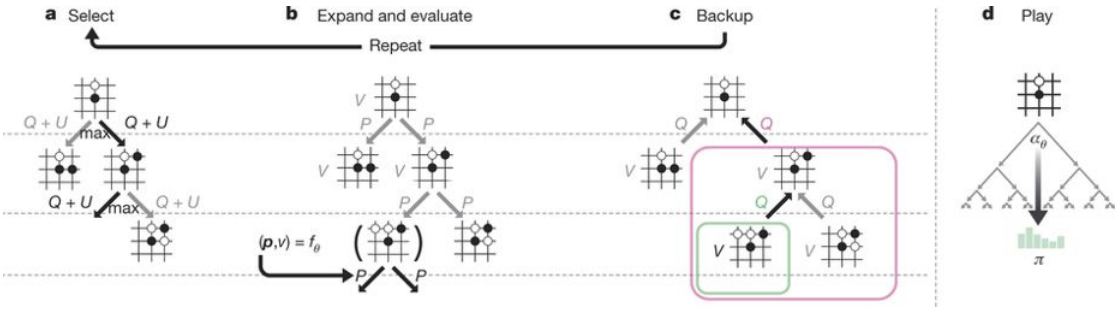

Figure 3.1: **Search agorithm**

AlphaGo Zero uses a similar implementation of the MCTS algorithm used in AlphaGo. Each node in the tree contains edges (s, a) for all legal actions $a \in A(s)$.. Each node stores the following statistics:

$$\{N(s,a), W(s,a), Q(s,a), P(s,a)\} \tag{3.1}$$

where $N(s, a)$ is the visit count, W(s,a) is the total action value, $Q(s, a)$ is the mean action value and $P(s, a)$ is the prior probability. The implemented algorithm follows the basic structure of the vanilla MCTS. The main difference between AlphaGo Zero and AlphaGo is that during play AlphaGo Zero does not do any random rollouts. It simply uses the neural network together with look-ahead search to select the best action.

Select (Figure 3.1a) At the beginning of each simulation, the node of the search tree s_0 is selected. The simulation finishes when it reaches a leaf node s_L at time step L. At each time-step $t < L$ an action is selected according to the PUCT algorithm. This step is completely identical to AlphaGo.

Expand and evaluate (Figure 3.1b) The leaf node is evaluated by the neural network:

$$d_i(\mathbf{p}, v) = f_\theta(d_i(s_L)) \tag{3.2}$$

where d_i is a random dihedral reflection. This is done to leverage Go's inherent symmetry for training data augmentation. The node is then expanded and each edge (s_L, a) is initialized to:

$$\{N(s_L, a) = 0, W(s_L, a) = 0, Q(s_L, a) = 0, P(s_L, a) = p_a\} \tag{3.3}$$

Backup (Figure 3.1c) After expansion, all edge statistics are updated in a backward pass through all previous time-steps: $t \leq L$.

$$N(s_t, a_t) = N(s_t, a_t) + 1 \tag{3.4}$$

$$W(s_t, a_t) = W(s_t, a_t) + v \tag{3.5}$$

$$Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)} \tag{3.6}$$

Play (Figure 3.1d) After the end of the search, AlphaGo Zero selects the move a to play in the root position s_0 according to:

$$\pi(a|s_0) = \frac{N(s_0, a)^{1/\tau}}{\sum_b N(s_0, b)^{1/\tau}} \tag{3.7}$$

where τ is a temperature parameter controlling exploration. The played node gets selected as the new root node s_0 and the tree above is discarded.

3.2 Self-play training pipeline

AlphaGo Zero uses one CNN with two regression heads to calculate the move probabilities $p_a = Pr(a|s)$ and the probability of the current player winning from the current position v. This network $f_\theta(s) = (\mathbf{p}, v)$ gets trained by a novel self-play reinforcement learning algorithm. The core idea is to use the MCTS itself as a powerful policy and value improvement operator. During training an MCTS, search guided by the neural network f_θ is carried out. The resulting move probabilities π are usually much stronger than the raw move probabilities $\mathbf{p}$ of the neural network $f_\theta(s)$. The network is then trained to better predict the improved move capabilities π. The outcome z of the self play games obtained by playing with the MCTS-based policy is used to train the network to better predict the winner of the game. The use of both operators is repeated in a policy iteration procedure leading to even better games carried out by the improving neural network f_θ in successive games. The training pipeline consists of three main components which are all executed asynchronously in parallel.

Optimization Each neural network f_θ is trained with mini batches that are sampled randomly from the last 500000 games of self-play. The network parameters θ are optimized by minimizing the loss function

$$(\mathbf{p}, v) = f_\theta(s) \text{ and } l = (z - v)^2 - \pi^T \log \mathbf{p} + c||\theta||^2 \tag{3.8}$$

by gradient descent. The first part of the loss function is the mean squared error between the predicted value v and the self-play winner z. The second part is the cross entropy loss between the predicted move probabilities $\mathbf{p}$ and the search probabilities π that were obtained using MCTS to augment $\mathbf{p}$. Because π is always a stronger prediction than $\mathbf{p}$, they achieve good convergence. The increasingly better play of the network also ensures that the accuracy of the value v improves steadily.

Evaluator To ensure the quality of the training data the network weights θ are not updated continuously. Instead, the performance of the updated neural network gets evaluated regularly against the previous version. Each evaluation consists of 400 games. If the new network wins by a margin of $> 55\%$, the network gets updated with the new weights. This suppresses noise during training.

Self-play The self-play dataset is continuously fed with new data from the actual network f_θ playing against itself. The difference to the non-self play algorithm is that the policy network gets augmented by 1600 Monte Carlo simulations. To ensure good training data, Dirichlet noise is added to the prior probabilities at the root node s_0. This increases exploration. To further increase exploration during the first 30 moves of the game, the temperature is set to $\theta = 1$. The temperature is set to $\theta = 0$ for the remainder of the game.

3.3 Neural network architecture

Much of the improved performance of AlphaGo Zero over AlphaGo can be attributed to the new network architecture. The two major changes are the use of residual nets and one single network instead of two.
When AlphaGo was released to the public in 2015, the concept of residual learning in the context of deep-learning was not introduced yet. This concept was introduced by He et al. one year later in 2016 [6]. The use of residual networks enables the training of deeper and thus more powerful networks by introducing skip connections. These connections represent a shortcut for the gradient during backpropagation thus solving the problem of vanishing gradients.
The use of one network for multiple related tasks is known as multi-task learning (MTL). MTL is successfully employed in many different fields like language processing, speech recognition, and computer vision [9]. MTL has a strong regularizing effect because the network has to find a common suitable representation for multiple tasks. This representation appears to be more general and performs better than two separate networks. Furthermore, the computational efficiency is significantly improved because most of the network is shared between the two regression heads.

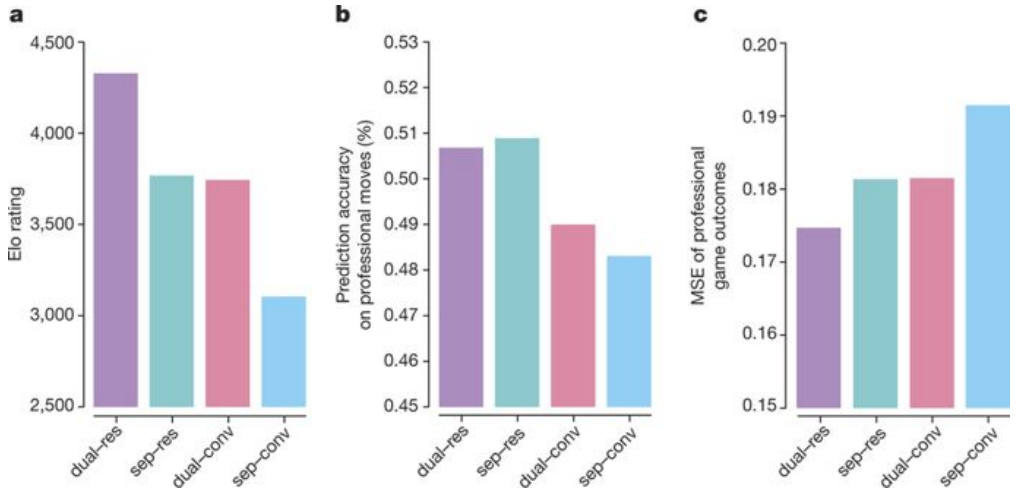

Figure 3.2: **Comparison of neural network architectures in AlphaGo Zero and AlphaGo.** To separate contributions of architecture and algorithm, a variant of AlphaGo Zero with the old architecture of AlphaGo was implemented. The 'sep-conv' architecture corresponds to the original architecture of AlphaGo with two separate networks. Changing to one single network ('dual-conv') considerably increases performing. It is now on par with two separate residual networks ('sep-res'). The highest performance is achieved by using one single residual network ('dual-res'). This architecture is slightly worse at predicting expert moves than 'sep-res' but clearly outperformed it at predicting game outcomes.

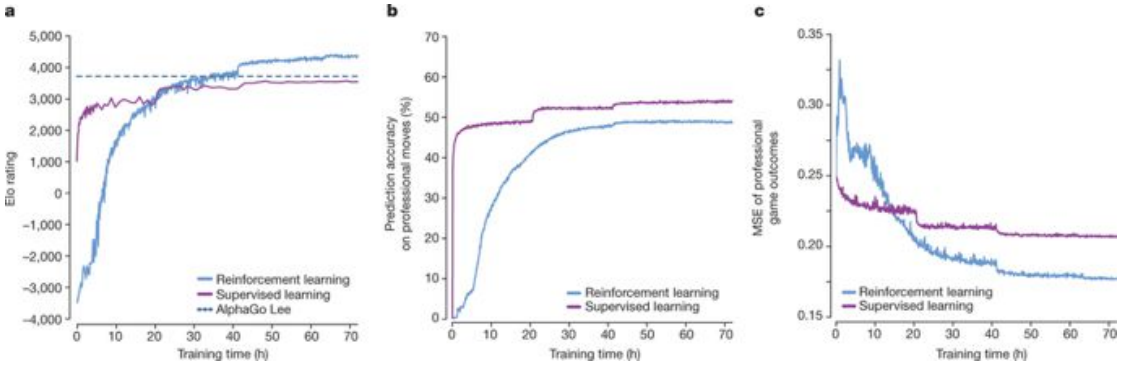

Figure 3.3: **Supervised vs reinforcement learning a** Performance comparison between AlphaGo, a supervised version of AlphaGo Zero, and the reinforcement learning version. The supervised version reaches the same performance as AlphaGo. This indicates that the human knowledge used for training creates a bottleneck. **b** Prediction accuracy on human professional moves. The supervised version converges considerably faster and reaches higher accuracy. **c** Mean-squared error (MSE) human professional game outcomes. The reinforcement learning version converges slower but ultimately performs much better.

3.4 Performance

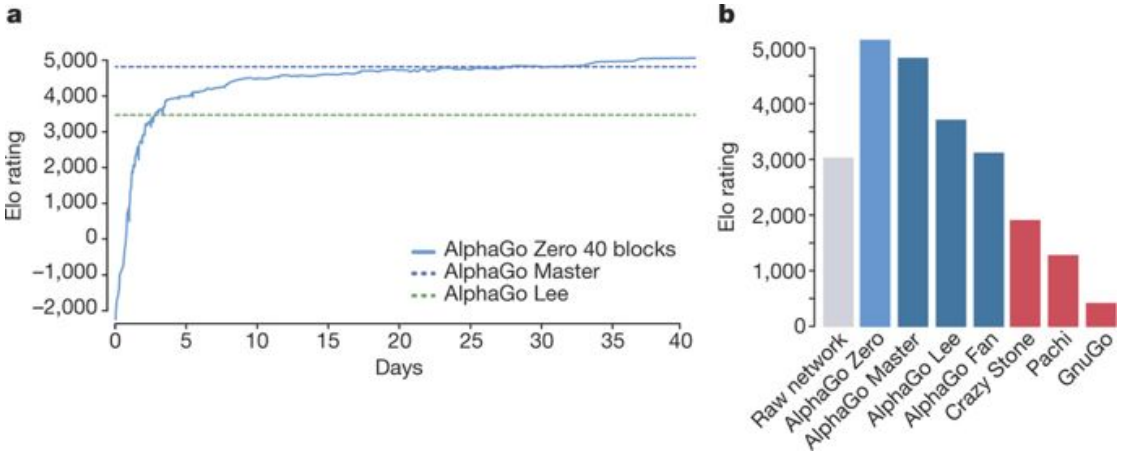

Figure 3.4: **Performance of AlphaGo Zero. a** The plot shows the performance of the best network of AlphaGo Zero over a period of 40 days. It is able to outperform the original version of AlphaGo (Lee) after only 3 days and is outperforming AlphaGo (Master) after 30 days. **b** AlphaGo Zero outperforms all previous approaches. The raw network (not using lookahead search) outperforms Crazy Stone by 1000 points. A 200-point gap corresponds to a 75% win rate.

AlphaGo Zero outperforms all previous approaches. This is particularly impressive because all prior attempts relied on huge datesets of human expert knowledge. The results imply that the quality of the training data set limits the performance. The performance improvement can only be attributed to the employed deep learning techniques because all tested programs rely on the same basic version of MCTS with a variant of UCT to balance exploitation and exploration. The only thing in which they differ is the way they compute their prior probabilities of actions $\mathbf{p}$ and the value of the states $v(s)$. Pachi relies on handcrafted heuristics, Crazy stone learns supervised to evaluate handcrafted features and AlphaGo learns an own feature representation. Particularly impressive is that the raw network outputs of AlphaGo Zero are able to defeat all programs prior to original version of AlphaGo. The network is good enough to play by intuition and does not rely on lookahead to reach human expert level performance. This has the side effect that AlphaGo Zero is computationally more efficient compared to AlphaGo. The concept of a strong network guiding the MCTS search is clearly superior to brute force simulation with a fast (but also very simple) rollout policy.

3.5 Learned knowledge

AlphaGo Zero does not use any domain knowledge and is trained without any human supervision. This means that it is not constrained to human play. Nonetheless it learned previously known joseki (standard moves), just to later drop these moves and develop knew previously unknown variants. It was able to quickly progress from completely random play towards a deep high level understanding of advanced Go concepts. These concepts were developed by professional players over centuries.

4 AlphaZero

AlphaZero is Google Deepmind's successor to AlphaGo Zero [1]. The fact that AlphaGo Zero only uses minimal domain knowledge and does not rely on the existence of an extensive dataset of expert level games, enables its use for any two player zero-sum game with perfect information. The authors evaluated the performance on Chess, Shogi, and Go.

4.1 Differences to AlphaGo Zero

While AlphaGo Zero only uses minimal domain specific techniques, several changes where needed in order to apply the algorithm to Chess and Shogi.

Data augmentation Go exhibits dihedral symmetry. This fact was used during training for data augmentation but also during MCTS search. The authors argue that this decreases bias. Both, Chess and Shogi, are highly asymmetric games. Thus, AlphaZero omits any data augmentations and makes no domain specific assumptions.

Value function The game outcome of Go is always either a win or a loss. In chess and Shogi there also exists the possibility of a draw. AlphaZero predicts the game outcome taking account of draws and other potential outcomes.

Evaluator The authors observed that the Evaluator from AlphaGo Zero is not strictly necessary and thus chose to omit it in AlphaZero. Instead, the weights of the network get updated continuously.

Hyperparameters The hyperparameters for AlphaGo Zero were tuned by Bayesian optimization. For AlphaZero, no game specific optimization is done. All games are played with the same set of hyperparameters.

4.2 Performance

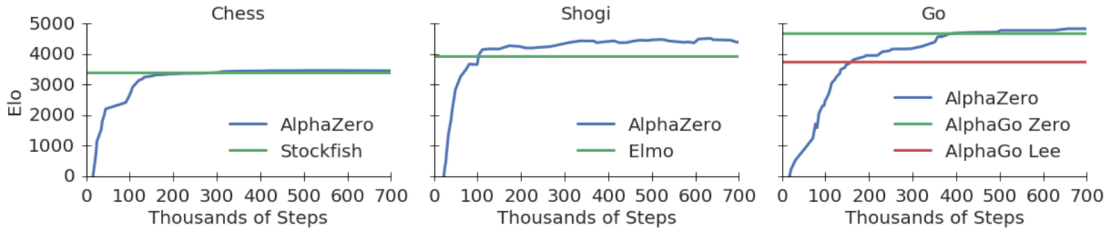

Figure 4.1: **Performance of AlphaZero in Chess, Shogi and Go.**

Chess Chess was long seen as the pinnacle of AI research. The previous best performer Stockfish [7] is based on alpha-beta search engine with many highly domain-specific adaptions. It evaluates 70 million positions per second, compared to AlphaZero which only evaluates 80 thousand positions per seconds. AlphaZero was able to outperform Stockfish without any human knowledge. This indicates that the learned knowledge present in the neural network is of considerably higher quality than the handcrafted heuristics of Stockfish.

Shogi Shogi is a Japanse board-game that is very similar to Chess. AlphaZero was able to outperform the previous top performer Elmo by a greater margin than it was able to against Stockfish. This highlights a limitation of approaches with handcrafted heuristics: They are game specific and not transferable to other problems. The considerably smaller research community led to less sophisticated heuristics and thus to worse performance than Stockfish is able to achieve for chess. The authors of AlphaZero did not have any expert knowledge about the game except its rules. This was enough to achieve state of the art performance.

Go AlphaZero outperforms AlphaGo Zero in Go by a small margin. This indicates that neither the absence of domain knowledge nor the absence of the evaluator degrades the performance. The increased performance can be explained by the fact that the training of AlphaGo Zero was stopped prematurely.

5 Conclusion

The authors were able to surpass state of the art performance in Go, Chess and Shogi while not relying on human training data. Their main contribution is that their approach is very general and easily applicable to any two person perfect information board game. Previous work relied on two main points: being able to evaluate as much of the decision tree as possible and using very domain specific handcrafted heuristics. The first point clearly showed its limits for games with a huge search space like AlphaGo and the second point severely limits transferability to other problems.
AlphaGo Zero and AlphaZero do not rely on the evaluation of as many positions as possible but instead on the quality of learned knowledge thus enabling more 'human-like' play and an AI that is able to develop new strategies instead of simply imitating human play.

Bibliography

[1] *[1712.01815] Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. URL: https://arxiv.org/abs/1712.01815 (visited on 08/30/2019).

[2] Petr BaudiÅ¡ And Jean-loup Gailly. "PACHI: State of the Art Open Source Go Program". In: *Advances in Computer Games*. Ed. by H. Jaap van den Herik and Aske Plaat. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 24–38. ISBN: 978-3-642-31866-5.

[3] Guillaume Chaslot et al. "Adding Expert Knowledge and Exploration in Monte-Carlo Tree Search". In: *Advances in Computer Games*. Ed. by H. Jaap van den Herik and Pieter Spronck. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 1–13. ISBN: 978-3-642-12993-3.

[4] RÃ©mi Coulom. "Monte-Carlo Tree Search in Crazy Stone". In: *Proc. Game Prog. Workshop, Tokyo, Japan* (2007), p. 26.

[5] *CrazyStone at Sensei's Library*. URL: https://senseis.xmp.net/?CrazyStone (visited on 08/27/2019).

[6] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016, pp. 770–778. URL: http://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html (visited on 08/29/2019).

[7] Tord Romstad, Marco Costalba, and Joona Kiiski. *Home - Stockfish - Open Source Chess Engine*. URL: https://stockfishchess.org/ (visited on 08/30/2019).

[8] Christopher D. Rosin. "Multi-Armed Bandits with Episode Context". In: *Annals of Mathematics and Artificial Intelligence* 61.3 (Mar. 1, 2011), pp. 203–230. ISSN: 1573-7470. DOI: 10.1007/s10472-011-9258-6. URL: https://doi.org/10.1007/s10472-011-9258-6 (visited on 08/30/2019).

[9] Sebastian Ruder. "An Overview of Multi-Task Learning in Deep Neural Networks". In: (June 15, 2017). arXiv: 1706.05098 [cs, stat]. URL: http://arxiv.org/abs/1706.05098 (visited on 08/29/2019).

[10] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: https://www.nature.com/articles/nature16961 (visited on 08/27/2019).

[11] David Silver et al. “Mastering the Game of Go without Human Knowledge”. In: *Nature* 550.7676 (Oct. 2017), pp. 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270. URL: https://www.nature.com/articles/nature24270 (visited on 08/28/2019).

[12] Nick Wedd. *Computer Go Tournaments on KGS*. URL: http://www.weddslist.com/kgs/ (visited on 08/27/2019).