Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver,[1*] Thomas Hubert,[1*] Julian Schrittwieser,[1*]
Ioannis Antonoglou,[1] Matthew Lai,[1] Arthur Guez,[1] Marc Lanctot,[1]
Laurent Sifre,[1] Dharshan Kumaran,[1] Thore Graepel,[1]
Timothy Lillicrap,[1] Karen Simonyan,[1] Demis Hassabis[1]

[1]DeepMind, 6 Pancras Square, London N1C 4AG.
[*]These authors contributed equally to this work.

Abstract

The game of chess is the most widely-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. In contrast, the *AlphaGo Zero* program recently achieved superhuman performance in the game of Go by *tabula rasa* reinforcement learning from games of self-play. In this paper, we generalise this approach into a single *AlphaZero* algorithm that can achieve, *tabula rasa*, superhuman performance in many challenging games. Starting from random play, and given no domain knowledge except the game rules, *AlphaZero* achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case.

The study of computer chess is as old as computer science itself. Babbage, Turing, Shannon, and von Neumann devised hardware, algorithms and theory to analyse and play the game of chess. Chess subsequently became the grand challenge task for a generation of artificial intelligence researchers, culminating in high-performance computer chess programs that perform at superhuman level (*9, 14*). However, these systems are highly tuned to their domain, and cannot be generalised to other problems without significant human effort, while general game-playing systems (*12, 25*) remain comparatively weak.

A long-standing ambition of artificial intelligence has been to create programs that can instead learn for themselves from first principles (*28*). Recently, the *AlphaGo Zero* algorithm achieved superhuman performance in the game of Go, by representing Go knowledge using deep convolutional neural networks (*23, 30*), trained solely by reinforcement learning from

games of self-play (*31*). In this paper, we apply a similar algorithm, which we call *AlphaZero*, to the games of chess and shogi as well as Go, without any additional domain knowledge except the rules of the game. Our results demonstrate that a general-purpose reinforcement learning algorithm can achieve, *tabula rasa*, superhuman performance across many challenging games.

A landmark for artificial intelligence was achieved in 1997 when Deep Blue defeated the human world champion (*9*). Computer chess programs continued to progress steadily beyond human level in the following two decades. These programs evaluate positions using features handcrafted by human grandmasters and carefully tuned weights, combined with a high-performance alpha-beta search that expands a vast search tree using a large number of clever heuristics and domain-specific adaptations. In the Methods we describe these augmentations, focusing on the 2016 Top Chess Engine Championship (TCEC) world-champion *Stockfish* (*27*); other strong chess programs, including Deep Blue, use very similar architectures (*9, 22*).

Shogi is a significantly harder game, in terms of computational complexity, than chess (*2, 15*): it is played on a larger board with a wider variety of pieces; any captured opponent piece changes sides and may subsequently be dropped anywhere on the board. The strongest shogi programs, such as Computer Shogi Association (CSA) world-champion *Elmo*, have only recently defeated human champions (*5*). These programs use a similar algorithm to computer chess programs, again based on a highly optimised alpha-beta search engine with many domain-specific adaptations.

Go is well suited to the neural network architecture used in *AlphaGo* because the rules of the game are translationally invariant (matching the weight sharing structure of convolutional networks), are defined in terms of *liberties* corresponding to the adjacencies between points on the board (matching the local structure of convolutional networks), and are rotationally and reflectionally symmetric (allowing for data augmentation and ensembling). Furthermore, the action space is simple (a stone may be placed at each possible location), and the game outcomes are restricted to binary wins or losses, both of which may help neural network training.

Chess and shogi are, arguably, less innately suited to *AlphaGo*'s neural network architectures. The rules are position-dependent (e.g. pawns may move two steps forward from the second rank and promote on the eighth rank) and asymmetric (e.g. pawns only move forward, and castling is different on kingside and queenside). The rules include long-range interactions (e.g. the queen may traverse the board in one move, or checkmate the king from the far side of the board). The action space for chess includes all legal destinations for all of the players' pieces on the board; shogi also allows captured pieces to be placed back on the board. Both chess and shogi may result in draws in addition to wins and losses; indeed it is believed that the optimal solution to chess is a draw (*18, 21, 32*).

The *AlphaZero* algorithm is a more generic version of the *AlphaGo Zero* algorithm that was first introduced in the context of Go (*31*). It replaces the handcrafted knowledge and domain-specific augmentations used in traditional game-playing programs with deep neural networks and a *tabula rasa* reinforcement learning algorithm.

Instead of a handcrafted evaluation function and move ordering heuristics, *AlphaZero* utilises a deep neural network $(\mathbf{p}, v) = f_\theta(s)$ with parameters θ. This neural network takes the board po-

sition s as an input and outputs a vector of move probabilities $\mathbf{p}$ with components $p_a = Pr(a|s)$ for each action a, and a scalar value v estimating the expected outcome z from position s, $v \approx \mathbb{E}[z|s]$. *AlphaZero* learns these move probabilities and value estimates entirely from self-play; these are then used to guide its search.

Instead of an alpha-beta search with domain-specific enhancements, *AlphaZero* uses a general-purpose Monte-Carlo tree search (MCTS) algorithm. Each search consists of a series of simulated games of self-play that traverse a tree from root s_{root} to leaf. Each simulation proceeds by selecting in each state s a move a with low visit count, high move probability and high value (averaged over the leaf states of simulations that selected a from s) according to the current neural network f_θ. The search returns a vector π representing a probability distribution over moves, either proportionally or greedily with respect to the visit counts at the root state.

The parameters θ of the deep neural network in *AlphaZero* are trained by self-play reinforcement learning, starting from randomly initialised parameters θ. Games are played by selecting moves for both players by MCTS, $a_t \sim \boldsymbol{\pi}_t$. At the end of the game, the terminal position s_T is scored according to the rules of the game to compute the game outcome z: -1 for a loss, 0 for a draw, and $+1$ for a win. The neural network parameters θ are updated so as to minimise the error between the predicted outcome v_t and the game outcome z, and to maximise the similarity of the policy vector $\mathbf{p}_t$ to the search probabilities $\boldsymbol{\pi}_t$. Specifically, the parameters θ are adjusted by gradient descent on a loss function l that sums over mean-squared error and cross-entropy losses respectively,

$$(\mathbf{p}, v) = f_\theta(s), \qquad\qquad l = (z - v)^2 - \boldsymbol{\pi}^\top \log \mathbf{p} + c||\theta||^2 \tag{1}$$

where c is a parameter controlling the level of L_2 weight regularisation. The updated parameters are used in subsequent games of self-play.

The *AlphaZero* algorithm described in this paper differs from the original *AlphaGo Zero* algorithm in several respects.

AlphaGo Zero estimates and optimises the probability of winning, assuming binary win/loss outcomes. *AlphaZero* instead estimates and optimises the expected outcome, taking account of draws or potentially other outcomes.

The rules of Go are invariant to rotation and reflection. This fact was exploited in *AlphaGo* and *AlphaGo Zero* in two ways. First, training data was augmented by generating 8 symmetries for each position. Second, during MCTS, board positions were transformed using a randomly selected rotation or reflection before being evaluated by the neural network, so that the Monte-Carlo evaluation is averaged over different biases. The rules of chess and shogi are asymmetric, and in general symmetries cannot be assumed. *AlphaZero* does not augment the training data and does not transform the board position during MCTS.

In *AlphaGo Zero*, self-play games were generated by the best player from all previous iterations. After each iteration of training, the performance of the new player was measured against the best player; if it won by a margin of 55% then it replaced the best player and self-play games were subsequently generated by this new player. In contrast, *AlphaZero* simply maintains a single neural network that is updated continually, rather than waiting for an iteration to complete.

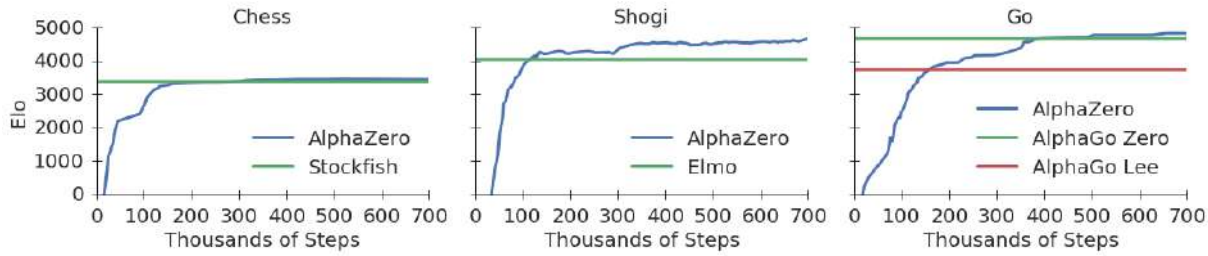

Figure 1: Training *AlphaZero* for 700,000 steps. Elo ratings were computed from evaluation games between different players when given one second per move. **a** Performance of *AlphaZero* in chess, compared to 2016 TCEC world-champion program *Stockfish*. **b** Performance of *AlphaZero* in shogi, compared to 2017 CSA world-champion program *Elmo*. **c** Performance of *AlphaZero* in Go, compared to *AlphaGo Lee* and *AlphaGo Zero* (20 block / 3 day) (*31*).

Self-play games are generated by using the latest parameters for this neural network, omitting the evaluation step and the selection of best player.

AlphaGo Zero tuned the hyper-parameter of its search by Bayesian optimisation. In *AlphaZero* we reuse the same hyper-parameters for all games without game-specific tuning. The sole exception is the noise that is added to the prior policy to ensure exploration (*31*); this is scaled in proportion to the typical number of legal moves for that game type.

Like *AlphaGo Zero*, the board state is encoded by spatial planes based only on the basic rules for each game. The actions are encoded by either spatial planes or a flat vector, again based only on the basic rules for each game (see Methods). Apart from these differences, the same algorithm settings, network architecture, and hyper-parameters were used for all games.

We applied the *AlphaZero* algorithm to chess, shogi, and also Go. We trained a separate instance of *AlphaZero* for each game. Training proceeded for 700,000 steps (mini-batches of size 4,096) starting from randomly initialised parameters, using 5,000 first-generation TPUs (*16*) to generate self-play games and 64 second-generation TPUs to train the neural networks; training lasted for approximately 9 hours in chess, 12 hours in shogi and 34 hours in Go (see Extended Table S3).[1] Further details of the training procedure are provided in the Methods.

Figure 1 shows the performance of *AlphaZero* during self-play reinforcement learning, as a function of training steps, on an Elo scale (*10*).[2] In chess, *AlphaZero* first outperformed *Stockfish* after just 4 hours (300k steps); in shogi, *AlphaZero* first outperformed *Elmo* after 2 hours (110k steps); and in Go, *AlphaZero* first outperformed *AlphaGo Lee* (*31*) after 8 hours (165k steps). Supplementary Figure S2 shows that the training algorithm is repeatable, achieving high performance in all independent runs.

We evaluated the fully trained instances of *AlphaZero* against *Stockfish*, *Elmo* and the previous version of *AlphaGo Zero* in chess, shogi and Go respectively. *AlphaZero* and *AlphaGo Zero* used a single machine with 4 TPUs. The chess match was played using 2016 TCEC world

[1]Note that the original *AlphaGo Zero* paper used GPUs to train the neural networks.

[2]The prevalence of draws in high-level chess tends to compress the Elo scale, compared to shogi or Go.

championship time controls of 3 hours per game, plus an additional 15 seconds for each move, against the 2016 TCEC world champion version of *Stockfish* (see Methods for details). The shogi match was played using the same time controls, against the 2017 CSA world champion version of *Elmo* (see Methods). The Go match was again played using the same time controls, against the previously published *AlphaGo Zero* (trained for 3 days[3]).

In chess, *AlphaZero* defeated *Stockfish*, winning 155 games and losing just 6 games out of 1,000 (see Figure 2). To verify the robustness of *AlphaZero*, we played additional matches that started from common human openings (see Figure 3). *AlphaZero* defeated *Stockfish* in each opening, suggesting that it has indeed mastered a wide spectrum of chess play. The frequency plots in Figure 3 and the timeline in Supplementary Figure S1 show that human openings were indeed independently discovered and played frequently by *AlphaZero* during self-play training. We also played a match that started from the set of opening positions used in the 2016 TCEC world championship; *AlphaZero* won convincingly in this match too[4] (see Supplementary Figure S3). We also played additional matches against the most recent development version of *Stockfish*, and a variant of *Stockfish* that uses a strong opening book[5]. *AlphaZero* won all matches by a large margin (see Figure 2). and Supplementary Table **??**).

Supplementary Table S5 shows 20 chess games played by *AlphaZero* in its matches against *Stockfish*. In several games *AlphaZero* sacrifices material for long-term strategic advantage (e.g. see Supplementary Table S5 swj:game X), suggesting that it has a more fluid, context-dependent positional evaluation than the rule-based evaluations used by previous chess programs.

In shogi, *AlphaZero* defeated *Elmo*, winning 98.2% of games when playing white, and 91.2% overall. We also played a match under the faster time controls used in the 2017 CSA world championship, and against another state-of-the-art shogi program; *AlphaZero* again won both matches by a wide margin (see Figure 2).

tkhubert:Supplementary Table **??** shows 20 shogi games played by *AlphaZero* in its matches against *Elmo*. The frequency plots in Figure 3 and the timeline in Supplementary Figure S1 show that *AlphaZero* frequently plays one of the two commonest human openings, but rarely plays the second, deviating on the very first move.

In Go, *AlphaZero* defeated the previous version of *AlphaGo Zero*, winning 60% of games against an opponent that was trained for double the time and exploited knowledge of board symmetry.

AlphaZero searches just 80 thousand positions per second in chess and 40 thousand in shogi, compared to 70 million for *Stockfish* and 35 million for *Elmo*. *AlphaZero* compensates for the lower number of evaluations by using its deep neural network to focus much more selectively on the most promising variations (Figure 4 provides an example from the match against *Stockfish*) – arguably a more "human-like" approach to search, as originally proposed by Shannon (*29*). *Al-*

[3] *AlphaGo Zero* was ultimately trained for 40 days; we do not reproduce that effort here.

[4] Many TCEC opening positions are unbalanced according to both *AlphaZero* and *Stockfish*, resulting in more losses for both players.

[5] To ensure diversity against a deterministic opening book, *AlphaZero* used a small amount of randomisation in its opening moves (see Methods); this avoided duplicate games but also resulted in more losses.

phaZero still defeated *Stockfish*with ten times less thinking time, and matched the performance of *Elmo*with a hundred times less (see Figure **??**). The high performance of *AlphaZero*, using MCTS, calls into question the widely held belief (*4, 11*) that alpha-beta search is inherently superior in these domains.

The game of chess represented the pinnacle of AI research over several decades. State-of-the-art programs are based on powerful engines that search many millions of positions, leveraging handcrafted domain expertise and sophisticated domain adaptations. *AlphaZero* is a generic reinforcement learning algorithm – originally devised for the game of Go – that achieved superior results within a few hours, searching a thousand times fewer positions, given no domain knowledge except the rules of chess. Furthermore, the same algorithm was applied without modification to the more challenging game of shogi, again outperforming the state of the art within a few hours.

References

1. Online chess games database, 365chess, 2017. URL: *https://www.365chess.com/*.

2. Victor Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, University of Limburg, Netherlands, 1994.

3. Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5366–5376, 2017.

4. Oleg Arenz. Monte Carlo chess. Master's thesis, Technische Universitat Darmstadt, 2012.

5. Computer Shogi Association. Results of the 27th world computer shogi championship. `http://www2.computer-shogi.org/wcsc27/index_e.html`. Retrieved November 29th, 2017.

6. J. Baxter, A. Tridgell, and L. Weaver. Learning to play chess using temporal differences. *Machine Learning*, 40(3):243–263, 2000.

7. Donald F. Beal and Martin C. Smith. Temporal difference learning for heuristic search and game playing. *Inf. Sci.*, 122(1):3–21, 2000.

8. Donald F. Beal and Martin C. Smith. Temporal difference learning applied to game playing and the results of application to shogi. *Theoretical Computer Science*, 252(1–2):105–119, 2001.

9. M. Campbell, A. J. Hoane, and F. Hsu. Deep Blue. *Artificial Intelligence*, 134:57–83, 2002.

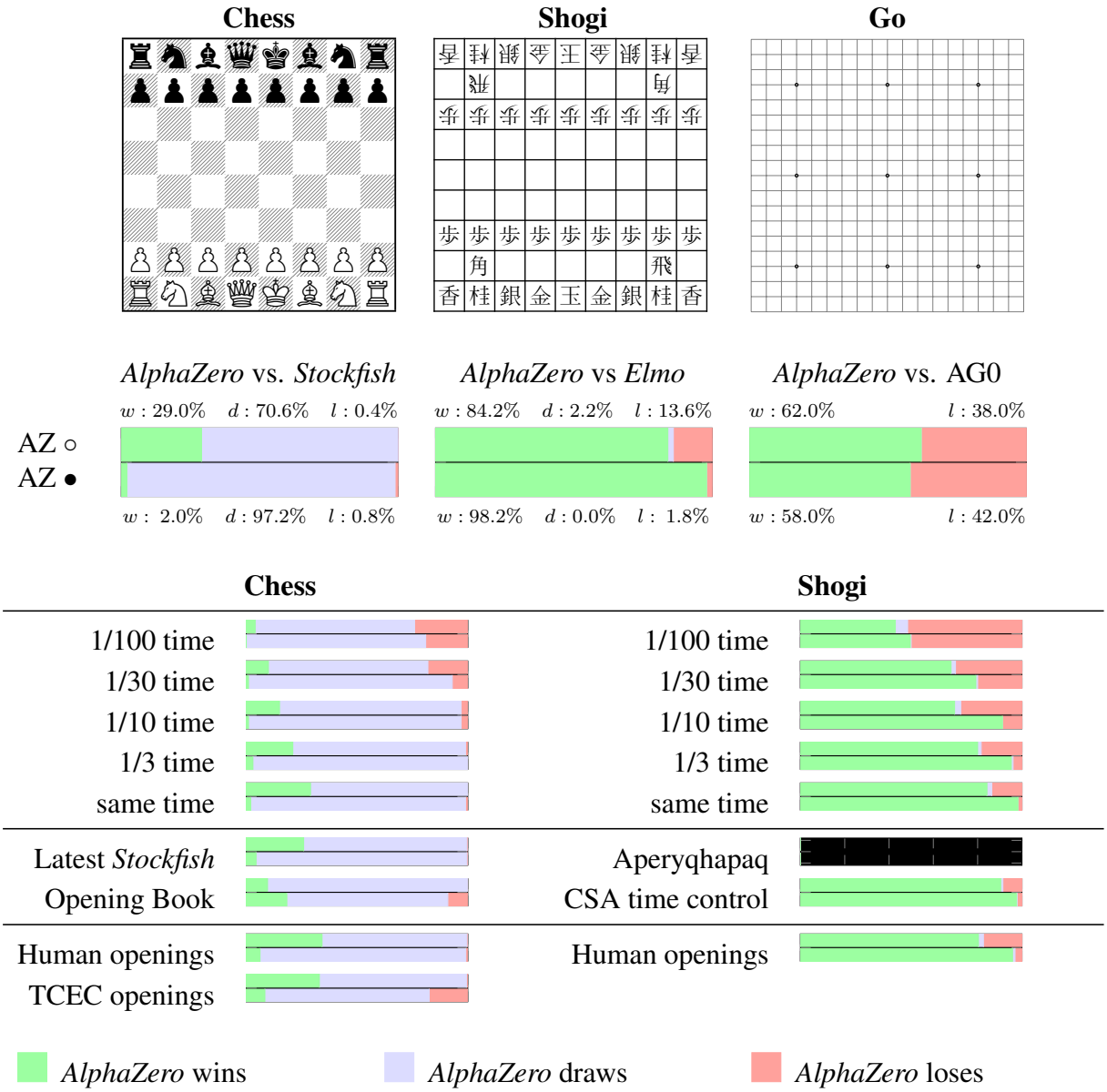

Figure 2: **a** Tournament evaluation of *AlphaZero* in chess, shogi, and Go, as games won, drawn or lost from *AlphaZero*'s perspective, in matches against respectively *Stockfish*, *Elmo*, and also against the previously published *AlphaGo Zero* after 3 days of training. In the top bar, *AlphaZero* (AZ) plays white; in the bottom bar *AlphaZero* plays black. Each bar shows the results from *AlphaZero*'s perspective: win ('w', green), draw ('d', white), loss ('l', red). **b** Scalability of *AlphaZero* with thinking time, compared to *Stockfish* and *Elmo*. *Stockfish* and *Elmo* always receive full time (3 hours + 15 seconds per move), time for *AlphaZero* is scaled down as given in the table. **c** Extra evaluations of *AlphaZero* in chess against the most recent version of *Stockfish*, and a version of *Stockfish* with a strong opening book. Extra evaluations of *AlphaZero* in shogi against *Elmo* under 2017 CSA world championship time controls (10 minutes per game + 10 seconds per move), and against another strong shogi program *Aperyqhapaq* tkhubert:Update with final results. **d** Average result of chess matches starting from different opening positions: either common human positions (see also Figure 3), or the 2016 TCEC world championship opening positions (see also Figure S3). Average result of shogi matches starting from common human positions (see also Figure 3).

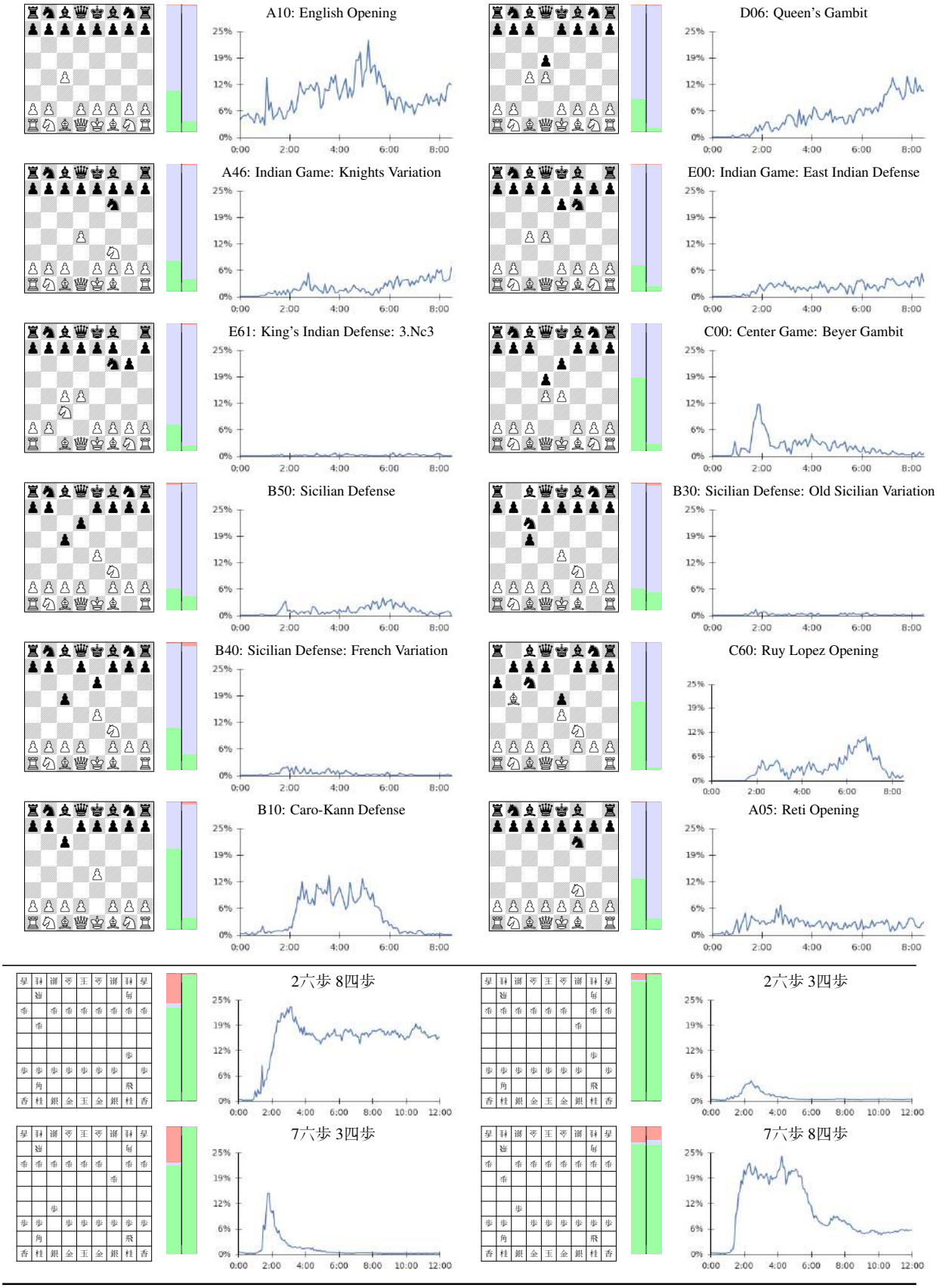

Figure 3: Chess matches starting from the 12 most popular human openings. In the left bar, *AlphaZero* plays white, starting from the given position; in the right bar *AlphaZero* plays black. Each bar shows the results from *AlphaZero*'s perspective: win (green), draw (white), loss (red).

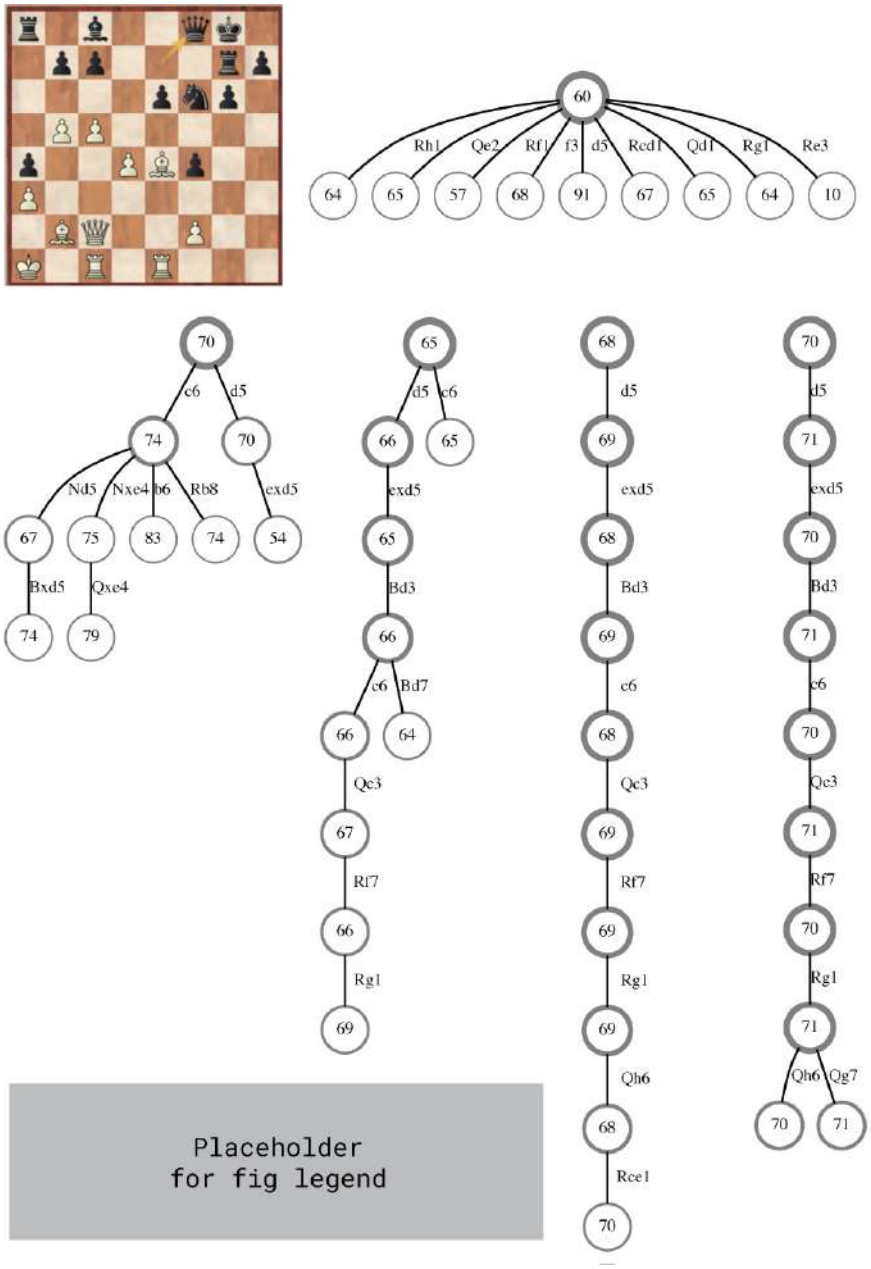

Figure 4: An illustration of *AlphaZero*'s search procedure, in a position (inset) from the match between *AlphaZero* (white) and *Stockfish* (black). The internal state of *AlphaZero*'s MCTS is summarised after $10^2, ..., 10^6$ simulations. Each summary shows the 10 most visited nodes. The estimated value is shown in each node (expressed as a winning percentage). The visit count, relative to the root node of each tree, is proportional to the thickness of the node boundary.

10. R. Coulom. Whole-history rating: A Bayesian rating system for players of time-varying strength. In *International Conference on Computers and Games*, pages 113–124, 2008.

11. Omid E David, Nathan S Netanyahu, and Lior Wolf. Deepchess: End-to-end deep neural network for automatic learning in chess. In *International Conference on Artificial Neural Networks*, pages 88–96. Springer, 2016.

12. Michael R. Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

13. Kunihito Hoki and Tomoyuki Kaneko. Large-scale optimization for evaluation functions with minimax search. *Journal of Artificial Intelligence Research (JAIR)*, 49:527–568, 2014.

14. Feng-hsiung Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, 2002.

15. Hiroyuki Iida, Makoto Sakuta, and Jeff Rollason. Computer shogi. *Artificial Intelligence*, 134:121–144, 2002.

16. Norman P. Jouppi, Cliff Young, Nishant Patil, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 1–12. ACM, 2017.

17. Tomoyuki Kaneko and Kunihito Hoki. Analysis of evaluation-function learning by comparison of sibling nodes. In *Advances in Computer Games - 13th International Conference, ACG 2011, Tilburg, The Netherlands, November 20-22, 2011, Revised Selected Papers*, pages 158–169, 2011.

18. John Knudsen. *Essential Chess Quotations*. iUniverse, 2000.

19. D. E. Knuth and R. W Moore. An analysis of alpha–beta pruning. *Artificial Intelligence*, 6(4):293—-326, 1975.

20. Matthew Lai. Giraffe: Using deep reinforcement learning to play chess. Master's thesis, Imperial College London, 2015.

21. Emanuel Lasker. *Common Sense in Chess*. Dover Publications, 1965.

22. David N. L. Levy and Monty Newborn. *How Computers Play Chess*. Ishi Press, 2009.

23. Chris J. Maddison, Aja Huang, Ilya Sutskever, and David Silver. Move evaluation in Go using deep convolutional neural networks. In *International Conference on Learning Representations*, 2015.

24. Tony Marsland. Computer chess methods. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence*. John Wiley & sons, New York, 1987.

25. Barney Pell. A strategic metagame player for general chess-like games. *Computational Intelligence*, 12:177–198, 1996.

26. Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. Understanding sampling style adversarial search methods. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2010.

27. Tord Romstad, Marco Costalba, Joona Kiiski, et al. Stockfish: A strong open source chess engine. `https://stockfishchess.org/`. Retrieved November 29th, 2017.

28. A. L. Samuel. Some studies in machine learning using the game of checkers II - recent progress. *IBM Journal of Research and Development*, 11(6):601–617, 1967.

29. Claude E Shannon. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

30. David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

31. David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

32. Wilhelm Steinitz. *The Modern Chess Instructor*. Edition Olms AG, 1990.

33. Sebastian Thrun. Learning to play the game of chess. In *Advances in neural information processing systems*, pages 1069–1076, 1995.

34. J. Veness, D. Silver, A. Blair, and W. Uther. Bootstrapping from game tree search. In *Advances in Neural Information Processing Systems*, pages 1937–1945, 2009.

Methods

Anatomy of a Computer Chess Program

In this section we describe the components of a typical computer chess program, focusing specifically on *Stockfish* (*27*), an open source program that won the 2016 TCEC computer chess world championship. For an overview of standard methods, see (*24*).

Each position s is described by a sparse vector of handcrafted features $\phi(s)$, including midgame/endgame-specific material point values, material imbalance tables, piece-square tables, mobility and trapped pieces, pawn structure, king safety, outposts, bishop pair, and other miscellaneous evaluation patterns. Each feature ϕ_i is assigned, by a combination of manual and automatic tuning, a corresponding weight w_i and the position is evaluated by a linear combination $v(s, w) = \phi(s)^\top w$. However, this raw evaluation is only considered accurate for positions that are "quiet", with no unresolved captures or checks. A domain-specialised *quiescence search* is used to resolve ongoing tactical situations before the evaluation function is applied.

The final evaluation of a position s is computed by a minimax search that evaluates each leaf using a quiescence search. Alpha-beta pruning is used to safely cut any branch that is provably dominated by another variation. Additional cuts are achieved using aspiration windows and principal variation search. Other pruning strategies include null move pruning (which assumes a pass move should be worse than any variation, in positions that are unlikely to be in *zugzwang*, as determined by simple heuristics), futility pruning (which assumes knowledge of the maximum possible change in evaluation), and other domain-dependent pruning rules (which assume knowledge of the value of captured pieces).

The search is focused on promising variations both by extending the search depth of promising variations, and by reducing the search depth of unpromising variations based on heuristics like history, static-exchange evaluation (SEE), and moving piece type. Extensions are based on domain-independent rules that identify singular moves with no sensible alternative, and domain-dependent rules, such as extending check moves. Reductions, such as late move reductions, are based heavily on domain knowledge.

The efficiency of alpha-beta search depends critically upon the order in which moves are considered. Moves are therefore ordered by iterative deepening (using a shallower search to order moves for a deeper search). In addition, a combination of domain-independent move ordering heuristics, such as killer heuristic, history heuristic, counter-move heuristic, and also domain-dependent knowledge based on captures (SEE) and potential captures (MVV/LVA).

A transposition table facilitates the reuse of values and move orders when the same position is reached by multiple paths. In some variants, a carefully tuned opening book may be used to select moves at the start of the game. An endgame tablebase, precalculated by exhaustive retrograde analysis of endgame positions, provides the optimal move in all positions with six and sometimes seven pieces or less.

Other strong chess programs, and also earlier programs such as Deep Blue, have used very similar architectures (*9, 24*) including the majority of the components described above, although

important details vary considerably.

None of the techniques described in this section are used by *AlphaZero*. It is likely that some of these techniques could further improve the performance of *AlphaZero*; however, we have focused on a pure self-play reinforcement learning approach and leave these extensions for future research.

Prior Work on Computer Chess and Shogi

In this section we discuss some notable prior work on reinforcement learning in computer chess.

NeuroChess (*33*) evaluated positions by a neural network that used 175 handcrafted input features. It was trained by temporal-difference learning to predict the final game outcome, and also the expected features after two moves. *NeuroChess* won 13% of games against *GnuChess* using a fixed depth 2 search.

Beal and Smith applied temporal-difference learning to estimate the piece values in chess (*7*) and shogi (*8*), starting from random values and learning solely by self-play.

KnightCap (*6*) evaluated positions by a neural network that used an attack-table based on knowledge of which squares are attacked or defended by which pieces. It was trained by a variant of temporal-difference learning, known as TD(leaf), that updates the leaf value of the principal variation of an alpha-beta search. *KnightCap* achieved human master level after training against a strong computer opponent with hand-initialised piece-value weights.

Meep (*34*) evaluated positions by a linear evaluation function based on handcrafted features. It was trained by another variant of temporal-difference learning, known as TreeStrap, that updated all nodes of an alpha-beta search. *Meep* defeated human international master players in 13 out of 15 games, after training by self-play with randomly initialised weights.

Kaneko and Hoki (*17*) trained the weights of a shogi evaluation function comprising a million features, by learning to select expert human moves during alpha-beta serach. They also performed a large-scale optimization based on minimax search regulated by expert game logs (*13*); this formed part of the *Bonanza* engine that won the 2013 World Computer Shogi Championship.

Giraffe (*20*) evaluated positions by a neural network that included mobility maps and attack and defend maps describing the lowest valued attacker and defender of each square. It was trained by self-play using TD(leaf), also reaching a standard of play comparable to international masters.

DeepChess (*11*) trained a neural network to performed pair-wise evaluations of positions. It was trained by supervised learning from a database of human expert games that was pre-filtered to avoid capture moves and drawn games. *DeepChess* reached a strong grandmaster level of play.

All of these programs combined their learned evaluation functions with an alpha-beta search enhanced by a variety of extensions.

An approach based on training dual policy and value networks using *AlphaZero*-like policy iteration was successfully applied to improve on the state-of-the-art in Hex (*3*).

MCTS and Alpha-Beta Search

For at least four decades the strongest computer chess programs have used alpha-beta search (*19, 24*). *AlphaZero* uses a markedly different approach that averages over the position evaluations within a subtree, rather than computing the minimax evaluation of that subtree. However, chess programs using traditional MCTS were much weaker than alpha-beta search programs, (*4, 26*); while alpha-beta programs based on neural networks have previously been unable to compete with faster, handcrafted evaluation functions.

AlphaZero evaluates positions using non-linear function approximation based on a deep neural network, rather than the linear function approximation used in typical chess programs. This provides a much more powerful representation, but may also introduce spurious approximation errors. MCTS averages over these approximation errors, which therefore tend to cancel out when evaluating a large subtree. In contrast, alpha-beta search computes an explicit minimax, which propagates the biggest approximation errors to the root of the subtree. Using MCTS may allow *AlphaZero* to effectively combine its neural network representations with a powerful, domain-independent search.

Domain Knowledge

1. The input features describing the position, and the output features describing the move, are structured as a set of planes; i.e. the neural network architecture is matched to the grid-structure of the board.

2. *AlphaZero* is provided with perfect knowledge of the game rules. These are used during MCTS, to simulate the positions resulting from a sequence of moves, to determine game termination, and to score any simulations that reach a terminal state.

3. Knowledge of the rules is also used to encode the input planes (i.e. castling, repetition, no-progress) and output planes (how pieces move, promotions, and piece drops in shogi).

4. The typical number of legal moves is used to scale the exploration noise (see below).

5. Chess and shogi games exceeding a maximum number of steps (determined by typical game length) were terminated and assigned a drawn outcome; Go games were terminated and scored with Tromp-Taylor rules, similarly to previous work (*31*).

AlphaZero did not use any form of domain knowledge beyond the points listed above. Specifically, *AlphaZero* did not use an opening book, endgame tablebases, or domain-specific heuristics.

Representation

In this section we describe the representation of the board inputs, and the representation of the action outputs, used by the neural network in *AlphaZero*. Other representations could have been used; in our experiments the training algorithm worked robustly for many reasonable choices.

The input to the neural network is an $N \times N \times (MT + L)$ image stack that represents state using a concatenation of T sets of M planes of size $N \times N$. Each set of planes represents the board position at a time-step $t - T + 1, ..., t$, and is set to zero for time-steps less than 1. The board is oriented to the perspective of the current player. The M feature planes are composed of binary feature planes indicating the presence of the player's pieces, with one plane for each piece type, and a second set of planes indicating the presence of the opponent's pieces. For shogi there are additional planes indicating the number of captured prisoners of each type. There are an additional L constant-valued input planes denoting the player's colour, the total move count, and the state of special rules: the legality of castling in chess (kingside or queenside); the repetition count for that position (3 repetitions is an automatic draw in chess; 4 in shogi); and the number of moves without progress in chess (50 moves without progress is an automatic draw). Input features are summarised in Table S1.

A move in chess may be described in two parts: selecting the piece to move, and then selecting among the legal moves for that piece. We represent the policy $\pi(a|s)$ by a $8 \times 8 \times 73$ stack of planes encoding a probability distribution over 4,672 possible moves. Each of the 8×8 positions identifies the square from which to "pick up" a piece. The first 56 planes encode possible 'queen moves' for any piece: a number of squares $[1..7]$ in which the piece will be moved, along one of eight relative compass directions $\{N, NE, E, SE, S, SW, W, NW\}$. The next 8 planes encode possible knight moves for that piece. The final 9 planes encode possible underpromotions for pawn moves or captures in two possible diagonals, to knight, bishop or rook respectively. Other pawn moves or captures from the seventh rank are promoted to a queen.

The policy in shogi is represented by a $9 \times 9 \times 139$ stack of planes similarly encoding a probability distribution over 11,259 possible moves. The first 64 planes encode 'queen moves' and the next 2 moves encode knight moves. An additional $64 + 2$ planes encode promoting queen moves and promoting knight moves respectively. The last 7 planes encode a captured piece dropped back into the board at that location.

The policy in Go is represented identically to *AlphaGo Zero* (*31*), using a flat distribution over $19 \times 19 + 1$ moves representing possible stone placements and the pass move. We also tried using a flat distribution over moves for chess and shogi; the final result was almost identical although training was slightly slower.

The action representations are summarised in Table S2. Illegal moves are masked out by setting their probabilities to zero, and re-normalising the probabilities for remaining moves.

Architecture

Apart from the representation of positions and actions described above, *AlphaZero* uses the same network architecture as *AlphaGo Zero* (*31*), briefly recapitulated here.

The neural network consists of a "body" followed by both policy and value "heads". The body consists of a rectified batch-normalised convolutional layer followed by 19 residual blocks (*?*). Each such block consists of two rectified batch-normalised convolutional layers with a skip connection. Each convolution applies 256 filters of kernel size 3×3 with stride 1. The policy head applies an additional rectified, batch-normalised convolutional layer, followed by a final convolution of 73 filters for chess or 139 filters for shogi, or a linear layer of size 362 for Go, representing the logits of the respective policies described above. The value head applies an additional rectified, batch-normalised convolution of 1 filter of kernel size 1×1 with stride 1, followed by a rectified linear layer of size 256 and a $\tanh$-linear layer of size 1.

Configuration

During training, each MCTS used 800 simulations. The number of games, positions, and thinking time varied per game due largely to different board sizes and game lengths, and are shown in Table S3. The learning rate was set to 0.2 for each game, and was dropped three times (after 100, 300 and 500 thousands of steps to 0.02, 0.002 and 0.0002 respectively) during the course of training. Moves are selected in proportion to the root visit count. Dirichlet noise $\mathrm{Dir}(\alpha)$ was added to the prior probabilities in the root node; this was scaled in inverse proportion to the approximate number of legal moves in a typical position, to a value of $\alpha = \{0.3, 0.15, 0.03\}$ for chess, shogi and Go respectively. Unless otherwise specified, the training and search algorithm and parameters are identical to *AlphaGo Zero* (*31*).

During evaluation, *AlphaZero* selects moves greedily with respect to the root visit count. Each MCTS was executed on a single machine with 4 TPUs.

Opponents

To evaluate performance in chess, we used *Stockfish* version 8 (official Linux release) as a baseline program. *Stockfish* was configured according to its 2016 TCEC world championship superfinal settings: 44 threads on 44 cores, a hash size of 32GB, syzygy endgame tablebases, at 3 hour time controls with 15 additional seconds per move. We also evaluated against the most recent version, *Stockfish* 9 (just released at time of writing), using the same configuration.

Stockfish does not have an opening book of its own and the main evaluations were performed without an opening book. We performed an additional evaluation in which opening moves were selected by the *Brainfish* program, using an opening book derived from *Stockfish*. However, we note that these matches were low in diversity, and *AlphaZero* and *Stockfish* tended to produce very similar games throughout the match, more than 90% of which were draws. When we forced *AlphaZero* to play with greater diversity (by softmax sampling for the first 30 moves) the

winning rate increased from 5.8% to 14%.

To evaluate performance in shogi, we used *Elmo* version WCSC27 in combination with YaneuraOu 2017 Early KPPT 4.79 64AVX2 TOURNAMENT as a baseline program, using 44 CPU threads and a hash size of 32GB with the usi options of EnteringKingRule set to CSARule27, MinimumThinkingTime set to 1000, BookFile set to standard_book.db, BookDepthLimit set to 0 and BookMoves set to 200. Additionally, we also evaluated against *Aperyqhapaq* combined with the same YaneuraOu version and BookFile まふ定跡横歩取角(改) . For *Aperyqhapaq*, we used the same usi options as for *Elmo* except for some book settings; namely we set BookMoves to 120 and ConsiderBookMoveCount to True.

Evaluation

We evaluated the relative strength of *AlphaZero* (Figure 1) by measuring the Elo rating of each player. We estimate the probability that player a will defeat player b by a logistic function $p(a \text{ defeats } b) = \frac{1}{1+\exp(c_{\text{elo}}(e(b)-e(a))}$, and estimate the ratings $e(\cdot)$ by Bayesian logistic regression, computed by the *BayesElo* program (*10*) using the standard constant $c_{\text{elo}} = 1/400$.

Elo ratings were computed from the results of a 1 second per move tournament between iterations of *AlphaZero* during training, and also a baseline player: either *Stockfish*, *Elmo* or *AlphaGo Lee* respectively. The Elo rating of the baseline players was anchored to publicly available values (*31*).

We also measured the head-to-head performance of *AlphaZero* against each baseline player. Matches consisted of 1000 games, except for the human openings (200 games as black and 200 games as white from each of 12 openings) and the 2016 TCEC openings (10 games as black and 10 games as white from each of the 50 openings). The 12 most frequently played human openings were chosen for this match; each opening was played more than 100,000 times in an online database (*1*).

Unless otherwise specified, all matches for chess, shogi and Go were set to 3 hours of main thinking time, plus 15 additional seconds of thinking time for each move. *AlphaZero* used a simple time control strategy: thinking for $1/20$th of the remaining time. Opponent programs were free to use their own time control strategy. Pondering was disabled for all players (particularly important for the asymmetric time controls in Figure **??**a).

Resignation was enabled for all players (-650 centipawns for 4 consecutive moves for *Stockfish*, -4500 centipawns for 10 consecutives moves for *Elmo*, 5% winrate for *AlphaZero*).

Example games

The Supplementary Data includes 110 games from the match between *AlphaZero* and *Stockfish*, starting from the initial board position, and 100 games from the match starting from 2016 TCEC world championship opening positions. For the match from the initial board position, one game was selected at random for each unique opening sequence of 15 moves; all *AlphaZero* losses

were also included. For the TCEC match, one game as white and one game as black were selected at random from the match starting from each opening position.

10 games were selected from each batch by GM Matthew Sadler, according to their interest to the chess community; these games are included in Supplementary Table S5.

Go		Chess		Shogi	
Feature	Planes	Feature	Planes	Feature	Planes
P1 stone	1	P1 piece	6	P1 piece	14
P2 stone	1	P2 piece	6	P2 piece	14
		Repetitions	2	Repetitions	3
				P1 prisoner count	7
				P2 prisoner count	7
Colour	1	Colour	1	Colour	1
		Total move count	1	Total move count	1
		P1 castling	2		
		P2 castling	2		
		No-progress count	1		
Total	17	Total	119	Total	362

Table S1: Input features used by *AlphaZero* in Go, Chess and Shogi respectively. The first set of features are repeated for each position in a $T = 8$-step history. Counts are represented by a single real-valued input; other input features are represented by a one-hot encoding using the specified number of binary input planes. The current player is denoted by P1 and the opponent by P2.

Table S5: Games played by *AlphaZero* against *Stockfish*, selected by GM Matthew Sadler. The first 10 games are from the initial board position, the remaining 10 games are from 2016 TCEC world championship start positions.

Chess		Shogi	
Feature	Planes	Feature	Planes
Queen moves	56	Queen moves	64
Knight moves	8	Knight moves	2
Underpromotions	9	Promoting queen moves	64
		Promoting knight moves	2
		Drop	7
Total	73	Total	139

Table S2: Action representation used by *AlphaZero* in Chess and Shogi respectively. The policy is represented by a stack of planes encoding a probability distribution over legal moves; planes correspond to the entries in the table.

	Chess	Shogi	Go
Mini-batches	700k	700k	700k
Training Time	9h	12h	34h
Training Games	44 million	24 million	21 million
Thinking Time	800 sims 40 ms	800 sims 80 ms	800 sims 200 ms

Table S3: Selected statistics of *AlphaZero* training in Chess, Shogi and Go.

Program	Chess	Shogi	Go
AlphaZero	80k	40k	16k
Stockfish	70,000k		
Elmo		35,000k	

Table S4: Evaluation speed (positions/second) of *AlphaZero*, *Stockfish*, and *Elmo* in chess, shogi and Go.

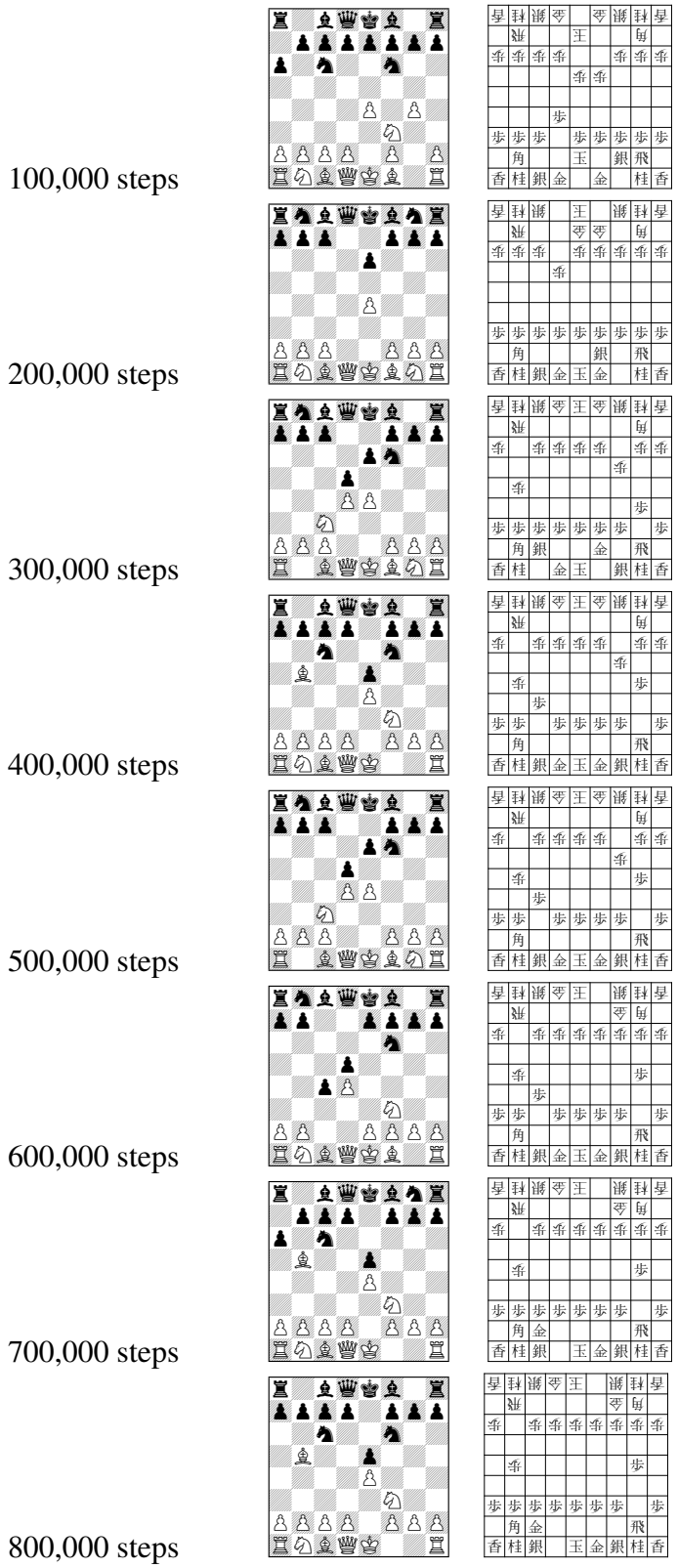

Figure S1: Chess and shogi openings preferred by AlphaZero at different stages of self-play training. The figure shows the most frequently selected 3-move opening played by *AlphaZero* during its games of self-play. Each move was generated by an MCTS with just 800 simulations per move.

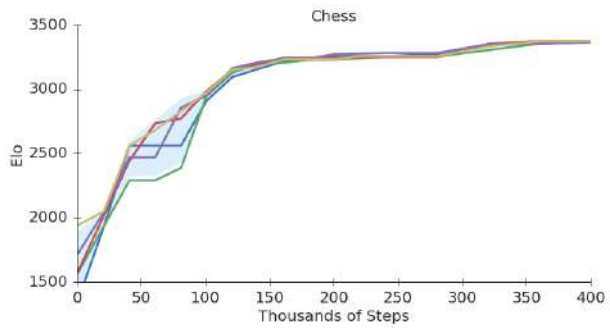

Figure S2: Repeatability of *AlphaZero* training on the game of chess. The figure shows 5 separate training runs of 400,000 steps (approximately 4 hours each). Elo ratings were computed from evaluation games between different players when given 800 simulations per move. Shaded region indicates 95% confidence interval.

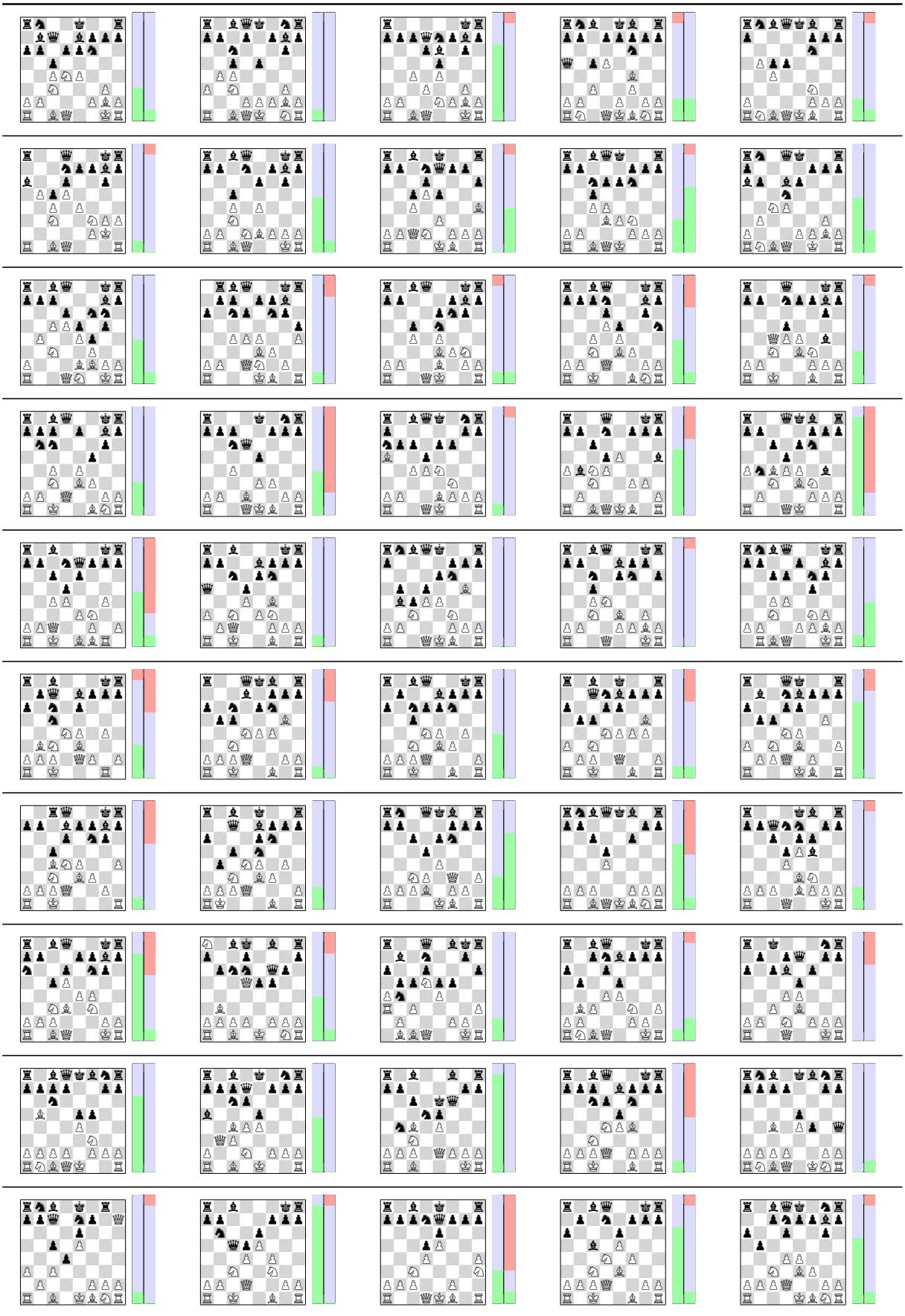

Figure S3: Chess matches beginning from the 2016 TCEC world championship start positions. In the top bar, *AlphaZero* plays white, starting from the given position; in the bottom bar *AlphaZero* plays black. Each bar shows the results from *AlphaZero*'s perspective: win (green), draw (white), loss (red). Many of these start positions are unbalanced according to both and *AlphaZero* and *Stockfish*, resulting in more losses for both players.