ARTICLE

doi:10.1038/nature24270

Mastering the game of Go without human knowledge

David Silver[1]*, Julian Schrittwieser[1]*, Karen Simonyan[1]*, Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

Much progress towards artificial intelligence has been made using supervised learning systems that are trained to replicate the decisions of human experts[1–4]. However, expert data sets are often expensive, unreliable or simply unavailable. Even when reliable data sets are available, they may impose a ceiling on the performance of systems trained in this manner[5]. By contrast, reinforcement learning systems are trained from their own experience, in principle allowing them to exceed human capabilities, and to operate in domains where human expertise is lacking. Recently, there has been rapid progress towards this goal, using deep neural networks trained by reinforcement learning. These systems have outperformed humans in computer games, such as Atari[6,7] and 3D virtual environments[8–10]. However, the most challenging domains in terms of human intellect—such as the game of Go, widely viewed as a grand challenge for artificial intelligence[11]—require a precise and sophisticated lookahead in vast search spaces. Fully general methods have not previously achieved human-level performance in these domains.

AlphaGo was the first program to achieve superhuman performance in Go. The published version[12], which we refer to as AlphaGo Fan, defeated the European champion Fan Hui in October 2015. AlphaGo Fan used two deep neural networks: a policy network that outputs move probabilities and a value network that outputs a position evaluation. The policy network was trained initially by supervised learning to accurately predict human expert moves, and was subsequently refined by policy-gradient reinforcement learning. The value network was trained to predict the winner of games played by the policy network against itself. Once trained, these networks were combined with a Monte Carlo tree search (MCTS)[13–15] to provide a lookahead search, using the policy network to narrow down the search to high-probability moves, and using the value network (in conjunction with Monte Carlo rollouts using a fast rollout policy) to evaluate positions in the tree. A subsequent version, which we refer to as AlphaGo Lee, used a similar approach (see Methods), and defeated Lee Sedol, the winner of 18 international titles, in March 2016.

Our program, AlphaGo Zero, differs from AlphaGo Fan and AlphaGo Lee[12] in several important aspects. First and foremost, it is trained solely by self-play reinforcement learning, starting from random play, without any supervision or use of human data. Second, it uses only the black and white stones from the board as input features. Third, it uses a single neural network, rather than separate policy and value networks. Finally, it uses a simpler tree search that relies upon this single neural network to evaluate positions and sample moves, without performing any Monte Carlo rollouts. To achieve these results, we introduce a new reinforcement learning algorithm that incorporates lookahead search inside the training loop, resulting in rapid improvement and precise and stable learning. Further technical differences in the search algorithm, training procedure and network architecture are described in Methods.

Reinforcement learning in AlphaGo Zero

Our new method uses a deep neural network f_θ with parameters θ. This neural network takes as an input the raw board representation s of the position and its history, and outputs both move probabilities and a value, $(\boldsymbol{p}, v) = f_\theta(s)$. The vector of move probabilities $\boldsymbol{p}$ represents the probability of selecting each move a (including pass), $p_a = \Pr(a|s)$. The value v is a scalar evaluation, estimating the probability of the current player winning from position s. This neural network combines the roles of both policy network and value network[12] into a single architecture. The neural network consists of many residual blocks[4] of convolutional layers[16,17] with batch normalization[18] and rectifier nonlinearities[19] (see Methods).

The neural network in AlphaGo Zero is trained from games of self-play by a novel reinforcement learning algorithm. In each position s, an MCTS search is executed, guided by the neural network f_θ. The MCTS search outputs probabilities $\boldsymbol{\pi}$ of playing each move. These search probabilities usually select much stronger moves than the raw move probabilities $\boldsymbol{p}$ of the neural network $f_\theta(s)$; MCTS may therefore be viewed as a powerful policy improvement operator[20,21]. Self-play with search—using the improved MCTS-based policy to select each move, then using the game winner z as a sample of the value—may be viewed as a powerful policy evaluation operator. The main idea of our reinforcement learning algorithm is to use these search operators

[1]DeepMind, 5 New Street Square, London EC4A 3TW, UK.
*These authors contributed equally to this work.

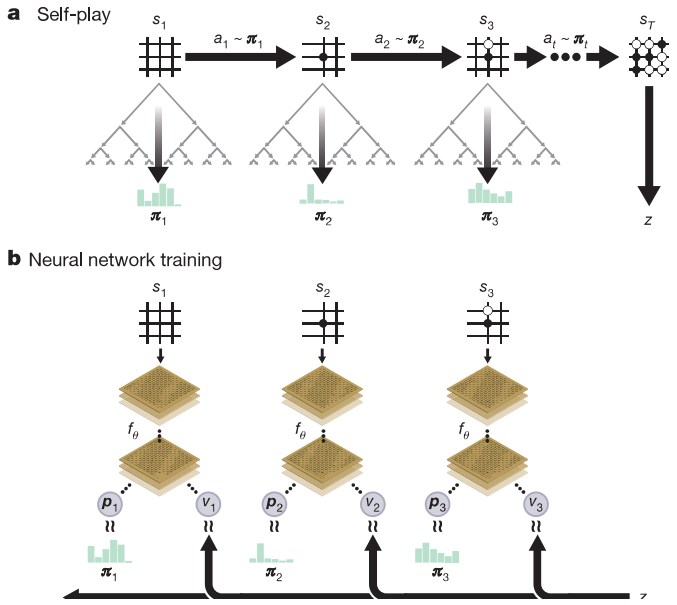

Figure 1 | Self-play reinforcement learning in AlphaGo Zero. a, The program plays a game $s_1, ..., s_T$ against itself. In each position s_t, an MCTS α_θ is executed (see Fig. 2) using the latest neural network f_θ. Moves are selected according to the search probabilities computed by the MCTS, $a_t \sim \pi_t$. The terminal position s_T is scored according to the rules of the game to compute the game winner z. **b**, Neural network training in AlphaGo Zero. The neural network takes the raw board position s_t as its input, passes it through many convolutional layers with parameters θ, and outputs both a vector $\boldsymbol{p}_t$, representing a probability distribution over moves, and a scalar value v_t, representing the probability of the current player winning in position s_t. The neural network parameters θ are updated to maximize the similarity of the policy vector $\boldsymbol{p}_t$ to the search probabilities π_t, and to minimize the error between the predicted winner v_t and the game winner z (see equation (1)). The new parameters are used in the next iteration of self-play as in **a**.

repeatedly in a policy iteration procedure[22,23]: the neural network's parameters are updated to make the move probabilities and value $(\boldsymbol{p}, v) = f_\theta(s)$ more closely match the improved search probabilities and self-play winner (π, z); these new parameters are used in the next iteration of self-play to make the search even stronger. Figure 1 illustrates the self-play training pipeline.

The MCTS uses the neural network f_θ to guide its simulations (see Fig. 2). Each edge (s, a) in the search tree stores a prior probability $P(s, a)$, a visit count $N(s, a)$, and an action value $Q(s, a)$. Each simulation starts from the root state and iteratively selects moves that maximize an upper confidence bound $Q(s, a) + U(s, a)$, where $U(s, a) \propto P(s, a) / (1 + N(s, a))$ (refs 12, 24), until a leaf node s' is encountered. This leaf position is expanded and evaluated only once by the network to generate both prior probabilities and evaluation, $(P(s', \cdot), V(s')) = f_\theta(s')$. Each edge (s, a) traversed in the simulation is updated to increment its visit count $N(s, a)$, and to update its action value to the mean evaluation over these simulations, $Q(s, a) = 1/N(s, a) \sum_{s'|s,a \to s'} V(s')$ where $s, a \to s'$ indicates that a simulation eventually reached s' after taking move a from position s.

MCTS may be viewed as a self-play algorithm that, given neural network parameters θ and a root position s, computes a vector of search probabilities recommending moves to play, $\pi = \alpha_\theta(s)$, proportional to the exponentiated visit count for each move, $\pi_a \propto N(s, a)^{1/\tau}$, where τ is a temperature parameter.

The neural network is trained by a self-play reinforcement learning algorithm that uses MCTS to play each move. First, the neural network is initialized to random weights θ_0. At each subsequent iteration $i \geq 1$, games of self-play are generated (Fig. 1a). At each time-step t, an MCTS search $\pi_t = \alpha_{\theta_{i-1}}(s_t)$ is executed using the previous iteration of neural network $f_{\theta_{i-1}}$ and a move is played by sampling the search probabilities π_t. A game terminates at step T when both players pass, when the search value drops below a resignation threshold or when the game exceeds a maximum length; the game is then scored to give a final reward of $r_T \in \{-1, +1\}$ (see Methods for details). The data for each time-step t is stored as (s_t, π_t, z_t), where $z_t = \pm r_T$ is the game winner from the perspective of the current player at step t. In parallel (Fig. 1b), new network parameters θ_i are trained from data (s, π, z) sampled uniformly among all time-steps of the last iteration(s) of self-play. The neural network $(\boldsymbol{p}, v) = f_{\theta_i}(s)$ is adjusted to minimize the error between the predicted value v and the self-play winner z, and to maximize the similarity of the neural network move probabilities $\boldsymbol{p}$ to the search probabilities π. Specifically, the parameters θ are adjusted by gradient descent on a loss function l that sums over the mean-squared error and cross-entropy losses, respectively:

$$(\boldsymbol{p}, v) = f_\theta(s) \quad \text{and} \quad l = (z - v)^2 - \boldsymbol{\pi}^{\mathrm{T}} \log \boldsymbol{p} + c\|\theta\|^2 \qquad (1)$$

where c is a parameter controlling the level of L2 weight regularization (to prevent overfitting).

Empirical analysis of AlphaGo Zero training

We applied our reinforcement learning pipeline to train our program AlphaGo Zero. Training started from completely random behaviour and continued without human intervention for approximately three days.

Over the course of training, 4.9 million games of self-play were generated, using 1,600 simulations for each MCTS, which corresponds to approximately 0.4 s thinking time per move. Parameters were updated

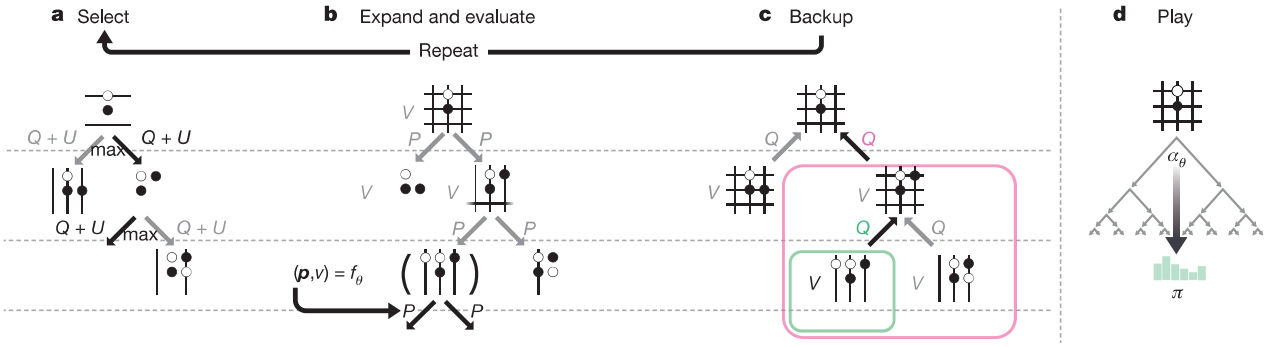

Figure 2 | MCTS in AlphaGo Zero. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus an upper confidence bound U that depends on a stored prior probability P and visit count N for that edge (which is incremented once traversed). **b**, The leaf node is expanded and the associated position s is evaluated by the neural network $(P(s, \cdot), V(s)) = f_\theta(s)$; the vector of P values are stored in the outgoing edges from s. **c**, Action value Q is updated to track the mean of all evaluations V in the subtree below that action. **d**, Once the search is complete, search probabilities π are returned, proportional to $N^{1/\tau}$, where N is the visit count of each move from the root state and τ is a parameter controlling temperature.

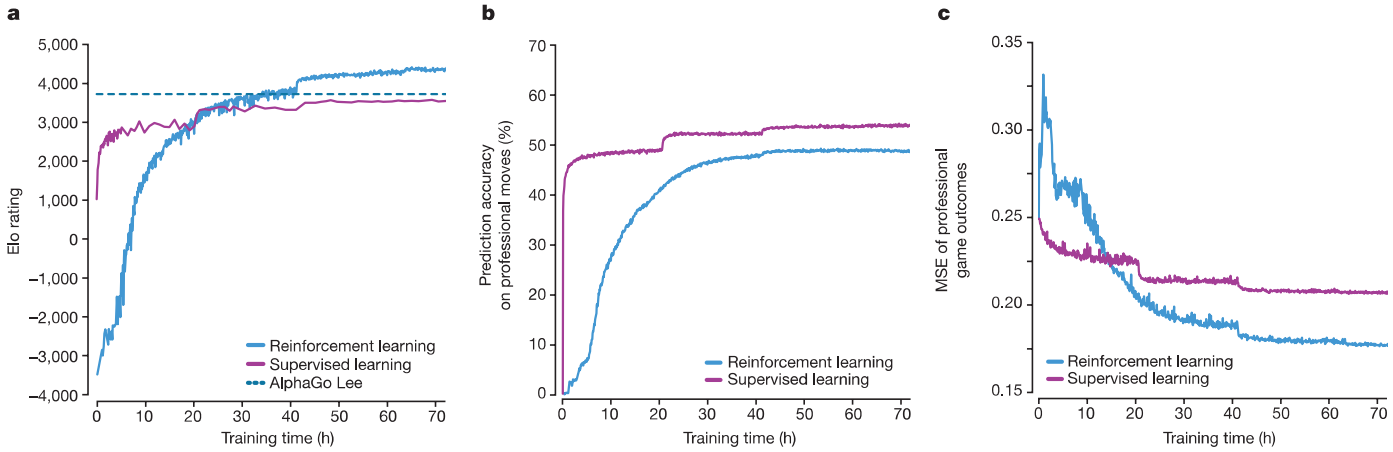

Figure 3 | Empirical evaluation of AlphaGo Zero. a, Performance of self-play reinforcement learning. The plot shows the performance of each MCTS player α_{θ_i} from each iteration i of reinforcement learning in AlphaGo Zero. Elo ratings were computed from evaluation games between different players, using 0.4 s of thinking time per move (see Methods). For comparison, a similar player trained by supervised learning from human data, using the KGS dataset, is also shown. **b**, Prediction accuracy on human professional moves. The plot shows the accuracy of the neural network f_{θ_i}, at each iteration of self-play i, in predicting human professional moves from the GoKifu dataset. The accuracy measures the percentage of positions in which the neural network assigns the highest probability to the human move. The accuracy of a neural network trained by supervised learning is also shown. **c**, Mean-squared error (MSE) of human professional game outcomes. The plot shows the MSE of the neural network f_{θ_i}, at each iteration of self-play i, in predicting the outcome of human professional games from the GoKifu dataset. The MSE is between the actual outcome $z \in \{-1, +1\}$ and the neural network value v, scaled by a factor of $\frac{1}{4}$ to the range of 0–1. The MSE of a neural network trained by supervised learning is also shown.

from 700,000 mini-batches of 2,048 positions. The neural network contained 20 residual blocks (see Methods for further details).

Figure 3a shows the performance of AlphaGo Zero during self-play reinforcement learning, as a function of training time, on an Elo scale[25]. Learning progressed smoothly throughout training, and did not suffer from the oscillations or catastrophic forgetting that have been suggested in previous literature[26–28]. Surprisingly, AlphaGo Zero outperformed AlphaGo Lee after just 36 h. In comparison, AlphaGo Lee was trained over several months. After 72 h, we evaluated AlphaGo Zero against the exact version of AlphaGo Lee that defeated Lee Sedol, under the same 2 h time controls and match conditions that were used in the man–machine match in Seoul (see Methods). AlphaGo Zero used a single machine with 4 tensor processing units (TPUs)[29], whereas AlphaGo Lee was distributed over many machines and used 48 TPUs. AlphaGo Zero defeated AlphaGo Lee by 100 games to 0 (see Extended Data Fig. 1 and Supplementary Information).

To assess the merits of self-play reinforcement learning, compared to learning from human data, we trained a second neural network (using the same architecture) to predict expert moves in the KGS Server dataset; this achieved state-of-the-art prediction accuracy compared to previous work[12,30–33] (see Extended Data Tables 1 and 2 for current and previous results, respectively). Supervised learning achieved a better initial performance, and was better at predicting human professional moves (Fig. 3). Notably, although supervised learning achieved higher move prediction accuracy, the self-learned player performed much better overall, defeating the human-trained player within the first 24 h of training. This suggests that AlphaGo Zero may be learning a strategy that is qualitatively different to human play.

To separate the contributions of architecture and algorithm, we compared the performance of the neural network architecture in AlphaGo Zero with the previous neural network architecture used in AlphaGo Lee (see Fig. 4). Four neural networks were created, using

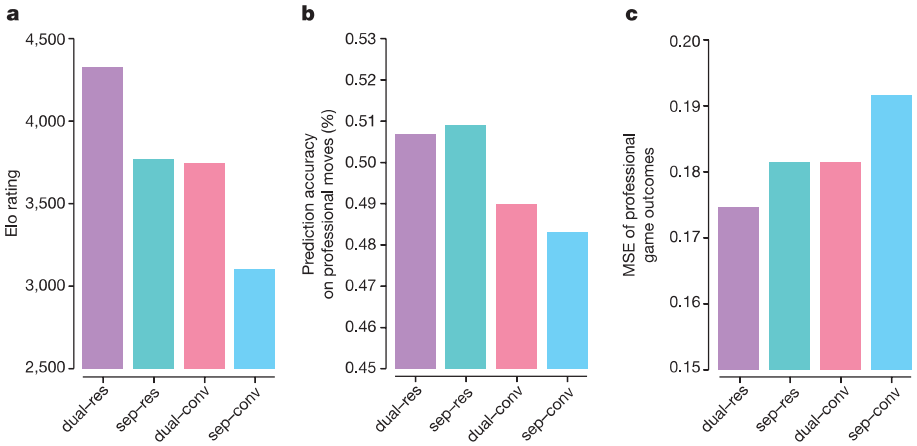

Figure 4 | Comparison of neural network architectures in AlphaGo Zero and AlphaGo Lee. Comparison of neural network architectures using either separate (sep) or combined policy and value (dual) networks, and using either convolutional (conv) or residual (res) networks. The combinations 'dual–res' and 'sep–conv' correspond to the neural network architectures used in AlphaGo Zero and AlphaGo Lee, respectively. Each network was trained on a fixed dataset generated by a previous run of AlphaGo Zero. **a**, Each trained network was combined with AlphaGo Zero's search to obtain a different player. Elo ratings were computed from evaluation games between these different players, using 5 s of thinking time per move. **b**, Prediction accuracy on human professional moves (from the GoKifu dataset) for each network architecture. **c** MSE of human professional game outcomes (from the GoKifu dataset) for each network architecture.

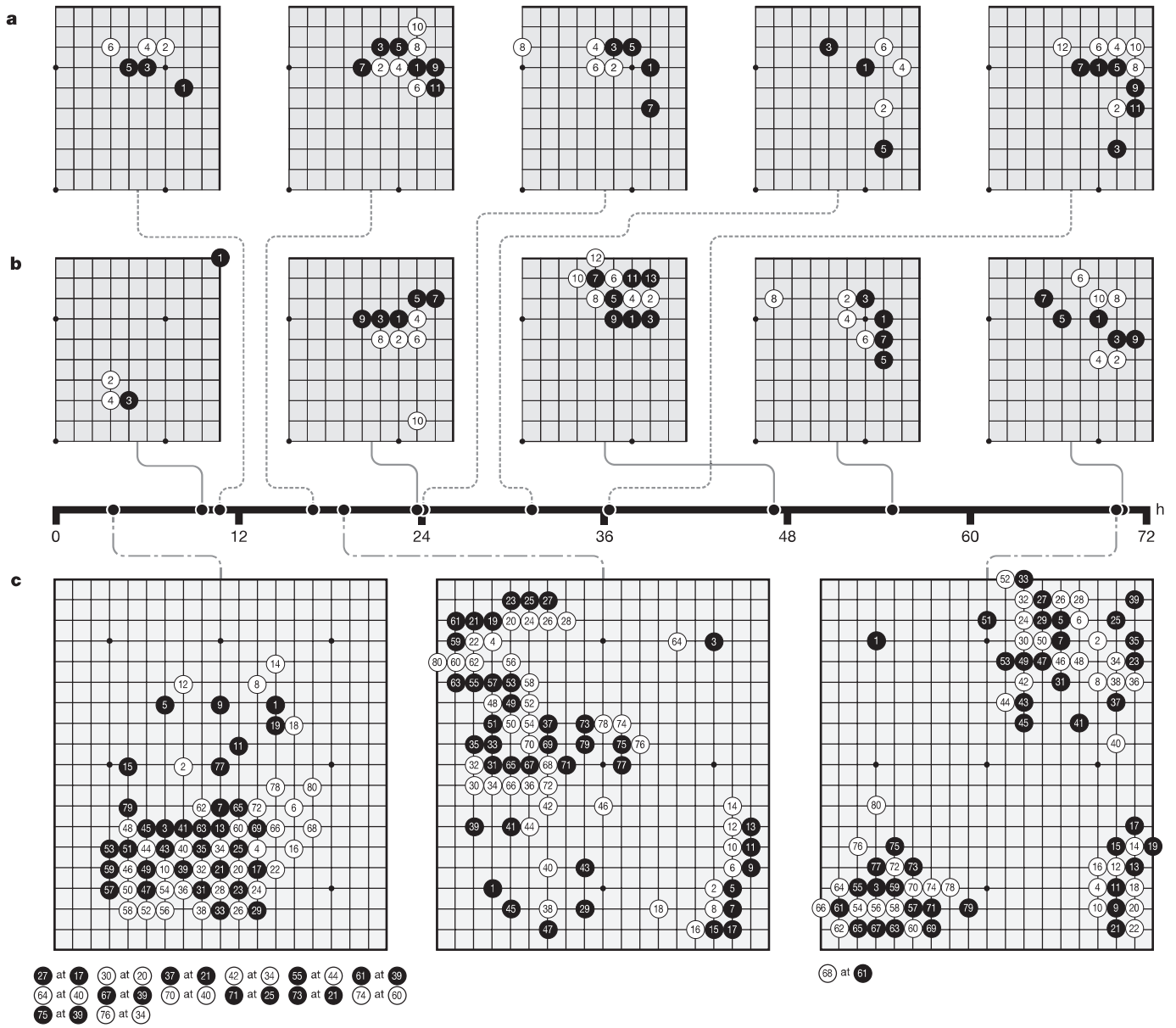

Figure 5 | Go knowledge learned by AlphaGo Zero. a, Five human *joseki* (common corner sequences) discovered during AlphaGo Zero training. The associated timestamps indicate the first time each sequence occurred (taking account of rotation and reflection) during self-play training. Extended Data Figure 2 provides the frequency of occurence over training for each sequence. **b**, Five *joseki* favoured at different stages of self-play training. Each displayed corner sequence was played with the greatest frequency, among all corner sequences, during an iteration of self-play training. The timestamp of that iteration is indicated on the timeline. At 10 h a weak corner move was preferred. At 47 h the 3–3 invasion was most frequently played. This *joseki* is also common in human professional play; however AlphaGo Zero later discovered and preferred a new variation. Extended Data Figure 3 provides the frequency of occurence over time for all five sequences and the new variation. **c**, The first 80 moves of three self-play games that were played at different stages of training, using 1,600 simulations (around 0.4 s) per search. At 3 h, the game focuses greedily on capturing stones, much like a human beginner. At 19 h, the game exhibits the fundamentals of life-and-death, influence and territory. At 70 h, the game is remarkably balanced, involving multiple battles and a complicated *ko* fight, eventually resolving into a half-point win for white. See Supplementary Information for the full games.

either separate policy and value networks, as were used in AlphaGo Lee, or combined policy and value networks, as used in AlphaGo Zero; and using either the convolutional network architecture from AlphaGo Lee or the residual network architecture from AlphaGo Zero. Each network was trained to minimize the same loss function (equation (1)), using a fixed dataset of self-play games generated by AlphaGo Zero after 72 h of self-play training. Using a residual network was more accurate, achieved lower error and improved performance in AlphaGo by over 600 Elo. Combining policy and value together into a single network slightly reduced the move prediction accuracy, but reduced the value error and boosted playing performance in AlphaGo by around another 600 Elo. This is partly due to improved computational efficiency, but more importantly the dual objective regularizes the network to a common representation that supports multiple use cases.

Knowledge learned by AlphaGo Zero

AlphaGo Zero discovered a remarkable level of Go knowledge during its self-play training process. This included not only fundamental elements of human Go knowledge, but also non-standard strategies beyond the scope of traditional Go knowledge.

Figure 5 shows a timeline indicating when professional *joseki* (corner sequences) were discovered (Fig. 5a and Extended Data

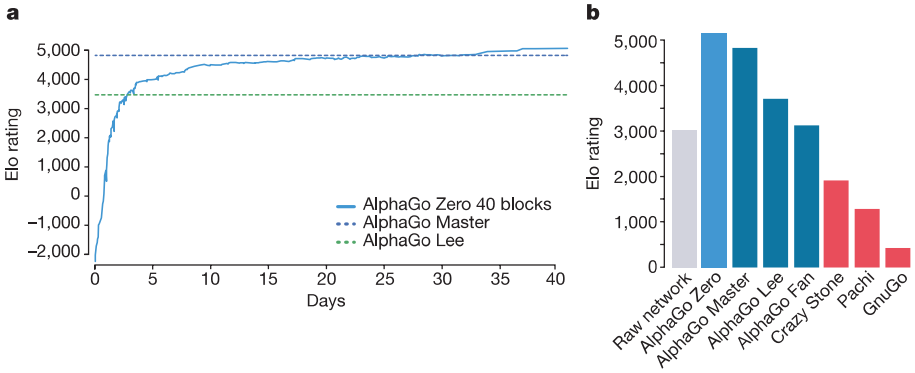

Figure 6 | Performance of AlphaGo Zero. a, Learning curve for AlphaGo Zero using a larger 40-block residual network over 40 days. The plot shows the performance of each player α_{θ_i} from each iteration i of our reinforcement learning algorithm. Elo ratings were computed from evaluation games between different players, using 0.4 s per search (see Methods). **b**, Final performance of AlphaGo Zero. AlphaGo Zero was trained for 40 days using a 40-block residual neural network. The plot shows the results of a tournament between: AlphaGo Zero, AlphaGo Master (defeated top human professionals 60–0 in online games), AlphaGo Lee (defeated Lee Sedol), AlphaGo Fan (defeated Fan Hui), as well as previous Go programs Crazy Stone, Pachi and GnuGo. Each program was given 5 s of thinking time per move. AlphaGo Zero and AlphaGo Master played on a single machine on the Google Cloud; AlphaGo Fan and AlphaGo Lee were distributed over many machines. The raw neural network from AlphaGo Zero is also included, which directly selects the move a with maximum probability p_a, without using MCTS. Programs were evaluated on an Elo scale[25]: a 200-point gap corresponds to a 75% probability of winning.

Fig. 2); ultimately AlphaGo Zero preferred new *joseki* variants that were previously unknown (Fig. 5b and Extended Data Fig. 3). Figure 5c shows several fast self-play games played at different stages of training (see Supplementary Information). Tournament length games played at regular intervals throughout training are shown in Extended Data Fig. 4 and in the Supplementary Information. AlphaGo Zero rapidly progressed from entirely random moves towards a sophisticated understanding of Go concepts, including *fuseki* (opening), *tesuji* (tactics), life-and-death, *ko* (repeated board situations), *yose* (endgame), capturing races, *sente* (initiative), shape, influence and territory, all discovered from first principles. Surprisingly, *shicho* ('ladder' capture sequences that may span the whole board)—one of the first elements of Go knowledge learned by humans—were only understood by AlphaGo Zero much later in training.

Final performance of AlphaGo Zero

We subsequently applied our reinforcement learning pipeline to a second instance of AlphaGo Zero using a larger neural network and over a longer duration. Training again started from completely random behaviour and continued for approximately 40 days.

Over the course of training, 29 million games of self-play were generated. Parameters were updated from 3.1 million mini-batches of 2,048 positions each. The neural network contained 40 residual blocks. The learning curve is shown in Fig. 6a. Games played at regular intervals throughout training are shown in Extended Data Fig. 5 and in the Supplementary Information.

We evaluated the fully trained AlphaGo Zero using an internal tournament against AlphaGo Fan, AlphaGo Lee and several previous Go programs. We also played games against the strongest existing program, AlphaGo Master—a program based on the algorithm and architecture presented in this paper but using human data and features (see Methods)—which defeated the strongest human professional players 60–0 in online games in January 2017[34]. In our evaluation, all programs were allowed 5 s of thinking time per move; AlphaGo Zero and AlphaGo Master each played on a single machine with 4 TPUs; AlphaGo Fan and AlphaGo Lee were distributed over 176 GPUs and 48 TPUs, respectively. We also included a player based solely on the raw neural network of AlphaGo Zero; this player simply selected the move with maximum probability.

Figure 6b shows the performance of each program on an Elo scale. The raw neural network, without using any lookahead, achieved an Elo rating of 3,055. AlphaGo Zero achieved a rating of 5,185, compared to 4,858 for AlphaGo Master, 3,739 for AlphaGo Lee and 3,144 for AlphaGo Fan.

Finally, we evaluated AlphaGo Zero head to head against AlphaGo Master in a 100-game match with 2-h time controls. AlphaGo Zero won by 89 games to 11 (see Extended Data Fig. 6 and Supplementary Information).

Conclusion

Our results comprehensively demonstrate that a pure reinforcement learning approach is fully feasible, even in the most challenging of domains: it is possible to train to superhuman level, without human examples or guidance, given no knowledge of the domain beyond basic rules. Furthermore, a pure reinforcement learning approach requires just a few more hours to train, and achieves much better asymptotic performance, compared to training on human expert data. Using this approach, AlphaGo Zero defeated the strongest previous versions of AlphaGo, which were trained from human data using handcrafted features, by a large margin.

Humankind has accumulated Go knowledge from millions of games played over thousands of years, collectively distilled into patterns, proverbs and books. In the space of a few days, starting *tabula rasa*, AlphaGo Zero was able to rediscover much of this Go knowledge, as well as novel strategies that provide new insights into the oldest of games.

Online Content Methods, along with any additional Extended Data display items and Source Data, are available in the online version of the paper; references unique to these sections appear only in the online paper.

Received 7 April; accepted 13 September 2017.

1. Friedman, J., Hastie, T. & Tibshirani, R. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Springer, 2009).
2. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521,** 436–444 (2015).
3. Krizhevsky, A., Sutskever, I. & Hinton, G. ImageNet classification with deep convolutional neural networks. In *Adv. Neural Inf. Process. Syst.* Vol. 25 (eds Pereira, F., Burges, C. J. C., Bottou, L. & Weinberger, K. Q.) 1097–1105 (2012).
4. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proc. 29th IEEE Conf. Comput. Vis. Pattern Recognit.* 770–778 (2016).
5. Hayes-Roth, F., Waterman, D. & Lenat, D. *Building Expert Systems* (Addison-Wesley, 1984).
6. Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015).
7. Guo, X., Singh, S. P., Lee, H., Lewis, R. L. & Wang, X. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Adv. Neural Inf. Process. Syst.* Vol. 27 (eds Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D. & Weinberger, K. Q.) 3338–3346 (2014).