Mastering the game of Go without human knowledge

在没有人类知识的情况下掌握围棋游戏

粗略翻译，如有指请正发邮箱至 hou.zg@foxmail.com

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.

人工智能长期以来的一个目标是创造一个能够在具有挑战性的领域，以超越人类的精通程度学习的算法，“tabula rasa”（译注：一种认知论观念，指个体在没有先天精神内容的情况下诞生，所有的知识都来自于后天的经验或感知）。此前，AlphaGo 成为首个在围棋中战胜人类世界冠军的系统。AlphaGo 的那些神经网络使用人类专家下棋的数据进行监督学习训练，同时也通过自我对弈进行强化学习。在这里，我们介绍一种仅基于强化学习的算法，不使用人类的数据、指导或规则以外的专业领域知识。AlphaGo 成了自己的老师。我们训练了一个神经网络来预测 AlphaGo 自己的落子选择和 AlphaGo 自我对弈的赢家。这种神经网络提高了树搜索的强度，使落子质量更高，自我对弈迭代更强。从“tabula rasa”开始，我们的新系统 AlphaGo Zero 实现了超人的表现，以 100：0 的成绩击败了此前发表的 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.

我们在人工智能方面取得了很大的进展，原因在于使用了经过训练的监督学习系统来复制人类专家的决策 [1-4]。但是，专家数据集通常很昂贵，不可靠或根本无法使用。即使有可靠的数据集，它们也可能会对以这种方式训练的系统的性能施加上限 [5]。相比之下，强化学习系统是根据他们自己的经验进行训练的，原则上允许他们超越人类能力，并且可以在缺乏人力专业知识的领域运作。最近，通过强化学习训练的深度神经网络，朝着这个目标快速发展。这些系统在计算机游戏中的表现优于人类，如 Atari[6,7] 和 3D 虚拟环境 [8-10]。然而，在人工智能方面最具挑战性的领域——比如被广泛认为是人工智能的巨大挑战的围棋游戏——需要在广阔的搜索空间中实现精确和复杂的前瞻。之前完全一般的方法没有在这些领域实现人类级别的表现。

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.

AlphaGo 是第一个在围棋中实现超人表现的项目。我们称之为 AlphaGo Fan 的发布版本 [12] 于 2015 年 10 月击败了欧洲冠军 Fan Hui。AlphaGo Fan 使用了两个深度神经网络：输出移动概率的策略网络和输出位置评估的价值网络。策略网络最初是通过监督学习来准确地预测人类专家的落子，随后通过策略升级强化学习进行了改进。价值网络通过策略网络自我对弈训练，用来预测游戏的赢家。一旦接受训练，这些网络就会与蒙特卡洛树搜索（MCTS）[13-15] 结合使用，从而提供先行搜索，使用策略网络将搜索范围缩小为高概率移动，并使用价值网络（结合蒙特卡洛的快速走子演算策略）来评估树中的位置。我们称之为 AlphaGo Lee 的后续版本使用了类似的方法（请参阅方法），并于 2016 年 3 月击败了获得 18 个国际冠军的 Lee Sedol。

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.

我们的项目 AlphaGo Zero 与 AlphaGo Fan 和 AlphaGo Lee[12] 在几个重要方面有所不同。 首先，它只是通过自我增强强化学习进行训练，从随机比赛开始，没有任何监督或使用人体数据。 其次，它只使用黑子和白子作为输入功能。 第三，它使用单一的神经网络，而不是单独的策略和价值网络。 最后，它使用更简单的树搜索，该搜索依赖于这个单一的神经网络来评估位置和样本移动，而无需执行任何蒙特卡洛走子演算（译者注：这是其他围棋程序使用的快速、随机游戏，用来预测哪一方会获胜，即快速价值网络）。 为了实现这些结果，我们引入了一种新的强化学习算法，该算法在训练环路内部结合了前瞻搜索，从而实现了快速改进和精确而稳定的学习。 在方法中描述了搜索算法，训练过程和网络体系结构中的其他技术差异。

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).

我们的新方法使用参数为 θ 的深度神经网络 f_θ。该神经网络将包含历史的棋局 s 作为输入，并输出移动概率和评估值，$(\boldsymbol{p}, v) = f_\theta(s)$。 移动概率向量 $\boldsymbol{p}$ 表示选择每个移动 a（包括跳过）的概率，$p_a = \Pr(a|s)$。 评估值标量 v，估计当前棋手从当前棋盘 s 获胜的概率。 这个神经网络将策略网络和价值网络 [12] 的角色结合到一个架构中。 神经网络由许多残差模块 [4,17]，批量归一化 [18] 以，非线性整流器 [19] 构成的卷积层组成（见方法）。

The neural network in AlphaGo Zero is trained from games of selfplay 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.

AlphaGo Zero 中的神经网络是由依赖于一种新型强化学习算法的自我对弈方法训练出来的。 在每个棋局 s，执行由神经网络 f_θ 指导的 MCTS。 MCTS 输出每次移动的概率 $\boldsymbol{\pi}$。 这些搜索概率通常选择比神经网络 $f_\theta(s)$ 的原始移动概率 $\boldsymbol{p}$ 更有效的移动；因此，MCTS 可被视为一个强有力的策略加强算子 [20,21]。 使用搜索进行自我对弈——使用改进的基于 MCTS 的策略来选择每个动作，然后使用游戏获胜者 z 作为价值的样本——可以被视为一个强大的策略评估算子。 我们强化学习算法的主要思想是使用这些搜索算子。

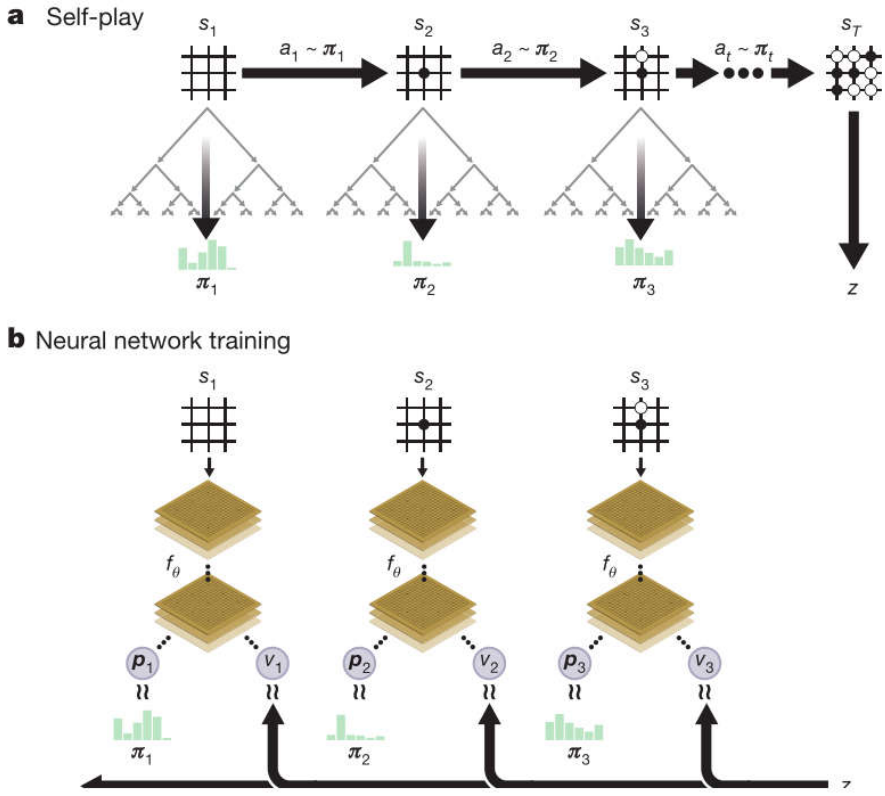


Figure **1 | Self-play reinforcement learning in AlphaGo Zero.**

a The program plays a game $s_1, \ldots, s_T$ 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, $a_t, \ldots, \boldsymbol{\pi}_t$.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 $\boldsymbol{\pi}_t$, 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**.

c 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 selfplay winner $(\boldsymbol{\pi}, z)$; 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.

a 该程序与自己进行一场游戏，棋盘状态为$s_1, \ldots, s_T$。 在每个棋盘 s_t中，一个使用最新的神经网络 f_θ的 MCTS α_θ被执行（见图 2）。 程序根据 MCTS 计算的搜索概率$a_t, \ldots, \boldsymbol{\pi}_t$选择移动。根据游戏规则对最终的棋盘 s_T 进行评分以计算游戏获胜者 z。

b AlphaGo Zero 中的神经网络训练。 神经网络将原始棋盘 s_t作为其输入，将其通过具有参数 θ的多个卷积层，并输出代表移动概率分布的向量$\boldsymbol{p}_t$ ，以及代表当前棋手在当前棋盘 s_t下赢的概率，标量值 v_t。 更新神经网络参数 θ以最大化策略向量$\boldsymbol{p}_t$与搜索概率 $\boldsymbol{\pi}_t$的相似性，并使预测获胜者 v_t和游戏赢者 z 之间的误差最小化（参见等式（1））。 新参数用于下一次自我对弈，如 a 中所示。

c 在策略迭代过程中重复 [22,23]：更新神经网络的参数以使移动概率和值$(\boldsymbol{p}, v) = f_\theta(s)$更接近匹配改进的搜索概率以及游戏的获胜者$(\boldsymbol{\pi}, z)$；这些新参数将用于下一次自我对弈，以使搜索更加强大。 图 1 说明了自我博弈训练工作流。

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 $U(s, a) \propto \frac{P(s,a)}{1 + N(s,a)}$ [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, $Q(s,a) = \frac{1}{N(s,a)} \sum_{s'|s,a\to s'} V(s')$, where $s,a \to s'$ indicates that a simulation eventually reached s′ after move a from position s.

MCTS 使用神经网络f_θ来指导其模拟（见图 2）。 搜索树中的每个边(s,a)存储先验概率 $P(s,a)$，访问计数 $N(s,a)$和动作值 $Q(s,a)$。 每个模拟从根状态开始并且迭代地选择可以使置信上限 $Q(s,a)+U(s,a)$最大化的移动，直到遇到叶节点 s' 结束，其中$U(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$ [12,24]。这个叶子棋盘只被网络扩展和评估一次，以产生先验概率和评估值，$(P(s',\cdot), V(s')) = f\theta(s')$。 在模拟中遍历的每个边$(s,a)$被更新以增加它的访问次数 $N(s,a)$，并且将它的动作值更新为在这些模拟上的平均评估，$Q(s,a) = \frac{1}{N(s,a)} \sum_{s'|s,a\to s'} V(s')$，其中，$s,a \to s'$表示移动 a 后最终到达了$s'$的模拟。

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, $\boldsymbol{\pi} = \alpha_\theta(s)$, proportional to the exponentiated visit count for each move, $\pi_a \propto N(s,a)^{\frac{1}{\tau}}$, where τ is a temperature parameter.

考虑到神经网络参数 θ 和根位置 s，MCTS 可以被看作是一种自我博弈算法，它计算一个搜索概率向量，推荐下棋的动作 $\boldsymbol{\pi} = \alpha_\theta(s)$，与每次移动的指数访问次数成正比，$\pi_a \propto N(s,a)^{\frac{1}{\tau}}$，其中 τ 是温度参数。

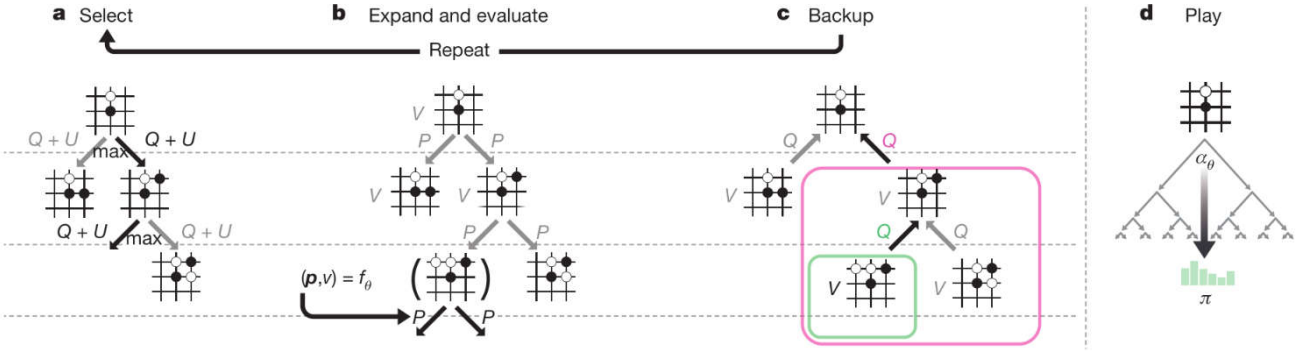


Figure 2 | MCTS in AlphaGo Zero.

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.

a 每次模拟通过选择具有最大动作值 Q 的边缘加上取决于所存储的先验概率 P 和该边缘的访问计数 N（其每次经过时递增）的上置信界限 U 来遍历树。

b 叶节点扩展并且相关位置 s 由神经网络评估$(\boldsymbol{P}(\boldsymbol{s},\cdot), V(s)) = f_\theta(s)$; P 值的向量存储在 s 的输出边上。

c 更新动作值 Q 以跟踪该动作下的子树中所有评估值 V 的均值。

d 一旦搜索完成，搜索概率 π 返回，与$N^{\frac{1}{\tau}}$ 成正比，其中 N 是从根状态到每个移动的访问计数，并且 τ 是控制温度的参数。

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 $\boldsymbol{\pi}_t = \alpha_{\theta_{i-1}}(s_t)$ is executed using the previous iteration of neural network $f_{\theta_{i-1}}$ and a move is played by sampling the search probabilities $\boldsymbol{\pi}_t$. 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 $(t_t, \boldsymbol{\pi}_t, z_t)$,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 $(s, \boldsymbol{\pi}, z)$ 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 $\boldsymbol{\pi}$. 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:

神经网络通过使用 MCTS 执行每个动作的自我增强强化学习算法进行训练。首先，神经网络被初始化为随机权重 θ_0。在随后的每次迭代 i≥1 时，产生自我对弈游戏（图 1a）。在每次时间步长 t 中，使用先前的神经网络 $f_{\theta_{i-1}}$ 迭代执行 MCTS 搜索 $\boldsymbol{\pi}_t = \alpha_{\theta_{i-1}}(s_t)$，并且通过对搜索概率 $\boldsymbol{\pi}_t$ 进行采样，得到落子的行为。当两个棋手都不落子，，或者搜索值下降到低于投降阈值或者当游戏超过最大长度时，游戏在步骤 T 终止；然后对游戏进行评分以给出最终奖励值 $r_T \in \{-1, +1\}$（详见 Methods）。每个时间步 t 的数据存储为 $(t_t, \boldsymbol{\pi}_t, z_t)$，其中 $z_t = \pm r_T$ 是从步骤 t 处当前棋手角度出发的游戏获胜者。并行地（图 1b），新的网络参数 θ_i 从最后一次自我对弈的所有时间步长中均匀采样的数据 $(s, \boldsymbol{\pi}, z)$ 中进行训练。调整神经网络 $(\boldsymbol{p}, v) = f_{\theta-i}(s)$ 以最小化预测值 v 与自我赢得者 z 之间的误差，并使神经网络移动概率 p 与搜索概率 π 的相似性最大化。具体而言，参数 θ 通过梯度下降在损失函数 l 上进行调整，所述损失函数 l 分别对 meansquared error 和 crossentropy loss 进行求和：

$$(\boldsymbol{p}, v) = f_\theta(s) \ \ and \ \ l = (z - v)^2 - \boldsymbol{\pi}^T \log p + c \parallel \theta \parallel^2 \tag{1}$$

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

其中 c 是控制 L2 权重正则化水平的参数（以防止过拟合）。

Empirical analysis of AlphaGo Zero training （AlphaGo Zero 训练的经验分析）

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.

我们使用强化学习途径来训练我们的计划 AlphaGo Zero。 训练从完全随机落子开始，持续约 3 天，无需人工干预。

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

在训练工作流中，每个 MCTS 使用 1,600 次模拟，每次移动的思考时间大约为 0.4s，从而产生了 490 万次自我游戏。 参数从 20 万个位置的 700,000 个小型库中更新。 神经网络包含 20 个残余块（详见方法）。

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).

图 3a 显示了 AlphaGo Zero 在自我训练强化学习过程中的表现，作为训练时间的函数，在 Elo scale[25] 上。 学习在整个训练过程中进展顺利，并没有遭受以前文献中提到的振荡或灾难性遗忘 [26-28]。 令人惊讶的是，AlphaGo Zero 仅仅 36 小时就跑赢了 AlphaGo Lee。 相比之下，AlphaGo Lee 进行了数月的训练。 72 小时后，我们根据在首尔 2 小时人机比赛中使用相同的控制和匹配条件（请参阅方法），对 AlphaGo Zero 与击败了 Lee Sedol 的 AlphaGo Lee 确切版本进行了评估。 AlphaGo Zero 使用带有 4 个张量处理单元（TPUs）[29] 的单台机器，而 AlphaGo Lee 分布在多台机器上并使用 48 个 TPU。 AlphaGo Zero 以 100：0 的成绩击败 AlphaGo Lee (参见扩展数据图 1 和补充信息)。

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.

为了评估自我强化学习的优点，与从人类数据中学习相比，我们训练了第二个神经网络（使用相同的体系结构）来预测 KGS 服务器数据集中的专家动作；与之前的工作 [12,30-33] 相比，这实现了预测的准确性（分别见当前和以前的结果的扩展数据表 1 和 2）。 监督式学习的初始表现更好，并且能更好地预测人类职业动作（图 3）。 值得注意的是，尽管监督学习获得了更高的移动预测准确度，但自学者的整体表现更好，在训练的前 24 小时内击败了训练有素的棋手。 这表明 AlphaGo Zero 可能正在学习一种与人类下棋十分不同的策略。

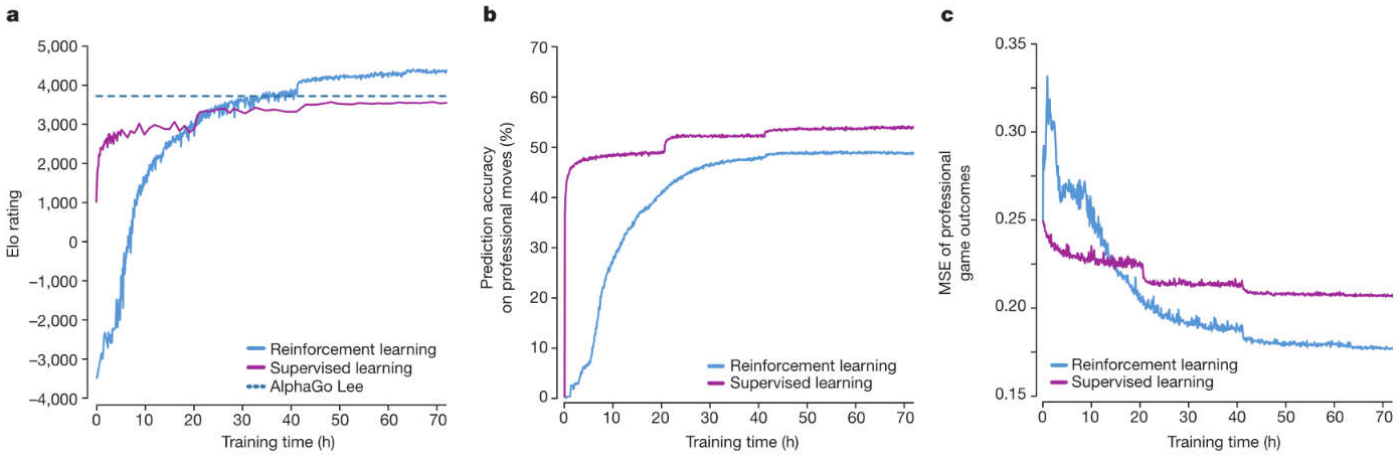


Figure 3 | Empirical evaluation of AlphaGo Zero.

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.

a 自我加强强化学习的表现。该图显示了每次在 AlphaGo Zero 中强化学习的迭代 i 中每个 MCTS 棋手 α_{θ_i}的表现。 Elo 评分是根据不同参与者之间的评估游戏计算得出的，每个步骤使用 0.4s 的思考时间（见方法）。为了比较，该图还显示了使用 KGS 数据集通过监督学习人类数据进行训练的类似棋手。

b 预测人类职业运动的准确性。该曲线图显示了在自我博弈 i 的每次迭代中，预测来自 GoKifu 数据集的人类职业移动的神经网络f_{θ_i}的准确性。精确度衡量了神经网络判定人类移动最高概率的位置的百分比。该图还显示了由监督学习训练的神经网络的准确性。

c 人类职业比赛结果的平均误差（MSE）。该图显示了在每次自我运动迭代时，神经网络f_{θ_i}的 MSE，用于预测 GoKifu 数据集中人类职业比赛的结果。 MSE 在实际结果输出$z \in \{-1, +1\}$和神经网络值 v之间，以 1/4 的比例缩放到 0-1 的范围。该图还显示了通过监督学习训练的神经网络的 MSE。

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 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.

为了分离架构和算法的贡献，我们将 AlphaGo Zero 中的神经网络架构的性能与以前在 AlphaGo Lee 中使用的的神经网络架构进行了比较 (见图 4)。 四个神经网络被创建，一个是 AlphaGo Lee 中使用的独立的策略和价值网络；一个是 AlphaGo Zero 中使用了将策略和价值网络组合的神经网络；还有一个是使用 AlphaGo Lee 的卷积网络架构；最后一个是 AlphaGo Zero 的残差网络架构。每个网络都经过训练，以最小化相同的损失函数（方程（1）），使用由 AlphaGo Zero 在 72 小时的自我博弈训练后产生的固定数据集。使用残差网络更准确，实现了更低的误差，AlphaGo 的性能提高了 600 多 Elo。将策略和价值组合在一起形成的网络，虽然略微降低了移动预测的准确性，但是将 AlphaGo 的价值误差和提高的播放性能减少了约 600 个 Elo。部分原因在于提高了计算效率，但更重要的是，双重目标将网络规范化为支持多种用例的常见表示。

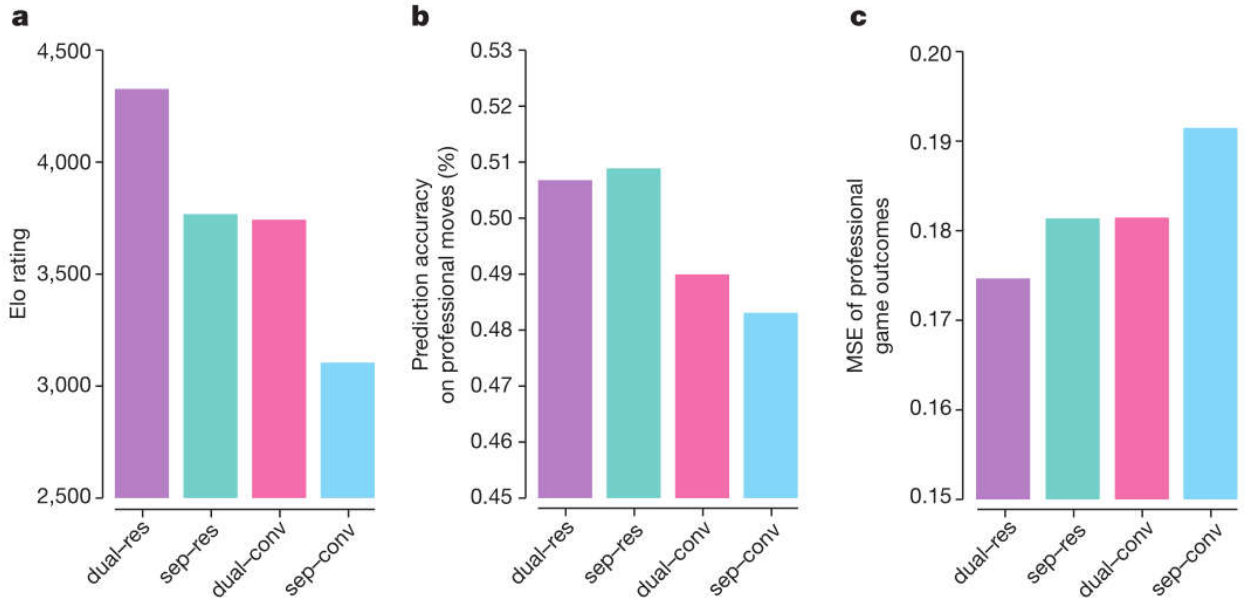


Figure 4 | Comparison of neural network architectures in AlphaGo Zero and AlphaGo Lee.

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.

比较使用单独的网络（sep）、组合的策略和价值网络（dual）、使用卷积网络（conv）以及残差网络（res）的神经网络架构的比较。 "dual-res" 和 "sep-conv" 组合分别对应于 AlphaGo Zero 和 AlphaGo Lee 中使用的神经网络架构。 每个网络都通过之前运行的 AlphaGo Zero 生成的固定数据集进行训练。

a 每个训练过的网络都与 AlphaGo Zero 的搜索相结合，以获得不同的棋手。 Elo 评级是根据这些不同参与者之间的评估游戏计算出来的，每个步骤使用 5 s 的思考时间。

b 对于每个网络架构的人类职业移动（来自 GoKifu 数据集）的预测准确性。 c 每个网络架构的人类职业游戏成果（来自 GoKifu 数据集）的 MSE。

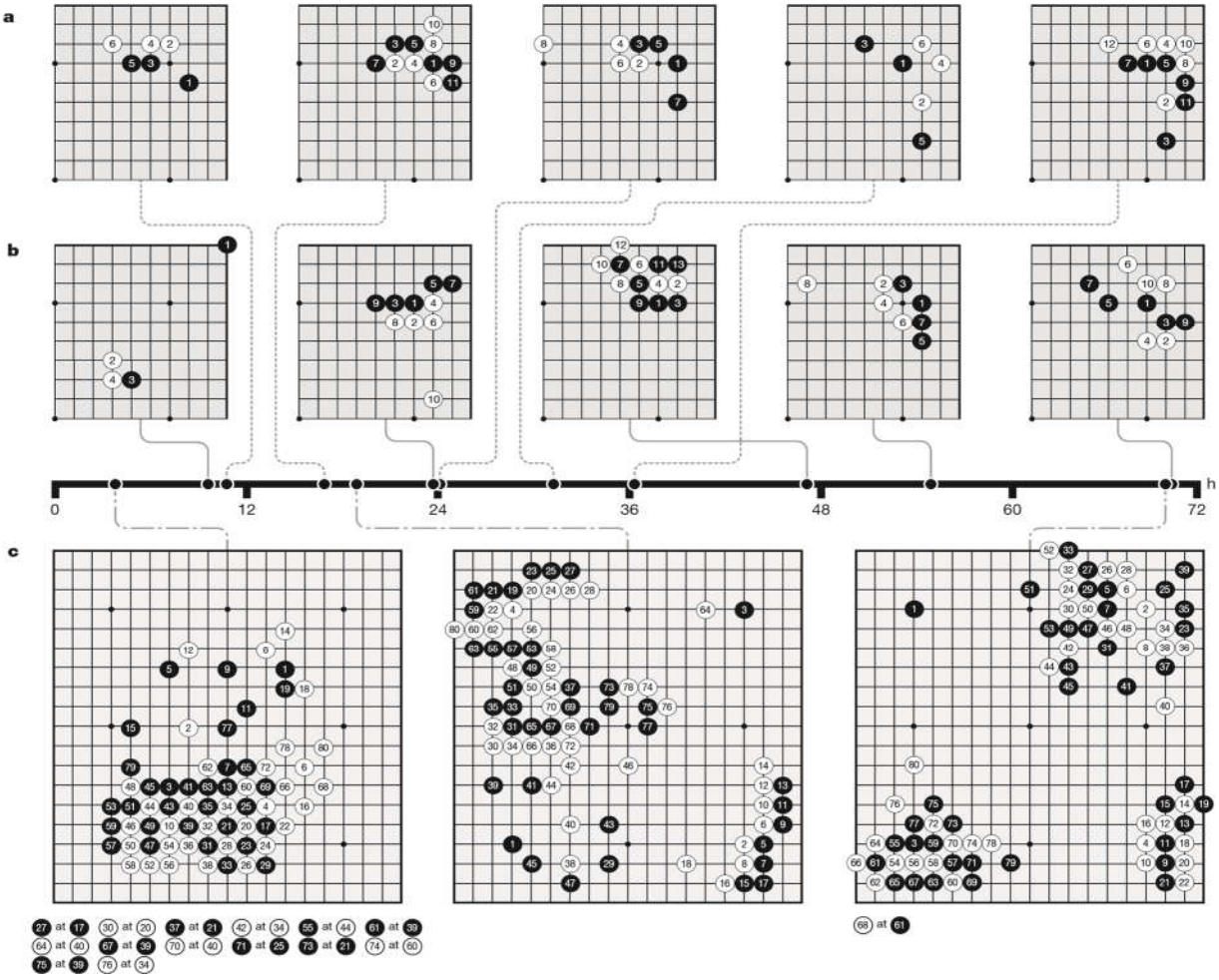


Figure 5 | Go knowledge learned by AlphaGo Zero.

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.

a AlphaGo Zero 训练期间发现的五种人类定式（共同角点序列）。相关时间戳指示在自我博弈训练期间每个序列的第一次发生（考虑旋转和反射）。扩展数据图 2 提供了每个序列的训练发生频率。

b 五定式在不同阶段的自我博弈训练频繁出现。在自我博弈训练的迭代期间，每个显示的角点序列在所有角点序列中以最大的频率出现。该迭代的时间戳在时间线上指示。在 10 小时的时候，一个弱的角落移动是首选。在 47 小时 3-3 侵入最频繁。这个 joseki 在人类专业游戏中也很常见;然而 AlphaGo Zero 后来发现并且偏爱了一个新的变体。扩展数据图 3 提供了所有五个序列随时间发生的频率和新变化。

c 在不同训练阶段进行的三场自我对弈的前 80 次移动中，每次搜索使用了 1,600 次模拟（约 0.4 秒）。在 3 小时，游戏贪婪地吃子，就像人类初学者一样。在 19 小时，游戏表现出对生死观，局势和实地的意识基础。 70 小时后，这场比赛非常平衡，包括多场战斗以及一场复杂的*劫* 战斗，最终结果为白方赢半子。请参阅完整游戏的补充信息。

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.

AlphaGo Zero 在其自我博弈训练过程中发现了非凡的围棋知识。 这不仅包括人类 Go 围棋知识的基本要素，还包括超出传统围棋知识范围的非标准策略。

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

图 5 显示了一个时间线，表明何时发现了专业定式（角点序列）（图 5a 和扩展数据图 2）;最终 AlphaGo Zero 更喜欢先前未知的新的定式变体（图 5b 和扩展数据图 3）。图 5c 显示了几种在不同训练阶段进行的快速自我博弈（参见补充信息）。在整个训练中定期进行的锦标赛长度比赛在扩展数据图 4 和补充信息中显示。 AlphaGo Zero 从完全随机移动迅速发展到对围棋概念的完善理解，包括 *fuseki*（布局），*tesuji*（手筋），生死观，*ko*（劫），*yose*（官子），提子，*sente*（先手），棋形，局势和实地，都是从最初的原则发现的。令人惊讶的是，*Shicho*（征子:可能横跨整个棋盘的"阶梯"捕获序列）—— 人类学习的围棋知识的第一要素之一 —— 只有在 AlphaGo Zero 的后期训练中才能被理解。

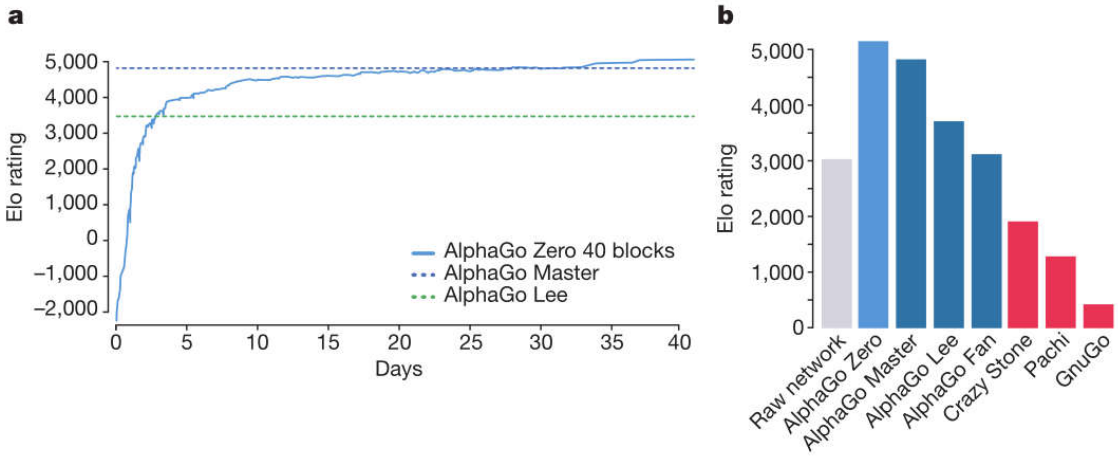


Figure 6 | Performance of AlphaGo Zero.

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.

a 40 天内，AlphaGo Zero 使用更大的 40 块残差网络的学习曲线。该图显示了在我们的强化学习算法的每次迭代 i 的每个参与者 α_{θ_i} 的表现。 Elo 评级是根据不同参与者之间的比赛评估计算得出的，每次搜索使用

0.4 s（请参阅方法）。

b AlphaGo Zero的最终表现。AlphaGo Zero使用40块残差神经网络了训练了40天。该图显示了AlphaGo Zero，AlphaGo Master（在线游戏中击败顶级人类专业人士 60-0），AlphaGo Lee（击败 Lee Sedol），AlphaGo Fan（击败 Fan Hui）以及之前的围棋项目 Crazy Stone，Pachi 和 GnuGo。每个程序都有 5s 的思考时间。 AlphaGo Zero 和 AlphaGo Master 在 Google Cloud 上的一台机器上下棋; AlphaGo Fan 和 AlphaGo Lee 分布在许多机器上。AlphaGo Zero 的原始神经网络也包含在内，它直接选择移动 a 的最大概率为p_a，而不使用 MCTS。程序按照 Elo 等级 [25] 进行评估：200 子的差距对应于获胜概率为 75%。

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.

随后我们使用更大的神经网络和更长的持续时间将我们的强化学习工作流应用于 AlphaGo Zero 的第二个实例。再次训练从完全随机落子开始并持续大约 40 天。

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.

在训练过程中，产生了 2900 万次自我博弈。参数每次从 310 万棋盘中提取 2048 个进行更新。神经网络包含 40 个残差块。学习曲线如图 6a 所示。 在整个训练过程中定期进行的比赛显示在扩展数据图 5 和补充信息中。

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.

我们让训练有素的 AlphaGo Zero 与 AlphaGo Fan，AlphaGo Lee 和之前的几个围棋项目进行内部竞标赛，从而对它评估。 我们还与最强大的现有程序 AlphaGo Master 进行了比赛，该程序基于本文提供的算法和体系结构，但使用了人类数据和特征（请参阅方法） - 它在 2017 年 1 月的在线比赛中 60-0 击败了最强的人类职业棋手。在我们的评估中，所有项目都允许每次行动有 5 秒的思考时间; AlphaGo Zero 和 AlphaGo Master 每台在带有 4 个 TPU 的单台机器上播放; AlphaGo Fan 和 AlphaGo Lee 分别分布有 176 个 GPU 和 48 个 TPU。 我们还包括一个完全基于 AlphaGo Zero 原始神经网络的参加者；该棋手只是以最大的概率选择了该移动。

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.

图 6b 显示了每个程序在 Elo 规模上的表现。 未使用任何预测的原始神经网络实现了 3,055 的 Elo 评级。AlphaGo Zero 获得了 5,185 的评分，而 AlphaGo Master 的 4,858，AlphaGo Lee 的 3,739 和 AlphaGo Fan 的 3,144。

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).

最后，我们评估了 AlphaGo Zero 头对阵 AlphaGo Master，并与 2h 时间控件进行了 100 场比赛。 AlphaGo Zero 赢得 89 场比赛，达到 11 场（参见扩展数据图 6 和补充信息）。

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.

我们的研究结果全面证明，即使在最具挑战性的领域中，纯粹的强化学习方法也是完全可行的：在没有给出超过基本规则的规则，没有关于领域的知识，没有人类的例子或指导的情况下，强化学习方法就可以训练到超人的水平。 此外，与对人类专家数据的训练相比，纯粹的强化学习方法只需要几个小时的训练时间，并且达到更好的渐进性能。 使用这种方法，AlphaGo Zero 大幅度击败了将人类数据作为特征输入从而训练出的 AlphaGo 最强大的先前版本。

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.

几千年来，人类已经积累了数百万次围棋对弈的知识，并全部提取为模式，谚语或者写成书籍。 在几天的时间里，AlphaGo Zero 发起了 tabula rasa，发现围棋的许多知识，为最古老的游戏提供了新的策略，带来了新的领悟。

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.**

Friedman, J., Hastie, T. & Tibshirani, R. The Elements of Statistical Learning: Data Mining, Inference, and Prediction (Springer, 2009).

LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521,** 436–444 (2015).

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).

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).

Hayes-Roth, F., Waterman, D. & Lenat, D. *Building Expert Systems* (AddisonWesley, 1984).

Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015).

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).

Mnih, V. *et al.* Asynchronous methods for deep reinforcement learning. In *Proc. 33rd Int. Conf. Mach. Learn.* Vol. 48 (eds Balcan, M. F. & Weinberger, K. Q.) 1928–1937 (2016).

Jaderberg, M. *et al.* Reinforcement learning with unsupervised auxiliary tasks. In *5th Int. Conf. Learn. Representations* (2017).

Dosovitskiy, A. & Koltun, V. Learning to act by predicting the future. In *5th Int. Conf. Learn. Representations* (2017).

Man´dziuk, J. in *Challenges for Computational Intelligence* (Duch, W. & Man´dziuk, J.) 407–442 (Springer, 2007).

Silver, D. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529,** 484–489 (2016).

Coulom, R. Efficient selectivity and backup operators in Monte-Carlo tree search. In *5th Int. Conf. Computers and Games* (eds Ciancarini, P. & van den Herik, H. J.) 72–83 (2006).

Kocsis, L. & Szepesvári, C. Bandit based Monte-Carlo planning. In *15th Eu. Conf. Mach. Learn.* 282–293 (2006).

Browne, C. *et al.* A survey of Monte Carlo tree search methods. *IEEE Trans. Comput. Intell. AI Games* **4,** 1–49 (2012).

Fukushima, K. Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybern.* **36,** 193–202 (1980).

LeCun, Y. & Bengio, Y. in *The Handbook of Brain Theory and Neural Networks* Ch. 3 (ed. Arbib, M.) 276–278 (MIT Press, 1995).

Ioffe, S. & Szegedy, C. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proc. 32nd Int. Conf. Mach. Learn.* Vol. 37 448–456 (2015).

Hahnloser, R. H. R., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J. & Seung, H. S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* **405,** 947–951 (2000).

Howard, R. Dynamic Programming and Markov Processes (MIT Press, 1960).

Sutton, R. & Barto, A. *Reinforcement Learning: an Introduction* (MIT Press, 1998).

Bertsekas, D. P. Approximate policy iteration: a survey and some new methods. *J. Control Theory Appl.* **9,** 310–335

(2011).
Scherrer, B. Approximate policy iteration schemes: a comparison. In *Proc. 31st Int. Conf. Mach. Learn*. Vol. 32 1314–1322 (2014).
Rosin, C. D. Multi-armed bandits with episode context. *Ann. Math. Artif. Intell.* **61,** 203–230 (2011).
Coulom, R. Whole-history rating: a Bayesian rating system for players of time-varying strength. In *Int. Conf. Comput. Games* (eds van den Herik, H. J., Xu, X. Ma, Z. & Winands, M. H. M.) Vol. 5131 113–124 (Springer, 2008).
Laurent, G. J., Matignon, L. & Le Fort-Piat, N. The world of independent learners is not Markovian. *Int. J. Knowledge-Based Intelligent Engineering Systems* **15,** 55–64 (2011).
Foerster, J. N. *et al.* Stabilising experience replay for deep multi-agent reinforcement learning. In *Proc. 34th Int. Conf. Mach. Learn.* Vol. 70 1146–1155 (2017).
Heinrich, J. & Silver, D. Deep reinforcement learning from self-play in imperfect-information games. In *NIPS Deep Reinforcement Learning Workshop* (2016).
Jouppi, N. P. *et al.* In-datacenter performance analysis of a Tensor Processing Unit. *Proc. 44th Annu. Int. Symp. Comp. Architecture* Vol. 17 1–12 (2017).
Maddison, C. J., Huang, A., Sutskever, I. & Silver, D. Move evaluation in Go using deep convolutional neural networks. In *3rd Int. Conf. Learn. Representations.* (2015).
Clark, C. & Storkey, A. J. Training deep convolutional neural networks to play Go. In *Proc. 32nd Int. Conf. Mach. Learn.* Vol. 37 1766–1774 (2015).
Tian, Y. & Zhu, Y. Better computer Go player with neural network and long-term prediction. In *4th Int. Conf. Learn. Representations* (2016).
Cazenave, T. Residual networks for computer Go. *IEEE Trans. Comput. Intell. AI Games* https://doi.org/10.1109/TCIAIG.2017.2681042 (2017).
Huang, A. AlphaGo master online series of games. https://deepmind.com/ research/AlphaGo/match-archive/master (2017).

Supplementary Information is available in the online version of the paper.

Acknowledgements We thank A. Cain for work on the visuals; A. Barreto, G. Ostrovski, T. Ewalds, T. Schaul, J. Oh and N. Heess for reviewing the paper; and the rest of the DeepMind team for their support.

Author Contributions D.S., J.S., K.S., I.A., A.G., L.S. and T.H. designed and implemented the reinforcement learning algorithm in AlphaGo Zero. A.H., J.S., M.L. and D.S. designed and implemented the search in AlphaGo Zero. L.B., J.S., A.H., F.H., T.H., Y.C. and D.S. designed and implemented the evaluation framework for AlphaGo Zero. D.S., A.B., F.H., A.G., T.L., T.G., L.S., G.v.d.D. and D.H. managed and advised on the project. D.S., T.G. and A.G. wrote the paper.

Author Information Reprints and permissions information is available at www.nature.com/reprints. The authors declare no competing financial interests. Readers are welcome to comment on the online version of the paper. Publisher's note: Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations. Correspondence and requests for materials should be addressed to D.S. (davidsilver@google.com).

reviewer Information *Nature* thanks S. Singh and the other anonymous reviewer(s) for their contribution to the peer review of this work.

METHODS

Reinforcement learning.

Policy iteration[20,21] is a classic algorithm that generates a sequence of improving policies, by alternating between policy evaluation— estimating the value function of the current policy—and policy improvement— using the current value function to generate a better policy. A simple approach to policy evaluation is to estimate the value function from the outcomes of sampled trajectories[35,36]. A simple approach to policy improvement is to select actions greedily with respect to the value function[20]. In large state spaces, approximations are necessary to evaluate each policy and to represent its improvement[22,23].

策略迭代 [20,21] 是一个经典的算法，用来生成一系列的改进策略，通过交替策略评估——估计当前策略的价值函数——以及策略的改进——使用当前价值函数去生成一个更好的策略。策略评估的一个简单的方法是估计从 trajectories[35,36] 采样得到的价值函数。策略改进的一种简单方法是就价值函数而言贪婪地选择落子 [20]。在大的状态空间下，近似的评估策略、近似的表现其改进是必要的 [22,23]。

Classification-based reinforcement learning[37] improves the policy using a simple Monte Carlo search. Many rollouts are executed for each action; the action with the maximum mean value provides a positive training example, while all other actions provide negative training examples; a policy is then trained to classify actions as positive or negative, and used in subsequent rollouts. This may be viewed as a precursor to the policy component of AlphaGo Zero's training algorithm when $\tau \to 0$.

基础分类强化学习 [37] 使用一个简单的蒙特卡洛搜索改进策略。对于每一个动作，许多走子演算被执行；这种根据最大平均值所得的动作，提供了一种积极的训练样例，而所有其他的动作提供负的训练样例；一个策略被用来训练积极或消极落子的分类，并用于后续的走子演算。当 $\tau \to 0$ 时，这可以看作是 AlphaGo Zero 训练算法策略组成的前身。

A more recent instantiation, classification-based modified policy iteration (CBMPI), also performs policy evaluation by regressing a value function towards truncated rollout values, similar to the value component of AlphaGo Zero; this achieved state-of-the-art results in the game of Tetris[38]. However, this previous work was limited to simple rollouts and linear function approximation using handcrafted features.

最近的一个实例，基于改进策略迭代（CBMPI）的分类器，同样通过回归价值函数进行策略评估从而推出截断的走子演算策略，类似于 AlphaGo Zero 的价值策略组成；在 tetris[38] 俄罗斯方块游戏中取得了体现最高水平的结果。然而，预先工作仅限于简单的走子演算，以及使用手工特征的线性函数逼近。

The AlphaGo Zero self-play algorithm can similarly be understood as an approximate policy iteration scheme in which MCTS is used for both policy improvement and policy evaluation. Policy improvement starts with a neural network policy, executes an MCTS based on that policy's recommendations, and then projects the (much stronger) search policy back into the function space of the neural network. Policy evaluation is applied to the (much stronger) search policy: the outcomes of self-play games are also projected back into the function space of the neural network. These projection steps are achieved by training the neural network parameters to match the search probabilities and self-play game outcome respectively.

AlphaGo Zero 自我对弈算法同样可以被理解为一个近似策略迭代方案，在该方案中改善策略和策略评价都使用了 MCTS。策略的完善从一个神经网络的策略开始，执行一个基于策略建议的 MCTS，然后将（更强的）搜索策略回归到神经网络的函数空间。（更强的）的搜索策略应用到策略评估：自我对弈的结果也被回归到神经网络的函数空间。这些回归步骤是通过训练神经网络参数来匹配搜索概率和自我对弈结果而实现的。

Guo *et al.*[7] also project the output of MCTS into a neural network, either by regressing a value network towards the search value, or by classifying the action selected by MCTS. This approach was used to train a neural network for playing Atari games; however, the MCTS was fixed—there was no policy iteration—and did not make any use of the trained networks.

Guo 等科学家 [7] 也将 MCTS 结果输出到一个神经网络，或者将搜索值回归到价值网络，或分类 MCTS 选择的行为。这种方法被用来训练一个神经网络，玩 Atari 游戏；然而，MCTS 是固定的——没有策略迭代——也没有用来训练网络。

Self-play reinforcement learning in games.

Our approach is most directly applicable to Zero-sum games of perfect information. We follow the formalism of alternating Markov games described in previous work[12], noting that algorithms based on value or policy iteration extend naturally to this setting[39].

我们的方法最直接适用于完全信息的零和博弈游戏。 我们遵循在先前工作 [12] 中描述的交替马尔可夫博弈的形式，注意到基于价值或策略迭代的算法自然延伸到该设定 [39]。

Self-play reinforcement learning has previously been applied to the game of Go. NeuroGo[40,41] used a neural network to represent a value function, using a sophisticated architecture based on Go knowledge regarding connectivity, territory and eyes. This neural network was trained by temporal-difference learning[42] to predict territory in games of self-play, building on previous work[43]. A related approach, RLGO[44], represented the value function instead by a linear combination of features, exhaustively enumerating all 3 ×3 patterns of stones; it was trained by temporal-difference learning to predict the winner in games of self-play. Both NeuroGo and RLGO achieved a weak amateur level of play.

自我对弈强化学习以前已应用于围棋游戏。 NeuroGo[40,41] 使用神经网络来表示价值函数，使用基于围棋关于连接，实地和眼知识的复杂架构。 这个神经网络是通过临时差异学习 [42] 来训练的，以预测自我对弈的领域，建立在以前的工作基础上 [43]。 一个相关的方法，RLGO[44]，代表了价值函数，而不是线性组合的功能，详尽列举所有 3×3 模式的棋子；它通过临时差异学习来预测自我对弈中的赢家。 NeuroGo 和 RLGO 的业余水平都很差。

MCTS may also be viewed as a form of self-play reinforcement learning[45]. The nodes of the search tree contain the value function for the positions encountered during search; these values are updated to predict the winner of simulated games of self-play. MCTS programs have previously achieved strong amateur level in Go[46,47], but used substantial domain expertise: a fast rollout policy, based on handcrafted features[13,48], that evaluates positions by running simulations until the end of the game; and a tree policy, also based on handcrafted features, that selects moves within the search tree[47].

MCTS 也可以被看作是一种自我强化学习的形式 [45]。 搜索树的节点包含搜索期间遇到位置的值函数；这些值被更新以预测模拟的自我对弈的胜利者。 MCTS 项目之前在业余围棋中取得了强大的水平 [46,47]，但使用了大量的专业领域知识：基于手工特征的快速走子策略 [13,48]；通过模拟到游戏结束评估棋局；以及基于手工特征的树策略，该策略选择搜索树内的移动 [47]。

Self-play reinforcement learning approaches have achieved high levels of performance in other games: chess[49–51], checkers[52], backgammon[53], othello[54], Scrabble[55] and most recently poker[56]. In all of these examples, a value function was trained by regression[54–56] or temporal-difference learning[49–53] from training data generated by self-play. The trained value function was used as an evaluation function in an alpha–beta search[49–54], a simple Monte Carlo search[55,57] or counterfactual regret minimization[56]. However, these methods used handcrafted input features[49–53,56] or handcrafted feature templates[54,55]. In addition, the learning process used supervised learning to initialize weights[58], hand-selected weights for piece values[49,51,52], handcrafted restrictions on the action space[56] or used pre-existing computer programs as training opponents[49,50], or to generate game records[51].

自我对弈强化学习方法在其他游戏中都取得了高水平的表现：chess[49-51]，cheers[52]，backgammon[53]，othello[54]，scrabble[55] 和最近出现的扑克比赛 [56]。 在所有这些例子中，价值函数都是通过回归 [54-56] 或临时差异学习 [49-53] 从自我对弈产生的训练数据进行训练的。 经过训练的价值函数被用作 alpha–beta 搜索的评估函数 [49-54]，简单的蒙特卡罗搜索 [55,57] 或反事实遗憾最小化 [56] 非完全信息的博弈。 但是，这些方法使用手工制作的输入功能 [49-53,56] 或人为制作的功能模板 [54,55]。 此外，学习过程使用监督学习来初始化权重 [58]，人为选择片段值权重 [49,51,52]，对动作空间 [56] 人为制作的限制或者使用预先存在的计算机程序作为训练对手 [49,50] 或生成游戏记录 [51]。

Many of the most successful and widely used reinforcement learning methods were first introduced in the context of Zero-sum games: temporal-difference learning was first introduced for a checkers-playing program[59], while MCTS was introduced for the game of Go[13]. However, very similar algorithms have subsequently proven highly effective in video games[6–8,10], robotics[60], industrial control[61–63] and online recommendation systems[64,65].

许多最成功和最广泛使用的强化学习方法首先在零和游戏中引入：临时差异学习首先被引入了一个棋盘游戏程序 [59]，而 MCTS 则被引入了围棋 [13] 游戏。 然而，非常相似的算法随后在视频游戏 [6-8,10]，机器人 [60]，工业控制 [61-63] 和在线推荐系统 [64,65] 中被证明是非常有效的。

AlphaGo versions.

We compare three distinct versions of AlphaGo:

(1) AlphaGo Fan is the previously published program[12] that played against Fan Hui in October 2015. This program was distributed over many machines using 176 GPUs.

(2) AlphaGo Lee is the program that defeated Lee Sedol 4–1 in March 2016. It was previously unpublished, but is similar in most regards to AlphaGo Fan[12]. However, we highlight several key differences to facilitate a fair comparison. First, the value network was trained from the outcomes of fast games of self-play by AlphaGo, rather than games of self-play by the policy network; this procedure was iterated several times—an initial step towards the *tabula rasa* algorithm presented in this paper. Second, the policy and value networks were larger than those described in the original paper—using 12 convolutional layers of 256 planes— and were trained for more iterations. This player was also distributed over many machines using 48 TPUs, rather than GPUs, enabling it to evaluate neural networks faster during search.

(3) AlphaGo Master is the program that defeated top human players by 60–0 in January 2017[34]. It was previously unpublished, but uses the same neural network architecture, reinforcement learning algorithm, and MCTS algorithm as described in this paper. However, it uses the same handcrafted features and rollouts as AlphaGo Lee[12] and training was initialized by supervised learning from human data.

(4) AlphaGo Zero is the program described in this paper. It learns from self-play reinforcement learning, starting from random initial weights, without using rollouts, with no human supervision and using only the raw board history as input features. It uses just a single machine in the Google Cloud with 4 TPUs (AlphaGo Zero could also be distributed, but we chose to use the simplest possible search algorithm).

我们比较 AlphaGo 的三个不同版本：

(1) AlphaGo Fan 是 2015 年 10 月发布的与 Fan Hui 对战的程序 [12].该程序分布在使用了 176 个 GPU 的多台的机器上。

(2) AlphaGo Lee 是 2016 年 3 月以 4-1 击败 Lee Sedol 的项目。它以前没有发表，但和 AlphaGo Fan[12] 的大多数问候都很相似。但是，我们强调几个关键差异以促进公平比较。首先，价值网络是通过 AlphaGo 的自我游戏快速游戏的结果进行训练的，而不是策略网络的自我对弈游戏;这个过程迭代了几次，这是本文提出的白板算法的第一步。其次，策略和价值网络大于原始论文中描述的那些——使用 12 个 256 层的卷积层——并且被训练用于更多迭代。该播放器还分布在许多使用 48 TPU 而非 GPU 的机器上，使其能够在搜索过程中更快地评估神经网络。

(3) AlphaGo Master 是 2017 年 1 月以 60-0 击败顶级人类棋手的项目 [34]。此前，该计划尚未发布，但采用了本文所述的相同的神经网络架构，强化学习算法和 MCTS 算法。然而，它使用与 AlphaGo Lee12 相同的人工特征及走子演算，并且通过从人类数据监督学习来初始化训练。

(4) AlphaGo Zero 是本文描述的程序。它应用自我对弈强化学习方法学习，从随机初始权重开始，不使用走子演算，不需要人工监督，只使用原始棋局历史作为输入特征。它只使用 Google Cloud 中的一台机器，配有 4 个 TPU（AlphaGo Zero 也可以分布部署，但我们选择使用最简单的搜索算法）。

Domain knowledge.

Our primary contribution is to demonstrate that superhuman performance can be achieved without human domain knowledge. To clarify this contribution, we enumerate the domain knowledge that AlphaGo Zero uses, explicitly or implicitly, either in its training procedure or its MCTS; these are the items of knowledge that would need to be replaced for AlphaGo Zero to learn a different (alternating Markov) game.

我们的主要贡献是证明没有人类知识就可以实现超越人类的表现。 为了清楚地说明这一贡献，我们列举了 AlphaGo Zero 在其训练程序或 MCTS 中明确或隐含地使用的领域知识；这些是 AlphaGo Zero 需要取代这些部分以学习不同的游戏（交替马尔可夫）。

(1) AlphaGo Zero is provided with perfect knowledge of the game rules. These are used during MCTS, to simulate the positions resulting from a sequence of moves, and to score any simulations that reach a terminal state. Games terminate when both players pass or after $19 \times 19 \times 2 = 722$ moves. In addition, the player is provided with the set of legal moves in each position.

(2) AlphaGo Zero uses Tromp–Taylor scoring[66] during MCTS simulations and self-play training. This is because human scores (Chinese, Japanese or Korean rules) are not well-defined if the game terminates before territorial boundaries are resolved. However, all tournament and evaluation games were scored using Chinese rules.

(3) The input features describing the position are structured as a 19×19 image; that is, the neural network architecture is matched to the grid-structure of the board.

(4) The rules of Go are invariant under rotation and reflection; this knowledge has been used in AlphaGo Zero both

by augmenting the dataset during training to include rotations and reflections of each position, and to sample random rotations or reflections of the position during MCTS (see Search algorithm). Aside from *komi*, the rules of Go are also invariant to colour transposition; this knowledge is exploited by representing the board from the perspective of the current player (see Neural network architecture).

(1) AlphaGo Zero 被提供了完美的游戏规则知识。用于在 MCTS 期间模拟由一系列落子产生的棋盘，并对达到终端状态的任何模拟进行评分。当双方棋手通过或 $19\times19\times2 = 722$ 移动后，游戏终止。另外，棋手在每个棋盘被提供一组合乎规则的落子动作。

(2) AlphaGo Zero 在 MCTS 模拟和自我训练中使用 Tromp-Taylor 得分评判 [66]。这是因为如果游戏在领土边界得到解决之前终止，那么人类分数（中国，日本或韩国的规则）就不能被很好的确定。不过，所有锦标赛和评估对局都使用中国规则进行评分。

(3) 描述棋局的输入特征被构造为 19×19 图像；也就是说，神经网络结构与棋盘的网格结构相匹配。

(4) 围棋的规则在旋转和反射变换下是不变的;通过在训练期间增加包括每个位置的旋转和反射以及在 MCTS 期间随机旋转或反射棋盘以扩展数据库（参见搜索算法），AlphaGo Zero 中已经使用了这种知识。除了贴目之外，围棋的规则也是颜色转换不变的;这种知识是通过从当前棋手的角度来代表棋盘而研究的（参见神经网络架构）。

AlphaGo Zero does not use any form of domain knowledge beyond the points listed above. It only uses its deep neural network to evaluate leaf nodes and to select moves (see 'Search algorithm'). It does not use any rollout policy or tree policy, and the MCTS is not augmented by any other heuristics or domain-specific rules. No legal moves are excluded—even those filling in the player's own eyes (a standard heuristic used in all previous programs[67]).

除了上面列出的要点之外，AlphaGo Zero 不使用任何形式的领域知识。 它只使用其深层神经网络来评估叶节点并选择移动（参见'搜索算法'）。 它不使用任何快速走子策略或树策略，并且 MCTS 不会增加任何其他启发式或域特定规则。 除了不合规则的落子——即使是填充棋手自己眼的落子（以前所有程序中使用的标准启发式 [67]）。

The algorithm was started with random initial parameters for the neural network. The neural network architecture (see 'Neural network architecture') is based on the current state of the art in image recognition[4,18], and hyperparameters for training were chosen accordingly (see 'Self-play training pipeline'). MCTS search parameters were selected by Gaussian process optimization[68], so as to optimize self-play performance of AlphaGo Zero using a neural network trained in a preliminary run. For the larger run (40 blocks, 40 days), MCTS search parameters were re-optimized using the neural network trained in the smaller run (20 blocks, 3 days). The training algorithm was executed autonomously without human intervention.

该算法从神经网络的随机初始参数开始。 神经网络体系结构（参见'神经网络体系结构'）基于图像识别技术的当前状态 [4,18]，并据此选择用于训练的超参数（请参阅'自我训练工作流'）。 通过高斯过程优化选择 MCTS 搜索参数 [68]，以便使用在初步运行中训练的神经网络来优化 AlphaGo Zero 的自我对弈性能。 对于较大的运行（40 个块，40 天），使用在较小运行（20 个块，3 天）中训练的神经网络重新优化 MCTS 搜索参数。 训练算法是在没有人为干预的情况下自动执行的。

Self-play training pipeline. 自对弈训练工作流

AlphaGo Zero's self-play training pipeline consists of three main components, all executed asynchronously in parallel. Neural network parameters θ_i are continually optimized from recent self-play data; AlphaGo Zero players α_{θ_i} are continually evaluated; and the best performing player so far, α_{θ_*}, is used to generate new self-play data.

AlphaGo Zero 的自对弈训练工作流由三个主要模块构成，所有模块均异步并行执行。 神经网络参数θ_i从最近的自对弈数据持续优化; 持续对 AlphaGo Zero 棋手α_{θ_i}进行评估; 迄今为止表现最好的棋手 α_{θ_*}用于生成新的自对弈数据。

Optimization. 优化器

Each neural network f_{θ_i} is optimized on the Google Cloud using TensorFlow, with 64 GPU workers and 19 CPU parameter servers. The batch-size is 32 per worker, for a total mini-batch size of 2,048. Each mini-batch of data is sampled uniformly at random from all positions of the most recent 500,000 games of self-play. Neural network parameters are optimized by stochastic gradient descent with momentum and learning rate annealing, using the loss in equation (1). The learning rate is annealed according to the standard schedule in Extended Data Table 3. The momentum parameter is set to 0.9. The cross-entropy and MSE losses are weighted equally (this is reasonable because rewards are unit scaled, $r \in \{-1, +1\}$) and the L2 regularization parameter is set to $c = 10^{-4}$. The optimization process produces a new checkpoint every 1,000 training steps. This checkpoint is evaluated by the evaluator and it may be used for generating the next batch of self-play games, as we explain next.

每个神经网络f_{θ_i}在 Google Cloud 上使用 TensorFlow 进行了优化，包括 64 位 GPU 工作节点和 19 个 CPU 参数

服务器。总最小批（mini-batch）大小为 2048，每个工作节点的批次（Batch）大小为 32，每一个最小批从最近的 500,000 场自对弈棋谱的所有状态联合随机抽样。使用带有动量和学习速率退火的随机梯度下降对神经网络参数进行优化，损失函数见公式（1）。 学习速率根据扩展数据表 3 中的标准时间表进行退火。动量参数设置为 0.9。 交叉熵和 MSE 损失权重相等（这是合理的，因为奖励值被归一化到 $r \in \{-1, +1\}$），L2 正则化系数设为$c = 10^{-4}$。优化过程每 1,000 个训练步骤产生一个新的检查点。 正如我们接下来要解释的那样，该检查点由评估模块评估，并可用于生成下一批自对弈棋谱。

Evaluator. 评估器

To ensure we always generate the best quality data, we evaluate each new neural network checkpoint against the current best network f_{θ_*} before using it for data generation. The neural network f_{θ_i} is evaluated by the performance of an MCTS search α_{θ_i} that uses f_{θ_i} to evaluate leaf positions and prior probabilities (see Search algorithm). Each evaluation consists of 400 games, using an MCTS with 1,600 simulations to select each move, using an infinitesimal temperature $\tau \to 0$ (that is, we deterministically select the move with maximum visit count, to give the strongest possible play). If the new player wins by a margin of $> 55\%$ (to avoid selecting on noise alone) then it becomes the best player α_{θ_*}, and is subsequently used for self-play generation, and also becomes the baseline for subsequent comparisons.

为了确保我们始终生成最佳质量数据，我们在将每个新神经网络检查点用于数据生成之前，针对当前最佳网络f_{θ_*}进行评估。 神经网络f_{θ_i} 通过使用f_{θ_i}的 MCTS 得到的叶子位置和先验概率α_{θ_i}的性能来评估 (参见搜索算法)。每次评估包括 400 场对局，MCTS 使用 1,600 次模拟来选择每次移动，将温度参数设置为无限小$\tau \to 0$（也就是说，我们确定性地选择最大访问次数的移动，以追求最强的棋手）。 如果新棋手获胜率为 55%以上 (为了避免单独选择噪声)，那么它将成为最佳棋手α_{θ_*}，并随后用于生成自对弈棋盘，并且设为后续比较的基准。

Self-play. 自对弈模块

The best current player α_{θ_*}, as selected by the evaluator, is used to generate data. In each iteration, α_{θ_*}plays 25,000 games of self-play, using 1,600 simulations of MCTS to select each move (this requires approximately 0.4 s per search). For the first 30 moves of each game, the temperature is set to $\tau = 1$; this selects moves proportionally to their visit count in MCTS, and ensures a diverse set of positions are encountered. For the remainder of the game, an infinitesimal temperature is used, $\tau \to 0$. Additional exploration is achieved by adding Dirichlet noise to the prior probabilities in the root node s_0, specifically $P(s, a) = (1 - \varepsilon)p_a + \varepsilon\eta_a$, where $\boldsymbol{\eta} \sim \mathrm{Dir}(0.03)$ and $\varepsilon = 0.25$; this noise ensures that all moves may be tried, but the search may still overrule bad moves. In order to save computation, clearly lost games are resigned. The resignation threshold v_{resign} is selected automatically to keep the fraction of false positives (games that could have been won if AlphaGo had not resigned) below 5%. To measure false positives, we disable resignation in 10% of self-play games and play until termination.

由评估器选择的最佳当前棋手α_{θ_*}用于生成数据。在每次迭代中，α_{θ_*}可以自我对弈 25,000 局，使用 1,600 次 MCTS 模拟来选择每个移动（每次搜索需要大约需要 0.4 s)。对于每场比赛的前 30 步，温度参数设置为 1，$\tau = 1$；这会根据他们在 MCTS 中的访问次数按比例选择移动，并确保遇到不同棋局（子节点）的移动。对于游戏的其余部分，设置温度参数为无穷小，$\tau \to 0$。将 Dirichlet 噪声添加到根节点s_0中的先验概率，$P(s, a) = (1 - \varepsilon)p_a + \varepsilon\eta_a$，其中 $\boldsymbol{\eta} \sim \mathrm{Dir}(0.03)$，$\varepsilon = 0.25$；这样可以确保所有的棋局都可以被尝试，但是搜索仍然可以否决不好的动作。为了节省计算，明显失败棋局会被丢弃。投降阈值v_{resign}被自动选择以确保错误正类率（如果 AlphaGo 没有投降可能赢得的比赛的比例）低于 5%。为了衡量错误正类率，我们在 10%的自对弈中的禁止投降，也就是说必须下完。

Supervised learning.

For comparison, we also trained neural network parameters θ_{SL} by supervised learning. The neural network architecture was identical to AlphaGo Zero. Mini-batches of data $(s, \boldsymbol{\pi}, z)$ were sampled at random from the KGS dataset, setting $\boldsymbol{\pi}_a = 1$ for the human expert move a. Parameters were optimized by stochastic gradient descent with momentum and learning rate annealing, using the same loss as in equation (1), but weighting the MSE component by a factor of 0.01. The learning rate was annealed according to the standard schedule in Extended Data Table 3. The momentum parameter was set to 0.9, and the L2 regularization parameter was set to $c = 10^{-4}$.

为了进行对比，我们还通过监督学习来训练参数为θ_{SL}的神经网络，结构与 AlphaGo Zero 相同。从 KGS 数据集中随机采样小批数据$(s, \boldsymbol{\pi}, z)$，为人类的落子策略 a 设置 $\boldsymbol{\pi}_a = 1$。使用与方程（1）中相同的损失函数，使用带有动量和学习速率退火的随机梯度下降对参数进行优化，不过将 MSE 分量权重设为 0.01。 学习速率根据扩展数据表 3 中的标准时间表进行退火。动量参数设置为 0.9。 动量参数设置为 0.9，L2 正则化系数设为$c = 10^{-4}$。

总结

By using a combined policy and value network architecture, and by using a low weight on the value component, it was possible to avoid overfitting to the values (a problem described in previous work[12]). After 72 h the move prediction accuracy exceeded the state of the art reported in previous work[12,30–33], reaching 60.4% on the KGS test set; the value prediction error was also substantially better than previously reported[12]. The validation set was composed of professional games from GoKifu. Accuracies and MSEs are reported in Extended Data Table 1 and Extended Data Table

2, respectively.

通过使用组合的策略和价值网络体系结构，并对价值组件使用较低的权重，可以避免过拟合这些值（前面的工作 12 中描述的问题）。72 小时后，下棋预测精度超过了之前工作中的预测 [12,30-33]，在 KGS 测试集上达到了 60.4%；价值预测误差也大大优于先前的结果 [12]。 验证集由 GoKifu 的专业棋局组成。 精度和 MSE 分别在扩展数据表 1 和扩展数据表 2 中展示。

Search algorithm.

AlphaGo Zero uses a much simpler variant of the asynchronous policy and value MCTS algorithm (APV-MCTS) used in AlphaGo Fan and AlphaGo Lee.

AlphaGo Zero 使用了应用于 AlphaGo Fan 和 AlphaGo Lee 的异步策略价值 MCTS 算法（APVMCTS）更简单变体。

Each node s in the search tree contains edges (s, a) for all legal actions $a \in \mathcal{A}(s)$. Each edge stores a set of statistics,$\{N(s,a), W(s,a), Q(s,a), P(s,a)\}$, 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 of selecting that edge. Multiple simulations are executed in parallel on separate search threads. The algorithm proceeds by iterating over three phases (Fig. 2a–c), and then selects a move to play (Fig. 2d).

搜索树中的每个节点 s 的每条边(s, a)都满足于下棋规则 $a \in \mathcal{A}(s)$ 的边。每个边存储一组统计信息，$\{N(s,a), W(s,a), Q(s,a), P(s,a)\}$，其中$N(s,a)$是访问计数，$W(s,a)$是总动作值，$Q(s,a)$是平均动作值，$P(s,a)$是选择该边的先验概率。 多个模拟在单独的搜索线程上并行执行。 该算法通过迭代三个阶段（图 2a-c），然后选择一个棋子移动（图 2d）。

Select (Fig. 2a). 选择

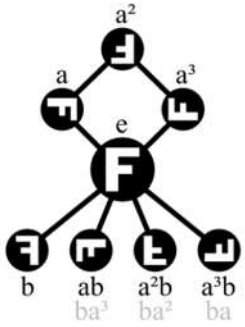
The selection phase is almost identical to AlphaGo Fan[12]; we recapitulate here for completeness. The first in-tree phase of each simulation begins at the root node of the search tree, s_0, and finishes when the simulation reaches a leaf node s_L at time-step L. At each of these time-steps, $t<L$, an action is selected according to the statistics in the search tree, $a_{t=\underset{a}{\operatorname{argmax}}(Q(s_t,a)+U(s_t,a))}$, using a variant of the PUCT algorithm[24],$U(s,a) = c_{puct}P(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$, where c_{puct} is a constant determining the level of exploration; this search control strategy initially prefers actions with high prior probability and low visit count, but asympotically prefers actions with high action value.

选择阶段几乎与 AlphaGo Fan[12] 相同；为了完整性,我们在这里重新叙述。 每个模拟的第一个扩展树阶段开始于搜索树的根节点s_0，并在时间步长 L 处，即到达叶节点s_L时结束。在每个时间步长 t(t <L)根据 PUCT 算法 [24] 的一个变种，计算搜索树中的统计概率。$a_{t=\underset{a}{\operatorname{argmax}}(Q(s_t,a)+U(s_t,a))}$，$U(s,a) = c_{puct}P(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$，$c_{puct}$是一个决定探索程度的常量；这种搜索控制策略最初倾向于具有高先验概率和低访问次数的落子，但是逐渐的会倾向于选择具有更高动作价值的落子。

expand and evaluate (Fig. 2b). 扩展和评估

The leaf node s_L is added to a queue for neural network evaluation,$(d_i(\boldsymbol{p}), v) = f_\theta(d_i(s_L))$, where d_i is a dihedral reflection or rotation selected uniformly at random from i in [1..8]. Positions in the queue are evaluated by the neural network using a mini-batch size of 8; the search thread is locked until evaluation completes. The leaf node is expanded and each edge (s_L, a) is initialized to$\{N(s_L,a) = 0, W(s_L,a) = 0, Q(s_L,a) = 0, P(s_L,a) = p_a\}$; the value v is then backed up.

将叶子节点 s_L 加到队列中等待输入至神经网络进行评估，$(d_i(\boldsymbol{p}), v) = f_\theta(d_i(s_L))$，其中 d_i 表示一个 1 至 8 的随机数来表示双方向镜面和旋转（译者注：从 8 个不同的方向进行评估，如下图所示，围棋棋型在很多情况如果从视觉角度来提取特征来说是同一个节点，极大的缩小了搜索空间）。 队列中的棋局由神经网络使用大小为 8 的小批量来评估；搜索线程将被锁定，直到评估完成。叶子节点被展开，每一条边(s_L, a)被初始化为$\{N(s_L,a) = 0, W(s_L,a) = 0, Q(s_L,a) = 0, P(s_L,a) = p_a\}$；然后将输出值 v 传回。



Backup (Fig. 2c).反向传播

The esdge statistics are updated in a backward pass through each step $t \leq L$. The visit counts are incremented, $N(s_t, a_t) = N(s_t, a_t) + 1$, and the action value is updated to the mean value, $W(s_t, a_t) = W(s_t, a_t) + v$, $Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)}$,We use virtual loss to ensure each thread evaluates different nodes[12,69].

在每个步骤 t≤L 后，边的统计信息反向传播更新（沿着扩展到叶子节点的路线回溯）。 访问次数递增，$N(s_t, a_t) = N(s_t, a_t) + 1$，并且行为价值更新为平均值$W(s_t, a_t) = W(s_t, a_t) + v$，$Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)}$（使用了神经网络的输出 v），我们使用虚拟损失来确保每个线程评估不同的节点 [12,69]。

Play (Fig. 2d). 落子

At the end of the search AlphaGo Zero selects a move a to play in the root position s_0, proportional to its exponentiated visit count, $\pi(a|s_0) = \frac{N(s_0,a)^{\frac{1}{\tau}}}{\sum_b N(s_0,b)}$, where τ is a temperature parameter that controls the level of exploration. The search tree is reused at subsequent time-steps: the child node corresponding to the played action becomes the new root node; the subtree below this child is retained along with all its statistics, while the remainder of the tree is discarded. AlphaGo Zero resigns if its root value and best child value are lower than a threshold value v_{resign}.

在搜索结束时，AlphaGo Zero 在棋盘s_0中选择落子动作 a，与访问次数成幂指数比例，$\pi(a|s_0) = \frac{N(s_0,a)^{\frac{1}{\tau}}}{\sum_b N(s_0,b)}$，其中其中 ττ 是一个温度常数用来控制探索等级（译者注：它是热力学玻尔兹曼分布的一种变形。温度较高的时候，分布更加均匀（走子多样性强）；温度降低的时候，分布更加尖锐（多样性弱，追求最强棋力））。 搜索树会在接下来的自对弈走子中复用，如果孩子节点和落子的位置吻合，它就成为新的根节点，保留子树的所有统计数据，同时丢弃其他的树。如果根的评价值和它最好孩子的评价值都低于v_{resign} AlphaGo Zero 就认输。

总结

Compared to the MCTS in AlphaGo Fan and AlphaGo Lee, the principal differences are that AlphaGo Zero does not use any rollouts; it uses a single neural network instead of separate policy and value networks; leaf nodes are always expanded, rather than using dynamic expansion; each search thread simply waits for the neural network evaluation, rather than performing evaluation and backup asynchronously; and there is no tree policy. A transposition table was also used in the large (40 blocks, 40 days) instance of AlphaGo Zero.

与 AlphaGo Fan 和 AlphaGo Lee 的 MCTS 相比，主要区别在于 AlphaGo Zero 不使用任何走子网络；它使用单个神经网络而不是单独的策略和价值网络；叶节点总是扩展，而不是使用动态扩展；每个搜索线程等待神经网络评估，而不是异步执行评估和反向传播；并没有使用树策略。 在 AlphaGo Zero 的大型实例中（使用 40 块残差神经网络，40 天训练时间）也使用了换位表。

Neural network architecture.

The input to the neural network is a 19 ×19 ×17 image stack comprising 17 binary feature planes. Eight feature planes, X_t, consist of binary values indicating the presence of the current player's stones ($X_t^i = 1$if intersection i contains a stone of the player's colour at time-step t; 0 if the intersection is empty, contains an opponent stone, or if t <0). A further 8 feature planes, Y_t, represent the corresponding features for the opponent's stones. The final feature plane, C, represents the colour to play, and has a constant value of either 1 if black is to play or 0 if white is to play. These planes are concatenated together to give input features $s_t = [X_t, Y_t, Y_{t-1}, X_{t-1}, \dots, X_{t-7}, Y_{t-7}, C]$. History features X_t, Y_t are necessary, because Go is not fully observable solely from the current stones, as repetitions are forbidden; similarly, the colour feature C is necessary, because the *komi* is not observable.

神经网络的输入是一个 19 ×19 ×17 的图像堆栈，包含 17 个二进制特征平面。八个特征平面X_t由二进制值组成，表示当前棋手的棋子的存在（如果交点 i 在时间步 t 为棋手颜色的棋子，则$X_t^i = 1$；如果交叉点没有棋子，或是对手棋子 t<0，则$X_t^i = 0$）。另外 8 个特征平面Y_t表示对手棋子的相应特征。最后的特征平面 C 表示要下棋的颜色，并且具有恒定的值，如果黑色要下棋，则为 1，如果白色下棋，则为 0。这些平面串联在一起，给出输入特征 $s_t = [X_t, Y_t, Y_{t-1}, X_{t-1}, \dots, X_{t-7}, Y_{t-7}, C]$。历史特征$X_t, Y_t$是必要的，因为落子过程不能完全从当前的棋子中观察到，而重复落子是被禁止的（避免循环劫）；同样，颜色特征 C 也是必要的，因为贴目是不可观察的（译者注：贴目，围棋术语。指黑方由于先手，在布局上占有一定的优势，为了公平起见，在最后计算双方所占地的多少时，黑棋必须扣减一定的目数或子数。）。

The input features s_t are processed by a residual tower that consists of a single convolutional block followed by either 19 or 39 residual blocks[4].

输入特征s_t由一个残差塔处理，该塔由单个卷积块组成，后面跟有 19 个或 39 个残余块 [4]。

The convolutional block applies the following modules:

(1) A convolution of 256 filters of kernel size 3 ×3 with stride 1
(2) Batch normalization[18]
(3) A rectifier nonlinearity

卷积模块应用以下模块：

(1) 由内核大小为 3×3，步幅为 1 的 256 个滤波器构成的卷积
(2) 批量正则化 [18]
(3) 整流器非线性

Each residual block applies the following modules sequentially to its input:

(1) A convolution of 256 filters of kernel size 3 ×3 with stride 1
(2) Batch normalization
(3) A rectifier nonlinearity
(4) A convolution of 256 filters of kernel size 3 ×3 with stride 1
(5) Batch normalization
(6) A skip connection that adds the input to the block
(7) A rectifier nonlinearity

每个残差块将下列模块顺序应用于其输入：

(1) 由内核大小为 3×3，步幅为 1 的 256 个滤波器构成的卷积
(2) 批量正则化
(3) 整流器非线性
(4) 由内核大小为 3×3，步幅为 1 的 256 个滤波器构成的卷积
(5) 批量正则化
(6) 跳转连接，在此增加一次输入
(7) 非线性整流器

The output of the residual tower is passed into two separate 'heads' for computing the policy and value.
The policy head applies the following modules:

(1) A convolution of 2 filters of kernel size 1 ×1 with stride 1
(2) Batch normalization
(3) A rectifier nonlinearity
(4) A fully connected linear layer that outputs a vector of size $19^2 + 1 = 362$,corresponding to logit probabilities for all intersections and the pass move .

The value head applies the following modules:

(1) convolution of 1 filter of kernel size 1 ×1 with stride 1
(2) Batch normalization
(3) A rectifier nonlinearity
(4) A fully connected linear layer to a hidden layer of size 256
(5) A rectifier nonlinearity
(6) A fully connected linear layer to a scalar
(7) A tanh nonlinearity outputting a scalar in the range $[-1, 1]$

输出的残差塔被分成两个独立的“头”，用于计算策略和价值。
策略头部应用以下模块：

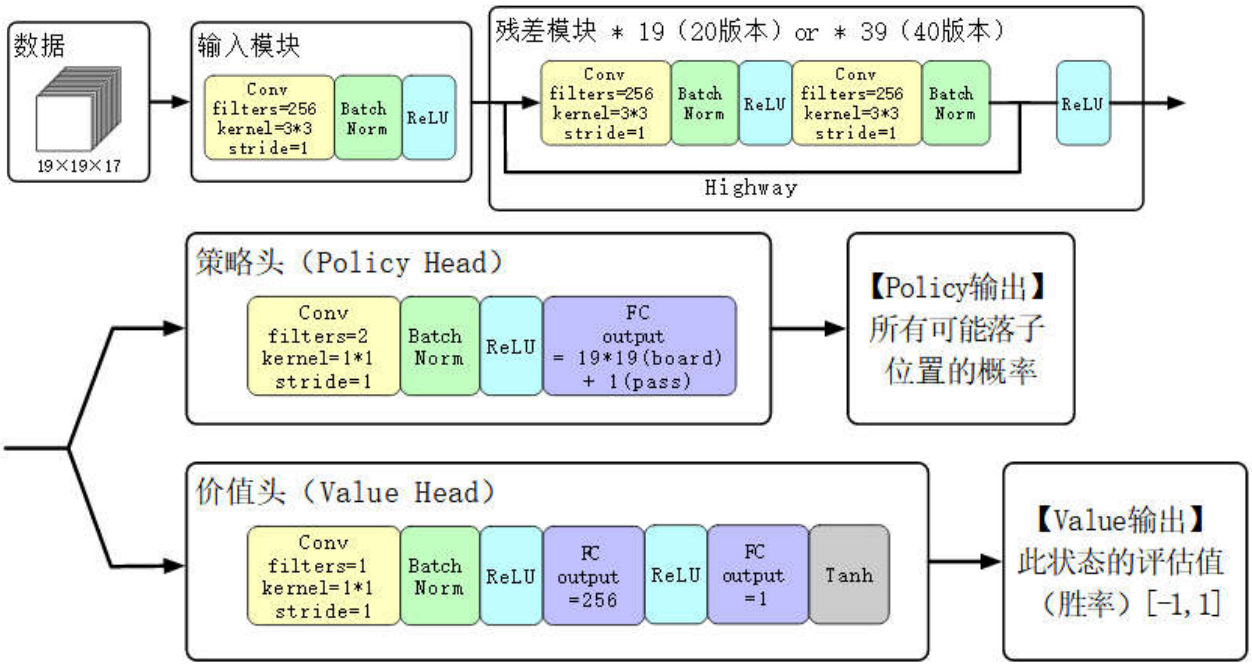
(1) 由内核大小为 1×1，步幅为 1 的 2 个过滤器组的卷积
(2) 批量正则化
(3) 整流器非线性
(4) 全连接层，输出为$19^2 + 1 = 362$的向量,对应于所有交点以及跳过的对数概率。

价值头部应用以下模块：

(1) 内核大小为 1×1 的 1 个滤波器与步幅 1 的卷积
(2) 批量正则化
(3) 非线性整流器
(4) 到大小为 256 的隐藏层的全连接层
(5) 非线性整流器
(6) 到标量的全连接层
(7) 一个 tanh 非线性函数输出范围在[- 1，1]内的标量

The overall network depth, in the 20- or 40-block network, is 39 or 79 parameterized layers, respectively, for the residual tower, plus an additional 2 layers for the policy head and 3 layers for the value head.

在 20 块或 40 块网络中，整体的网络深度分别为 39（19*2+1）或 79（20*2+1）个参数层，对于残差塔，分别另加 2 层作为策略头，3 层作为价值头。



We note that a different variant of residual networks was simultaneously applied to computer Go[33] and achieved an amateur dan-level performance; however, this was restricted to a single-headed policy network trained solely by supervised learning.

我们注意到残差网络的不同变体同时应用于围棋 [33] 计算机，并取得了业余级的性能；然而，这仅限于一个完全由监督学习训练的单一策略网络。

Neural network architecture comparison.

Figure 4 shows the results of a comparison between network architectures. Specifically, we compared four different neural networks:

(1) dual–res: the network contains a 20-block residual tower, as described above, followed by both a policy head and a value head. This is the architecture used in AlphaGo Zero.
(2) sep–res: the network contains two 20-block residual towers. The first tower is followed by a policy head and the second tower is followed by a value head.
(3) dual–conv: the network contains a non-residual tower of 12 convolutional blocks, followed by both a policy head and a value head.
(4) sep–conv: the network contains two non-residual towers of 12 convolutional blocks. The first tower is followed by a policy head and the second tower is followed by a value head. This is the architecture used in AlphaGo Lee.

图 4 显示了网络体系结构之间的比较结果。 具体地，我们比较了四种不同的神经网

(1) dual–res:：如上所述，网络包含一个 20 块残差塔，其后是策略头和价值头。 这是 AlphaGo Zero 中使用的架构。
(2) sep-res：网络包含两个 20 块残差塔。 第一个塔后面是策略头，第二个塔后面是价值头。
(3) dual-conv：网络包含一个由 12 个卷积块组成的残差塔，随后是策略头和价值头。
(4) sep-conv：网络包含由 12 个卷积块构成的两个残差塔。第一个塔后面是策略头，第二个塔后面是价值头。这是 AlphaGo Lee 使用的架构。

Each network was trained on a fixed dataset containing the final 2 million games of self-play data generated by a previous run of AlphaGo Zero, using stochastic gradient descent with the annealing rate, momentum and regularization hyperparameters described for the supervised learning experiment; however, cross-entropy and MSE components were weighted equally, since more data was available.

每个网络都是在一个固定的数据集上进行训练的，这个数据集包含了上述的 AlphaGo Zero 产生的最后 200 万场自我博弈对局数据，将带有动量和学习速率退火的随机梯度下降用于监督学习； 然而，由于可以获取到更多的数据，所以对 crossentropy 和 MSE 成分设置的权重是相等的。

Evaluation.

We evaluated the relative strength of AlphaGo Zero (Figs 3a, 6) by measuring the Elo rating of each player. We estimate the probability that player a will defeat player b by a logistic function $P\ (a\ \text{defeats}\ b) = \frac{1}{1+e^{c_{elo}(e(b)-e(a))}}$, and estimate the ratings $e(\cdot)$ by Bayesian logistic regression, computed by the BayesElo program[25] using the standard constant $c_{elo} = \frac{1}{400}$.

我们通过测量每个棋手的 Elo 等级来评估 AlphaGo Zero 的相对强度（图 3a，6）。 我们通过逻辑函数 P 估计棋手 a 击败棋手 b 的概率$P\ (a\ \text{defeats}\ b) = \frac{1}{1+e^{c_{elo}(e(b)-e(a))}}$，并且通过贝叶斯逻辑回归来估计等级 e (·)，由 BayesElo 程序 [25] 使用标准常数$c_{elo} = \frac{1}{400}$计算。

Elo ratings were computed from the results of a 5 s per move tournament between AlphaGo Zero, AlphaGo Master, AlphaGo Lee and AlphaGo Fan. The raw neural network from AlphaGo Zero was also included in the tournament. The Elo ratings of AlphaGo Fan, Crazy Stone, Pachi and GnuGo were anchored to the tournament values from previous work[12], and correspond to the players reported in that work. The results of the matches of AlphaGo Fan against Fan Hui and AlphaGo Lee against Lee Sedol were also included to ground the scale to human references, as otherwise the Elo ratings of AlphaGo are unrealistically high due to self-play bias.

Elo 评分是根据 AlphaGo Zero，AlphaGo Master，AlphaGo Lee 和 AlphaGo Fan 之间每场 5 秒的比赛结果计算出来的。 来自 AlphaGo Zero 的原始神经网络也包含在锦标赛中。 AlphaGo Fan，Crazy Stone，Pachi 和 GnuGo 的 Elo 评分是根据之前论文中的比赛成绩确定的 [12]，并与该论文中提出的棋手相对应。AlphaGo Fan 与 Fan Hui 和 AlphaGo Lee 对 Lee Sedol 的比赛结果也包括在内，以便将比例基准提供给人类参考，否则 AlphaGo 的 Elo 评分由于自我偏见而显得不切实际。

The Elo ratings in Figs 3a, 4a, 6a were computed from the results of evaluation games between each iteration of player α_{θ_i} during self-play training. Further evaluations were also performed against baseline players with Elo ratings anchored to the previously published values[12].

图 3a，4a，6a 中的 Elo 等级是根据在自我对弈训练期间每个迭代棋手α_{θ_i} 之间的对局评估的结果计算的。 进一步的评估也针对基准棋手进行，Elo 评级与以前公布的数值相关 [12]。

We measured the head-to-head performance of AlphaGo Zero against AlphaGo Lee, and the 40-block instance of AlphaGo Zero against AlphaGo Master, using the same player and match conditions that were used against Lee Sedol in Seoul, 2016. Each player received 2 h of thinking time plus 3 byoyomi periods of 60 s per move. All games were scored using Chinese rules with a *komi* of 7.5 points.

我们测量了 AlphaGo Zero 对阵 AlphaGo Lee，以及由四十层块构成的 AlphaGo Zero 对阵 AlphaGo Master 的王牌对王牌表现，使用了与 2016 年首尔对阵 Lee Sedol 相同的棋手和比赛条件。每名棋手都获得了 2 小时的思考时间 加上每次移动 60 秒的 3 次读秒周期。 所有的比赛都使用中国规则进行打分，贴目输为 7.5。

Data availability.

The datasets used for validation and testing are the GoKifu dataset (available from http://gokifu.com/) and the KGS dataset (available from https://u-go.net/gamerecords/).

Barto, A. G. & Duff, M. Monte Carlo matrix inversion and reinforcement learning. *Adv. Neural Inf. Process. Syst.* **6,** 687–694 (1994).

Singh, S. P. & Sutton, R. S. Reinforcement learning with replacing eligibility traces. *Mach. Learn.* **22,** 123–158 (1996).

Lagoudakis, M. G. & Parr, R. Reinforcement learning as classification: leveraging modern classifiers. In *Proc. 20th Int. Conf. Mach. Learn.* 424–431 (2003).

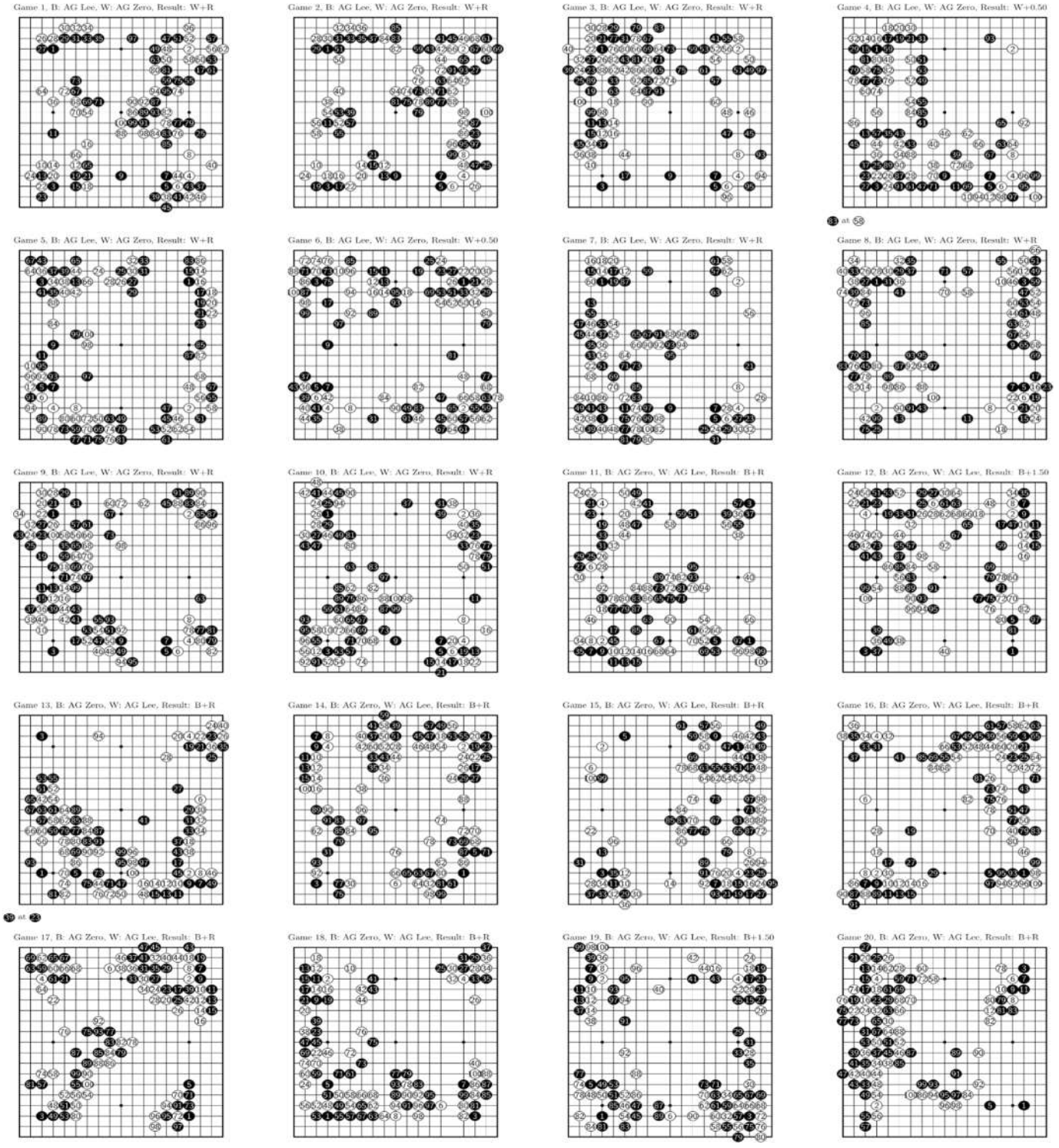
Scherrer, B., Ghavamzadeh, M., Gabillon, V., Lesner, B. & Geist, M. Approximate modified policy iteration and its application to the game of Tetris. *J. Mach. Learn. Res.* **16,** 1629–1676 (2015).

Littman, M. L. Markov games as a framework for multi-agent reinforcement learning. In *Proc. 11th Int. Conf. Mach. Learn.* 157–163 (1994).

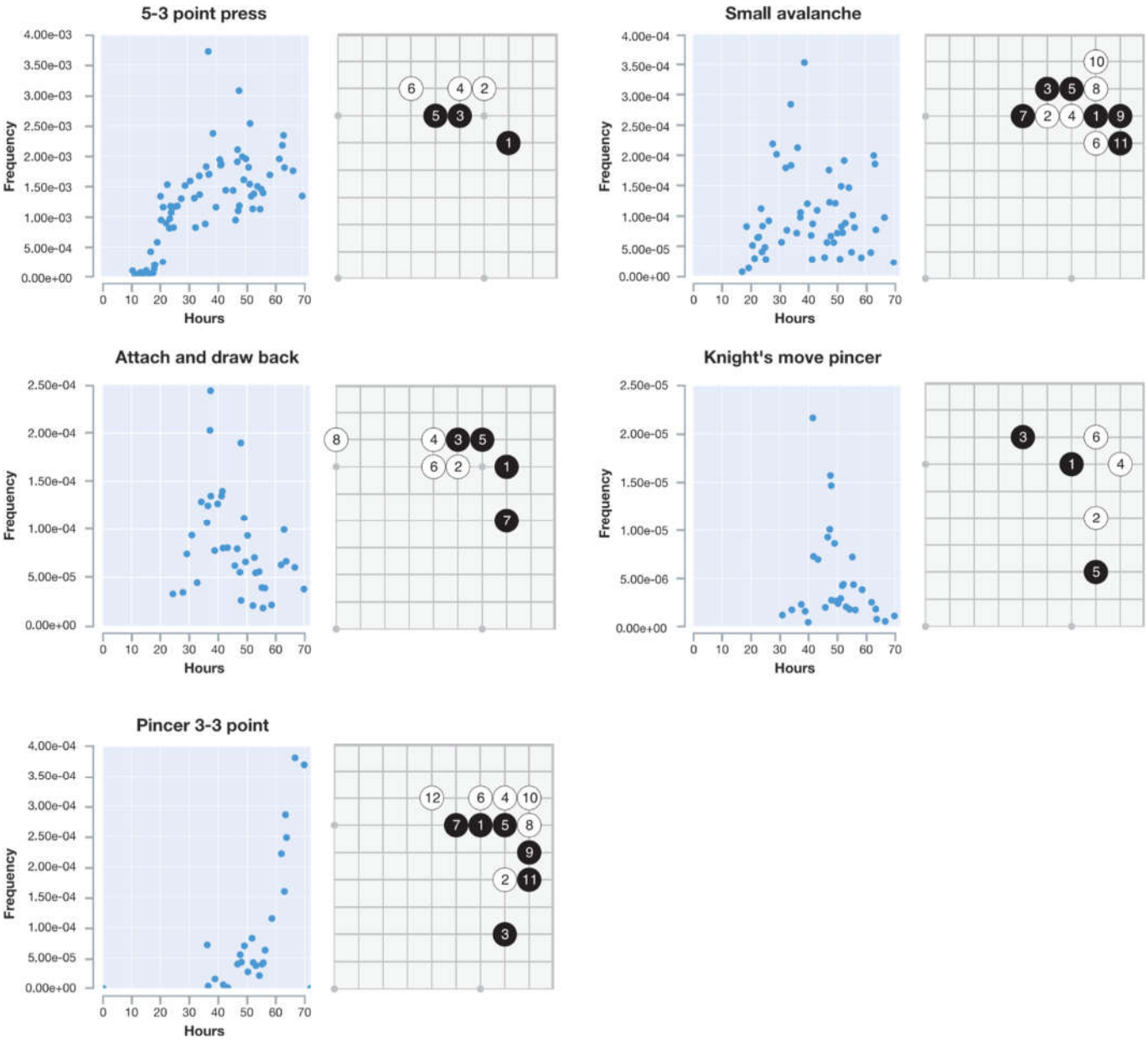
Enzenberger, M. The integration of a priori knowledge into a Go playing neural network. http://www.cgl.ucsf.edu/go/Programs/neurogo-html/neurogo.html (1996).

Enzenberger, M. in *Advances in Computer Games* (eds Van Den Herik, H. J., Iida, H. & Heinz, E. A.) 97–108 (2003).

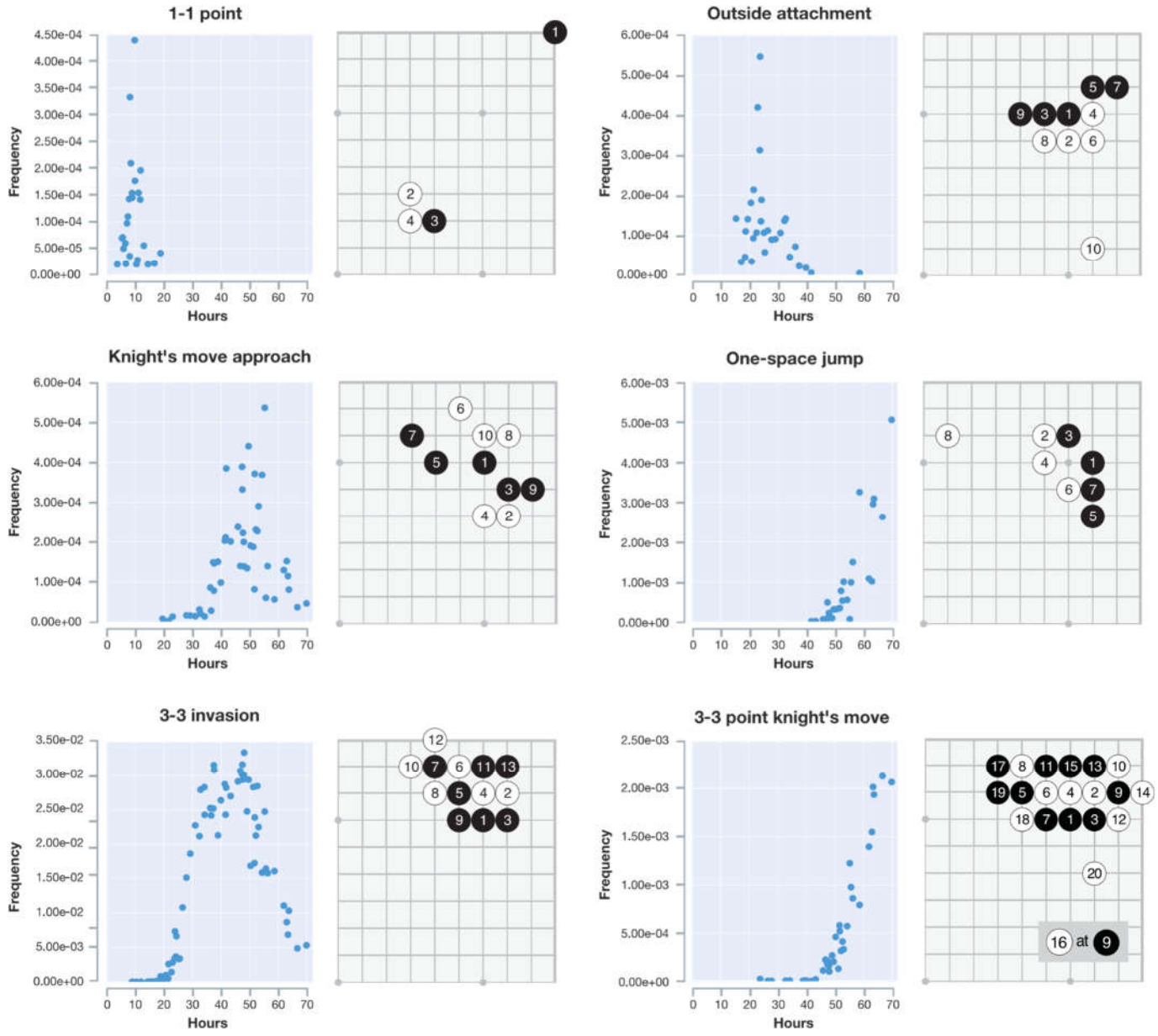
Sutton, R. Learning to predict by the method of temporal differences. *Mach. Learn.* **3,** 9–44 (1988).
Schraudolph, N. N., Dayan, P. & Sejnowski, T. J. Temporal difference learning of position evaluation in the game of Go. *Adv. Neural Inf. Process. Syst.* **6,** 817–824 (1994).
Silver, D., Sutton, R. & Müller, M. Temporal-difference search in computer Go. *Mach. Learn.* **87,** 183–219 (2012).
Silver, D. Reinforcement Learning and Simulation-Based Search in Computer Go. PhD thesis, Univ. Alberta, Edmonton, Canada (2009).
Gelly, S. & Silver, D. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artif. Intell.* **175,** 1856–1875 (2011).
Coulom, R. Computing Elo ratings of move patterns in the game of Go. *Int. Comput. Games Assoc. J.* **30,** 198–208 (2007).
Gelly, S., Wang, Y., Munos, R. & Teytaud, O. Modification of UCT with patterns in Monte-Carlo Go. Report No. 6062 (INRIA, 2006).
Baxter, J., Tridgell, A. & Weaver, L. Learning to play chess using temporal differences. *Mach. Learn.* **40,** 243–263 (2000).
Veness, J., Silver, D., Blair, A. & Uther, W. Bootstrapping from game tree search. In *Adv. Neural Inf. Process. Syst.* 1937–1945 (2009).
Lai, M. Giraffe: Using Deep Reinforcement Learning to Play Chess. MSc thesis, Imperial College London (2015).
Schaeffer, J., Hlynka, M. & Jussila, V. Temporal difference learning applied to a high-performance game-playing program. In *Proc. 17th Int. Jt Conf. Artif. Intell.* Vol. 1 529–534 (2001).
Tesauro, G. TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput.* **6,** 215–219 (1994).
Buro, M. From simple features to sophisticated evaluation functions. In *Proc. 1st Int. Conf. Comput. Games* 126–145 (1999).
Sheppard, B. World-championship-caliber Scrabble. *Artif. Intell.* **134,** 241–275 (2002).
Moravcˇík, M. *et al.* DeepStack: expert-level artificial intelligence in heads-up no-limit poker. *Science* **356,** 508–513 (2017).
Tesauro, G & Galperin, G. On-line policy improvement using Monte-Carlo search. In *Adv. Neural Inf. Process. Syst.* 1068–1074 (1996).
Tesauro, G. Neurogammon: a neural-network backgammon program. In *Proc. Int. Jt Conf. Neural Netw.* Vol. 3, 33–39 (1990).
Samuel, A. L. Some studies in machine learning using the game of checkers II - recent progress. *IBM J. Res. Develop.* **11,** 601–617 (1967).
Kober, J., Bagnell, J. A. & Peters, J. Reinforcement learning in robotics: a survey. *Int. J. Robot. Res.* **32,** 1238–1274 (2013).
Zhang, W. & Dietterich, T. G. A reinforcement learning approach to job-shop scheduling. In *Proc. 14th Int. Jt Conf. Artif. Intell.* 1114–1120 (1995).
Cazenave, T., Balbo, F. & Pinson, S. Using a Monte-Carlo approach for bus regulation. In *Int. IEEE Conf. Intell. Transport. Syst.* 1–6 (2009).
Evans, R. & Gao, J. Deepmind AI reduces Google data centre cooling bill by 40%. https://deepmind.com/blog/deepmind-ai-reduces-google-data-centrecooling-bill-40/ (2016).
Abe, N. *et al.* Empirical comparison of various reinforcement learning strategies for sequential targeted marketing. In *IEEE Int. Conf. Data Mining* 3–10 (2002).
Silver, D., Newnham, L., Barker, D., Weller, S. & McFall, J. Concurrent reinforcement learning from customer interactions. In *Proc. 30th Int. Conf. Mach. Learn.* Vol. 28 924–932 (2013).
Tromp, J. Tromp–Taylor rules. http://tromp.github.io/go.html (1995).
Müller, M. Computer Go. *Artif. Intell.* **134,** 145–179 (2002).
Shahriari, B., Swersky, K., Wang, Z., Adams, R. P. & de Freitas, N. Taking the human out of the loop: a review of Bayesian optimization. *Proc. IEEE* **104,** 148–175 (2016).
Segal, R. B. On the scalability of parallel UCT. *Comput. Games* **6515,** 36–47 (2011).



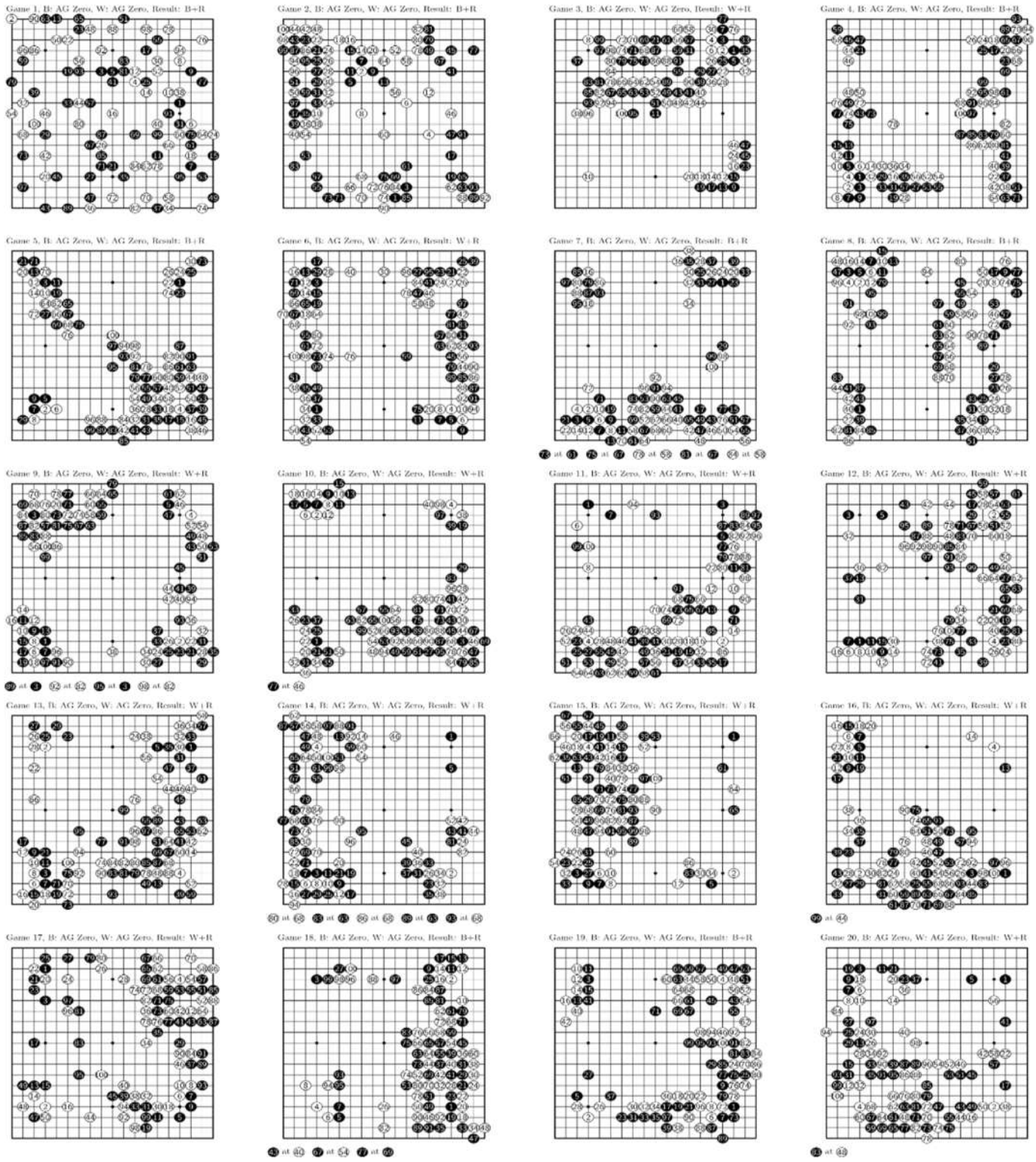
Extended Data Figure 1 | Tournament games between AlphaGo Zero (20 blocks, 3 days) versus AlphaGo Lee using 2 h time controls. One hundred moves of the first 20 games are shown; full games are provided in the Supplementary Information.



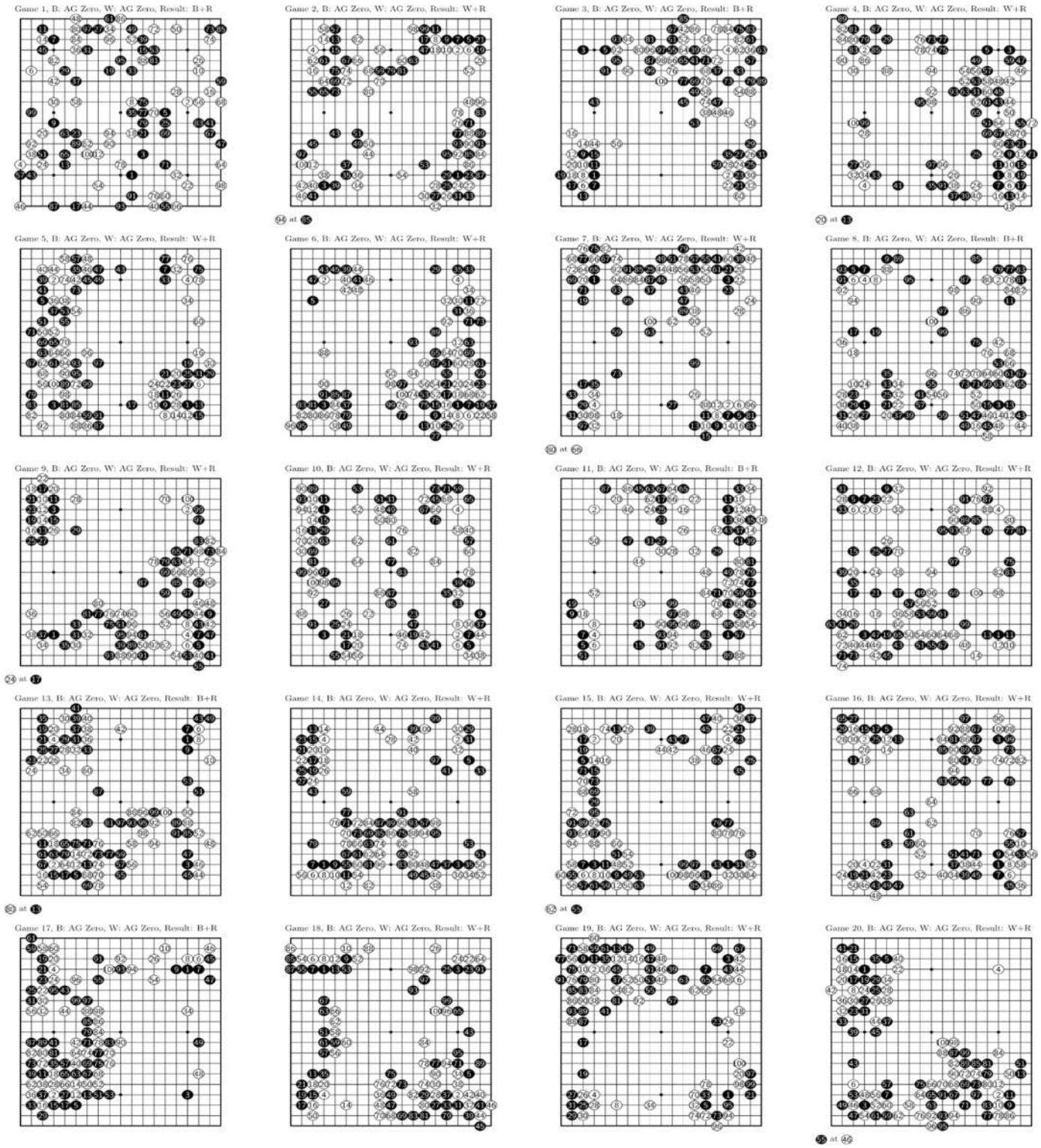
Extended Data Figure 2 | Frequency of occurence over time during training, for each *joseki* from Fig. 5a (corner sequences common in professional play that were discovered by AlphaGo Zero). The corresponding *joseki* are shown on the right.



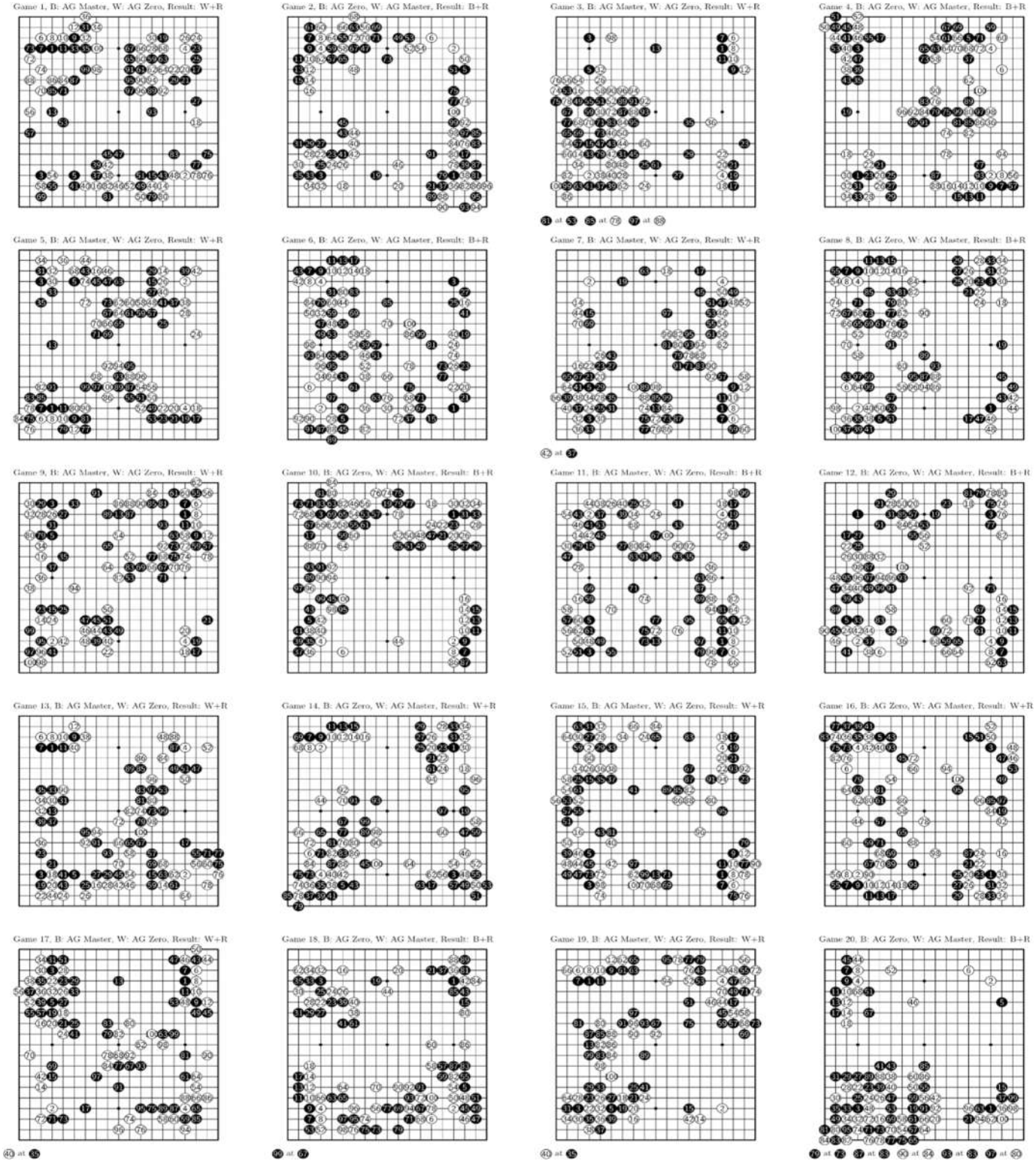
Extended Data Figure 3 | Frequency of occurence over time during training, for each *joseki* from Fig. 5b (corner sequences that AlphaGo Zero favoured for at least one iteration), and one additional variation. The corresponding *joseki* are shown on the right.



Extended Data Figure 4 | AlphaGo Zero (20 blocks) self-play games. The 3-day training run was subdivided into 20 periods. The best player from each period (as selected by the evaluator) played a single game against itself, with 2 h time controls. One hundred moves are shown for each game; full games are provided in the Supplementary Information.



Extended Data Figure 5 | AlphaGo Zero (40 blocks) self-play games. The 40-day training run was subdivided into 20 periods. The best player from each period (as selected by the evaluator) played a single game against itself, with 2 h time controls. One hundred moves are shown for each game; full games are provided in the Supplementary Information.



Extended Data Figure 6 | AlphaGo Zero (40 blocks, 40 days) versus AlphaGo Master tournament games using 2 h time controls. One hundred moves of the first 20 games are shown; full games are provided in the Supplementary Information.

extended data table 1 | Move prediction accuracy

	KGS train	*KGS* test	*GoKifu* validation
Supervised learning (20 block)	62.0	60.4	54.3
Supervised learning (12 layer [12])	59.1	55.9	-
Reinforcement learning (20 block)	-	-	49.0
Reinforcement learning (40 block)	-	-	51.3

Percentage accuracies of move prediction for neural networks trained by reinforcement learning (that is, AlphaGo Zero) or supervised learning. For supervised learning, the network was trained for 3 days on KGS data (amateur games); comparative results are also shown from ref. 12. For reinforcement learning, the 20-block network was trained for 3 days and the 40-block network was trained for 40 days. Networks were also evaluated on a validation set based on professional games from the GoKifu dataset.

extended data table 2 | Game outcome prediction error

	KGS train	*KGS* test	*GoKifu* validation
Supervised learning (20 block)	0.177	0.185	0.207
Supervised learning (12 layer [12])	0.19	0.37	-
Reinforcement learning (20 block)	-	-	0.177
Reinforcement learning (40 block)	-	-	0.180

Mean squared error on game outcome predictions for neural networks trained by reinforcement learning (that is, AlphaGo Zero) or supervised learning. For supervised learning, the network was trained for 3 days on KGS data (amateur games); comparative results are also shown from ref. 12. For reinforcement learning, the 20 block network was trained for 3 days and the 40 block network was trained for 40 days. Networks were also evaluated on a validation set based on professional games from the GoKifu dataset.

extended data table 3 | Learning rate schedule

Thousands of steps	Reinforcement learning	Supervised learning
0–200	10^{-2}	10^{-1}
200–400	10^{-2}	10^{-2}
400–600	10^{-3}	10^{-3}
600–700	10^{-4}	10^{-4}
700–800	10^{-4}	10^{-5}
>800	10^{-4}	-

Learning rate used during reinforcement learning and supervised learning experiments, measured in thousands of steps (mini-batch updates).

参考：

https://charlesliuyx.github.io/2017/10/18/%E6%B7%B1%E5%85%A5%E6%B5%85%E5%87%BA%E7%9C%8B%E6%87%82AlphaGo%E5%85%83/ 深入浅出看懂 AlphaGo 元