Mastering the Game of Go with Deep Neural Networks and Tree Search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1], Demis Hassabis[1].

[1] Google DeepMind, 5 New Street Square, London EC4A 3TW.
[2] Google, 1600 Amphitheatre Parkway, Mountain View CA 94043.

*These authors contributed equally to this work.

Correspondence should be addressed to either David Silver (davidsilver@google.com) or Demis Hassabis (demishassabis@google.com).

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence due to its enormous search space and the difficulty of evaluating board positions and moves. We introduce a new approach to computer Go that uses *value networks* to evaluate board positions and *policy networks* to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte-Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte-Carlo simulation with value and policy networks. Using this search algorithm, our program *AlphaGo* achieved a 99.8% winning rate against other Go programs, and defeated the European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an *optimal value function*, $v^*(s)$, which determines the outcome of the game, from every board position or *state* s, under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game's breadth (number

运用深度神经网络与树搜索技术精通围棋

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1],Arthur Guez[1],Laurent Sifre[1],George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1],Veda Panneershelvam[1],Marc Lanctot[1], Sander Dieleman[1], 多米尼克·格雷韦[1], 约翰·南[2],纳尔·卡尔奇布伦纳[1],伊利亚·苏茨克维尔[2],蒂莫西·利利克拉普[1], 玛德琳·利奇[1], 科拉伊·卡武克楚奥卢[1],托尔·格雷佩尔[1],德米斯·哈萨比斯[1]。

[1] 谷歌DeepMind，伦敦EC4A 3TW新街广场5号。[2] 谷歌，加利福尼亚州山景城圆形剧场大道1600号，邮编94043。

*这些作者对本工作贡献相同。

相关事宜请联系David Silver（davidsilver@google.com）或Demis Hassabis（demishassabis@google.com）。

围棋因其海量搜索空间及棋局评估难度，长期被视为人工智能领域最具挑战性的经典博弈。本文提出新型计算机围棋方案：采用价值网络评估棋局状态，策略网络选择行棋方案。这些深度神经网络通过创新性结合两类训练实现：基于人类专家对局的监督学习，以及基于自对弈的强化学习。 在完全不预判的情况下，该神经网络展现出媲美顶尖蒙特卡罗树搜索程序的水平——后者需模拟数千局随机自对弈。我们还创新性地将蒙特卡罗模拟与价值网络、策略网络相结合，构建出新型搜索算法。 运用该搜索算法，我们的AlphaGo程序在与其他围棋程序的对弈中取得99.8%胜率，并以5比0的比分击败欧洲围棋冠军。这是计算机程序首次在标准围棋对局中战胜人类职业棋手，此前人们认为这一成就至少需要十年才能实现。

所有完全信息博弈都存在最优价值函数 $v^*(s)$，该函数在所有玩家完美博弈的情况下，从任意棋局状态 s中决定游戏结果。此类博弈可通过递归计算最优价值函数来求解，该计算在包含约 b^d 种可能走法序列的搜索树中进行，其中 b 为博弈的广度（走法数量）。

of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35, d \approx 80$) [1] and especially Go ($b \approx 250, d \approx 150$) [1], exhaustive search is infeasible [2,3], but the effective search space can be reduced by two general principles. First, the depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s. This approach has led to super-human performance in chess [4], checkers [5] and othello [6], but it was believed to be intractable in Go due to the complexity of the game [7]. Second, the breadth of the search may be reduced by sampling actions from a *policy* $p(a|s)$ that is a probability distribution over possible moves a in position s. For example, *Monte-Carlo rollouts* [8] search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p. Averaging over such rollouts can provide an effective position evaluation, achieving super-human performance in backgammon [8] and Scrabble [9], and weak amateur level play in Go [10].

Monte-Carlo tree search (MCTS) [11,12] uses Monte-Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant values become more accurate. The policy used to select actions during search is also improved over time, by selecting children with higher values. Asymptotically, this policy converges to optimal play, and the evaluations converge to the optimal value function [12]. The strongest current Go programs are based on MCTS, enhanced by policies that are trained to predict human expert moves [13]. These policies are used to narrow the search to a beam of high probability actions, and to sample actions during rollouts. This approach has achieved strong amateur play [13–15]. However, prior work has been limited to shallow policies [13–15] or value functions [16] based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains: for example image classification [17], face recognition [18], and playing Atari games [19]. They use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localised representations of an image [20]. We employ a similar architecture for the game of Go. We pass in the board position as a 19×19 image and use convolutional layers

每局棋局中合法走法数量)和 d 是其深度（游戏长度）。在大型棋类游戏中，如国际象棋（$b \approx 35, d \approx 80$） [1]尤其是围棋（$b \approx 250, d \approx 150$）1，穷举搜索不可行 2,3，但可通过两条通用原则有效缩减搜索空间。 首先，通过局势评估可缩减搜索深度：在状态 s 处截断搜索树，并将 s 以下子树替换为近似价值函数 $v(s) \approx v^*(s)$，该函数可预测从状态 s开始的局势结果。 该方法已在国际象棋[4]、跳棋 [5] 和黑白棋[6]中实现超越人类的性能，但因围棋的复杂性 [7]，人们曾认为其在围棋领域不可行。 其次，可通过从策略 $p(a|s)$中采样动作来缩减搜索广度，该策略是对位置 s中可能走法 a 的概率分布。例如蒙特卡洛展开 [8] 在完全不分支的情况下进行最大深度搜索，通过从策略 p中为双方采样长动作序列。 对这类展开进行平均可获得有效的局面评估，在双陆棋 [8] 和拼字游戏9中能达到超越人类的表现，而在围棋中则仅能达到业余弱手水平 [10]。

蒙特卡洛树搜索（MCTS） [11,12] 通过蒙特卡洛展开法估算搜索树中每个状态的价值。随着模拟次数增加，搜索树不断扩展，相关价值评估也日益精确。搜索过程中选择动作的策略会持续优化——通过优先选择价值更高的子节点。 渐近地，该策略将收敛至最优玩法，评估值亦趋近最优价值函数 [12]。当前最强围棋程序基于MCTS构建，并通过训练策略预测人类专家棋步 [13]加以强化。这些策略用于将搜索范围缩小至高概率行动束，并在展开过程中采样行动。 该方法已实现业余高手水平的对弈能力 [13–15]。然而，先前研究仅限于基于输入特征线性组合的浅层策略 [13–15] 或价值函数 [16] 。

近期，深度卷积神经网络在视觉领域取得突破性进展：例如图像分类[17]、人脸识别[18]及Atari游戏 [19]。该架构通过多层神经元（以重叠瓦片状排列）构建图像的抽象局部表征 [20]。我们采用类似架构应用于围棋领域。 我们将棋盘状态作为图像输入，并运用卷积层进行处理

to construct a representation of the position. We use these neural networks to reduce the effective depth and breadth of the search tree: evaluating positions using a *value network*, and sampling actions using a *policy network*.

We train the neural networks using a pipeline consisting of several stages of machine learning (Figure 1). We begin by training a supervised learning (SL) policy network, p_σ, directly from expert human moves. This provides fast, efficient learning updates with immediate feedback and high quality gradients. Similar to prior work [13,15], we also train a fast policy p_π that can rapidly sample actions during rollouts. Next, we train a reinforcement learning (RL) policy network, p_ρ, that improves the SL policy network by optimising the final outcome of games of self-play. This adjusts the policy towards the correct goal of winning games, rather than maximizing predictive accuracy. Finally, we train a value network v_θ that predicts the winner of games played by the RL policy network against itself. Our program *AlphaGo* efficiently combines the policy and value networks with MCTS.

1 Supervised Learning of Policy Networks

For the first stage of the training pipeline, we build on prior work on predicting expert moves in the game of Go using supervised learning[13,21–24]. The SL policy network $p_\sigma(a|s)$ alternates between convolutional layers with weights σ, and rectifier non-linearities. A final softmax layer outputs a probability distribution over all legal moves a. The input s to the policy network is a simple representation of the board state (see Extended Data Table 2). The policy network is trained on randomly sampled state-action pairs (s, a), using stochastic gradient ascent to maximize the likelihood of the human move a selected in state s,

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}. \tag{1}$$

We trained a 13 layer policy network, which we call the *SL policy network*, from 30 million positions from the KGS Go Server. The network predicted expert moves with an accuracy of

构建棋局状态的表示模型。我们利用这些神经网络来降低搜索树的实际深度和广度：通过价值网络评估棋局状态，并借助策略网络采样行动方案。

我们通过包含多个机器学习阶段的管道训练神经网络（图1）。首先直接基于人类专家棋步训练监督学习策略网络 p_σ，该方法可提供即时反馈与高质量梯度，实现快速高效的学习更新。参照先前的研究13,15，我们同时训练快速策略网络 p_π，使其能在推演过程中快速采样动作。 随后，我们训练强化学习策略网络 p_ρ，通过优化自对弈的最终结果来改进监督策略网络。这将策略导向正确的目标——赢得比赛，而非最大化预测准确率。最后，我们训练价值网络 v_θ，用于预测强化策略网络自对弈的胜负结果。我们的AlphaGo程序通过蒙特卡洛树搜索（MCTS）高效整合了策略网络与价值网络。

1 策略网络的监督学习

在训练管道的第一阶段，我们基于先前利用监督学习预测围棋专家棋步的研究成果[13,21–24]。策略网络 $p_\sigma(a|s)$交替采用具有权重 σ的卷积层与整流非线性函数。最终的softmax层输出所有合法棋步 a的概率分布。 策略网络的输入 s 是对棋盘状态的简化表示（详见扩展数据表2）。该网络通过随机采样的状态-动作对(s, a)进行训练，采用随机梯度上升法最大化人类在状态 s中选择动作 a 的概率。

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}. \tag{1}$$

我们从KGS围棋服务器中选取3000万个棋局位置，训练出13层策略网络（命名为SL策略网络）。该网络预测专家棋步的准确率达到

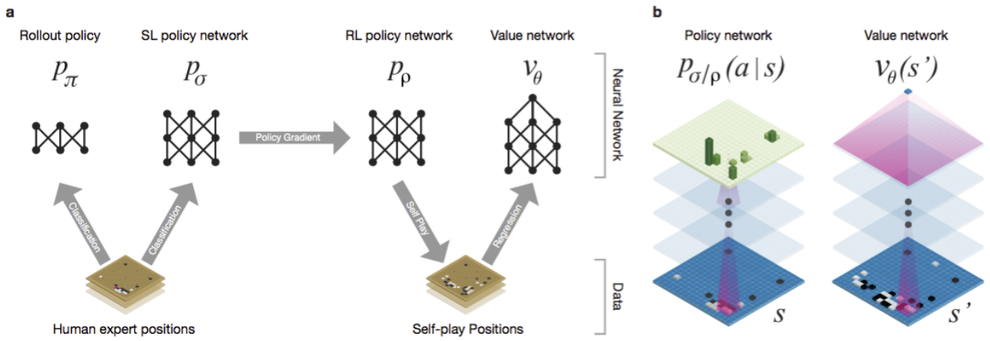


Figure 1: **Neural network training pipeline and architecture. a** A fast rollout policy p_π and supervised learning (SL) policy network p_σ are trained to predict human expert moves in a data-set of positions. A reinforcement learning (RL) policy network p_ρ is initialised to the SL policy network, and is then improved by policy gradient learning to maximize the outcome (i.e. winning more games) against previous versions of the policy network. A new data-set is generated by playing games of self-play with the RL policy network. Finally, a value network v_θ is trained by regression to predict the expected outcome (i.e. whether the current player wins) in positions from the self-play data-set. **b** Schematic representation of the neural network architecture used in *AlphaGo*. The policy network takes a representation of the board position s as its input, passes it through many convolutional layers with parameters σ (SL policy network) or ρ (RL policy network), and outputs a probability distribution $p_\sigma(a|s)$ or $p_\rho(a|s)$ over legal moves a, represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters θ, but outputs a scalar value $v_\theta(s')$ that predicts the expected outcome in position s'.

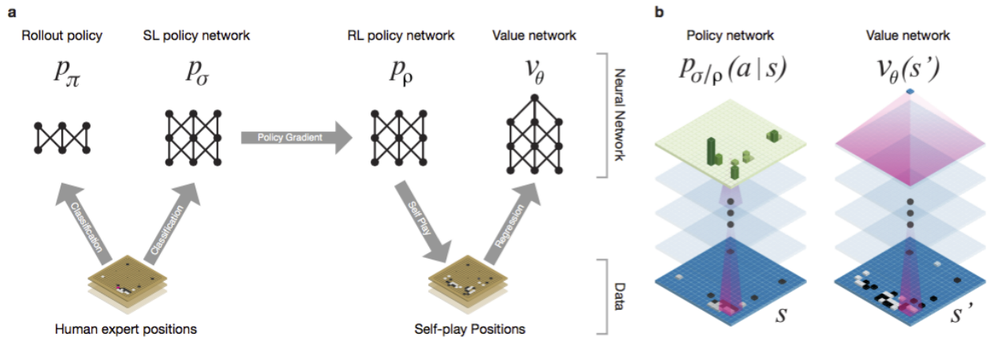


图1：神经网络训练流程与架构。a 快速展开策略 p_π 及su-

监督学习（SL）策略网络 p_σ 通过训练数据集中的棋局预测人类专家棋步。强化学习（RL）策略网络 p_ρ 以SL策略网络为初始化基础，通过策略梯度学习持续优化，以最大化相较于策略网络前代版本的胜率（即赢得更多对局）。 通过RL策略网络进行自我对弈生成新数据集。最后，采用回归训练价值网络 v_θ，使其能预测自我对弈数据集中棋局的预期结果（即当前玩家胜负概率）。b AlphaGo所用神经网络架构示意图。 策略网络以棋局状态表示 s 为输入，经由参数集 σ （SL策略网络）或 ρ （RL策略网络）的多个卷积层处理，输出覆盖合法走法 a的概率分布 $p_\sigma(a|s)$或 $p_\rho(a|s)$，该分布以棋盘概率图形式呈现。 价值网络同样采用多个卷积层（参数 θ），但输出标量值 $v_\theta(s')$，用于预测当前棋局 s'的预期结果。

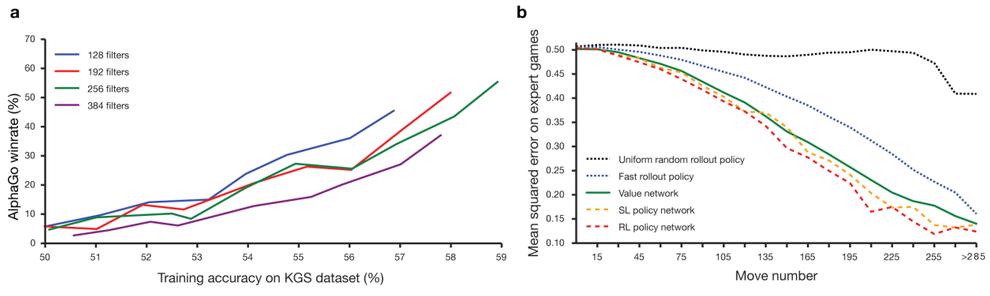


Figure 2: **Strength and accuracy of policy and value networks.** **a** Plot showing the playing strength of policy networks as a function of their training accuracy. Policy networks with 128, 192, 256 and 384 convolutional filters per layer were evaluated periodically during training; the plot shows the winning rate of *AlphaGo* using that policy network against the match version of *AlphaGo*. **b** Comparison of evaluation accuracy between the value network and rollouts with different policies. Positions and outcomes were sampled from human expert games. Each position was evaluated by a single forward pass of the value network v_θ, or by the mean outcome of 100 rollouts, played out using either uniform random rollouts, the fast rollout policy p_π, the SL policy network p_σ or the RL policy network p_ρ. The mean squared error between the predicted value and the actual game outcome is plotted against the stage of the game (how many moves had been played in the given position).

57.0% on a held out test set, using all input features, and 55.7% using only raw board position and move history as inputs, compared to the state-of-the-art from other research groups of 44.4% at date of submission [24] (full results in Extended Data Table 3). Small improvements in accuracy led to large improvements in playing strength (Figure 2,a); larger networks achieve better accuracy but are slower to evaluate during search. We also trained a faster but less accurate rollout policy $p_\pi(a|s)$, using a linear softmax of small pattern features (see Extended Data Table 4) with weights π; this achieved an accuracy of 24.2%, using just $2\,\mu s$ to select an action, rather than $3\,ms$ for the policy network.

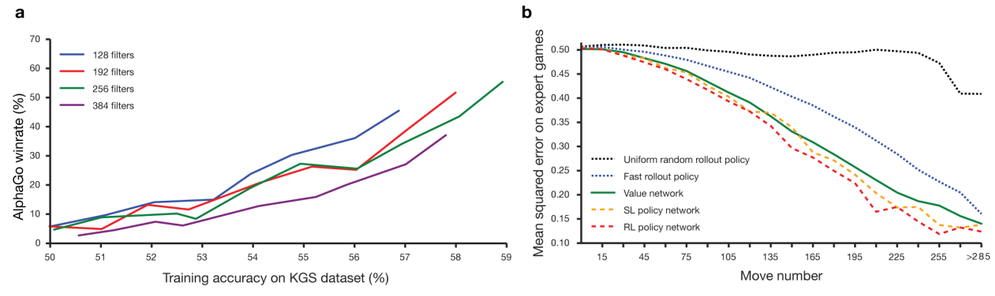


图2：策略网络与价值网络的强度与准确性。a图展示了对弈过程

政策网络的强度与其训练精度呈函数关系。训练过程中定期评估了每层分别采用128、192、256及384个卷积滤波器的政策网络；图示展示了采用该政策网络的AlphaGo对抗比赛版AlphaGo时的胜率。b 价值网络与不同政策下的推演结果在评估精度上的对比。 棋局位置与结果均采样自人类专家对局。每个棋局位置通过价值网络 v_θ单次前向传播评估，或通过100次展开的平均结果评估——展开方式包括均匀随机展开、快速展开策略 p_π、SL策略网络 p_σ 或RL策略网络 p_ρ。 图中绘制了预测值与实际赛果之间的均方误差，并按比赛阶段（即特定位置已进行的步数）进行标注。

在保留测试集上，使用全部输入特征时准确率达57.0%，仅使用原始棋盘位置和行棋历史时为55.7%，而其他研究团队在提交时点的最先进水平仅为44.4% [24] （完整结果见扩展数据表3）。微小的准确率提升带来显著的棋力增强（图2a）； 更大规模网络虽能提升准确率，但在搜索过程中评估速度较慢。我们还训练了更快速但准确率稍低的展开策略$p_\pi(a|s)$，该策略采用小规模模式特征的线性软最大化（详见扩展数据表4）及权重π；其准确率达24.2%，动作选择仅需2微秒，远低于策略网络所需的3毫秒。

2 Reinforcement Learning of Policy Networks

The second stage of the training pipeline aims at improving the policy network by policy gradient reinforcement learning (RL) [25,26]. The RL policy network p_ρ is identical in structure to the SL policy network, and its weights ρ are initialised to the same values, $\rho = \sigma$. We play games between the current policy network p_ρ and a randomly selected previous iteration of the policy network. Randomising from a pool of opponents stabilises training by preventing overfitting to the current policy. We use a reward function $r(s)$ that is zero for all non-terminal time-steps $t < T$. The *outcome* $z_t = \pm r(s_T)$ is the terminal reward at the end of the game from the perspective of the current player at time-step t: $+1$ for winning and -1 for losing. Weights are then updated at each time-step t by stochastic gradient ascent in the direction that maximizes expected outcome [25],

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t \,. \tag{2}$$

We evaluated the performance of the RL policy network in game play, sampling each move $a_t \sim p_\rho(\cdot|s_t)$ from its output probability distribution over actions. When played head-to-head, the RL policy network won more than 80% of games against the SL policy network. We also tested against the strongest open-source Go program, *Pachi* [14], a sophisticated Monte-Carlo search program, ranked at 2 amateur *dan* on KGS, that executes 100,000 simulations per move. Using no search at all, the RL policy network won 85% of games against *Pachi*. In comparison, the previous state-of-the-art, based only on supervised learning of convolutional networks, won 11% of games against *Pachi* [23] and 12% against a slightly weaker program *Fuego* [24].

3 Reinforcement Learning of Value Networks

The final stage of the training pipeline focuses on position evaluation, estimating a value function $v^p(s)$ that predicts the outcome from position s of games played by using policy p for both players [27–29],

$$v^p(s) = \mathbb{E}\left[z_t \mid s_t = s, a_{t\ldots T} \sim p\right] . \tag{3}$$

2 策略网络的强化学习

训练管道的第二阶段旨在通过策略梯度强化学习（RL） [25,26]改进策略网络。RL策略网络 p_ρ 与SL策略网络结构完全相同，其权重 ρ 初始化为相同值 $\rho = \sigma$。我们让当前策略网络 p_ρ 与随机选取的策略网络历史迭代版本进行对弈。 通过从对手池中随机抽取，可避免对当前策略的过拟合从而稳定训练过程。我们采用的奖励函数 $r(s)$对所有非终止时间步 $t < T$取值为零。 结果 $z_t = \pm r(s_T)$是当前玩家在时间步 t结束时获得的终局奖励：获胜奖励 $+1$，失败奖励 -1。随后在每个时间步 t 通过随机梯度上升更新权重，方向为最大化预期结果 [25]。

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t \,. \tag{2}$$

我们在实战中评估了RL策略网络的性能，通过其动作概率分布采样每步棋$a_t \sim p_\rho(\cdot|s_t)$。在对抗测试中，RL策略网络对阵SL策略网络时胜率超过80%。 我们还测试了最强开源围棋程序Pachi 14——这款复杂的蒙特卡罗搜索程序在KGS平台排名业余二段，每步执行10万次模拟。 在完全不使用搜索的情况下，RL策略网络对战Pachi的胜率达85%。相比之下，仅基于卷积神经网络监督学习的先前最先进方法，对战Pachi的胜率仅为11% [23]，对战稍弱程序Fuego [24]时胜率为12%。

3 价值网络的强化学习

训练管道的最终阶段聚焦于局势评估，通过计算价值函数$v^p(s)$来预测基于局势 s 的对局结果——该函数基于双方采用策略 p 进行的27–[29]局对弈数据进行预测。

$$v^p(s) = \mathbb{E}\left[z_t \mid s_t = s, a_{t\ldots T} \sim p\right] . \tag{3}$$

Ideally, we would like to know the optimal value function under perfect play $v^*(s)$; in practice, we instead estimate the value function v^{p_ρ} for our strongest policy, using the RL policy network p_ρ. We approximate the value function using a *value network* $v_\theta(s)$ with weights θ, $v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$. This neural network has a similar architecture to the policy network, but outputs a single prediction instead of a probability distribution. We train the weights of the value network by regression on state-outcome pairs (s, z), using stochastic gradient descent to minimize the mean squared error (MSE) between the predicted value $v_\theta(s)$, and the corresponding outcome z,

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial \theta}(z - v_\theta(s))\,. \tag{4}$$

The naive approach of predicting game outcomes from data consisting of complete games leads to overfitting. The problem is that successive positions are strongly correlated, differing by just one stone, but the regression target is shared for the entire game. When trained on the KGS dataset in this way, the value network memorised the game outcomes rather than generalising to new positions, achieving a minimum MSE of 0.37 on the test set, compared to 0.19 on the training set. To mitigate this problem, we generated a new self-play data-set consisting of 30 million distinct positions, each sampled from a separate game. Each game was played between the RL policy network and itself until the game terminated. Training on this data-set led to MSEs of 0.226 and 0.234 on the training and test set, indicating minimal overfitting. Figure 2,b shows the position evaluation accuracy of the value network, compared to Monte-Carlo rollouts using the fast rollout policy p_π; the value function was consistently more accurate. A single evaluation of $v_\theta(s)$ also approached the accuracy of Monte-Carlo rollouts using the RL policy network p_ρ, but using 15,000 times less computation.

4 Searching with Policy and Value Networks

AlphaGo combines the policy and value networks in an MCTS algorithm (Figure 3) that selects actions by lookahead search. Each edge (s, a) of the search tree stores an *action value* $Q(s, a)$, *visit*

理想情况下，我们希望在完美对弈下求得最优价值函数 $v^*(s)$；实际中则通过强化学习策略网络 p_ρ，对最强策略的价值函数 v^{p_ρ} 进行估计。我们通过价值网络 $v_\theta(s)$及其权重 θ，$v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$来近似价值函数。该神经网络架构与策略网络相似，但输出单一预测值而非概率分布。我们通过状态-结果对(s, z)的回归训练价值网络权重，采用随机梯度下降法最小化预测值 $v_\theta(s)$与对应结果z之间的均方误差(MSE)。

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial \theta}(z - v_\theta(s))\,. \tag{4}$$

基于完整对局数据预测赛果的简单方法会导致过拟合。问题在于连续棋局位置高度相关——仅差一子之隔，但回归目标却需覆盖整盘棋局。采用KGS数据集进行此类训练时，价值网络会记忆棋局结果而非泛化至新局面，测试集均方误差最低值达0.37（训练集为0.19）。为解决此问题，我们生成包含3000万个独立局面的自对弈数据集，每个局面均来自独立对局。每局对弈均由RL策略网络与自身反复博弈直至结束。基于该数据集训练后，训练集与测试集的均方误差分别降至0.226和0.234，表明过拟合现象显著减少。图2b对比了价值网络与基于快速展开策略 p_π的蒙特卡罗展开评估精度，价值函数始终展现出更高准确性。单次评估 $v_\theta(s)$的精度亦接近基于RL策略网络 p_ρ的蒙特卡罗展开，但计算量仅需后者的1/15000。

4 策略网络与价值网络的搜索机制

AlphaGo通过MCTS算法（图3）整合策略网络与价值网络，通过前瞻搜索选择行动。搜索树中每条边（s, a）存储行动值 $Q(s, a)$，访问

count $N(s,a)$, and *prior probability* $P(s,a)$. The tree is traversed by simulation (i.e. descending the tree in complete games without backup), starting from the root state. At each time-step t of each simulation, an action a_t is selected from state s_t,

$$a_t = \underset{a}{\operatorname{argmax}} \left(Q(s_t,a) + u(s_t,a)\right), \tag{5}$$

so as to maximize action value plus a bonus $u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$ that is proportional to the prior probability but decays with repeated visits to encourage exploration. When the traversal reaches a leaf node s_L at step L, the leaf node may be expanded. The leaf position s_L is processed just once by the SL policy network p_σ. The output probabilities are stored as prior probabilities P for each legal action a, $P(s,a) = p_\sigma(a|s)$. The leaf node is evaluated in two very different ways: first, by the value network $v_\theta(s_L)$; and second, by the outcome z_L of a random rollout played out until terminal step T using the fast rollout policy p_π; these evaluations are combined, using a mixing parameter λ, into a leaf evaluation $V(s_L)$,

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L\,. \tag{6}$$

At the end of simulation n, the action values and visit counts of all traversed edges are updated. Each edge accumulates the visit count and mean evaluation of all simulations passing through that edge,

$$N(s,a) = \sum_{i=1}^{n} \mathbf{1}(s,a,i) \tag{7}$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} \mathbf{1}(s,a,i) V(s_L^i)\,, \tag{8}$$

where s_L^i is the leaf node from the ith simulation, and $\mathbf{1}(s,a,i)$ indicates whether an edge (s,a) was traversed during the ith simulation. Once the search is complete, the algorithm chooses the most visited move from the root position.

The SL policy network p_σ performed better in *AlphaGo* than the stronger RL policy network p_ρ, presumably because humans select a diverse beam of promising moves, whereas RL optimizes

计数 $N(s,a)$ 及先验概率 $P(s,a)$。通过模拟（即在无备份的完整对局中沿树向下遍历）从根节点开始探索树结构。每次模拟的 t 时间步中，从状态 s_t 选择动作 a_t，

$$a_t = \underset{a}{\operatorname{argmax}} \left(Q(s_t,a) + u(s_t,a)\right), \tag{5}$$

以最大化行动价值并附加奖励项 $u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$——该奖励项与先验概率成正比，但会随重复访问次数衰减以激励探索行为。当遍历在步骤 L到达叶节点 s_L 时，该叶节点可被展开。叶节点位置 s_L 仅由SL策略网络 p_σ处理一次。输出概率将作为每个合法动作 a，$P(s,a) = p_\sigma(a|s)$的先验概率 P 存储。叶节点通过两种截然不同的方式评估：首先由价值网络 $v_\theta(s_L)$评估；其次通过快速展开策略 p_π进行随机展开，直至终止步 T 所得结果 z_L；这两种评估通过混合参数 λ组合为叶节点评估 $V(s_L)$，

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L\,. \tag{6}$$

在 n模拟结束时，所有遍历边的动作值与访问计数均被更新。每条边累积了所有通过该边的模拟的访问计数与平均评估值，

$$N(s,a) = \sum_{i=1}^{n} \mathbf{1}(s,a,i) \tag{7}$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} \mathbf{1}(s,a,i) V(s_L^i)\,, \tag{8}$$

其中s_L^i 表示第 i次模拟的叶节点，$\mathbf{1}(s,a,i)$表示第 i次模拟中是否遍历了边(s,a)。搜索完成后，算法从根节点位置选择访问次数最多的走法。

在AlphaGo系统中，强化学习策略网络 p_σ 的表现优于更强大的随机强化学习策略网络p_ρ，这可能是因为人类能筛选出多样化的潜在优选棋步，而随机强化学习则侧重优化

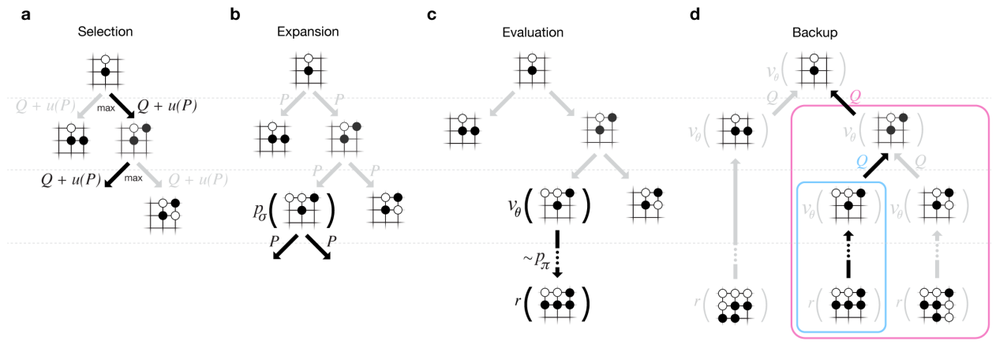


Figure 3: **Monte-Carlo tree search in *AlphaGo*. a** Each simulation traverses the tree by selecting the edge with maximum action-value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b** The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c** At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d** Action-values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

for the single best move. However, the value function $v_\theta(s) \approx v^{p_\rho}(s)$ derived from the stronger RL policy network performed better in *AlphaGo* than a value function $v_\theta(s) \approx v^{p_\sigma}(s)$ derived from the SL policy network.

Evaluating policy and value networks requires several orders of magnitude more computation than traditional search heuristics. To efficiently combine MCTS with deep neural networks, *AlphaGo* uses an asynchronous multi-threaded search that executes simulations on CPUs, and computes policy and value networks in parallel on GPUs. The final version of *AlphaGo* used 40 search threads, 48 CPUs, and 8 GPUs. We also implemented a distributed version of *AlphaGo* that exploited multiple machines, 40 search threads, 1202 CPUs and 176 GPUs. The Methods section provides full details of asynchronous and distributed MCTS.

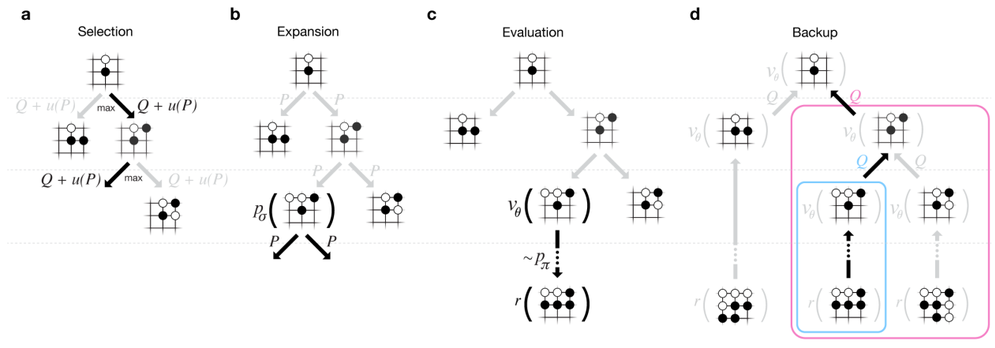


图3：AlphaGo中的蒙特卡洛树搜索。a 每次模拟通过选择具有最大动作值 Q的边进行树遍历，并附加取决于该边存储先验概率 P 的奖励 $u(P)$。 b 叶节点可能被展开；新节点由策略网络 p_σ 处理一次，输出概率作为每个动作的先验概率 P 存储。 c 模拟结束时，叶节点通过两种方式评估：使用价值网络 v_θ；以及采用快速展开策略 p_π将局面展开至终局，再通过函数 r计算胜者。 d 动作价值 Q 通过追踪该动作子树下所有评估 $r(\cdot)$ 和 $v_\theta(\cdot)$ 的均值进行更新。

在寻找单步最佳棋局时，AlphaGo采用强化学习策略网络推导出的价值函数 $v_\theta(s) \approx v^{p_\rho}(s)$ 表现优于基于弱化学习策略网络推导的 $v_\theta(s) \approx v^{p_\sigma}(s)$。

策略网络与价值网络的评估所需计算量远超传统搜索启发式算法。为高效融合蒙特卡洛树搜索（MCTS）与深度神经网络，AlphaGo采用异步多线程搜索架构：在CPU上执行模拟计算，同时在GPU上并行处理策略与价值网络。最终版本的AlphaGo配置了40个搜索线程、48个CPU核心及8个GPU核心。 我们还实现了AlphaGo的分布式版本，利用多台机器、40个搜索线程、1202个CPU和176个GPU协同工作。具体实现细节详见方法论章节中关于异步与分布式MCTS的完整说明。

5 Evaluating the Playing Strength of *AlphaGo*

To evaluate *AlphaGo*, we ran an internal tournament among variants of *AlphaGo* and several other Go programs, including the strongest commercial programs *Crazy Stone* [13] and *Zen*, and the strongest open source programs *Pachi* [14] and *Fuego* [15]. All of these programs are based on high-performance MCTS algorithms. In addition, we included the open source program *GnuGo*, a Go program using state-of-the-art search methods that preceded MCTS. All programs were allowed 5 seconds of computation time per move.

The results of the tournament (see Figure 4,a) suggest that single machine *AlphaGo* is many *dan* ranks stronger than any previous Go program, winning 494 out of 495 games (99.8%) against other Go programs. To provide a greater challenge to *AlphaGo*, we also played games with 4 handicap stones (i.e. free moves for the opponent); *AlphaGo* won 77%, 86%, and 99% of handicap games against *Crazy Stone*, *Zen* and *Pachi* respectively. The distributed version of *AlphaGo* was significantly stronger, winning 77% of games against single machine *AlphaGo* and 100% of its games against other programs.

We also assessed variants of *AlphaGo* that evaluated positions using just the value network ($\lambda = 0$) or just rollouts ($\lambda = 1$) (see Figure 4,b). Even without rollouts *AlphaGo* exceeded the performance of all other Go programs, demonstrating that value networks provide a viable alternative to Monte-Carlo evaluation in Go. However, the mixed evaluation ($\lambda = 0.5$) performed best, winning $\geq 95\%$ against other variants. This suggests that the two position evaluation mechanisms are complementary: the value network approximates the outcome of games played by the strong but impractically slow p_ρ, while the rollouts can precisely score and evaluate the outcome of games played by the weaker but faster rollout policy p_π. Figure 5 visualises *AlphaGo*'s evaluation of a real game position.

Finally, we evaluated the distributed version of *AlphaGo* against Fan Hui, a professional 2 *dan*, and the winner of the 2013, 2014 and 2015 European Go championships. On 5–9th October

5 评估AlphaGo的棋力水平

为评估AlphaGo，我们组织了内部对抗赛，参赛方包括AlphaGo的多个变体版本及其他围棋程序，涵盖最强商业程序Crazy Stone [13] 和Zen，以及最强开源程序Pachi [14] 和Fuego [15]。所有程序均基于高性能MCTS算法。 此外，我们还纳入了开源程序GnuGo——该围棋程序采用先于MCTS的尖端搜索方法。所有程序每步棋均被允许5秒的计算时间。

赛事结果（见图4a）表明，单台AlphaGo的棋力远超以往任何围棋程序，在与其他围棋程序的495局对弈中取得494胜（胜率99.8%）。 为提升挑战难度，我们采用让子棋（即对手免费落子）模式与AlphaGo对弈：在与Crazy Stone、Zen和Pachi的让子棋中，AlphaGo分别取得77%、86%和99%的胜率。分布式版本的AlphaGo表现更为强劲，在与单机AlphaGo的对战中胜率达77%，对其他程序则保持100%胜率。

我们还评估了仅使用价值网络（$\lambda = 0$）或仅使用推演（$\lambda = 1$）进行局势评估的AlphaGo变体（见图4b）。 即使不使用推演，AlphaGo的性能仍超越所有其他围棋程序，证明价值网络可作为围棋领域中蒙特卡罗评估的可行替代方案。然而混合评估（$\lambda = 0.5$）表现最佳，对其他变体胜率达 $\geq 95\%$。 这表明两种局面评估机制具有互补性：价值网络近似模拟了强大但运行速度过慢的 p_ρ策略的对局结果，而展开策略能精确评估较弱但运行更快的 p_π策略的对局结果。图5可视化呈现了AlphaGo对真实棋局的评估过程。

最终，我们让AlphaGo的分布式版本与职业二段棋手范谟展开对弈——范谟曾于2013、2014及2015年三度夺得欧洲围棋锦标赛冠军。对局于10月5日至9日举行。

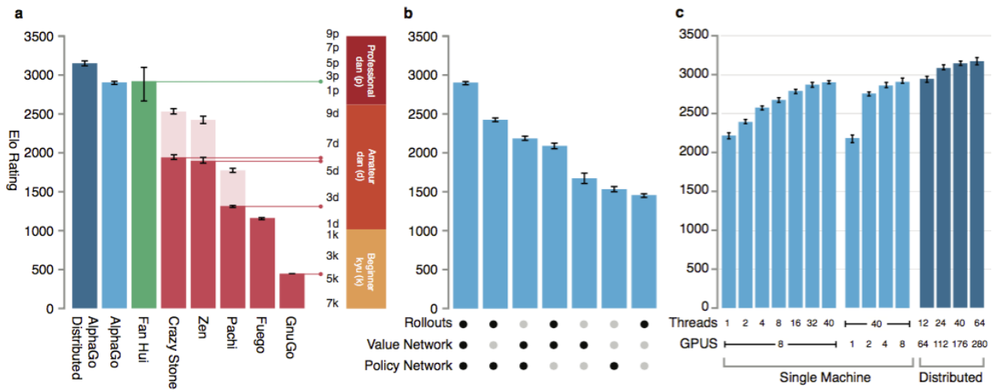


Figure 4: **Tournament evaluation of *AlphaGo*. a** Results of a tournament between different Go programs (see Extended Data Tables 6 to 11). Each program used approximately 5 seconds computation time per move. To provide a greater challenge to *AlphaGo*, some programs (pale upper bars) were given 4 handicap stones (i.e. free moves at the start of every game) against all opponents. Programs were evaluated on an *Elo* scale [30]: a 230 point gap corresponds to a 79% probability of winning, which roughly corresponds to one amateur *dan* rank advantage on KGS [31]; an approximate correspondence to human ranks is also shown, horizontal lines show KGS ranks achieved online by that program. Games against the human European champion Fan Hui were also included; these games used longer time controls. 95% confidence intervals are shown. **b** Performance of *AlphaGo*, on a single machine, for different combinations of components. The version solely using the policy network does not perform any search. **c** Scalability study of Monte-Carlo tree search in *AlphaGo* with search threads and GPUs, using asynchronous search (light blue) or distributed search (dark blue), for 2 seconds per move.

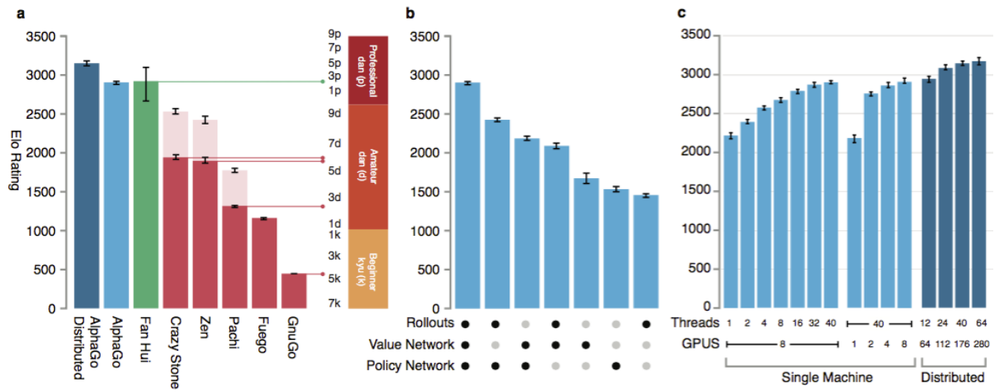


图4：AlphaGo赛事评估结果。a 不同围棋程序的赛事对弈结果（详见扩展数据表6至11）。各程序每步棋耗时约5秒。为增强对AlphaGo的挑战性，部分程序（浅色上端柱状图）在对阵所有对手时获得4子让子（即每局开局免费行棋）。 程序采用埃洛等级分 [30]评估：230分差距对应79%胜率，约等同于KGS平台业余段位优势 [31]；图中同时标注了近似对应的人类段位，横线表示该程序在线上KGS平台达到的段位。图中还包含与欧洲人机围棋冠军范谟的对弈记录，这些对局采用更长的用时限制。 图中显示95%置信区间。b 单台机器上AlphaGo不同组件组合的性能表现。仅使用策略网络的版本不进行任何搜索。c AlphaGo中蒙特卡洛树搜索的可扩展性研究，采用搜索线程和GPU，分别使用异步搜索（浅蓝）或分布式搜索（深蓝），每步耗时2秒。

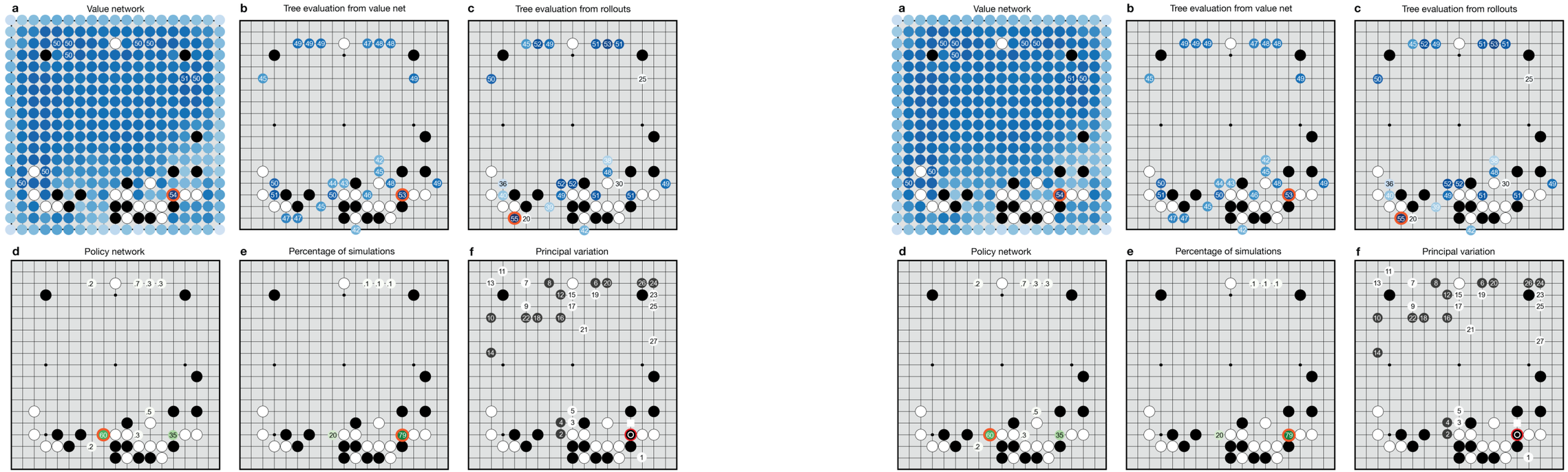


Figure 5: **How *AlphaGo* (black, to play) selected its move in an informal game against Fan Hui.** For each of the following statistics, the location of the maximum value is indicated by an orange circle. **a** Evaluation of all successors s' of the root position s, using the value network $v_\theta(s')$; estimated winning percentages are shown for the top evaluations. **b** Action-values $Q(s,a)$ for each edge (s,a) in the tree from root position s; averaged over value network evaluations only ($\lambda = 0$). **c** Action-values $Q(s,a)$, averaged over rollout evaluations only ($\lambda = 1$). **d** Move probabilities directly from the SL policy network, $p_\sigma(a|s)$; reported as a percentage (if above 0.1%). **e** Percentage frequency with which actions were selected from the root during simulations. **f** The principal variation (path with maximum visit count) from *AlphaGo*'s search tree. The moves are presented in a numbered sequence. *AlphaGo* selected the move indicated by the red circle; Fan Hui responded with the move indicated by the white square; in his post-game commentary he preferred the move (1) predicted by *AlphaGo*.

图5：人工智能如何实现**在与范文钊的非正式对局中，phaGo（执黑方）通过自主决策选定下一步棋。**Hui。以下各项统计数据中，最大值位置均以橙色圆圈标注：a 通过价值网络$v_\theta(s')$评估根节点位置 s的所有后继节点 s'；显示顶级评估结果的预计胜率。 b 根节点位置 s树中每条边(s,a)的行动值 $Q(s,a)$；仅基于价值网络评估的平均值($\lambda = 0$)。 c 仅基于展开评估的平均动作值 $Q(s,a)$（$\lambda = 1$）。d 直接来自SL策略网络的走法概率 $p_\sigma(a|s)$；以百分比形式呈现（若高于0.1%）。 e 模拟过程中从根节点选取动作的百分比频率。f AlphaGo搜索树中的主变异（访问次数最多的路径）。走法按编号顺序呈现。AlphaGo选择了红圈标记的走法；范谟以白方框标记的走法回应；他在赛后评论中更倾向于AlphaGo预测的走法(1)。

2015 *AlphaGo* and Fan Hui competed in a formal five game match. *AlphaGo* won the match 5 games to 0 (see Figure 6 and Extended Data Table 1). This is the first time that a computer Go program has defeated a human professional player, without handicap, in the full game of Go; a feat that was previously believed to be at least a decade away [3,7,32].

6 Discussion

In this work we have developed a Go program, based on a combination of deep neural networks and tree search, that plays at the level of the strongest human players, thereby achieving one of artificial intelligence's "grand challenges" [32–34]. We have developed, for the first time, effective move selection and position evaluation functions for Go, based on deep neural networks that are trained by a novel combination of supervised and reinforcement learning. We have introduced a new search algorithm that successfully combines neural network evaluations with Monte-Carlo rollouts. Our program *AlphaGo* integrates these components together, at scale, in a high-performance tree search engine.

During the match against Fan Hui, *AlphaGo* evaluated thousands of times fewer positions than *Deep Blue* did in its chess match against Kasparov [4]; compensating by selecting those positions more intelligently, using the policy network, and evaluating them more precisely, using the value network – an approach that is perhaps closer to how humans play. Furthermore, while *Deep Blue* relied on a handcrafted evaluation function, *AlphaGo*'s neural networks are trained directly from game-play purely through general-purpose supervised and reinforcement learning methods.

Go is exemplary in many ways of the difficulties faced by artificial intelligence [34,35]: a challenging decision-making task; an intractable search space; and an optimal solution so complex it appears infeasible to directly approximate using a policy or value function. The previous major breakthrough in computer Go, the introduction of Monte-Carlo tree search, led to corresponding advances in many other domains: for example general game-playing, classical planning, partially observed planning, scheduling, and constraint satisfaction [36,37]. By combining tree search with

2015年AlphaGo与范芮进行正式五番棋对决。AlphaGo以5比0的比分获胜（见图6及扩展数据表1）。这是计算机围棋程序首次在无让子条件下，于完整围棋对局中战胜人类职业棋手——这一成就此前被认为至少需要十年才能实现 [3,7,32]。

6 讨论

在本研究中，我们开发了一款基于深度神经网络与树搜索相结合的围棋程序，其水平已达到顶尖人类棋手级别，从而实现了人工智能领域的"重大挑战"之一 [32–34]。我们首次基于深度神经网络开发出高效的棋步选择与局势评估功能，该网络通过监督学习与强化学习的创新组合方式进行训练。 我们引入了一种新型搜索算法，成功将神经网络评估与蒙特卡罗推演相结合。我们的程序AlphaGo将这些组件整合到一个大规模的高性能树搜索引擎中。

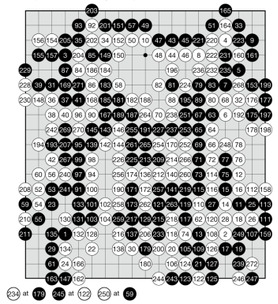
在与范谟的对弈中，AlphaGo评估的棋局位置比深蓝在与卡斯帕罗夫的国际象棋对决中评估的数量少数千倍 [4]；其通过策略网络更智能地筛选棋局位置，并借助价值网络更精准地评估——这种方法或许更接近人类的下棋方式。 此外，深蓝依赖人工设计的评估函数，而AlphaGo的神经网络则直接通过通用监督学习和强化学习方法，从实战中获得训练。

围棋在多方面体现了人工智能面临的挑战：决策任务极具难度；搜索空间难以处理；最优解过于复杂，难以通过策略或价值函数直接近似。 计算机围棋领域的重大突破——蒙特卡罗树搜索的引入——推动了诸多领域的同步进展：例如通用博弈、经典规划、部分可观察规划、调度及约束满足问题 [36,37]。通过将树搜索与

Game 1

Fan Hui (Black), AlphaGo (White)

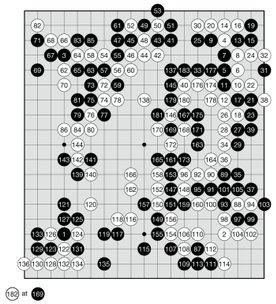
AlphaGo wins by 2.5 points



Game 2

AlphaGo (Black), Fan Hui (White)

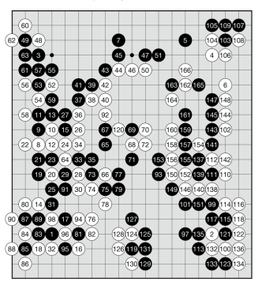
AlphaGo wins by resignation



Game 3

Fan Hui (Black), AlphaGo (White)

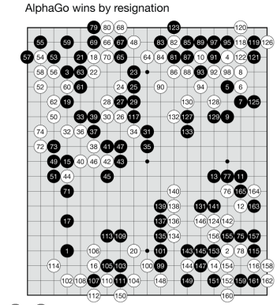
AlphaGo wins by resignation



Game 4

AlphaGo (Black), Fan Hui (White)

AlphaGo wins by resignation



Game 5

Fan Hui (Black), AlphaGo (White)

AlphaGo wins by resignation

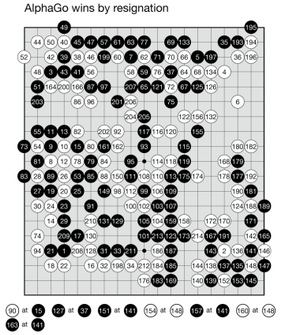
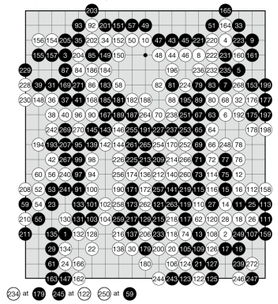


Figure 6: **Games from the match between *AlphaGo* and the human European champion, Fan Hui**. Moves are shown in a numbered sequence corresponding to the order in which they were played. Repeated moves on the same intersection are shown in pairs below the board. The first move number in each pair indicates when the repeat move was played, at an intersection identified by the second move number.

Game 1

Fan Hui (Black), AlphaGo (White)

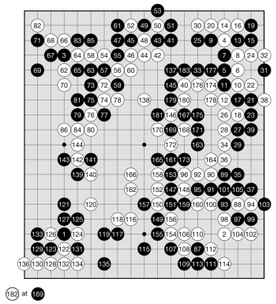
AlphaGo wins by 2.5 points



Game 2

AlphaGo (Black), Fan Hui (White)

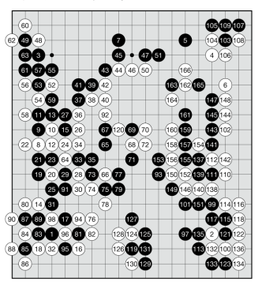
AlphaGo wins by resignation



Game 3

Fan Hui (Black), AlphaGo (White)

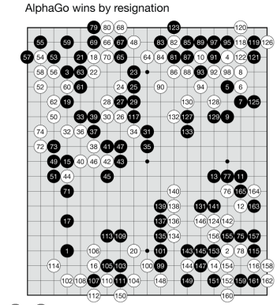
AlphaGo wins by resignation



Game 4

AlphaGo (Black), Fan Hui (White)

AlphaGo wins by resignation



Game 5

Fan Hui (Black), AlphaGo (White)

AlphaGo wins by resignation

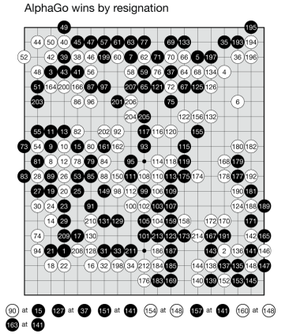


图6：AlphaGo与欧洲人棋手范文烨对弈的比赛局面

棋局以编号顺序展示棋手落子过程。同一交叉点的重复落子以成对形式显示于棋盘下方，每组编号中首位数字标注重复落子时刻，次位数字则标识落子交叉点位置。

policy and value networks, *AlphaGo* has finally reached a professional level in Go, providing hope that human-level performance can now be achieved in other seemingly intractable artificial intelligence domains.

通过策略网络与价值网络的协同作用，AlphaGo最终达到围棋职业水平，为其他看似难以攻克的人工智能领域实现人类级表现带来了希望。

References

1. Allis, L. V. *Searching for Solutions in Games and Artificial Intelligence*. Ph.D. thesis, University of Limburg, Maastricht, The Netherlands (1994).

2. van den Herik, H., Uiterwijk, J. W. & van Rijswijck, J. Games solved: Now and in the future. *Artificial Intelligence* **134**, 277–311 (2002).

3. Schaeffer, J. The games computers (and people) play. *Advances in Computers* **50**, 189–266 (2000).

4. Campbell, M., Hoane, A. & Hsu, F. Deep Blue. *Artificial Intelligence* **134**, 57–83 (2002).

5. Schaeffer, J. *et al.* A world championship caliber checkers program. *Artificial Intelligence* **53**, 273–289 (1992).

6. Buro, M. From simple features to sophisticated evaluation functions. In *1st International Conference on Computers and Games*, 126–145 (1999).

7. Müller, M. Computer Go. *Artificial Intelligence* **134**, 145–179 (2002).

8. Tesauro, G. & Galperin, G. On-line policy improvement using Monte-Carlo search. In *Advances in Neural Information Processing*, 1068–1074 (1996).

9. Sheppard, B. World-championship-caliber Scrabble. *Artificial Intelligence* **134**, 241–275 (2002).

10. Bouzy, B. & Helmstetter, B. Monte-Carlo Go developments. In *10th International Conference on Advances in Computer Games*, 159–174 (2003).

参考文献

1. Allis, L. V.《游戏与人工智能中的解法探索》。博士论文，荷兰马斯特里赫特林堡大学（1994）。2. van den Herik, H., Uiterwijk, J. W. & van Rijswijck, J.《已解决的游戏：现状与未来》。《人工智能》134期，277–311页 3. 舍费尔，J.《计算机（与人类）玩的游戏》。《计算机进展》50卷，189–266页（2000年）。4. 坎贝尔，M.，霍恩，A. & 徐，F.《深蓝》。《人工智能》134卷，57–83页（2002年）。 5. 舍费尔, J. 等. 世界冠军级跳棋程序. 人工智能 53, 273–289 (1992). 6. 布罗, M. 从简单特征到复杂评估函数. 收录于第一届计算机与游戏国际会议论文集, 126–145 (1999). 7. M¨uller, M. 计算机围棋. 人工智能 134, 145–179 (2002).8. Tesauro, G. & Galperin, G. 基于蒙特卡洛搜索的在线策略改进. 收录于《神经信息处理进展》, 1068–1074 (1996). 9. 谢帕德，B. 世界冠军级拼字游戏。人工智能 134, 241–275 (2002)。10. 布齐，B. & 赫尔姆斯泰特，B. 蒙特卡洛围棋发展。载于第十届计算机游戏进展国际会议，159–174 (2003)。

15

11. Coulom, R. Efficient selectivity and backup operators in Monte-Carlo tree search. In *5th International Conference on Computer and Games*, 72–83 (2006).

12. Kocsis, L. & Szepesvári, C. Bandit based Monte-Carlo planning. In *15th European Conference on Machine Learning*, 282–293 (2006).

13. Coulom, R. Computing Elo ratings of move patterns in the game of Go. *International Computer Games Association Journal* **30**, 198–208 (2007).

14. Baudiš, P. & Gailly, J.-L. Pachi: State of the art open source Go program. In *Advances in Computer Games*, 24–38 (Springer, 2012).

15. Müller, M., Enzenberger, M., Arneson, B. & Segal, R. Fuego — an open-source framework for board games and Go engine based on Monte-Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games* **2**, 259–270 (2010).

16. Gelly, S. & Silver, D. Combining online and offline learning in UCT. In *17th International Conference on Machine Learning*, 273–280 (2007).

17. Krizhevsky, A., Sutskever, I. & Hinton, G. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 1097–1105 (2012).

18. Lawrence, S., Giles, C. L., Tsoi, A. C. & Back, A. D. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks* **8**, 98–113 (1997).

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

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

21. Stern, D., Herbrich, R. & Graepel, T. Bayesian pattern ranking for move prediction in the game of Go. In *International Conference of Machine Learning*, 873–880 (2006).

22. Sutskever, I. & Nair, V. Mimicking Go experts with convolutional neural networks. In *International Conference on Artificial Neural Networks*, 101–110 (2008).

11. Coulom, R. モンテカルロ樹搜索中的高效选择性与备份操作符. 发表于第五届计算机与游戏国际会议，第72–83页（2006年）。

12. Kocsis, L.& Szepesv´ari, C. Bandit based Monte-Carlo planning. In 15th European Conference on Machine Learning, 282–293 (2006).

13. Coulom, R. 围棋棋步模式的Elo评级计算。《国际计算机游戏协会期刊》30卷，198–208页（2007年）。

14. Baudi˘s, P. & Gailly, J.-L. Pachi: 尖端开源围棋程序。载于《计算机游戏进展》，第24–38页（Springer出版社，2012年）。

15. M¨uller, M., Enzenberger, M., Arneson, B. & Segal, R. Fuego — 基于蒙特卡洛树搜索的棋类游戏与围棋引擎开源框架。IEEE计算智能与游戏人工智能汇刊 2, 259–270 (2010)。

16. Gelly, S. & Silver, D. 结合在线与离线学习的UCT算法。载于第17届国际机器学习会议论文集，273–280页（2007）。

17. Krizhevsky, A., Sutskever, I. & Hinton, G. 基于深度卷积神经网络的ImageNet分类研究. 载于《神经信息处理系统进展》，第1097–1105页（2012年）。

18. Lawrence, S., Giles, C. L., Tsoi, A. C. & Back, A. D. 面部识别：卷积神经网络方法。IEEE神经网络汇刊 8, 98–113 (1997).

19. Mnih, V. 等. 通过深度强化学习实现人类级控制。《自然》518, 529–533 (2015).

20. LeCun, Y., Bengio, Y. & Hinton, G. 深度学习. Nature 521, 436–444 (2015).

21. Stern, D., Herbrich, R. & Graepel, T. 围棋棋步预测的贝叶斯模式排序. 收录于《国际机器学习会议论文集》，第873–880页（2006年）。

22. Sutskever, I. & Nair, V. 卷积神经网络对围棋专家的模拟研究. 载于《国际人工神经网络会议论文集》，第101–110页（2008年）。

23. Maddison, C. J., Huang, A., Sutskever, I. & Silver, D. Move evaluation in Go using deep convolutional neural networks. *3rd International Conference on Learning Representations* (2015).

24. Clark, C. & Storkey, A. J. Training deep convolutional neural networks to play go. In *32nd International Conference on Machine Learning*, 1766–1774 (2015).

25. Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* **8**, 229–256 (1992).

26. Sutton, R., McAllester, D., Singh, S. & Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 1057–1063 (2000).

27. Schraudolph, N. N., Dayan, P. & Sejnowski, T. J. Temporal difference learning of position evaluation in the game of Go. *Advances in Neural Information Processing Systems* 817–817 (1994).

28. Enzenberger, M. Evaluation in Go by a neural network using soft segmentation. In *10th Advances in Computer Games Conference*, 97–108 (2003).

29. Silver, D., Sutton, R. & Müller, M. Temporal-difference search in computer Go. *Machine learning* **87**, 183–219 (2012).

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

31. KGS: Rating system math. URL `http://www.gokgs.com/help/rmath.html`.

32. Levinovitz, A. The mystery of Go, the ancient game that computers still can't win. *Wired Magazine* (2014).

33. Mechner, D. All Systems Go. *The Sciences* **38** (1998).

23. Maddison, C. J., Huang, A., Sutskever, I. & Silver, D. 基于深度卷积神经网络的围棋棋步评估. 第三届国际学习表示会议论文集 (2015).

24. Clark, C. & Storkey, A. J. 训练深度卷积神经网络进行围棋对弈. 载于第32届国际机器学习会议论文集, 1766–1774 (2015).

25. Williams, R. J. 用于联结主义强化学习的简单统计梯度追踪算法。《机器学习》第8卷, 229–256页（1992年）。

26. Sutton, R., McAllester, D., Singh, S. & Mansour, Y. 基于函数逼近的强化学习策略梯度方法. 收录于《神经信息处理系统进展》, 第1057–1063页（2000年）。

27. Schraudolph, N. N., Dayan, P. & Sejnowski, T. J. 围棋中位置评估的时间差学习。《神经信息处理系统进展》817–817 (1994).

28. 恩岑贝格，M.《基于软分割的神经网络围棋评估》，载于第十届计算机游戏进展会议论文集，第97–108页（2003年）。

29. Silver, D., Sutton, R. & Müller, M. 计算机围棋中的时差搜索。机器学习 87, 183–219 (2012).

30. Coulom, R. 全历史评级：适用于实力随时间变化的棋手的贝叶斯评级系统。载于国际计算机与游戏会议论文集，第113–124页（2008年）。

31. KGS：评级系统数学原理。网址 http://www.gokgs.com/help/rmath.html。32. Levinovitz, A.《围棋之谜：这古老游戏至今仍令计算机束手无策》。《连线》杂志（2014）。33. Mechner, D.《全系统就绪》。《科学》杂志第38卷（1998）。

34. Mandziuk, J. Computational intelligence in mind games. In *Challenges for Computational Intelligence*, 407–442 (2007).

35. Berliner, H. A chronology of computer chess and its literature. *Artificial Intelligence* **10**, 201–214 (1978).

36. Browne, C. *et al.* A survey of Monte-Carlo tree search methods. *IEEE Transactions of Computational Intelligence and AI in Games* **4**, 1–43 (2012).

37. Gelly, S. *et al.* The grand challenge of computer Go: Monte Carlo tree search and extensions. *Communications of the ACM* **55**, 106–113 (2012).

Author Contributions

A.H., G.v.d.D., J.S., I.A., M.La., A.G., T.G., D.S. designed and implemented the search in *AlphaGo*. C.M., A.G., L.S., A.H., I.A., V.P., S.D., D.G., N.K., I.S., K.K., D.S. designed and trained the neural networks in *AlphaGo*. J.S., J.N., A.H., D.S. designed and implemented the evaluation framework for *AlphaGo*. D.S., M.Le., T.L., T.G., K.K., D.H. managed and advised on the project. D.S., T.G., A.G., D.H. wrote the paper.

Acknowledgements

We thank Fan Hui for agreeing to play against *AlphaGo*; Toby Manning for refereeing the match; R. Munos and T. Schaul for helpful discussions and advice; A. Cain and M. Cant for work on the visuals; P. Dayan, G. Wayne, D. Kumaran, D. Purves, H. van Hasselt, A. Barreto and G. Ostrovski for reviewing the paper; and the rest of the DeepMind team for their support, ideas and encouragement.

34. Mandziuk, J. 计算智能在智力游戏中的应用。《计算智能面临的挑战》，第407–442页（2007年）。

35. Berliner, H. 计算机国际象棋及其文献编年史。《人工智能》10卷，201–214页（1978年）。

36. Browne, C. 等. 蒙特卡洛树搜索方法综述. 《IEEE计算智能与游戏人工智能汇刊》第4卷，1–43页（2012年）。

37. Gelly, S. 等. 计算机围棋的重大挑战：蒙特卡洛树搜索及其扩展。《ACM通讯》55卷，106–113页（2012年）。

作者贡献

A.H.、G.v.d.D.、J.S.、I.A.、M.La.、A.G.、T.G.、D.S.设计并实现了AlphaGo的搜索算法。C.M.、A.G.、L.S.、A.H.、I.A.、V.P.、S.D.、D.G.、N.K.、 I.S.、K.K.、D.S.设计并训练了AlphaGo中的神经网络。 J.S.、J.N.、A.H.、D.S.设计并实现了AlphaGo的评估框架。D.S.、M.Le.、T.L.、T.G.、K.K.、D.H.负责项目管理与指导。D.S.、T.G.、A.G.、D.H.共同撰写了本论文。

鸣谢

我们感谢范谟同意与AlphaGo对弈；托比·曼宁担任比赛裁判； R. Munos与T. Schaul的建设性讨论与建议；A. Cain与M. Cant的视觉设计工作；P. Dayan、G. Wayne、D. Kumaran、D. Purves、H. van Hasselt、A. Barreto及G. Ostrovski的论文审阅；以及DeepMind团队其他成员的支持、创意与鼓励。

Methods

Problem setting Many games of perfect information, such as chess, checkers, othello, backgammon and Go, may be defined as alternating Markov games [38]. In these games, there is a state space $\mathcal{S}$ (where state includes an indication of the current player to play); an action space $\mathcal{A}(s)$ defining the legal actions in any given state $s \in \mathcal{S}$; a state transition function $f(s, a, \xi)$ defining the successor state after selecting action a in state s and random input ξ (e.g. dice); and finally a reward function $r^i(s)$ describing the reward received by player i in state s. We restrict our attention to two-player zero sum games, $r^1(s) = -r^2(s) = r(s)$, with deterministic state transitions, $f(s, a, \xi) = f(s, a)$, and zero rewards except at a terminal time-step T. The *outcome* of the game $z_t = \pm r(s_T)$ is the terminal reward at the end of the game from the perspective of the current player at time-step t. A policy $p(a|s)$ is a probability distribution over legal actions $a \in \mathcal{A}(s)$. A value function is the expected outcome if all actions for both players are selected according to policy p, that is, $v^p(s) = \mathbb{E}\left[z_t \mid s_t = s, a_{t\ldots T} \sim p\right]$. Zero sum games have a unique optimal value function $v^*(s)$ that determines the outcome from state s following perfect play by both players,

$$v^*(s) = \begin{cases} z_T & \text{if } s = s_T, \\ \max\limits_a \; -v^*(f(s,a)) & \text{otherwise.} \end{cases}$$

Prior work The optimal value function can be computed recursively by *minimax* (or equivalently *negamax*) search [39]. Most games are too large for exhaustive minimax tree search; instead, the game is truncated by using an approximate value function $v(s) \approx v^*(s)$ in place of terminal rewards. Depth-first minimax search with $\alpha - \beta$ pruning [39] has achieved super-human performance in chess [4], checkers [5] and othello [6], but it has not been effective in Go [7].

Reinforcement learning can learn to approximate the optimal value function directly from games of self-play [38]. The majority of prior work has focused on a linear combination $v_\theta(s) = \phi(s) \cdot \theta$ of features $\phi(s)$ with weights θ. Weights were trained using temporal-difference learning [40] in chess [41,42], checkers [43,44] and Go [29]; or using linear regression in othello [6] and Scrabble [9]. Temporal-difference learning has also been used to train a neural network to approximate the optimal value function, achieving super-human performance in backgammon [45]; and achieving

方法论

问题设定许多完全信息博弈（如国际象棋、跳棋、黑白棋、双陆棋和围棋）可定义为交替马尔可夫博弈 [38]。此类博弈包含：状态空间 $\mathcal{S}$ （状态包含当前执棋方标识）； 动作空间 $\mathcal{A}(s)$定义任意状态 $s \in \mathcal{S}$下的合法操作；状态转移函数 $f(s, a, \xi)$规定在状态 s 选择动作 a 后的后继状态；随机输入 ξ (如骰子投掷结果)； 最后是描述玩家 i 在状态 s中获得奖励的奖励函数 $r^i(s)$。 我们聚焦于双人零和博弈， $r^1(s) = -r^2(s) = r(s)$，其状态转换具有确定性：$f(s, a, \xi) = f(s, a)$，除终止时间步 T外奖励值为零。游戏结果$z_t = \pm r(s_T)$指当前玩家在时间步 t视角下游戏终结时的终止奖励。 策略 $p(a|s)$是合法行动 $a \in \mathcal{A}(s)$上的概率分布。 价值函数是当双方玩家所有行动均按策略 p选择时的预期结果，即 $v^p(s) = \mathbb{E}\left[z_t \mid s_t = s, a_{t\ldots T} \sim p\right]$。零和博弈具有唯一的最优价值函数 $v^*(s)$，该函数决定双方玩家完美博弈后从状态 s 出发的结果。

$$v^*(s) = \begin{cases} z_T & \text{if } s = s_T, \\ \max\limits_a \; -v^*(f(s,a)) & \text{otherwise.} \end{cases}$$

前期研究 最优价值函数可通过最小最大（或等效的负最大）搜索递归计算 [39]。多数棋类游戏规模过大，无法进行穷举最小最大树搜索；因此采用近似价值函数 $v(s) \approx v^*(s)$替代终端奖励来截断游戏。 深度优先最小最大搜索结合 $\alpha - \beta$ 剪枝 [39] 已在国际象棋4、跳棋 [5] 和黑白棋6中实现超越人类的性能，但在围棋 [7]领域尚未奏效。

强化学习可通过自我对弈 [38]直接近似最优价值函数。现有研究多聚焦于特征 $\phi(s)$的线性组合 $v_\theta(s) = \phi(s) \cdot \theta$及其权重 θ。 在国际象棋41,42、跳棋12和围棋13中，权重通过时差学习11进行训练；而在黑白棋14和拼字游戏15中，则采用线性回归训练。 时差学习还被用于训练神经网络近似最优价值函数，在双陆棋中实现超越人类的表现 [45]；并在

weak *kyu* level performance in small-board Go [27,28,46] using convolutional networks.

An alternative approach to minimax search is Monte-Carlo tree search (MCTS) [11,12], which estimates the optimal value of interior nodes by a double approximation, $V^n(s) \approx v^{P^n}(s) \approx v^*(s)$. The first approximation, $V^n(s) \approx v^{P^n}(s)$, uses n Monte-Carlo simulations to estimate the value function of a *simulation policy* P^n. The second approximation, $v^{P^n}(s) \approx v^*(s)$, uses a simulation policy P^n in place of minimax optimal actions. The simulation policy selects actions according to a search control function $\underset{a}{\operatorname{argmax}}\ (Q^n(s,a) + u(s,a))$, such as UCT [12], that selects children with higher *action-values*, $Q^n(s,a) = -V^n(f(s,a))$, plus a bonus $u(s,a)$ that encourages exploration; or in the absence of a search tree at state s, it samples actions from a fast rollout policy $p_\pi(a|s)$. As more simulations are executed and the search tree grows deeper, the simulation policy becomes informed by increasingly accurate statistics. In the limit, both approximations become exact and MCTS (e.g., with UCT) converges [12] to the optimal value function $\lim_{n\to\infty} V^n(s) = \lim_{n\to\infty} v^{P^n}(s) = v^*(s)$. The strongest current Go programs are based on MCTS [13–15,37].

MCTS has previously been combined with a policy that is used to narrow the beam of the search tree to high probability moves [13]; or to bias the bonus term towards high probability moves [47]. MCTS has also been combined with a value function that is used to initialise action-values in newly expanded nodes [16], or to mix Monte-Carlo evaluation with minimax evaluation [48]. In contrast, *AlphaGo*'s use of value functions is based on truncated Monte-Carlo search algorithms [8,9], which terminate rollouts before the end of the game and use a value function in place of the terminal reward. *AlphaGo*'s position evaluation mixes full rollouts with truncated rollouts, resembling in some respects the well-known temporal-difference learning algorithm TD(λ). *AlphaGo* also differs from prior work by using slower but more powerful representations of the policy and value function; evaluating deep neural networks is several orders of magnitudes slower than linear representations and must therefore occur asynchronously.

The performance of MCTS is to a large degree determined by the quality of the rollout policy. Prior work has focused on handcrafted patterns [49] or learning rollout policies by supervised learning [13], reinforcement learning [16], simulation balancing [50,51] or online adaptation [29,52]; how-

在小棋盘围棋中展现弱段位水平 [27,28,46] 采用卷积神经网络实现。

替代最小最大搜索的方法是蒙特卡罗树搜索（MCTS）[11,12]，该方法通过双重近似估算内部节点的最优值：$V^n(s) \approx v^{P^n}(s) \approx v^*(s)$。第一近似 $V^n(s) \approx v^{P^n}(s)$ 采用 n 蒙特卡罗模拟来估计模拟策略 P^n 的价值函数。第二近似 $v^{P^n}(s) \approx v^*(s)$ 采用模拟策略 P^n 替代最小最大最优动作。该策略依据搜索控制函数 $\underset{a}{\operatorname{argmax}}\ (Q^n(s,a) + u(s,a))$ 选择动作，例如 UCT 12 算法——其优先选择具有更高动作价值的子状态 a，即 $Q^n(s,a) = -V^n(f(s,a))$，并附加鼓励探索的奖励项 $u(s,a)$；若状态 s下无搜索树，则从快速展开策略$p_\pi(a|s)$采样动作。随着更多模拟的执行和搜索树的深入，模拟策略将获得越来越精确的统计数据支持。在极限情况下，两种近似都变得精确，MCTS（例如与UCT结合）收敛 [12] 到最优价值函数$\lim_{n\to\infty} V^n(s) = \lim_{n\to\infty} v^{P^n}(s) = v^*(s)$。当前最强的围棋程序基于MCTS [13–15,37]。

MCTS曾与策略结合使用，用于将搜索树的范围缩小至高概率走法 [13]；或使奖励项向高概率走法倾斜 [47]。MCTS也曾与价值函数结合，用于初始化新扩展节点中的行动价值16，或将蒙特卡罗评估与最小最大评估混合 [48]。相较之下，AlphaGo采用基于截断蒙特卡罗搜索算法的价值函数8,9，该算法在对局结束前终止推演，并以价值函数替代终局奖励。AlphaGo的位置评估混合了完整展开与截断展开，在某些方面类似于著名的时差学习算法TD（λ）。AlphaGo还通过使用更慢但更强大的策略和价值函数表示形式区别于先前工作；深度神经网络的评估速度比线性表示慢几个数量级，因此必须异步进行。

MCTS的性能在很大程度上取决于展开策略的质量。先前研究侧重于人工设计模式 [49] 或通过监督学习13、强化学习16、模拟平衡 [50,51] 或在线适应 [29,52]来学习展开策略；如何——

ever, it is known that rollout-based position evaluation is frequently inaccurate [53]. *AlphaGo* uses relatively simple rollouts, and instead addresses the challenging problem of position evaluation more directly using value networks.

然而众所周知，基于推演的局势评估常存在偏差 [53]。AlphaGo采用相对简单的推演算法，转而通过价值网络更直接地解决局势评估这一难题。

Search Algorithm To efficiently integrate large neural networks into *AlphaGo*, we implemented an *asynchronous policy and value* MCTS algorithm (APV-MCTS). 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,

搜索算法 为高效整合大型神经网络至AlphaGo系统，我们实现了异步策略与价值蒙特卡洛树搜索算法（APV-MCTS）。搜索树中每个节点 s 包含所有合法操作 $a \in \mathcal{A}(s)$的边（s, a）。每条边存储一组统计数据，

$$\{P(s,a), N_v(s,a), N_r(s,a), W_v(s,a), W_r(s,a), Q(s,a)\},$$

where $P(s, a)$ is the prior probability, $W_v(s, a)$ and $W_r(s, a)$ are Monte-Carlo estimates of total action-value, accumulated over $N_v(s, a)$ and $N_r(s, a)$ leaf evaluations and rollout rewards respectively, and $Q(s, a)$ is the combined mean action-value for that edge. Multiple simulations are executed in parallel on separate search threads. The APV-MCTS algorithm proceeds in the four stages outlined in Figure 3.

其中 $P(s, a)$ 表示先验概率， $W_v(s, a)$ 与 $W_r(s, a)$ 分别是通过 $N_v(s, a)$ 及 $N_r(s, a)$叶节点评估与推演奖励的累积值， $Q(s, a)$则为该边界的综合平均动作价值。多线程搜索中并行执行多次模拟。APV-MCTS算法遵循图3所示的四阶段流程。

Selection (Figure 4a). The first *in-tree phase* of each simulation begins at the root of the search tree and finishes when the simulation reaches a leaf node at time-step L. At each of these time-steps, $t < L$, an action is selected according to the statistics in the search tree, $a_t = \underset{a}{\operatorname{argmax}} \big(Q(s_t, a) + u(s_t, a)\big)$, using a variant of the PUCT algorithm [47],

选择（图4a）。每次模拟的首次树内阶段始于搜索树根节点，当模拟在时间步 L到达叶节点时结束。 在每个时间步 $t < L$，根据搜索树中的统计数据$a_t = \operatorname{argmax}\big(Q(s_t, a) + u(s_t, a)\big)$选择动作，采用PUCT算法的变体47$_a$

$$u(s,a) = c_{\text{puct}} P(s,a) \frac{\sqrt{\sum_b N_r(s,b)}}{1 + N_r(s,a)}$$

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.

其中 c_{puct} 为决定探索深度的常数；该搜索控制策略初始阶段优先选择先验概率高且访问次数低的动作，但渐近地倾向于选择动作价值高的动作。

Evaluation (Figure 4c). The leaf position s_L is added to a queue for evaluation $v_\theta(s_L)$ by the value network, unless it has previously been evaluated. The second *rollout phase* of each simulation begins at leaf node s_L and continues until the end of the game. At each of these time-steps, $t \geq L$, actions are selected by both players according to the rollout policy, $a_t \sim p_\pi(\cdot|s_t)$. When the game reaches a terminal state, the outcome $z_t = \pm r(s_T)$ is computed from the final

评估（图4c）。叶节点位置 s_L 会被价值网络加入评估队列 $v_\theta(s_L)$进行评估，除非该节点已被先前评估过。每次模拟的第二轮展开阶段从叶节点 s_L 开始，持续至游戏结束。在每个时间步 $t \geq L$，双方玩家均依据展开策略 $a_t \sim p_\pi(\cdot|s_t)$选择行动。当对局进入终局状态时，最终结果 $z_t = \pm r(s_T)$将根据终局状态计算得出。

score.

Backup (Figure 4d). At each in-tree step $t \leq L$ of the simulation, the rollout statistics are updated as if it had lost $-n_{\text{vl}}$ games, $N_r(s_t, a_t) \leftarrow N_r(s_t, a_t) + n_{\text{vl}}; W_r(s_t, a_t) \leftarrow W_r(s_t, a_t) - n_{\text{vl}}$; this virtual loss [54] discourages other threads from simultaneously exploring the identical variation. At the end of the simulation, the rollout statistics are updated in a backward pass through each step $t \leq L$, replacing the virtual losses by the outcome, $N_r(s_t, a_t) \leftarrow N_r(s_t, a_t) - n_{\text{vl}} + 1; W_r(s_t, a_t) \leftarrow W_r(s_t, a_t) + n_{\text{vl}} + z_t$. Asynchronously, a separate backward pass is initiated when the evaluation of the leaf position s_L completes. The output of the value network $v_\theta(s_L)$ is used to update value statistics in a second backward pass through each step $t \leq L$, $N_v(s_t, a_t) \leftarrow N_v(s_t, a_t) + 1, W_v(s_t, a_t) \leftarrow W_v(s_t, a_t) + v_\theta(s_L)$. The overall evaluation of each state-action is a weighted average of the Monte-Carlo estimates, $Q(s, a) = (1 - \lambda)\frac{W_v(s,a)}{N_v(s,a)} + \lambda\frac{W_r(s,a)}{N_r(s,a)}$, that mixes together the value network and rollout evaluations with weighting parameter λ. All updates are performed lock-free [55].

Expansion (Figure 4b). When the visit count exceeds a threshold, $N_r(s, a) > n_{\text{thr}}$, the successor state $s' = f(s, a)$ is added to the search tree. The new node is initialized to $\{N_v(s', a) = N_r(s', a) = 0, W_v(s', a) = W_r(s', a) = 0, P(s', a) = p_\sigma(a|s')\}$, using a *tree policy* $p_\tau(a|s')$ (similar to the rollout policy but with more features, see Extended Data Table 4) to provide placeholder prior probabilities for action selection. The position s' is also inserted into a queue for asynchronous GPU evaluation by the policy network. Prior probabilities are computed by the SL policy network $p_\sigma^\beta(\cdot|s')$ with a softmax temperature set to β; these replace the placeholder prior probabilities, $P(s', a) \leftarrow p_\sigma^\beta(a|s')$, using an atomic update. The threshold n_{thr} is adjusted dynamically to ensure that the rate at which positions are added to the policy queue matches the rate at which the GPUs evaluate the policy network. Positions are evaluated by both the policy network and the value network using a mini-batch size of 1 to minimize end-to-end evaluation time.

We also implemented a distributed APV-MCTS algorithm. This architecture consists of a single master machine that executes the main search, many remote worker CPUs that execute asynchronous rollouts, and many remote worker GPUs that execute asynchronous policy and value

score.

备份（图4d）。在模拟的每次树内步骤 $t \leq L$ 中，滚动统计数据均按其已输掉 $-n_{\text{vl}}$ 局比赛的方式更新，$N_r(s_t, a_t) \leftarrow N_r(s_t, a_t) + n_{\text{vl}}$; $W_r(s_t, a_t) \leftarrow W_r(s_t, a_t) - n_{\text{vl}}$；此虚拟失利 [54] 可抑制其他线程同时探索相同变体。模拟结束时，通过回溯遍历每个步骤$t \leq L$更新展开统计数据，用实际结果替换虚拟损失：$N_r(s_t, a_t) \leftarrow N_r(s_t, a_t) - n_{\text{vl}} + 1$; $W_r(s_t, a_t) \leftarrow W_r(s_t, a_t) + n_{\text{vl}} + z_t$。当叶节点位置评估 s_L 完成时，将异步启动独立的反向传播。价值网络 $v_\theta(s_L)$的输出用于更新价值统计数据，通过二次反向传播遍历每个步骤：$t \leq L$, $N_v(s_t, a_t) \leftarrow N_v(s_t, a_t) + 1, W_v(s_t, a_t) \leftarrow W_v(s_t, a_t) + v_\theta(s_L)$. 每个状态动作的总体评估是蒙特卡罗估计值的加权平均值，$Q(s, a) = (1 - \lambda) W_v(s,a) / N_v(s,a) + \lambda W_r(s,a) / N_r(s,a)$，该评估通过权重参数 λ混合了价值网络与推演评估结果。所有更新操作均采用无锁方式 [55]执行。

扩展（图4b）。当访问次数超过阈值 $N_r(s, a) > n_{\text{thr}}$时，将后继状态 $s' = f(s, a)$添加至搜索树。新节点初始化为：$\{N_v(s', a) = N_r(s', a) = 0, W_v(s', a) = W_r(s', a) = 0, P(s', a) = p_\sigma(a|s')\}$，采用树策略 $p_\tau(a|s')$（类似于展开策略但包含更多特征，详见扩展数据表4）为动作选择提供占位先验概率。位置 s' 同时被插入队列，供策略网络进行异步GPU评估。先验概率由SL策略网络 $p_\sigma^\beta(\cdot|s')$计算，其softmax温度设定为 β；这些概率通过原子更新机制替换占位符先验概率 $P(s', a) \leftarrow p_\sigma^\beta(a|s')$。阈值 n_{thr} 动态调整以确保策略队列的添加速率与GPU评估策略网络的速率匹配。策略网络和价值网络均采用1的小批量大小评估位置，以最小化端到端评估时间。

我们还实现了分布式APV-MCTS算法。该架构包含：执行主搜索的单个主机、执行异步推演的多个远程CPU节点，以及执行异步策略与价值计算的多个远程GPU节点。

network evaluations. The entire search tree is stored on the master, which only executes the in-tree phase of each simulation. The leaf positions are communicated to the worker CPUs, which execute the rollout phase of simulation, and to the worker GPUs, which compute network features and evaluate the policy and value networks. The prior probabilities of the policy network are returned to the master, where they replace placeholder prior probabilities at the newly expanded node. The rewards from rollouts and the value network outputs are each returned to the master, and backed up the originating search path.

At the end of search *AlphaGo* selects the action with maximum visit count; this is less sensitive to outliers than maximizing action-value [15]. 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. The match version of *AlphaGo* continues searching during the opponent's move. It extends the search if the action maximizing visit count and the action maximizing action-value disagree. Time controls were otherwise shaped to use most time in the middle-game [56]. *AlphaGo* resigns when its overall evaluation drops below an estimated 10% probability of winning the game, i.e. $\max_a Q(s,a) < -0.8$.

AlphaGo does not employ the all-moves-as-first [10] or rapid action-value estimation [57] heuristics used in the majority of Monte-Carlo Go programs; when using policy networks as prior knowledge, these biased heuristics do not appear to give any additional benefit. In addition *AlphaGo* does not use progressive widening [13], dynamic komi [58] or an opening book [59].

Rollout Policy The rollout policy $p_\pi(a|s)$ is a linear softmax based on fast, incrementally computed, local pattern-based features consisting of both "response" patterns around the previous move that led to state s, and "non-response" patterns around the candidate move a in state s. Each non-response pattern is a binary feature matching a specific 3×3 pattern centred on a, defined by the colour (black, white, empty) and liberty count $(1, 2, \geq 3)$ for each adjacent intersection. Each response pattern is a binary feature matching the colour and liberty count in a 12-point diamond-shaped pattern [21] centred around the previous move that led to s. Additionally, a small number of

网络评估。整个搜索树存储在主节点上，该节点仅执行每次模拟的树内阶段。 叶节点位置信息传递至：- 执行模拟展开阶段的从属CPU- 计算网络特征并评估策略/价值网络的从属GPU策略网络的先验概率返回主节点，用于替换新扩展节点中的占位先验概率。展开阶段的奖励值与价值网络输出均返回主节点，并追溯至原始搜索路径。

在搜索结束时，AlphaGo选择访问次数最多的动作；相较于最大化动作价值 [15]，该方法对异常值的敏感度更低。搜索树在后续时间步中被复用：对应已执行动作的子节点成为新根节点；该子节点下的子树及其所有统计数据被保留，其余树结构则被舍弃。 比赛版AlphaGo在对手行棋期间持续搜索。若最大化访问次数的操作与最大化操作价值的操作不一致，则扩展搜索范围。 时间控制机制设计为在中盘阶段消耗最多时间 [56]。当整体评估值跌破获胜概率10%的预估值时，AlphaGo将认输，即满足$\max_a Q(s,a) < -0.8$.的条件。

AlphaGo未采用多数蒙特卡洛围棋程序使用的"所有着法均作为首手" [10] 或"快速行动价值评估" [57] 启发式策略；当政策网络作为先验知识时，这些偏倚性启发式策略似乎并无额外优势。 此外，AlphaGo 既不采用渐进式拓宽 [13], 策略，也不使用动态贴目 [58] 或开局棋谱库 [59]。

展开策略 展开策略 $p_\pi(a|s)$基于快速增量计算的局部模式特征，采用线性软最大化算法。该特征包含两部分：围绕导致状态 s的前手棋的"响应"模式，以及围绕状态 s中候选棋 a的"非响应"模式。 每个非响应模式均为二进制特征，匹配特定以 a为中心的 3×3 模式，该模式由相邻交点的颜色（黑/白/空）及自由度计数（1/2/≥ 3）定义。 每个响应模式均为二进制特征，匹配以导致当前状态 s的上一步棋为中心的12点菱形区域 [21] 内的颜色与自由度分布。此外，少量

handcrafted local features encode common-sense Go rules (see Extended Data Table 4). Similar to the policy network, the weights π of the rollout policy are trained from 8 million positions from human games on the Tygem server to maximize log likelihood by stochastic gradient descent. Rollouts execute at approximately 1,000 simulations per second per CPU thread on an empty board.

Our rollout policy $p_\pi(a|s)$ contains less handcrafted knowledge than state-of-the-art Go programs [13]. Instead, we exploit the higher quality action selection within MCTS, which is informed both by the search tree and the policy network. We introduce a new technique that caches all moves from the search tree and then plays similar moves during rollouts; a generalisation of the *last good reply* heuristic [52]. At every step of the tree traversal, the most probable action is inserted into a hash table, along with the 3×3 pattern context (colour, liberty and stone counts) around both the previous move and the current move. At each step of the rollout, the pattern context is matched against the hash table; if a match is found then the stored move is played with high probability.

Symmetries In previous work, the symmetries of Go have been exploited by using rotationally and reflectionally invariant filters in the convolutional layers [24,27,28]. Although this may be effective in small neural networks, it actually hurts performance in larger networks, as it prevents the intermediate filters from identifying specific asymmetric patterns [23]. Instead, we exploit symmetries at run-time by dynamically transforming each position s using the dihedral group of 8 reflections and rotations, $d_1(s), ..., d_8(s)$. In an *explicit symmetry ensemble*, a mini-batch of all 8 positions is passed into the policy network or value network and computed in parallel. For the value network, the output values are simply averaged, $\bar{v}_\theta(s) = \frac{1}{8}\sum_{j=1}^{8} v_\theta(d_j(s))$. For the policy network, the planes of output probabilities are rotated/reflected back into the original orientation, and averaged together to provide an ensemble prediction, $\bar{p}_\sigma(\cdot|s) = \frac{1}{8}\sum_{j=1}^{8} d_j^{-1}(p_\sigma(\cdot|d_j(s)))$; this approach was used in our raw network evaluation (see Extended Data Table 3). Instead, APV-MCTS makes use of an *implicit symmetry ensemble* that randomly selects a single rotation/reflection $j \in [1, 8]$ for each evaluation. We compute exactly one evaluation for that orientation only; in each simulation we compute the value of leaf node s_L by $v_\theta(d_j(s_L))$, and allow the search procedure to average over these evaluations. Similarly, we compute the policy network for a single, randomly selected

手工设计的局部特征编码了围棋常识规则（详见扩展数据表4）。与策略网络类似，展开策略的权重 π 通过随机梯度下降法，从Tygem服务器上人类对局的800万个棋局位置中训练得出，以最大化对数似然值。在空棋盘上，每个CPU线程每秒执行约1000次模拟展开。

我们的推演策略 $p_\pi(a|s)$相较于顶尖围棋程序 [13]，包含更少的人工设计知识。取而代之的是，我们利用蒙特卡洛树搜索（MCTS）中更高质量的行动选择机制——该机制同时受搜索树和策略网络的双重指导。 我们提出一种新方法：缓存搜索树中的所有棋步，并在推演过程中复用相似棋步，这是"最后良好回应启发式" [52]的泛化版本。在每次树遍历步骤中，系统将最可能的行动与 3×3 模式上下文（包含前手与当前棋步的颜色、自由度及子数统计）共同插入哈希表。 在展开的每个步骤中，模式上下文都会与哈希表进行匹配；若找到匹配项，则存储的走法将以高概率被执行。

对称性先前研究通过卷积层中的旋转不变与反射不变滤波器 [24,27,28]利用围棋的对称性。尽管此方法在小型神经网络中有效，但在大型网络中反而损害性能——因其阻碍了中间滤波器识别特定非对称模式 [23]。 我们转而采用运行时动态对称性处理方案：通过八面体群（含8种反射与旋转组合） $d_1(s)$ ... $d_8(s)$ 动态变换每个棋局位置 s 。在显式对称性集合中，将包含全部8种位置的迷你批次同时输入策略网络或价值网络进行并行计算。 对于价值网络，输出值直接取平均值： $\bar{v}_\theta(s) = \frac{1}{8}\sum_{j=1}^{8} v_\theta(d_j(s))$。 策略网络中，输出概率平面经旋转/反射恢复至原始方向后进行平均，形成集合预测： $\bar{p}_\sigma(\cdot|s) = \frac{1}{8}\sum_{j=1}^{8} d_j^{-1}(p_\sigma(\cdot|d_j(s)))$。此方法用于原始网络评估（详见扩展数据表3）。 相反，APV-MCTS利用隐式对称集合，每次评估随机选取单一旋转/反射 $j \in [1, 8]$ 。 我们仅针对该方向精确计算一次评估；每次模拟中通过 $v_\theta(d_j(s_L))$计算叶节点 s_L 的值，并允许搜索过程对这些评估结果进行平均。同理，我们为随机选取的单一方向计算策略网络。

rotation/reflection, $d_j^{-1}(p_\sigma(\cdot|d_j(s)))$.

Policy Network: Classification We trained the policy network p_σ to classify positions according to expert moves played in the KGS data set. This data set contains 29.4 million positions from 160,000 games played by KGS 6 to 9 *dan* human players; 35.4% of the games are handicap games. The data set was split into a test set (the first million positions) and a training set (the remaining 28.4 million positions). Pass moves were excluded from the data set. Each position consisted of a raw board description s and the move a selected by the human. We augmented the data set to include all 8 reflections and rotations of each position. Symmetry augmentation and input features were precomputed for each position. For each training step, we sampled a randomly selected mini-batch of m samples from the augmented KGS data-set, $\{s^k, a^k\}_{k=1}^m$ and applied an asynchronous stochastic gradient descent update to maximize the log likelihood of the action, $\Delta\sigma = \frac{\alpha}{m}\sum_{k=1}^m \frac{\partial \log p_\sigma(a^k|s^k)}{\partial\sigma}$. The step-size α was initialized to 0.003 and was halved every 80 million training steps, without momentum terms, and a mini-batch size of $m = 16$. Updates were applied asynchronously on 50 GPUs using DistBelief [60]; gradients older than 100 steps were discarded. Training took around 3 weeks for 340 million training steps.

Policy Network: Reinforcement Learning We further trained the policy network by policy gradient reinforcement learning [25,26]. Each iteration consisted of a mini-batch of n games played in parallel, between the current policy network p_ρ that is being trained, and an opponent p_{ρ^-} that uses parameters ρ^- from a previous iteration, randomly sampled from a pool $\mathcal{O}$ of opponents, so as to increase the stability of training. Weights were initialized to $\rho = \rho^- = \sigma$. Every 500 iterations, we added the current parameters ρ to the opponent pool. Each game i in the mini-batch was played out until termination at step T^i, and then scored to determine the outcome $z_t^i = \pm r(s_{T^i})$ from each player's perspective. The games were then replayed to determine the policy gradient update, $\Delta\rho = \frac{\alpha}{n}\sum_{i=1}^n\sum_{t=1}^{T^i}\frac{\partial \log p_\rho(a_t^i|s_t^i)}{\partial\rho}(z_t^i - v(s_t^i))$, using the REINFORCE algorithm [25] with baseline $v(s_t^i)$ for variance reduction. On the first pass through the training pipeline, the baseline was set to zero; on the second pass we used the value network $v_\theta(s)$ as a baseline; this provided a small performance boost. The policy network was trained in this way for 10,000 mini-batches of 128

旋转/翻转, d_j^{-1} $(p_\sigma(\cdot|d_j(s)))$。

策略网络：分类我们训练策略网络 p_σ 根据KGS数据集中的专家棋谱对棋局进行分类。该数据集包含16万局KGS六段至九段人类棋手的对局，共计2940万个棋局；其中35.4%为让子棋。 数据集被划分为测试集（前100万局棋局）和训练集（剩余2840万局棋局）。弃子操作被排除在数据集之外。每局棋局包含原始棋盘描述 s 和人类棋手选择的棋步 a 。我们对数据集进行了扩展，包含每局棋局的8种镜像与旋转变体。 对每个棋局预先计算了对称性扩展与输入特征。每次训练步骤中，从扩展后的KGS数据集中随机抽取 m 个样本组成迷你批次 $\{s^k, a^k\}_{k=1}^m$ ，并采用异步随机梯度下降更新以最大化动作的对数似然值 $\Delta\sigma = \frac{\alpha}{m}\sum_{k=1}^m \frac{\partial \log p_\sigma(a^k|s^k)}{\partial\sigma}$ 。 步长初始化为0.003，每8000万训练步减半（无动量项），小批量大小为 $m = 16$。更新通过DistBelief在50块GPU上异步执行，弃用超过100步的梯度。 训练历时约3周，共执行3.4亿次训练步数。

策略网络：强化学习 我们通过策略梯度强化学习 [25,26]进一步训练策略网络。 每次迭代包含 n 场并行进行的迷你批次对弈，由当前训练中的策略网络 p_ρ 与对手p_{ρ^-} 对战，该对手采用前次迭代的参数 ρ^- ，随机抽取自对手池 $\mathcal{O}$ ， 以增强训练稳定性。 权重初始化为 $\rho = \rho^- = \sigma$。每500次迭代，将当前参数 ρ 加入对手池。小批量中每局游戏 i 均执行至步骤 T^i终止，随后评分以确定双方视角下的结果$z_t^i = \pm r(s_{T^i})$。 随后通过重播游戏确定策略梯度更新 $\Delta\rho = \frac{\alpha}{n}\sum_{i=1}^n\sum_{t=1}^{T^i}\frac{\partial \log p_\rho(a_t^i|s_t^i)}{\partial\rho}(z_t^i - v(s_t^i))$，采用REINFORCE算法 [25] 并以基线$v(s_t^i)$降低方差。 在训练管道首次运行时，基线设为零；第二次运行时采用价值网络 $v_\theta(s)$ 作为基线，此举带来小幅性能提升。策略网络以此方式训练了10,000个128元素的小批次。

games, using 50 GPUs, for one day.

Value Network: Regression We trained a value network $v_\theta(s) \approx v^{p_\rho}(s)$ to approximate the value function of the RL policy network p_ρ. To avoid overfitting to the strongly correlated positions within games, we constructed a new data-set of uncorrelated self-play positions. This data-set consisted of over 30 million positions, each drawn from a unique game of self-play. Each game was generated in three phases by randomly sampling a time-step $U \sim \mathbf{unif}\{1, 450\}$, and sampling the first $t = 1, ..., U-1$ moves from the SL policy network, $a_t \sim p_\sigma(\cdot|s_t)$; then sampling one move uniformly at random from available moves, $a_U \sim \mathbf{unif}\{1, 361\}$ (repeatedly until a_U is legal); then sampling the remaining sequence of moves until the game terminates, $t = U + 1, ..., T$, from the RL policy network, $a_t \sim p_\rho(\cdot|s_t)$. Finally, the game is scored to determine the outcome $z_t = \pm r(s_T)$. Only a single training example (s_{U+1}, z_{U+1}) is added to the data-set from each game. This data provides unbiased samples of the value function $v^{p_\rho}(s_{U+1}) = \mathbb{E}\left[z_{U+1} \mid s_{U+1}, a_{U+1,...,T} \sim p_\rho\right]$. During the first two phases of generation we sample from noisier distributions so as to increase the diversity of the data-set. The training method was identical to SL policy network training, except that the parameter update was based on mean squared error between the predicted values and the observed rewards, $\Delta\theta = \frac{\alpha}{m}\sum_{k=1}^{m}\left(z^k - v_\theta(s^k)\right)\frac{\partial v_\theta(s^k)}{\partial\theta}$. The value network was trained for 50 million mini-batches of 32 positions, using 50 GPUs, for one week.

Features for Policy / Value Network Each position s was preprocessed into a set of 19×19 feature planes. The features that we use come directly from the raw representation of the game rules, indicating the status of each intersection of the Go board: stone colour, liberties (adjacent empty points of stone's chain), captures, legality, turns since stone was played, and (for the value network only) the current colour to play. In addition, we use one simple tactical feature that computes the outcome of a ladder search [7]. All features were computed relative to the current colour to play; for example, the stone colour at each intersection was represented as either *player* or *opponent* rather than *black* or *white*. Each integer is split into K different 19×19 planes of binary values (one-hot encoding). For example, separate binary feature planes are used to represent whether an intersection has 1 liberty, 2 liberties, ..., ≥ 8 liberties. The full set of feature planes are

游戏，使用50个GPU，持续一天。

价值网络：回归我们训练了一个价值网络 $v_\theta(s) \approx v^{p_\rho}(s)$来近似强化学习策略网络 p_ρ的价值函数。为避免对棋局中高度相关位置的过拟合，我们构建了新的无关联自对弈位置数据集。 该数据集包含逾3000万个局面，每个局面均来自独立的自对弈对局。每局对弈通过三阶段生成：随机采样时间步长 $U \sim \mathbf{unif}\{1, 450\}$，并从SL策略网络 $a_t \sim p_\sigma(\cdot|s_t)$中采样前 $t = 1, ..., U-1$ 步棋； 随后从可用走法中均匀随机采样一步， $a_U \sim \mathbf{unif}\{1, 361\}$ (重复直至 a_U 合法）； 最后从RL策略网络 $a_t \sim p_\rho(\cdot|s_t)$中采样剩余走法序列直至对局结束， $t = U + 1, ..., T$。 最后，对游戏进行评分以确定结果 $z_t = \pm r(s_T)$。 每局游戏仅向数据集添加单个训练样本（s_{U+1}, z_{U+1}）。该数据提供价值函数 $v^{p_\rho}(s_{U+1})$ $= \mathbb{E}\left[z_{U+1} \mid s_{U+1}, a_{U+1,...,T} \sim p_\rho\right]$的无偏样本。在生成阶段的前两阶段，我们从噪声更大的分布中采样以增加数据集多样性。 训练方法与SL策略网络训练完全一致，区别在于参数更新基于预测值与观测奖励之间的均方误差， $\Delta\theta = \frac{\alpha}{m}\sum_{k=1}^{m}\left(z^k - v_\theta(s^k)\right)\frac{\partial v_\theta(s^k)}{\partial\theta}$。 价值网络使用50个GPU，以32个位置为单位进行5000万次小批量训练，历时一周。

– ——

策略/价值网络特征每个位置 s 均经预处理转化为一组 19×19特征平面。 所用特征直接源自棋局规则的原始表示，反映围棋棋盘每个交叉点的状态：棋子颜色、自由度（棋子连线相邻的空点）、吃子情况、合法性、落子后回合数，以及（仅价值网络使用）当前执子方。 此外，我们采用一项简单的战术特征来计算梯形搜索结果 [7]。所有特征均基于当前执子方计算；例如，各交点的棋子颜色以"己方"或"对手"表示，而非黑白二色。每个整数被拆分为 K 个不同的二进制值平面（one-hot编码）。 例如，通过独立的二进制特征平面分别表示交点拥有1个自由度、2个自由度... ≥ 8 个自由度。完整特征平面集为：

listed in Extended Data Table 2.

Neural Network Architecture The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23×23 image, then convolves k filters of kernel size 5×5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21×21 image, then convolves k filters of kernel size 3×3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1×1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of *AlphaGo* used $k = 192$ filters; Figure 2,b and Extended Data Table 3 additionally show the results of training with $k = 128, 256, 384$ filters.

The input to the value network is also a $19 \times 19 \times 48$ image stack, with an additional binary feature plane describing the current colour to play. Hidden layers 2 to 11 are identical to the policy network, hidden layer 12 is an additional convolution layer, hidden layer 13 convolves 1 filter of 1×1 with stride 1, and hidden layer 14 is a fully connected linear layer with 256 rectifier units. The output layer is a fully connected linear layer with a single *tanh* unit.

Evaluation We evaluated the relative strength of computer Go programs by running an internal tournament and measuring the Elo rating of each program. We estimate the probability that program a will beat program b by a logistic function $p(a \text{ beats } b) = \frac{1}{1+\exp(c_{\text{elo}}(e(b)-e(a)))}$, and estimate the ratings $e(\cdot)$ by Bayesian logistic regression, computed by the *BayesElo* program [30] using the standard constant $c_{\text{elo}} = 1/400$. The scale was anchored to the *BayesElo* rating of professional Go player Fan Hui (2908 at date of submission) [61]. All programs received a maximum of 5 seconds computation time per move; games were scored using Chinese rules with a *komi* of 7.5 points (extra points to compensate white for playing second). We also played handicap games where *AlphaGo* played white against existing Go programs; for these games we used a non-standard handicap system in which *komi* was retained but black was given additional stones on the usual handicap points. Using these rules, a handicap of K stones is equivalent to giving $K - 1$ free moves to black, rather than $K - 1/2$ free moves using standard no-*komi* handicap rules. We used these handicap rules

详见扩展数据表2。

神经网络架构策略网络的输入为包含48个特征层的 $19 \times 19 \times 48$ 图像堆栈。首个隐藏层通过零填充将输入扩展为 23×23 图像，随后使用步长为1的 k 滤波器（卷积核尺寸为 5×5）对输入图像进行卷积运算，并应用整流非线性函数。后续第2至12层隐层均将前一层输入零填充为 21×21 图像，随后对 k 个卷积核（尺寸 3×3，步长1）进行卷积运算，并再次应用整流非线性函数。最终层采用步长为1的 1×1 核尺寸卷积滤波器（每个位置偏置不同），并应用softmax函数。AlphaGo比赛版本使用 $k = 192$ 滤波器；图2b和扩展数据表3还展示了使用 $k = 128$、256、384滤波器训练的结果。

价值网络的输入同样为 $19 \times 19 \times 48$ 图像堆栈，并附加描述当前可玩颜色的二进制特征平面。隐藏层2至11与策略网络完全相同，隐藏层12为额外卷积层，隐藏层13采用步长为1的 1×1 单滤波器卷积，隐藏层14为含256个整流单元的全连接线性层。输出层为单一tanh单元的全连接线性层。

评估 我们通过内部锦标赛运行并测量各程序的Elo等级分，评估计算机围棋程序的相对实力。我们通过逻辑函数 $p(a \text{ 胜 } b) = \frac{1}{1+\exp(c_{\text{elo}}(e(b)-e(a)))}$，并通过贝叶斯逻辑回归估算评级 $e(\cdot)$，由BayesElo程序 [30] 使用标准常数 $c_{\text{elo}} = 1/400$ 计算得出。评级体系以职业棋手范谟（提交时等级分2908）的BayesElo评级 [61] 为基准锚定。所有程序每步棋最多耗时5秒；采用中国规则计分，贴目7.5点（补偿后手方）。我们还安排了让子对局，由AlphaGo执白对抗现有围棋程序。此类对局采用非标准让子体系：保留贴目规则，同时在常规让子点位额外给予黑方子数。按此规则，K 子的让子量相当于给予黑方 $K-1$ 个自由手，而非标准无贴目让子规则下的 $K - 1/2$ 个自由手。我们采用这些让子规则

because *AlphaGo*'s value network was trained specifically to use a *komi* of 7.5.

With the exception of distributed *AlphaGo*, each computer Go program was executed on its own single machine, with identical specs, using the latest available version and the best hardware configuration supported by that program (see Extended Data Table 6). In Figure 4, approximate ranks of computer programs are based on the highest KGS rank achieved by that program; however, the KGS version may differ from the publicly available version.

The match against Fan Hui was arbitrated by an impartial referee. 5 formal games and 5 informal games were played with 7.5 *komi*, no handicap, and Chinese rules. *AlphaGo* won these games 5–0 and 3–2 respectively (Figure 6 and Extended Data Figure 6). Time controls for formal games were 1 hour main time plus 3 periods of 30 seconds *byoyomi*. Time controls for informal games were 3 periods of 30 seconds *byoyomi*. Time controls and playing conditions were chosen by Fan Hui in advance of the match; it was also agreed that the overall match outcome would be determined solely by the formal games. To approximately assess the relative rating of Fan Hui to computer Go programs, we appended the results of all 10 games to our internal tournament results, ignoring differences in time controls.

References

38. Littman, M. L. Markov games as a framework for multi-agent reinforcement learning. In *11th International Conference on Machine Learning*, 157–163 (1994).

39. Knuth, D. E. & Moore, R. W. An analysis of alpha-beta pruning. *Artificial Intelligence* **6**, 293–326 (1975).

40. Sutton, R. Learning to predict by the method of temporal differences. *Machine Learning* **3**, 9–44 (1988).

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

因为AlphaGo的价值网络是专门针对7.5目让子的规则训练的。

除分布式AlphaGo外，所有计算机围棋程序均在配置相同的独立机器上运行，采用该程序支持的最新可用版本及最佳硬件配置（详见扩展数据表6）。图4中计算机程序的近似排名基于其在KGS平台获得的最高段位；但需注意KGS版本可能与公开版本存在差异。

与范谟的对弈由独立裁判监督。双方进行了5场正式赛与5场友谊赛，均采用7.5目贴目、无让子、中国围棋规则。AlphaGo分别以5-0和3-2的比分获胜（见图6及扩展数据图6）。正式赛时限为1小时主时限加3轮30秒读秒。 非正式赛采用三段30秒读秒制。赛制与条件均由范谟赛前确定，双方同意仅以正式赛结果判定总胜负。为粗略评估范谟相对于计算机围棋程序的等级分，我们将全部十局赛果纳入内部锦标赛成绩统计，未计入计时差异。

参考文献

38. Littman, M. L. 马尔可夫博弈作为多智能体强化学习框架。载于第11届国际机器学习会议论文集，157–163页（1994）。39. Knuth, D. E. & Moore, R. W. α-β剪枝算法分析。《人工智能》第6卷，293–326页（1975）。 40. 萨顿，R. 基于时间差分法的预测学习。《机器学习》第3卷，第9–44页（1988年）。41. 巴克斯特，J.，特里德格尔，A. & 威弗，L. 运用时间差分法学习国际象棋。《机器学习》第40卷，第243–263页（2000年）。

42. Veness, J., Silver, D., Blair, A. & Uther, W. Bootstrapping from game tree search. In *Advances in Neural Information Processing Systems* (2009).

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

44. Schaeffer, J., Hlynka, M. & Jussila, V. Temporal difference learning applied to a high-performance game-playing program. In *17th International Joint Conference on Artificial Intelligence*, 529–534 (2001).

45. Tesauro, G. TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation* **6**, 215–219 (1994).

46. Dahl, F. Honte, a Go-playing program using neural nets. In *Machines that learn to play games*, 205–223 (Nova Science, 1999).

47. Rosin, C. D. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence* **61**, 203–230 (2011).

48. Lanctot, M., Winands, M. H. M., Pepels, T. & Sturtevant, N. R. Monte Carlo tree search with heuristic evaluations using implicit minimax backups. In *IEEE Conference on Computational Intelligence and Games*, 1–8 (2014).

49. Gelly, S., Wang, Y., Munos, R. & Teytaud, O. Modification of UCT with patterns in Monte-Carlo Go. Tech. Rep. 6062, INRIA (2006).

50. Silver, D. & Tesauro, G. Monte-Carlo simulation balancing. In *26th International Conference on Machine Learning*, 119 (2009).

51. Huang, S.-C., Coulom, R. & Lin, S.-S. Monte-Carlo simulation balancing in practice. In *7th International Conference on Computers and Games*, 81–92 (Springer-Verlag, 2011).

52. Baier, H. & Drake, P. D. The power of forgetting: Improving the last-good-reply policy in Monte Carlo Go. *IEEE Transactions on Computational Intelligence and AI in Games* **2**, 303–309 (2010).

42. Veness, J., Silver, D., Blair, A. & Uther, W. Bootstrapping from game tree search. In Advances in Neural Information Processing Systems (2009).

43. Samuel, A. L. 《基于跳棋游戏的机器学习研究Ⅱ——近期进展》。IBM研究与开发期刊 11, 601–617 (1967)。

44. Schaeffer, J., Hlynka, M. & Jussila, V. 时差学习在高性能游戏程序中的应用。载于第17届国际人工智能联合会议论文集，529–534页（2001）。

45. Tesauro, G. TD-gammon，一款自学习双陆棋程序，达到大师级水平。《神经计算》第6卷，215–219页（1994年）。

46. Dahl, F. Honte,《基于神经网络的围棋程序》。载于《会玩游戏的机器》，205–223页（Nova Science出版社，1999年）。

47. Rosin, C. D. 具有事件上下文的多臂老虎机问题。《数学与人工智能年鉴》61卷，203–230页（2011年）。

48. Lanctot, M., Winands, M. H. M., Pepels, T. & Sturtevant, N. R. 基于隐式最小最大备份的启发式评估蒙特卡罗树搜索。收录于IEEE计算智能与游戏会议论文集，第1–8页（2014年）。

49. Gelly, S., Wang, Y., Munos, R. & Teytaud, O. 《蒙特卡洛围棋中基于模式的UCT改进》. 技术报告 6062, INRIA (2006).

50. Silver, D. & Tesauro, G. 蒙特卡洛模拟平衡法. 载于第26届国际机器学习会议论文集, 119 (2009).

51. 黄世昌、Coulom R.、林世盛. 实践中的蒙特卡洛模拟平衡. 载于第七届计算机与游戏国际会议论文集，第81–92页（Springer-Verlag出版社，2011年）。

52. Baier, H. & Drake, P. D. 遗忘的力量：改进蒙特卡洛围棋中的最佳应手策略。《IEEE游戏计算智能与人工智能汇刊》2卷，303–309页（2010年）。

53. Huang, S. & Müller, M. Investigating the limits of Monte-Carlo tree search methods in computer Go. In *8th International Conference on Computers and Games*, 39–48 (2013).

54. Segal, R. B. On the scalability of parallel UCT. *Computers and Games* **6515**, 36–47 (2011).

55. Enzenberger, M. & Müller, M. A lock-free multithreaded Monte-Carlo tree search algorithm. In *12th Advances in Computer Games Conference*, 14–20 (2009).

56. Huang, S.-C., Coulom, R. & Lin, S.-S. Time management for Monte-Carlo tree search applied to the game of Go. In *International Conference on Technologies and Applications of Artificial Intelligence*, 462–466 (2010).

57. Gelly, S. & Silver, D. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence* **175**, 1856–1875 (2011).

58. Baudiš, P. Balancing MCTS by dynamically adjusting the komi value. *International Computer Games Association* **34**, 131 (2011).

59. Baier, H. & Winands, M. H. Active opening book application for Monte-Carlo tree search in 19× 19 Go. In *Benelux Conference on Artificial Intelligence*, 3–10 (2011).

60. Dean, J. *et al.* Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, 1223–1231 (2012).

61. Go ratings. URL `http://www.goratings.org`.

53. 黄思廷 & 穆勒, M. 《计算机围棋中蒙特卡洛树搜索方法的极限研究》. 载于第八届计算机与游戏国际会议论文集, 39–48 (2013).

54. Segal, R. B. 关于并行UCT的可扩展性研究。《计算机与游戏》65卷15期，36–47页（2011年）。

55. 恩岑贝格, M. & 穆勒, M. 无锁多线程蒙特卡罗树搜索算法. 载于第12届计算机游戏进展会议论文集, 14–20 (2009).

56. 黄世昌、Coulom R.、林世盛. 围棋领域蒙特卡洛树搜索的时间管理策略. 收录于《人工智能技术与应用国际会议论文集》，第462–466页（2010年）。

57. Gelly, S. & Silver, D. 蒙特卡洛树搜索与计算机围棋中的快速行动价值评估。《人工智能》175, 1856–1875 (2011).

58. Baudiˇs, P. 通过动态调整让子值实现平衡MCTS算法。国际计算机游戏协会第34卷, 131页（2011年）。

59. Baier, H. & Winands, M. H. 基于蒙特卡洛树搜索的围棋主动开局书应用。载于《比荷卢人工智能会议论文集》，第3–10页（2011年）。

60. Dean, J. 等. 大规模分布式深度网络. 载于《神经信息处理系统进展》，第1223–1231页（2012年）。

61. 围棋评级。网址：http://www.goratings.org。

Date	Black	White	Category	Result
5/10/15	Fan Hui	*AlphaGo*	Formal	*AlphaGo* wins by 2.5 points
5/10/15	Fan Hui	*AlphaGo*	Informal	Fan Hui wins by resignation
6/10/15	*AlphaGo*	Fan Hui	Formal	*AlphaGo* wins by resignation
6/10/15	*AlphaGo*	Fan Hui	Informal	*AlphaGo* wins by resignation
7/10/15	Fan Hui	*AlphaGo*	Formal	*AlphaGo* wins by resignation
7/10/15	Fan Hui	*AlphaGo*	Informal	*AlphaGo* wins by resignation
8/10/15	*AlphaGo*	Fan Hui	Formal	*AlphaGo* wins by resignation
8/10/15	*AlphaGo*	Fan Hui	Informal	*AlphaGo* wins by resignation
9/10/15	Fan Hui	*AlphaGo*	Formal	*AlphaGo* wins by resignation
9/10/15	*AlphaGo*	Fan Hui	Informal	Fan Hui wins by resignation

Extended Data Table 1: **Details of match between *AlphaGo* and Fan Hui**. The match consisted of five formal games with longer time controls, and five informal games with shorter time controls. Time controls and playing conditions were chosen by Fan Hui in advance of the match.

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Extended Data Table 2: **Input features for neural networks.** Feature planes used by the policy network (all but last feature) and value network (all features).

日期	黑棋	白棋	分类	结果
2015年5月10日	范谟	*AlphaGo*	正式	AlphaGo以2.5分的优势获胜。
2015年5月10日	范谟	*AlphaGo*	非正式	范谟获胜（对手认输）
2015年6月10日	*AlphaGo*	范谟	正式	AlphaGo以认输方式获胜
2015年6月10日	*AlphaGo*	范谟	非正式	AlphaGo以认输方式获胜
2015年7月10日	范谟	*AlphaGo*	正式	AlphaGo以认输方式获胜
2015年7月10日	范谟	*AlphaGo*	非正式	AlphaGo以认输方式获胜
2015年8月10日	*AlphaGo*	范谟	正式	AlphaGo以认输方式获胜
2015年8月10日	*AlphaGo*	范谟	非正式	AlphaGo以认输方式获胜
2015年9月10日	范谟	*AlphaGo*	正式	AlphaGo以认输方式获胜
2015年9月10日	*AlphaGo*	范谟	非正式	范谟获胜（对手认输）

扩展数据表1：AlphaGo与范芮对弈详情。本次对决包含五局正式比赛（采用较长时限）及五局非正式比赛（采用较短时限）。所有时限与比赛条件均由范芮在赛前预先设定。

功能	# of planes	说明
石色	3	执子手/落子/空位
Ones	1	一个充满1的常数平面
回合数自	8	自上次行棋以来经过多少回合
自由裁量权	8	自由点数量（相邻空点）
捕获尺寸	8	将被吃掉多少颗对手棋子
自杀点数量	8	将有多少己方棋子被吃掉
行棋后的自由度	8	此手落后的自由度数量
梯位捕获	1	当前这一步棋是否构成成功的梯形捕获
逃出阶梯	1	当前这一步棋是否成功逃脱梯子局
合理性	1	判断棋步是否合法且不填塞自身眼位
零	1	一个充满0的常数平面
玩家颜色	1	当前执黑方

扩展数据表2：神经网络输入特征。策略网络使用的特征平面（除最后一项特征外）与价值网络使用的特征平面（所有特征）。

Architecture			Evaluation				
Filters	Symmetries	Features	Test accuracy %	Train accuracy %	Raw net wins %	*AlphaGo* wins %	Forward time (ms)
128	1	48	54.6	57.0	36	53	2.8
192	1	48	55.4	58.0	50	50	4.8
256	1	48	55.9	59.1	67	55	7.1
256	2	48	56.5	59.8	67	38	13.9
256	4	48	56.9	60.2	69	14	27.6
256	8	48	57.0	60.4	69	5	55.3
192	1	4	47.6	51.4	25	15	4.8
192	1	12	54.7	57.1	30	34	4.8
192	1	20	54.7	57.2	38	40	4.8
192	8	4	49.2	53.2	24	2	36.8
192	8	12	55.7	58.3	32	3	36.8
192	8	20	55.8	58.4	42	3	36.8

Extended Data Table 3: **Supervised learning results for the policy network.** The policy network architecture consists of 128, 192 or 256 filters in convolutional layers; an explicit symmetry ensemble over 2, 4 or 8 symmetries; using only the first 4, 12 or 20 input feature planes listed in Extended Data Table 2. The results consist of the test and train accuracy on the KGS data set; and the percentage of games won by given policy network against *AlphaGo*'s policy network (highlighted row 2): using the policy networks to select moves directly (raw wins); or using *AlphaGo*'s search to select moves (*AlphaGo* wins); and finally the computation time for a single evaluation of the policy network.

Feature	# of patterns	Description
Response	1	Whether move matches one or more response features
Save atari	1	Move saves stone(s) from capture
Neighbour	8	Move is 8-connected to previous move
Nakade	8192	Move matches a *nakade* pattern at captured stone
Response pattern	32207	Move matches 12-point diamond pattern near previous move
Non-response pattern	69338	Move matches 3×3 pattern around move
Self-atari	1	Move allows stones to be captured
Last move distance	34	Manhattan distance to previous two moves
Non-response pattern	32207	Move matches 12-point diamond pattern centred around move

Extended Data Table 4: **Input features for rollout and tree policy.** Features used by the rollout policy (first set) and tree policy (first and second set). Patterns are based on stone colour (black/white/empy) and liberties $(1, 2, \geq 3)$ at each intersection of the pattern.

架构			评估				
过滤器	对称性特征		测试准确性racy %	训练准确性 %	原始文本 net 胜率 %	*AlphaGo* 胜率 %	转译时间（毫秒）
128	1	48	54.6	57.0	36	53	2.8
192	1	48	55.4	58.0	50	50	4.8
256	1	48	55.9	59.1	67	55	7.1
256	2	48	56.5	59.8	67	38	13.9
256	4	48	56.9	60.2	69	14	27.6
256	8	48	57.0	60.4	69	5	55.3
192	1	4	47.6	51.4	25	15	4.8
192	1	12	54.7	57.1	30	34	4.8
192	1	20	54.7	57.2	38	40	4.8
192	8	4	49.2	53.2	24	2	36.8
192	8	12	55.7	58.3	32	3	36.8
192	8	20	55.8	58.4	42	3	36.8

扩展数据表3：策略网络的监督学习结果。策略网络架构包含卷积层中128、192或256个滤波器；覆盖2、4或8种对称性的显式对称集合；仅使用扩展数据表2中列出的前4、12或20个输入特征平面。 结果包含KGS数据集的测试与训练准确率；特定策略网络对战AlphaGo策略网络的胜率（高亮第2行）：采用策略网络直接选步（原始胜局）；或借助AlphaGo搜索选步（AlphaGo胜局）；以及策略网络单次评估的计算时间。

功能	模式数量	说明
响应	1	是否与一个或多个响应特征匹配
保存阿塔里	1	此步棋使子免遭吃掉
邻居	8	当前棋子与前一步棋子呈八连通状态
中田	8192	该着法符合中路吃子后的中路定式
响应模式	32207	棋步需匹配前一步附近的12点菱形布局
无响应模式	69338	将 3×3 pattern 移动到 3×3 pattern 周围
自杀式投子	1	"吃子"指捕获对方棋子
最后一步距离	34	与前两步棋的曼哈顿距离
无响应模式	32207	棋步匹配以棋步为中心的12点菱形图案

扩展数据表4：推演策略与树策略的输入特征。推演策略（第一组）与树策略（第一、二组）使用的特征。模式基于棋子颜色（黑/白/空）及模式中每个交点的自由度（$1, 2, \geq 3$）。

Symbol	Parameter	Value
β	Softmax temperature	0.67
λ	Mixing parameter	0.5
n_{vl}	Virtual loss	3
n_{thr}	Expansion threshold	40
c_{puct}	Exploration constant	5

Extended Data Table 5: **Parameters used by *AlphaGo*.**

符号	参数	价值
β	Softmax温度	0.67
λ	混合参数	0.5
n_{vl}	虚拟失利	3
n_{thr}	扩展阈值	40
c_{puct}	探索常数	5

扩展数据表5：AlphaGo使用的参数。

Short name	Computer Player	Version	Time settings	CPUs	GPUs	KGS Rank	Elo
α^d_{rvp}	Distributed *AlphaGo*	See Methods	5 seconds	1202	176	–	3140
α_{rvp}	*AlphaGo*	See Methods	5 seconds	48	8	–	2890
CS	CrazyStone	2015	5 seconds	32	–	6d	1929
ZN	Zen	5	5 seconds	8	–	6d	1888
PC	Pachi	10.99	400,000 sims	16	–	2d	1298
FG	Fuego	svn1989	100,000 sims	16	–	–	1148
GG	GnuGo	3.8	level 10	1	–	5k	431
CS_4	CrazyStone	4 handicap stones	5 seconds	32	–	–	2526
ZN_4	Zen	4 handicap stones	5 seconds	8	–	–	2413
PC_4	Pachi	4 handicap stones	400,000 sims	16	–	–	1756

Extended Data Table 6: **Results of a tournament between different Go programs.** Each program played with a maximum of 5 seconds thinking time per move; the games against Fan Hui were conducted using longer time controls, as described in Methods. CS_4, ZN_4 and PC_4 were given 4 handicap stones; *komi* was 7.5 in all games. Elo ratings were computed by *BayesElo*.

短名称	计算机棋手	版本	时间设置	CPU	GPU	KGS段位	Elo
α^d_{rvp}	分布式AlphaGo	参见方法论	5 seconds	1202	176	–	3140
α_{rvp}	*AlphaGo*	参见方法论	5 seconds	48	8	–	2890
CS	CrazyStone	2015	5 seconds	32	–	6d	1929
ZN	禅	5	5 seconds	8	–	6d	1888
PC	Pachi	10.99	400 000 次模拟	16	–	2d	1298
FG	Fuego	svn1989	100,000次模拟	16	–	–	1148
GG	GnuGo	3.8	level 10	1	–	5k	431
CS_4	CrazyStone	让四子	5 seconds	32	–	–	2526
ZN_4	禅	让四子	5 seconds	8	–	–	2413
PC_4	Pachi	让四子	400,000次模拟	16	–	–	1756

扩展数据表6：不同围棋程序的比赛结果。每款程序每步棋最多思考5秒；与范谟的对局采用更长时限，具体规则详见方法部分。CS_4、ZN_4 和 PC_4 均获4子让子；所有对局均采用7.5目贴目。Elo等级分由BayesElo计算得出。

Short name	Policy network	Value network	Rollouts	Mixing constant	Policy GPUs	Value GPUs	Elo rating
α_{rvp}	p_σ	v_θ	p_π	$\lambda = 0.5$	2	6	2890
α_{vp}	p_σ	v_θ	–	$\lambda = 0$	2	6	2177
α_{rp}	p_σ	–	p_π	$\lambda = 1$	8	0	2416
α_{rv}	$[p_\tau]$	v_θ	p_π	$\lambda = 0.5$	0	8	2077
α_v	$[p_\tau]$	v_θ	–	$\lambda = 0$	0	8	1655
α_r	$[p_\tau]$	–	p_π	$\lambda = 1$	0	0	1457
α_p	p_σ	–	–	–	0	0	1517

Extended Data Table 7: **Results of a tournament between different variants of *AlphaGo*.** Evaluating positions using rollouts only (α_{rp}, α_r), value nets only (α_{vp}, α_v), or mixing both ($\alpha_{rvp}, \alpha_{rv}$); either using the policy network p_σ ($\alpha_{rvp}, \alpha_{vp}, \alpha_{rp}$), or no policy network ($\alpha_{rvp}, \alpha_{vp}, \alpha_{rp}$), i.e. instead using the placeholder probabilities from the tree policy p_τ throughout. Each program used 5 seconds per move on a single machine with 48 CPUs and 8 GPUs. Elo ratings were computed by *BayesElo*.

AlphaGo	Search threads	CPUs	GPUs	Elo
Asynchronous	1	48	8	2203
Asynchronous	2	48	8	2393
Asynchronous	4	48	8	2564
Asynchronous	8	48	8	2665
Asynchronous	16	48	8	2778
Asynchronous	32	48	8	2867
Asynchronous	40	48	8	2890
Asynchronous	40	48	1	2181
Asynchronous	40	48	2	2738
Asynchronous	40	48	4	2850
Distributed	12	428	64	2937
Distributed	24	764	112	3079
Distributed	40	1202	176	3140
Distributed	64	1920	280	3168

Extended Data Table 8: **Results of a tournament between *AlphaGo* and distributed *AlphaGo*, testing scalability with hardware.** Each program played with a maximum of 2 seconds computation time per move. Elo ratings were computed by *BayesElo*.

简短 name	政策网络	价值网络	推演	混合常量	政策 GPU	价值 GPU	Elo 评分
α_{rvp}	p_σ	v_θ	p_π	$\lambda = 0.5$	2	6	2890
α_{vp}	p_σ	v_θ		$\lambda = 0$	2	6	2177
α_{rp}	p_σ		p_π	$\lambda = 1$	8	0	2416
α_{rv}	$[p_\tau]$	v_θ	p_π	$\lambda = 0.5$	0	8	2077
α_v	$[p_\tau]$	v_θ		$\lambda = 0$	0	8	1655
α_r	$[p_\tau]$		p_π	$\lambda = 1$	0	0	1457
α_p	p_σ	–	–	–	0	0	1517

扩展数据表7：不同版本AlphaGo的对战结果。仅使用展开评估（α_{rp}, α_r）、仅使用价值网络（α_{vp}, α_v）或混合两者（$\alpha_{rvp}, \alpha_{rv}$）的局面评估； 采用策略网络 p_σ ($\alpha_{rvp}, \alpha_{vp}, \alpha_{rp}$)或完全不使用策略网络（$\alpha_{rvp}, \alpha_{vp}, \alpha_{rp}$）， 即全程使用树策略 p_τ 中的占位符概率。 每个程序在配备48个CPU和8个GPU的单台机器上每步耗时5秒。Elo等级分由BayesElo计算得出。

AlphaGo	搜索相关讨论	CPU	GPU	Elo
异步	1	48	8	2203
异步	2	48	8	2393
异步	4	48	8	2564
异步	8	48	8	2665
异步	16	48	8	2778
异步	32	48	8	2867
异步	40	48	8	2890
异步	40	48	1	2181
异步	40	48	2	2738
异步	40	48	4	2850
分布式	12	428	64	2937
分布式	24	764	112	3079
分布式	40	1202	176	3140
分布式	64	1920	280	3168

扩展数据表8：AlphaGo与分布式AlphaGo的对弈结果，通过硬件测试其可扩展性。每个程序每步棋的计算时间上限为2秒。Elo等级分由BayesElo计算得出。

	α_{rvp}	α_{vp}	α_{rp}	α_{rv}	α_r	α_v	α_p
α_{rvp}	-	1 [0; 5]	5 [4; 7]	0 [0; 4]	0 [0; 8]	0 [0; 19]	0 [0; 19]
α_{vp}	99 [95; 100]	-	61 [52; 69]	35 [25; 48]	6 [1; 27]	0 [0; 22]	1 [0; 6]
α_{rp}	95 [93; 96]	39 [31; 48]	-	13 [7; 23]	0 [0; 9]	0 [0; 22]	4 [1; 21]
α_{rv}	100 [96; 100]	65 [52; 75]	87 [77; 93]	-	0 [0; 18]	29 [8; 64]	48 [33; 65]
α_r	100 [92; 100]	94 [73; 99]	100 [91; 100]	100 [82; 100]	-	78 [45; 94]	78 [71; 84]
α_v	100 [81; 100]	100 [78; 100]	100 [78; 100]	71 [36; 92]	22 [6; 55]	-	30 [16; 48]
α_p	100 [81; 100]	99 [94; 100]	96 [79; 99]	52 [35; 67]	22 [16; 29]	70 [52; 84]	-
CS	100 [97; 100]	74 [66; 81]	98 [94; 99]	80 [70; 87]	5 [3; 7]	36 [16; 61]	8 [5; 14]
ZN	99 [93; 100]	84 [67; 93]	98 [93; 99]	92 [67; 99]	6 [2; 19]	40 [12; 77]	100 [65; 100]
PC	100 [98; 100]	99 [95; 100]	100 [98; 100]	98 [89; 100]	78 [73; 81]	87 [68; 95]	55 [47; 62]
FG	100 [97; 100]	99 [93; 100]	100 [96; 100]	100 [91; 100]	78 [73; 83]	100 [65; 100]	65 [55; 73]
GG	100 [44; 100]	100 [34; 100]	100 [68; 100]	100 [57; 100]	99 [97; 100]	67 [21; 94]	99 [95; 100]
CS_4	77 [69; 84]	12 [8; 18]	53 [44; 61]	15 [8; 24]	0 [0; 3]	0 [0; 30]	0 [0; 8]
ZN_4	86 [77; 92]	25 [16; 38]	67 [56; 76]	14 [7; 27]	0 [0; 12]	0 [0; 43]	-
PC_4	99 [97; 100]	82 [75; 88]	98 [95; 99]	89 [79; 95]	32 [26; 39]	13 [3; 36]	35 [25; 46]

Extended Data Table 9: **Cross-table of percentage win rates between programs.** 95% Agresti-Coull confidence intervals in grey. Each program played with a maximum of 5 seconds computation time per move. CN_4, ZN_4 and PC_4 were given 4 handicap stones; *komi* was 7.5 in all games. Distributed *AlphaGo* scored 77% [70; 82] against α_{rvp} and 100% against all other programs (no handicap games were played).

	α_{rvp}	α_{vp}	α_{rp}	α_{rv}	α_r	α_v	α_p
α_{rvp}	-	1 [0; 5]	5 [4; 7]	0 [0; 4]	0 [0; 8]	0 [0; 19]	0 [0; 19]
α_{vp}	99 [95; 100]	-	61 [52; 69]	35 [25; 48]	6 [1; 27]	0 [0; 22]	1 [0; 6]
α_{rp}	95 [93; 96]	39 [31; 48]	-	13 [7; 23]	0 [0; 9]	0 [0; 22]	4 [1; 21]
α_{rv}	100 [96; 100]	65 [52; 75]	87 [77; 93]	-	0 [0; 18]	29 [8; 64]	48 [33; 65]
α_r	100 [92; 100]	94 [73; 99]	100 [91; 100]	100 [82; 100]	-	78 [45; 94]	78 [71; 84]
α_v	100 [81; 100]	100 [78; 100]	100 [78; 100]	71 [36; 92]	22 [6; 55]	-	30 [16; 48]
α_p	100 [81; 100]	99 [94; 100]	96 [79; 99]	52 [35; 67]	22 [16; 29]	70 [52; 84]	-
CS	100 [97; 100]	74 [66; 81]	98 [94; 99]	80 [70; 87]	5 [3; 7]	36 [16; 61]	8 [5; 14]
ZN	99 [93; 100]	84 [67; 93]	98 [93; 99]	92 [67; 99]	6 [2; 19]	40 [12; 77]	100 [65; 100]
PC	100 [98; 100]	99 [95; 100]	100 [98; 100]	98 [89; 100]	78 [73; 81]	87 [68; 95]	55 [47; 62]
FG	100 [97; 100]	99 [93; 100]	100 [96; 100]	100 [91; 100]	78 [73; 83]	100 [65; 100]	65 [55; 73]
GG	100 [44; 100]	100 [34; 100]	100 [68; 100]	100 [57; 100]	99 [97; 100]	67 [21; 94]	99 [95; 100]
CS_4	77 [69; 84]	12 [8; 18]	53 [44; 61]	15 [8; 24]	0 [0; 3]	0 [0; 30]	0 [0; 8]
ZN_4	86 [77; 92]	25 [16; 38]	67 [56; 76]	14 [7; 27]	0 [0; 12]	0 [0; 43]	-
PC_4	99 [97; 100]	82 [75; 88]	98 [95; 99]	89 [79; 95]	32 [26; 39]	13 [3; 36]	35 [25; 46]

扩展数据表9：程序间胜率交叉表。灰色区域为95% Agresti-Coull置信区间。 各程序每步棋计算时间上限为5秒。 CN_4、 ZN_4 和 PC_4 获让4子；所有对局均采用7.5目贴目制。分布式AlphaGo对战 [70时胜率77%；对战 82] 时胜率100%，对战其他程序时胜率100%（未进行让子对局）。

Threads		1	2	4	8	16	32	40	40	40	40
	GPU	8	8	8	8	8	8	8	4	2	1
1	8	-	70 [61;78]	90 [84;94]	94 [83;98]	86 [72;94]	98 [91;100]	98 [92;99]	100 [76;100]	96 [91;98]	38 [25;52]
2	8	30 [22;39]	-	72 [61;81]	81 [71;88]	86 [76;93]	92 [83;97]	93 [86;96]	83 [69;91]	84 [75;90]	26 [17;38]
4	8	10 [6;16]	28 [19;39]	-	62 [53;70]	71 [61;80]	82 [71;89]	84 [74;90]	81 [69;89]	78 [63;88]	18 [10;28]
8	8	6 [2;17]	19 [12;29]	38 [30;47]	-	61 [51;71]	65 [51;76]	73 [62;82]	74 [59;85]	64 [55;73]	12 [3;34]
16	8	14 [6;28]	14 [7;24]	29 [20;39]	39 [29;49]	-	52 [41;63]	61 [50;71]	52 [41;64]	41 [32;51]	5 [1;25]
32	8	2 [0;9]	8 [3;17]	18 [11;29]	35 [24;49]	48 [37;59]	-	52 [42;63]	44 [32;57]	26 [17;36]	0 [0;30]
40	8	2 [1;8]	7 [4;14]	16 [10;26]	27 [18;38]	39 [29;50]	48 [37;58]	-	43 [30;56]	41 [26;58]	4 [1;18]
40	4	0 [0;24]	17 [9;31]	19 [11;31]	26 [15;41]	48 [36;59]	56 [43;68]	57 [44;70]	-	29 [18;41]	2 [0;11]
40	2	4 [2;9]	16 [10;25]	22 [12;37]	36 [27;45]	59 [49;68]	74 [64;83]	59 [42;74]	71 [59;82]	-	5 [1;17]
40	1	62 [48;75]	74 [62;83]	82 [72;90]	88 [66;97]	95 [75;99]	100 [70;100]	96 [82;99]	98 [89;100]	95 [83;99]	-

Extended Data Table 10: **Cross-table of percentage win rates between programs in the single-machine scalability study.** 95% Agresti-Coull confidence intervals in grey. Each program played with 2 seconds per move; *komi* was 7.5 in all games.

线程		1	2	4	8	16	32	40	40	40	40
	GPU	8	8	8	8	8	8	8	4	2	1
1	8	-	70 [61;78]	90 [84;94]	94 [83;98]	86 [72;94]	98 [91;100]	98 [92;99]	100 [76;100]	96 [91;98]	38 [25;52]
2	8	30 [22;39]	-	72 [61;81]	81 [71;88]	86 [76;93]	92 [83;97]	93 [86;96]	83 [69;91]	84 [75;90]	26 [17;38]
4	8	10 [6;16]	28 [19;39]	-	62 [53;70]	71 [61;80]	82 [71;89]	84 [74;90]	81 [69;89]	78 [63;88]	18 [10;28]
8	8	6 [2;17]	19 [12;29]	38 [30;47]	-	61 [51;71]	65 [51;76]	73 [62;82]	74 [59;85]	64 [55;73]	12 [3;34]
16	8	14 [6;28]	14 [7;24]	29 [20;39]	39 [29;49]	-	52 [41;63]	61 [50;71]	52 [41;64]	41 [32;51]	5 [1;25]
32	8	2 [0;9]	8 [3;17]	18 [11;29]	35 [24;49]	48 [37;59]	-	52 [42;63]	44 [32;57]	26 [17;36]	0 [0;30]
40	8	2 [1;8]	7 [4;14]	16 [10;26]	27 [18;38]	39 [29;50]	48 [37;58]	-	43 [30;56]	41 [26;58]	4 [1;18]
40	4	0 [0;24]	17 [9;31]	19 [11;31]	26 [15;41]	48 [36;59]	56 [43;68]	57 [44;70]	-	29 [18;41]	2 [0;11]
40	2	4 [2;9]	16 [10;25]	22 [12;37]	36 [27;45]	59 [49;68]	74 [64;83]	59 [42;74]	71 [59;82]	-	5 [1;17]
40	1	62 [48;75]	74 [62;83]	82 [72;90]	88 [66;97]	95 [75;99]	100 [70;100]	96 [82;99]	98 [89;100]	95 [83;99]	-

扩展数据表10：单机可扩展性研究中程序间胜率交叉表。灰色区域为95% Agresti-Coull置信区间。所有对局均采用每手2秒时限，让子数为7.5。

Threads			40	12	24	40	64
	GPU		8	64	112	176	280
		CPU	48	428	764	1202	1920
40	8	48	-	52 [43; 61]	68 [59; 76]	77 [70; 82]	81 [65; 91]
12	64	428	48 [39; 57]	-	64 [54; 73]	62 [41; 79]	83 [55; 95]
24	112	764	32 [24; 41]	36 [27; 46]	-	36 [20; 57]	60 [51; 69]
40	176	1202	23 [18; 30]	38 [21; 59]	64 [43; 80]	-	53 [39; 67]
64	280	1920	19 [9; 35]	17 [5; 45]	40 [31; 49]	47 [33; 61]	-

Extended Data Table 11: **Cross-table of percentage win rates between programs in the distributed scalability study**. 95% Agresti-Coull confidence intervals in grey. Each program played with 2 seconds per move; *komi* was 7.5 in all games.

线程			40	12	24	40	64
	GPU		8	64	112	176	280
		CPU	48	428	764	1202	1920
40	8	48	-	52 [43; 61]	68 [59; 76]	77 [70; 82]	81 [65; 91]
12	64	428	48 [39; 57]	-	64 [54; 73]	62 [41; 79]	83 [55; 95]
24	112	764	32 [24; 41]	36 [27; 46]	-	36 [20; 57]	60 [51; 69]
40	176	1202	23 [18; 30]	38 [21; 59]	64 [43; 80]	-	53 [39; 67]
64	280	1920	19 [9; 35]	17 [5; 45]	40 [31; 49]	47 [33; 61]	-

扩展数据表11：分布式可扩展性研究中程序间胜率交叉表。灰色区域为95% Agresti-Coull置信区间。所有对局均采用每手2秒时限，让子数统一为7.5目。