**Mastering the game of Go with deep neural networks and tree search**

David Silver1\*, Aja Huang1\*,

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here 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 stateof-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 human 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.

围棋由于它巨大的搜索空间并且很难评估棋盘中落子的位置和决定下一步的落子，长期以来被视为人工智能处理经典游戏中最大的挑战。本文阐述了一种新的方法去处理围棋，这种新的方法使用‘价值网络’去评估棋盘中已落子的位置，‘策略网络’去选择下次落子的位置。这些深度神经网络是用一种新奇的方法训练的，它结合监督学习的方法学习人类专家玩过的棋盘，并且使用强化学习在游戏中自我博弈。没有任何先前的搜索处理，神经网络在蒙特卡洛树搜索随机仿真了成千上万把自我博弈的基础之上学会下围棋。我们也提出了一种新的搜索算法，这种算法结合了蒙特卡洛搜索、价值网络和策略网络。应用这种搜索算法，我们以99.8%的胜率赢得其他围棋软件。并且以5：0赢了欧洲围棋冠军。这是在至少几十年以来，第一次计算机程序在围棋上打败人类职业选手。

The effective search space can be reduced by two general principles.

有两条法则可以有效地减少搜索空间的大小。

First, the depth of the search may be reduced by position evaluation:

1.博弈树的深度太深可以用位置评估函数来解决。

truncating the search tree at state s and replacing the subtree below s by an approximate value function v(s) ≈ v\*(s) that predicts the outcome from state s.

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.

2.博弈树宽度太宽的问题可以用策略网络来解决，它是基于状态s而得出动作的一个概率分布。

Alpha Go中含有四大部件：

|  |
| --- |
|  |

·**SL policy network**：用人类棋手里面的对弈记录进行训练，输入为48张图片，以输出为人类棋手的落子为目标。

·**Rollout policy**：将很复杂的卷积神经网络去掉就能够得到一个快速走子网络。它与SL policy network的功能一样，两者间的不同是：Rollout policy网络简单，落子更快，准确率较低。

·**RL policy network**：最开始由SL policy network复制而来，两者开始对弈，在对弈的结果里面我们依据最终的胜负来修正权重。优化之后再更新对手的权重，再接着对弈，两者共同优化。

·**Value network**：用RL policy network自我对弈得到的数据来训练价值网络。输入是一个棋面，输出是这个棋面的胜率。为什么不直接用人类棋手的数据来训练价值网络呢？这是因为人类棋手对局的数据很少，有效样本也很少，很容易产生过拟合，对局的水平也不是很高。

We begin by training a supervised learning (SL) policy network pσ directly from expert human moves.

我们最开始使用监督学习算法训练一个策略网络SL policy network，学习的资料是来自于人类棋手的对弈棋局。

we train a reinforcement learning (RL) policy network pρ that improves the SL policy network by optimizing the final outcome of games of selfplay.

接下来我们训练一个RL policy network通过自我对弈以获得最终的胜利为目标来优化策略网络。

Finally, we train a value network vθ that predicts the winner of games played by the RL policy network against itself.

最后，我们将训练一个价值网络vθ来预测RL policy network自我对弈的胜率。

·**Supervised learning of policy networks**

The SL policy network pσ(a | s) alternates between convolutional layers with weights σ, and rectifier nonlinearities. A final softmax layer outputs a probability distribution over all legal moves a.

SL policy network采用卷积神经网络和ReLu激活函数，最后一层使用softmax输出落子概率。

The input s to the policy network is a simple representation of the board state (see Extended Data Table 2).

输入策略网络中的状态是一个棋盘状态的表达。如下图所示

|  |
| --- |
|  |

using stochastic gradient ascent to maximize the likelihood of the human move a selected in state s

用随机梯度上升去最大化其与人类棋手落子的相似度。

We also trained a faster but less accurate rollout policy pπ(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 μs to select an action, rather than 3 ms for the policy network.

Deep Mind团队也训练一个快速走子网络rollout policy，rollout policy是一个网络结构简单的神经网络，也是使用softmax作为输出，能够达到24.2%的精确度，决策一步棋仅需2 μs，而SL网络需要3 ms。

·**Reinforcement learning of policy networks**

The RL policy network pρ is identical in structure to the SL policy network, and its weights ρ are initialized to the same values, ρ = σ.

RL policy network的网络结构是和SL policy network的网络结构一模一样，并且RL网络权重的初始化是拷贝SL网络的参数。

We play games between the current policy network pρ and a randomly selected previous iteration of the policy network.

随着自我博弈的进行，RL网络就会慢慢进化。RL policy network随机选择之前版本的RL policy network进行对弈。

Randomizing from a pool of opponents in this way stabilizes training by preventing overfitting to the current policy.

用这种从对手池中选择对手的方式能够更稳定地训练，防止过拟合。最终胜利的话，奖励为+1，输了的话，奖励为-1。

这种自我对弈出来的RL policy network其实就具备很高的围棋水平了。

**·Reinforcement learning of value networks**

we instead estimate the value function v pρ for our strongest policy, using the RL policy network pρ.

使用RL policy network强大的策略网络来评估价值函数。

This neural network has a similar architecture to the policy network, but outputs a single prediction instead of a probability distribution.

价值网络的架构与策略网络的架构是非常相似的，只是将其输出变为一个单一的值，而不是策略网络中的动作分布概率。

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 vθ(s), and the corresponding outcome z

用回归的方法训练神经网络，以最小化均方差做随机梯度下降。误差来自**预测的输出**和**这盘局相应地最终奖励输出**。

we generated a new self-play data set consisting of 30 million distinct positions, each sampled from a separate game.

为了防止过拟合等问题，他们从强化学习自我博弈的不同的棋局里面选出30万把不同位置的棋盘来做训练。

**·Searching with policy and value networks**

在说这个之前我们先要说一下蒙特卡洛树搜索https://www.cnblogs.com/yifdu25/p/8303462.html

MCTS也就是蒙特卡罗树搜索（Monte Carlo Tree Search），是一类树搜索算法的统称，可以较为有效地解决一些探索空间巨大的问题，例如一般的围棋算法都是基于MCTS实现的。

这类算法要解决的问题是这样的，我们把围棋的每一步所有可能选择都作为树的节点，第零层只有1个根节点，第1层就有361种下子可能和节点，第2层有360种下子可能和节点，这是一颗非常大的树，我们要在每一层树节点中搜索出赢概率最大的节点，也就是下子方法。

MCTS的算法分为四步，第一步是Selection，就是在树中找到一个最好的值得探索的节点，一般策略是先选择未被探索的子节点，如果都探索过就选择UCB值最大的子节点。第二步是Expansion，就是在前面选中的子节点中走一步创建一个新的子节点，一般策略是随机自行一个操作并且这个操作不能与前面的子节点重复。第三步是Simulation，就是在前面新Expansion出来的节点开始模拟游戏，直到到达游戏结束状态，这样可以收到到这个expansion出来的节点的得分是多少。第四步是Backpropagation，就是把前面expansion出来的节点得分反馈到前面所有父节点中，更新这些节点的quality value和visit times，方便后面计算UCB值。

基本思路就是这样的，通过不断的模拟得到大部分节点的UCB值，然后下次模拟的时候根据UCB值有策略得选择值得利用和值得探索的节点继续模拟，在搜索空间巨大并且计算能力有限的情况下，这种启发式搜索能更集中地、更大概率找到一些更好的节点。

|  |
| --- |
|  |

UCB:

|  |
| --- |
|  |

但是在alpha go里面，将上述公式做了一定的改变。上面这种表述可能有点抽象。这里再附上具体的博客操作链接。

https://blog.csdn.net/sinat\_23588337/article/details/78322375

·**Monte-Carlo tree search in AlphaGo**

首先我们先有一个博弈树，博弈树是有根节点和子节点。我们把围棋的每一步所有可能选择都作为树的节点，第零层只有1个根节点，第1层就有361种下子可能和节点，第2层有360种下子可能和节点，这是一颗非常大的树，我们要在每一层树节点中搜索出赢概率最大的节点，也就是下子方法。

每条蒙特卡洛路径上面都有四个值，Q、N、P、V。Q代表这条路径的好坏，假设我们探索一条路径，最终获胜了，那么这条路径的Q值就会提升，输了的话Q值就会下降。N表示模拟走子经过这条路径的次数。最开始的Q、P都是0，P代表，是人类棋手在这条边落子的概率(之后随着网络的改进就不代表人类棋手了)。最开始的时候我们是没有子节点的，也就是与每个子节点的连接都是虚线，随着探索的增加，实边越来越多。假设我们新探索出了一条新的路径，那么这条路径的Q值与N值都将大于0，这个P一直都是，就是SL policy network，也就是说SL这个网络不仅是用于与强化学习对抗博弈，还用于蒙特卡洛树搜索的P值。

当我们停留在某个节点的时候，我们如何来选择动作呢？

|  |  |
| --- | --- |
|  | (1) |

我们依据上式可以知道，如果是一条没有走过的路径，那么Q值为0，但是探索度项会帮助我们提升走这条路的概率。

我们用快速走子网络进行对弈，一直下到终止，尽管快速走子网络下棋很臭，但是黑白双方都是使用快速走子网络进行对弈，从统计概念上说，他们的胜负会取决于当前棋盘的好坏，而不是网络的好坏。这个时候算出一个中间变量V：

|  |  |
| --- | --- |
|  | (2) |

再用这个中间变量V去表示Q

|  |  |
| --- | --- |
|  | (3) |

当搜索完成之后，算法会选择最常走的那条路径进行移动。