法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

关注 小象学院

第7讲　策略梯度法与Alpha Go

强化学习

主讲人：叶梓

上海交通大学博士

主要研究方向：机器学习、深度学习、人工智能

本章内容

- 策略梯度
- 基于价值学习方法的不足
- 如何评价策略？
- 策略的表现形式
- REINFORCE
- Alpha Go的实现
- Alpha Go Zero的实现简介
- 案例

策略梯度

- 上一讲中，主要是利用函数近似来将$V_\pi(s)$与$Q_\pi(s,a)$参数化：

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

- 本讲中直接对策略进行参数化，

$$\pi_\theta = P[a|s,\theta]$$

- 策略P(a|s)将从一个概率集合变成函数本身π(s,a)，
- 这个策略函数表明在给定一个状态s的情况下，采取任何可能行为的概率。
- 它是一个概率密度函数，即在实际应用策略的时候，是按照这个概率分布进行action采样的。

为什么要用基于策略的学习？

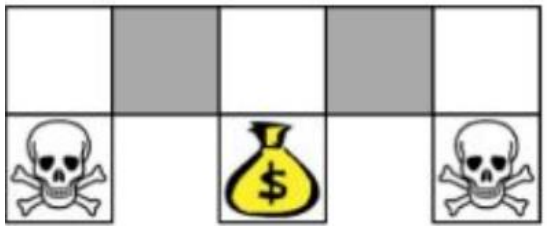
- 基于策略的学习可能会具有更好的收敛性，这是因为基于策略的学习总是朝着好的方向在改善；
  - 有些价值函数在后期会一直围绕最优价值函数持续小的震荡而不收敛。
- 在对于那些拥有高维度或连续状态空间来说，使用基于价值函数的学习，制定策略时需要比较各种行为对应的价值大小，带来计算上的困难。
- 能够学到一些随机策略，基于价值函数的学习通常是学不到随机策略的。
- 有时候价值函数很难计算。

运用基于策略学习的场景

- 有时随机策略才是最优策略。比如“剪刀石头布”这个游戏，如果按照某一种策略来出拳的话，很容易让别人抓住规律，那就输了。
- 所以最好的策略就是随机出拳，让别人猜不到。

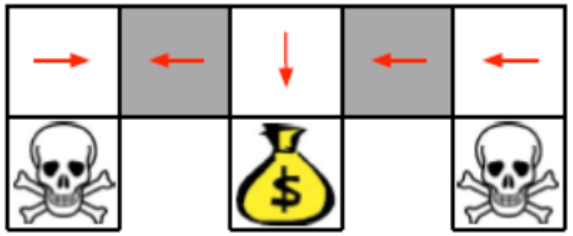
基于价值学习方法的不足

- 在下图中，agent需要尽量不遇到骷髅而去拿到钱包。agent所在的状态就是图中上方5个格子。
- 对于这个环境的描述，可以使用坐标来表示每个格子(状态)，也可以使用格子某个方向是否有墙来作为特征描述这些格子。
- 如使用后者描述，就会发现两个灰色格子的状态描述是一样的，也就是所谓的重名(Aliased)。



基于价值学习方法的不足

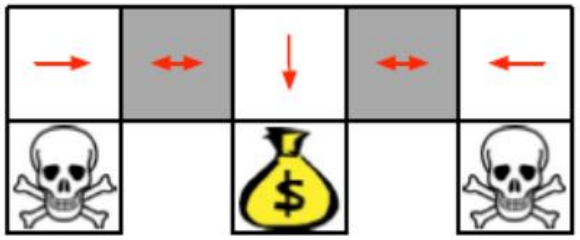
- 可以看到当agent处于这两个灰色格子时，应该采取的行动是不一样的。
  - 当agent在左边的灰色格子时，它的策略应该是向东走，当它在右边的灰色格子时，它应该向西走。
- 这就会出现在同一状态下，需要采取策略不同的情况。而对于基于价值的学习方法，策略是固定的，所以就会出现上图的策略。



当然，也可能两个灰色格子都是朝东走。但总之是确定策略。

基于价值学习方法的不足

- 所以出现这种状态相同，需要采取的行动不同的这种情况时，随机策略就会比确定性的策略好，即下图。
  - π_θ(南北有墙，往东走) = 0.5
  - π_θ(南北有墙，往西走) = 0.5
- 基于策略的学习才能够学到优化的随机策略。



策略梯度方法

- PG方法是一大类基于梯度的策略学习方法。经典的REINFORCE算法和AC算法都可以纳入其框架。
- 基本框架如下：
  - 先定义policy为参数θ的函数，记作π(θ)，
  - 然后定义J(θ)为目标函数（Objective function）
  - 接着通过利用目标函数相对于参数的梯度更新参数θ，从而达到目标函数的局部最大值。
  - 记作：

$$\Delta\theta = \alpha\nabla_\theta J(\theta)$$

策略梯度

- Policy Gradient 就是通过更新 Policy Network 来直接更新策略的。
- 那什么是 Policy Network？实际上就是一个神经网络，输入是状态，输出直接就是动作（不是Q值）。
- 一般输出有两种方式：
  - 一种是概率的方式，即输出某一个动作的概率；
  - 另一种是确定性的方式，即输出具体的某一个动作。

策略梯度目标函数

- 设计一个关于策略的目标函数，通过梯度下降算法优化参数，最终使得惩罚最小化（奖励最大化）。对于策略网络，目标函数其实是比较容易给定的，假定直接按长期回报来算：

$$L(\theta) = \mathbb{E}(r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots | \pi(, \theta))$$

- 由于reward是环境给出的,这个损失函数和策略网络的联系不直观，如何才能更新参数θ?
- 一个非常朴素的想法：
  - 如果某一个动作得到的reward多，那么就使其出现的概率增大；反之就小。

如何评价策略？

$$log(\mathcal{L}) = log(\prod_{i=1}^{N} p_i) = \sum_{i=1}^{N} log(p_i)$$

- 如何去评价一个策略π_θ的好坏呢？如果能够构造一个好的动作**评判指标**，来判断一个动作的好与坏，那么就可以通过改变动作的出现概率来优化策略。
- 假设这个评价指标是 f(s,a)，Policy Network 输出的 $\pi(a|s,\theta)$ 是概率，那么可以通过极大似然估计的方法来优化这个目标。
- 例如可以构造如下目标函数：

$$L(\theta) = \sum log\pi(a|s,\theta) f(s,a)$$

用另一种方法来分析

- f(s,a)不仅仅可以作为动作的评价指标，还可以作为目标函数的期望。就如同下棋，评价指标就是局面赢或输，而目标就是结果赢。
- 因此，问题就变成对f(s,a)求关于参数的梯度，f(x)即f(s,a)。

$$
\begin{aligned}
\nabla_\theta E_x[f(x)] &= \nabla_\theta \sum_x p(x) f(x) && \text{definition of expectation} \\
&= \sum_x \nabla_\theta p(x) f(x) && \text{swap sum and gradient} \\
&= \sum_x p(x) \frac{\nabla_\theta p(x)}{p(x)} f(x) && \text{both multiply and divide by } p(x) \\
&= \sum_x p(x) \nabla_\theta \log p(x) f(x) && \text{use the fact that } \nabla_\theta \log(z) = \frac{1}{z} \nabla_\theta z \\
&= E_x[f(x) \nabla_\theta \log p(x)] && \text{definition of expectation}
\end{aligned}
$$

D. Silver的推导方式

- 直接将目标函数和策略网络建立联系：考虑一个非常简单的one-step的MDPs。
  - 初始状态s服从d(s)
  - 执行一步后结束，获得奖励r = $R_{s,a}$。

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\
&= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a)\mathcal{R}_{s,a}
\end{aligned}
$$

$$
\begin{aligned}
\nabla_\theta \pi_\theta(s,a) &= \pi_\theta(s,a)\frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \\
&= \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)
\end{aligned}
$$

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\mathcal{R}_{s,a} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r]
\end{aligned}
$$

策略梯度定理

$$E_x[f(x)\nabla_\theta \log p(x)]$$

- 将上页中的即时奖励r换成长期值$Q^\pi(s,a)$
- 则根据策略梯度定理，无论是上面说的三种目标函数（start state, average reward and average value）中的哪一种，它们的梯度都是一样的，如下所示：

Theorem

For any differentiable policy $\pi_\theta(s,a)$,
for any of the policy objective functions $J = J_1, J_{avR}$, *or* $\frac{1}{1-\gamma}J_{avV}$,
the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\overbrace{\nabla_\theta \log \pi_\theta(s,a)}^{\text{score function}}\ Q^{\pi_\theta}(s,a)\right]$$

如何评价策略？

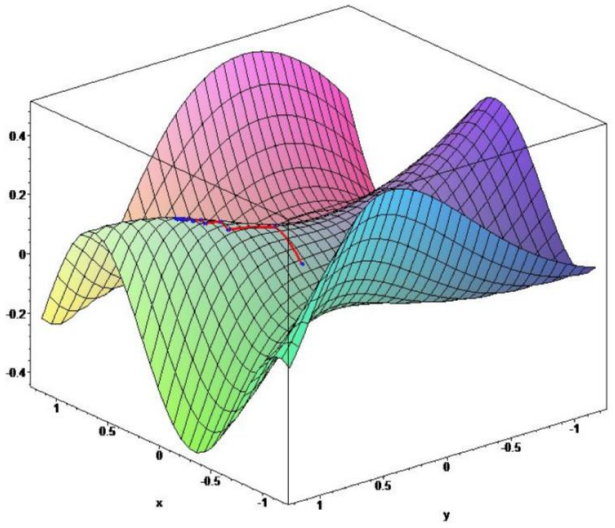
- 在 Policy Gradient 中，如何确定评价指标 f(s,a) 是关键。

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^{\pi}(s_t, a_t)$: state-action value function.
5. $A^{\pi}(s_t, a_t)$: advantage function.
6. $r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$: TD residual.

- 简单的方法是：根据回合的输赢来判断这个回合中的每一步到底是好是坏。
- 但其实更好的方法是：每走一步就能够获取到这一步的具体评价，因此出现了很多其他的直接给出某个时刻的评估的评价方式→AC。

求解目标函数的极值

- 找到的是目标函数，接下来就是如何优化目标函数，因此策略梯度就是一个优化问题。
- 策略梯度(Policy Gradient)
  - 令$J(\theta)$为任何类型的策略目标函数，策略梯度算法可以使$J(\theta)$沿着其梯度上升至局部最大值。
  - 其中，$\Delta\theta = \alpha\nabla_\theta J(\theta)$是策略梯度：



$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

有限差分策略梯度

- 上面策略梯度的算法可能比较复杂，特别是梯度函数本身很难得到的时候。
- 具体做法：就是对θ的每个分量都单独求差分（增加一个微小的扰动）来估算梯度：

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

微小扰动

  - 其中，u_k是一个单位向量，只有在第k维上的值为1，其它为0。
- 这种算法简单，而且不需要策略函数是可导的，可以用于任意的策略。
- 但它有噪声，而且很多时候效率也不高。

策略的表现形式：softmax策略

- Softmax策略是针对一些具有离散的行为常用的一个策略。其实就是针对action是离散的情况，softmax就可以输出在某个状态下所有可能执行的动作的概率。
- 把行为看成是多个特征加权线性代数和：$\phi(s,a)^T\theta$；
- 而采取某一具体行为的概率与e的该值次幂成正比：

$$\pi_\theta(s,a) \propto e^{\phi(s,a)^\top \theta}$$

S状态下，依策略做出的所有动作的均值

- 其score function为：

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}\left[\phi(s,\cdot)\right]$$

- 其中，等号右边第一部分是采取某行为的Score，第二部分是当前状态的期望分值。

策略的表现形式：高斯策略

- 高斯策略用于连续的动作空间。比如，如果控制机器人行走要调整流经某个电机的电流值，而这个电流值是连续值。
- 高斯策略就需要用到均值和方差，
  - 一般对于均值来说会用参数化来表示，例如用线性组合：

$$\mu(s) = \phi(s)^{\top}\theta$$

  - 方差可以使用固定值，也可以像均值那样参数化。

策略的表现形式：高斯策略

- 行为(action)是对应一个具体的数值，这个数值就是从以μ(s)为均值，σ为标准差的高斯分布中随机采样产生的：

$$a \sim \mathcal{N}(\mu(s), \sigma^2)$$

- 所以score function就变成如下公式：
- 其实就是对高斯函数进行求导就得到

$$\nabla_\theta \log \pi_\theta(s,a) = \frac{(a-\mu(s))\phi(s)}{\sigma^2} \qquad f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

MCPG的算法（REINFORCE）

- 原始的 Policy Gradient 往往采用的回合更新，也就是要到一轮结束后才能进行更新。其更新过程如下：

  - Update parameters by stochastic gradient ascent
  - Using policy gradient theorem
  - Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

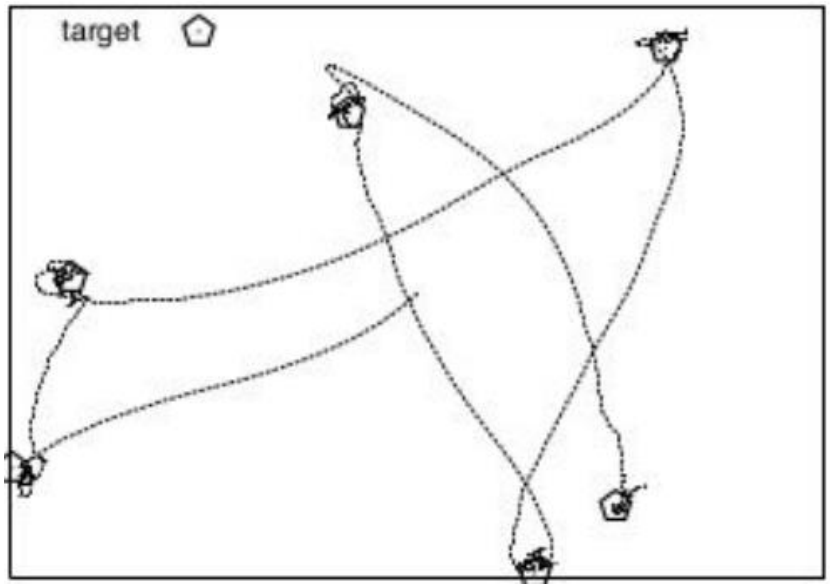
$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

这里用的是v

function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

蒙特卡罗策略梯度：Puck世界示例

- 在区域里追踪目标的例子：有一个五边形的目标物体，同时还有一个Agent。
- 状态空间：个体观察自己的位置(x, y),速度(v_x, v_y)以及目标物体（图中的五边形）的位置（t_x, t_y），共6个特征。



蒙特卡罗策略梯度：Puck世界示例

- 行为空间：个体控制自己在上、下、左、右四个方向上的油门（速率的增量）和不操作，共5个行为。
- 环境动力学：将个体的行为转化为其速度和位置的变化。目标物体出现位置随机，每隔一段时间更新位置。
- 奖励：奖励值的大小基于个体与目标物体之间的距离，距离越小奖励越大。
- 用蒙特卡洛策略梯度算法收敛速度慢，需要的迭代次数长，还存在较高的变异性。

AlphaGo

- 引爆最近一轮AI热潮的，就是AlphaGo及其相关成果。

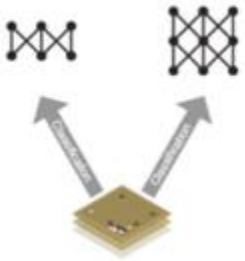
- 它应用了DRL技术将围棋智能水平达到了一个全新的高度。2013年时，人们认为计算机要战胜职业棋手至少还需10年以上，但AlphaGo让它提前了好多年实现。
- AlphaGo虽然也基于DRL，但其意义与之前基于DQN的工作大相径庭，否则《Nature》也不会让它发两次。
  - 如果说之前的最开始提出的DQN还是一种算法的话，AlphaGo这时就已经是一套为围棋精心设计的算法框架了。
  - 还有，DQN的意义在于算法具有通用性，而AlphaGo的意义在于围棋问题本身的搜索空间非常之大，同时也是经典的NP-hard问题。

减少搜索空间

- 搜索空间的有效减少，可以采用两种通用的原则。
- 第一，搜索的深度可以通过棋局评估降低：在状态 s 时对搜索树进行剪枝，然后用一个近似估值函数$v(s) \approx v^*(s)$ 取代状态s下面的子树，这个近似估值函数预测状态 s 之后的对弈结果。
- 第二，搜索的广度可以通过来自策略 $p(a \mid s)$ 的采样动作来降低，这个策略是一个在位置 s 的可能下棋走子a 概率分布。

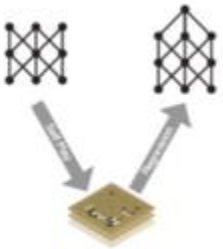
AlphaGo的架构

监督学习（模仿学习）



+

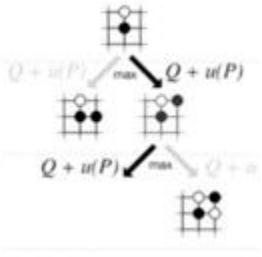
增强学习（自主学习）



AlphaGo

+

MCTS（人工设定）



整体架构

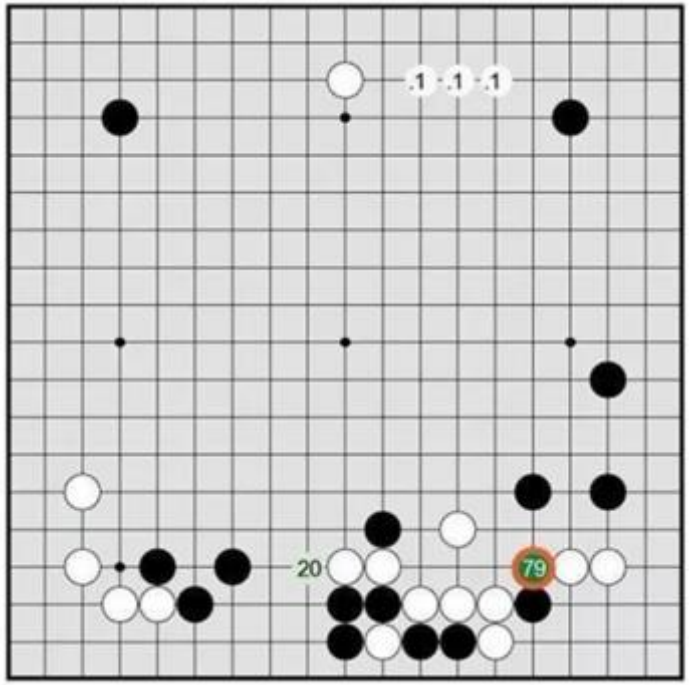
AlphaGo 流程

- 基于人类专家的数据集通过监督学习训练SL policy network。另外训练一个单独的策略用于在之后的rollout中快速选取动作。这一步是为了结合人类已有经验，用于后面几步作先验知识。
- 接着，将SL policy network的参数来初始化RL policy network 是通过RL的PG方法学习策略，通过随机梯度上升算法来更新RL policy network参数。
- value network用于预测RL policy network玩游戏时的胜算，这一步使用随机梯度下降（SGD）来最小化outcome（对当前状态而言围棋结束时的回报，即输赢）与V函数之间的差（MSE）。
- 有了上面的policy network和value network，就可以据此用MCST来进行搜索了。

MCTS

- 蒙特卡洛树，其实就是把它蒙特卡洛方法移到树结构上罢了。
- 它的思想也是通过进行大量的模拟（rollouts），最后想要将答案收敛到某个分支。

- 没有落子的地方都是可能下子的。
- 但在模拟中，右下那步被模拟的次数达到总数的79%，就选这步了。
- “模拟”次数“最多”的走法就是统计上“最优”的走法。



AlphaGo

- AlphaGo通过蒙特卡洛搜索树（Monte-Carlo Tree Search，MCTS）和DRL结合使得这个大规模搜索问题在可计算范围内。
- 其实AlphaGo更多是个工程上的杰作。它基于前面的DRL基础上，做了很多针对围棋问题的特有技术，且用了分布式的技术加速计算。
- 简单概括一下AlphaGo的基本原理：在MCTS的框架下引入两个卷积神经网络policy network和value network以改进纯随机的Monte Carlo模拟，并借助监督学习和强化学习训练这两个网络。

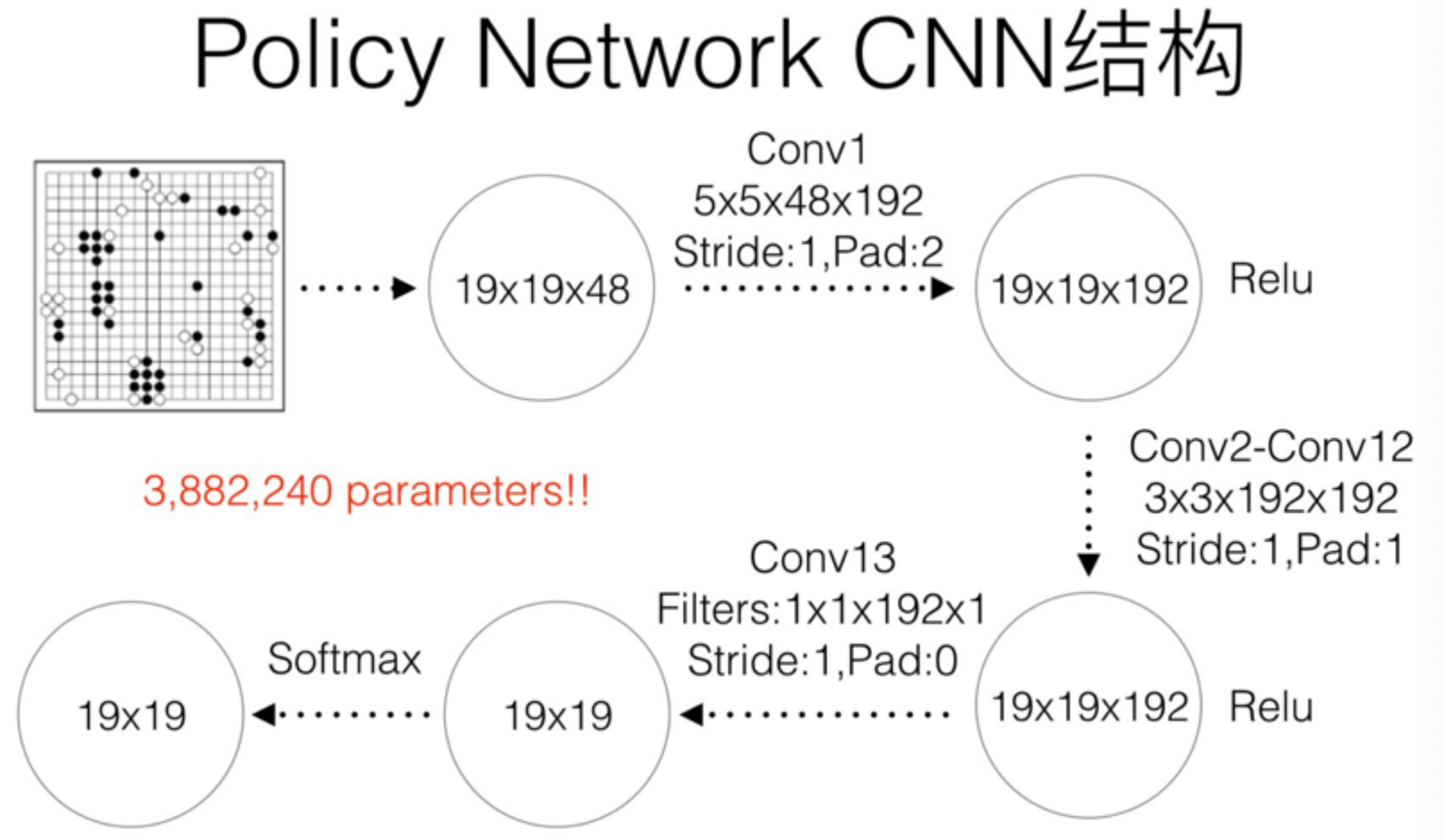
有监督学习的Policy Networks

- 有监督学习策略网络p_σ：
  - 通过有监督学习（SL），让神经网络学习专业选手的走子，这个训练通过立即的反馈和高质量的梯度提供了快速有效的学习更新。
- 经过人类高手三千万步围棋走法的训练后，SL policy network模拟人类落子的准确率已经达到了57%；相应地，网络的棋力也得到大大的提升。
- 但是，如果直接用这个网络与人类高手，甚至是MCTS的程序进行对弈，依然是输面大。

SL策略网络

- SL策略网络是一个13层的深度卷积神经网络，网络的输入是由48个特征平面组成的19 x 19 x 48(通道)的图层。
- 第1个隐藏层把输入处理为23 x 23的图像，边界补0。然后用核为5 x 5，跨度为1的k个滤波器对图像进行卷积。
- 第2-12个隐藏层将前面的输出图像补0成21 x 21的图像，同样用k个滤波器进行卷积（核3 x 3，步长为1），详见下一页。
- 最后的输出应用softmax函数。

策略网络的架构


Policy Network CNN结构
Conv1
5x5x48x192
Stride:1,Pad:2
19x19x48
19x19x192
Relu
3,882,240 parameters!!
Conv2-Conv12
3x3x192x192
Stride:1,Pad:1
Conv13
Filters:1x1x192x1
Stride:1,Pad:0
19x19x192
Relu
Softmax
19x19
19x19

AlphaGo的输入

- 输入：19*19*48层的特征。

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

策略网络的训练细节

- Cost：交叉熵
- 训练：50个GPU，3周时间
- 训练步数：340,000,000
- Batchsize：16
- LR：初始为0.003，80,000,000步后为0.0015

- 训练用的数据是从KGS服务器上弄下来的2940万个棋盘的盘面数据。

有监督学习的Policy Networks

- 但p_σ这个网络的走子太慢了，平均每步3ms的响应时间，使得这个网络很难被直接用于MCTS的rollout中进行策略的随机。
- 为此，deepmind团队通过提取一些pattern features又训练了一个更快速（响应时间达到了2μs）但准确率有所降低（24.2%）的rollout policy network： p_π。
- p_π是可以快速决策的策略网络：
  - 是为了让在下棋时可以更快地给出走子策略，
  - 大型网络比较耗时，但质量上p_π比p_σ要差一些。

AlphaGo的架构

- 用强化学习训练策略网络p_ρ：
  - 使用增强学习方法，通过自我对弈来优化策略网络的目的——从提高预测人类走子到提高胜率。
- 训练价值网络v_θ：
  - 这个神经网络是用来预测一个盘面下，使用RL策略网络自我对弈的胜率。
- MCTS：
  - 将策略网络与价值网络有效结合起来，对最优方案进行搜索。

强化学习的Policy Networks

- 为了进一步提高policy network的对弈能力，又采用一种policy gradient 技术，训练了一个RL policy network：p_ρ。
- 这个网络的结构与SL policy network的网络结构相同，依然是一个输出为给定状态下落子概率的卷积神经网络。
- 网络的参数被初始化为p_σ的参数；接下来，通过不断地与历史版本进行自我对弈，网络的权重向着收益最大化的方向进化。
- 此时，网络的学习目标不再是模拟人类的走法，而是更为终极的目标：赢棋。

强化学习的Policy Networks

- 具体来说，我们定义了一个reward function $r(s_t)$：对于非终止的时间步$t < T_t < T$，总有$r(s_t)=0$。
- 每一步的收益$z(t)$被定义为$\pm r(s_T)$：即对当前玩家而言对弈的最终结果（+1代表赢棋；−1代表输棋）。
- 网络的权重通过随机梯度上升法进行调整：

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

- 通过这种方式训练出来的RL policy network，在与SL policy network对弈时已有80%的赢面。即便是与依赖Monte Carlo搜索的围棋博弈程序相比，不依赖任何搜索的RL policy network，也已经达到了85%的赢面。

强化学习的Value Networks

- 然后，DeepMind团队又开始寻求一个能快速预估棋面价值（棋势）的Value Network。
- 一个盘面的价值函数$v^p(s)$，被定义为在给定的一组对弈策略p的情况下，从状态s出发，最终的期望收益（也即赢棋的概率）：
- 然而，我们并不知道什么才是最优的策略。因此，在实际应用中，DeepMind团队采用了当前最强的策略函数p_ρ（RL policy network）来计算一个棋面的价值$v^{p_\rho}(s)$，并训练了一个value network $v_\theta(s)$ 来拟合这个价值函数：$v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$。

强化学习的Value Networks

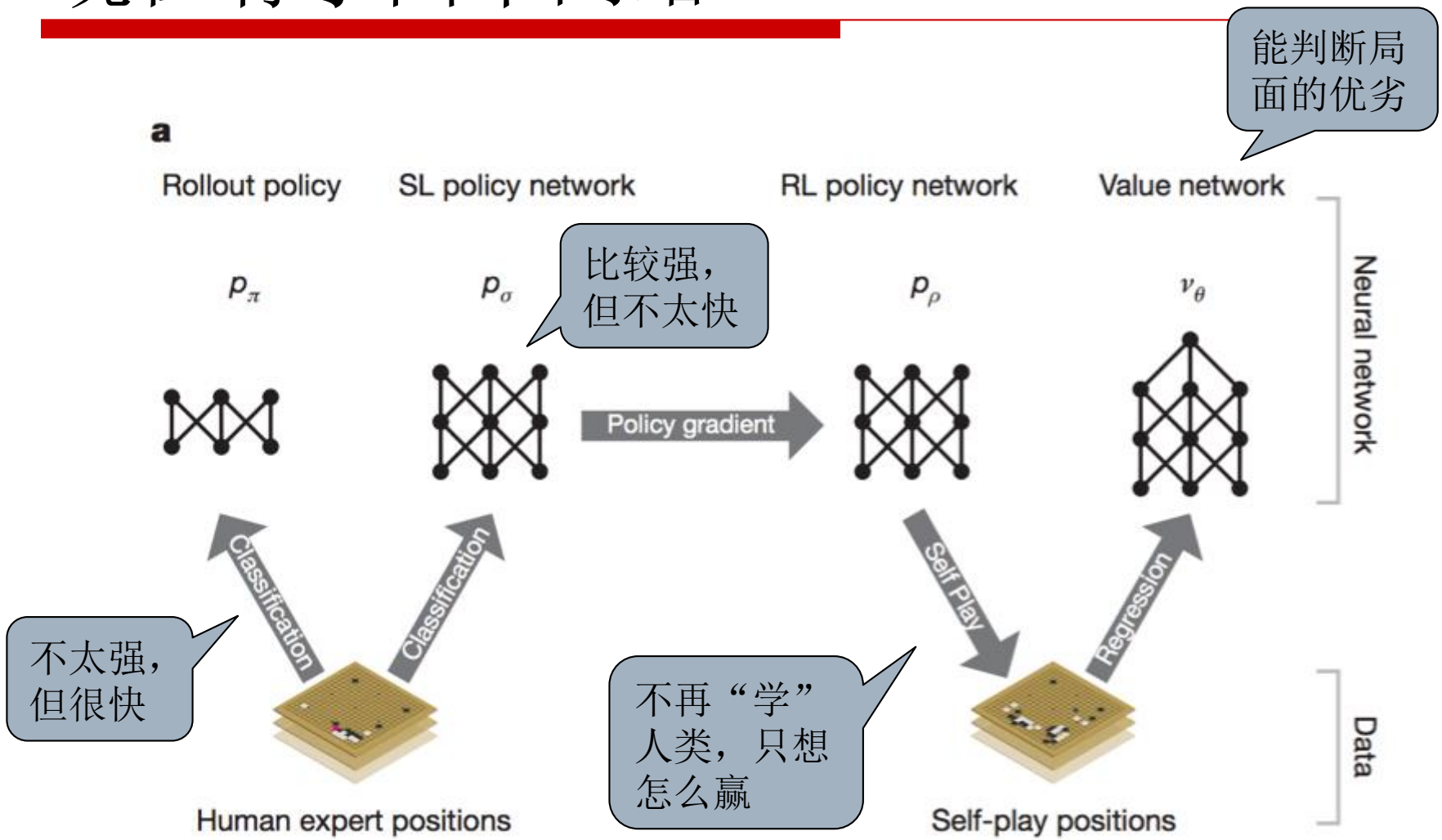
- Value Network的网络结构与前面的Policy Network类似，也是一个卷积神经网络，只是输出层变成了一个单神经元的标量。
- 可以通过构造一组(s, z)的训练数据，并用随机梯度下降法最小化网络的输出$v_\theta(s)$与目标收益z的均方差，来调整网络的参数：

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

强化学习的Value Networks

- 在构造训练数据时有一些技巧。如果我们从人类对弈的完整棋局中抽取足够数量的训练数据，很容易出现过拟合的问题。
- 这是因为在同一轮棋局中的两个棋面的相关性很强（往往只相差几个棋子）；此时，网络很容易记住这些棋面的最终结果，而对新棋面的泛化能力很弱。
- 为了解决这个问题，需要通过RL policy network的自我对弈，产生了三千万个从不同棋局中提取出来的棋面－收益组合的训练数据。

现在有了四个网络

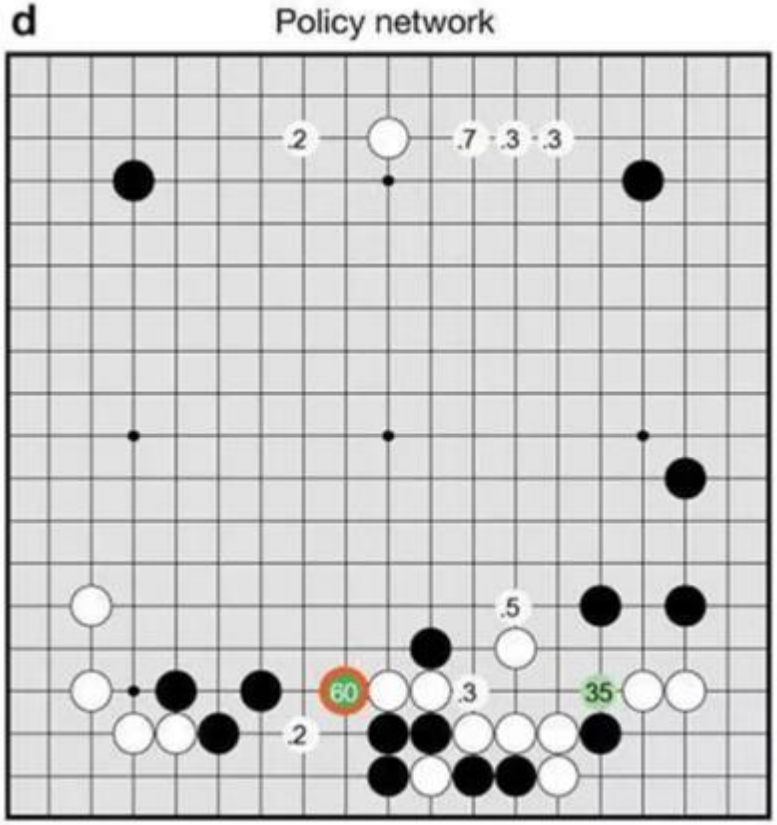

能判断局面的优劣
a
Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
比较强，但不太快
Policy gradient
Neural network
Classification
Classification
Self Play
Regression
不太强，但很快
不再“学”人类，只想怎么赢
Human expert positions
Self-play positions
Data

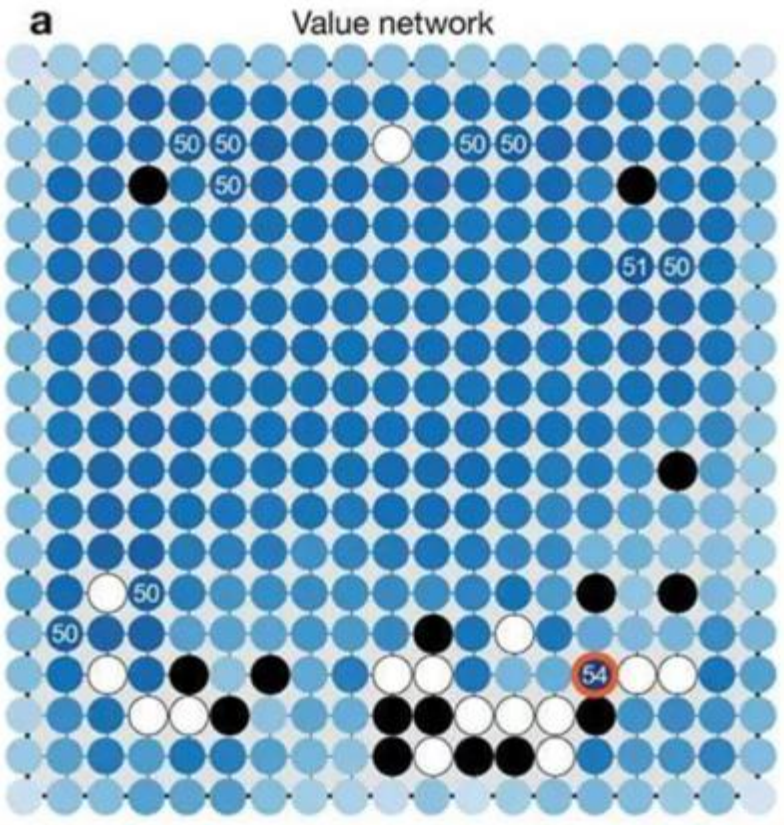
AlphaGo的网络说明

- 策略网络把棋局状态 s 当作输入，策略网络把 s 传输通过很多卷积层（这些卷积层是参数为σ的SL策略网络或者参数为ρ 的RL策略网络），然后输出一个关于下棋动作 a 的概率分布 $p_\sigma(a \mid s)$ 或 $p_\rho(a \mid s)$，用一个棋盘的概率地图来表示。
- 估值网络类似的使用了很多参数θ 的卷积层，但是输出一个标量值 $v_\theta(s')$ 用来预测棋局状态 s′后的结局。

策略网络与价值网络

- 策略网络“眼中”的棋局，和价值网络“眼中”的棋局。





AlphaGo 架构与流程

- 在最后一个阶段中，算法会通过MC模拟从根状态来往下单向遍历搜索树，这一阶段又可以分为四步。
- 这其实也是MCST算法框架中的经典的四个步骤。只是AlphaGo在这个MCTS算法中结合了policy和value network。

  - a、选取（selection）；b、扩展（expansion）；
  - c、评估（evaluation）；d、回溯（backup）。

MCTS的四个步骤

- 1. 选择（Selection）：从根节点开始，不断递归选择最优的子节点，直到达到一个叶结点。
- 根据公式选择最优子节点：$$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$$
  - 其中v_i是节点估计的值，n_i是节点被访问的次数，而N则是其父节点已经被访问的总次数，C是一个常量。
- 2. 扩展（Expansion）：当前节点不是终止节点（游戏未结束），并且访问该节点次数超过某个设定的阈值，则在创建一个或多个子节点。
- 3. 模拟（Simulation）：从当前叶结点开始，通过随机获采用某个走子策略p，模拟走子，直到游戏结束。
- 4. 回传（Back propagation）：根据模拟的最终结果，反向更新搜索树。

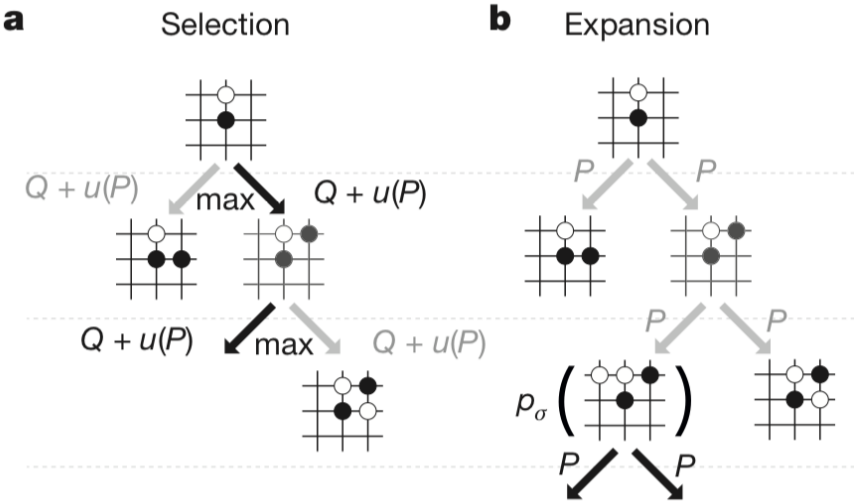
AlphaGo中的蒙特卡洛树搜索

- AlphaGo在把策略网络、估值网络和MCTS算法结合，MCTS通过预测搜索选择下棋动作。
- 每一个搜索树的边(s, a)存储着一个动作估值Q(s, a)，访问计数 N(s, a)和先验概率 P(s, a)。
- 这棵树从根节点开始，通过模拟来遍历（比如在完整的博弈中沿着树无没有备份地向下搜索）。

AlphaGo中的蒙特卡洛树搜索

- a 每一次模拟遍历搜索树，通过选择拥有最大下棋动作估值Q的边，加上一个额外奖励u(P)（依赖于存储的该边的先验概率P）。
- b叶节点可能被展开，新的结点被策略网络p_θ执行一次，然后结果概率存储下来作为每一个下棋动作的先验概率。

$$u(s,a) \sim \frac{P(s,a)}{1+N(s,a)}$$



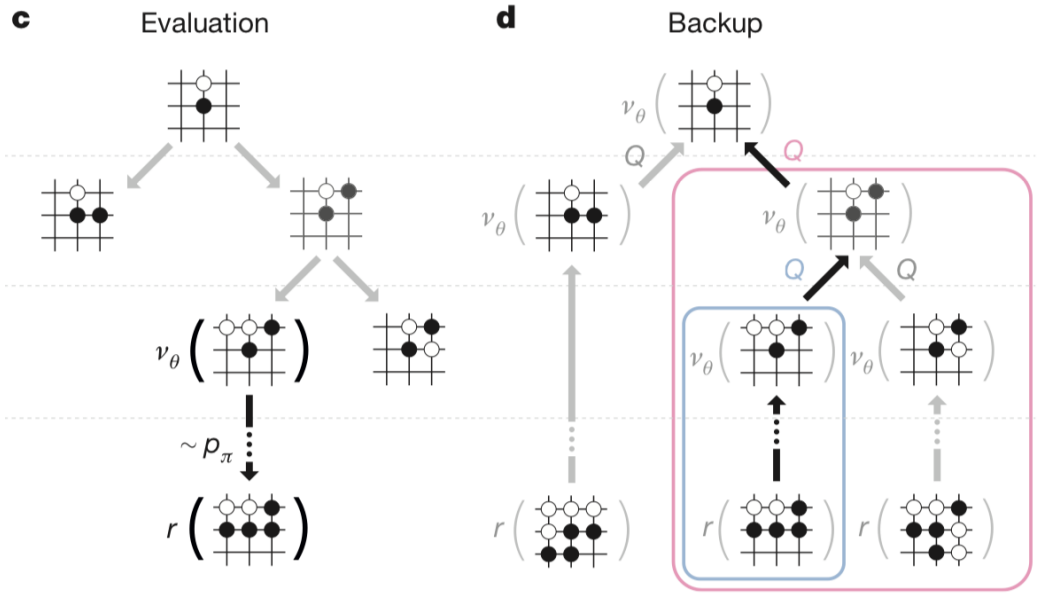
AlphaGo中的蒙特卡洛树搜索

- c在一次模拟的结尾，叶节点由两种方式评估：使用估值网络v_θ；运行一个走子策略直到博弈结尾（使用了p_π），然后对于赢者运算函数r。
- d下棋动作估值Q得到更新，用来跟踪所有评估r(.)的平均值和该下棋动作的子树的$v_\theta(\cdot)$

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

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

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

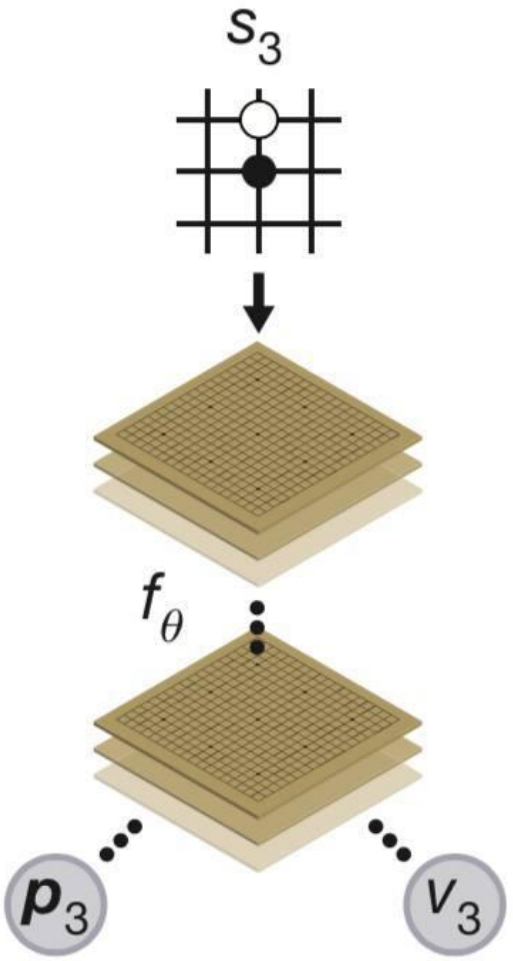


AlphaGo Zero

- **神经网络权值完全随机初始化。**AlphaGo Zero不利用任何人类专家的经验或数据，随机初始化神经网络的权值进行策略选择。
- **无需先验知识。**AlphaGo Zero不再需要人工设计特征，而是仅利用原始棋盘情况输入到神经网络中，以此得到结果。
- **神经网络结构复杂性降低。**AlphaGo Zero将原先两个结构独立的策略网络和价值网络合为一体，合并成一个。
- **舍弃快速走子网络。**AlphaGo Zero不再使用快速走子网络替换随机模拟，而是完全将神经网络得到的结果替换为随机模拟。
- **神经网络引入残差结构。**AlphaGo Zero的神经网络采用基于残差网络结构的模块进行搭建，用更深的神经网络进行特征表征提取，从而在更加复杂的棋盘局面中进行学习。
- **硬件资源需求更少。**AlphaGo Lee需176块GPU和48块TPU，而到现在的AlphaGo Zero只需要单机4块TPU便可完成。

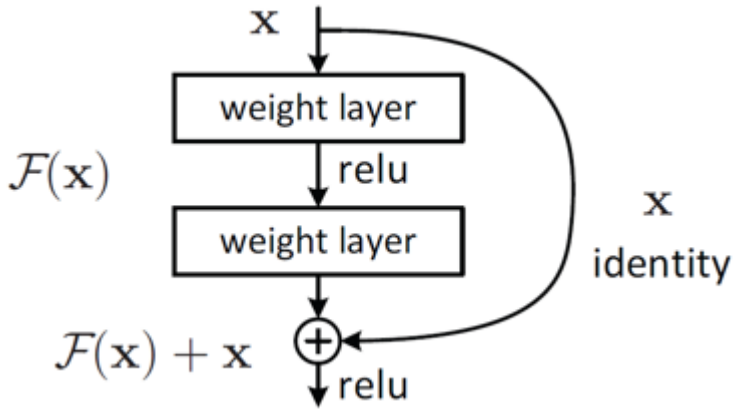
AlphaGo Zero：双头架构

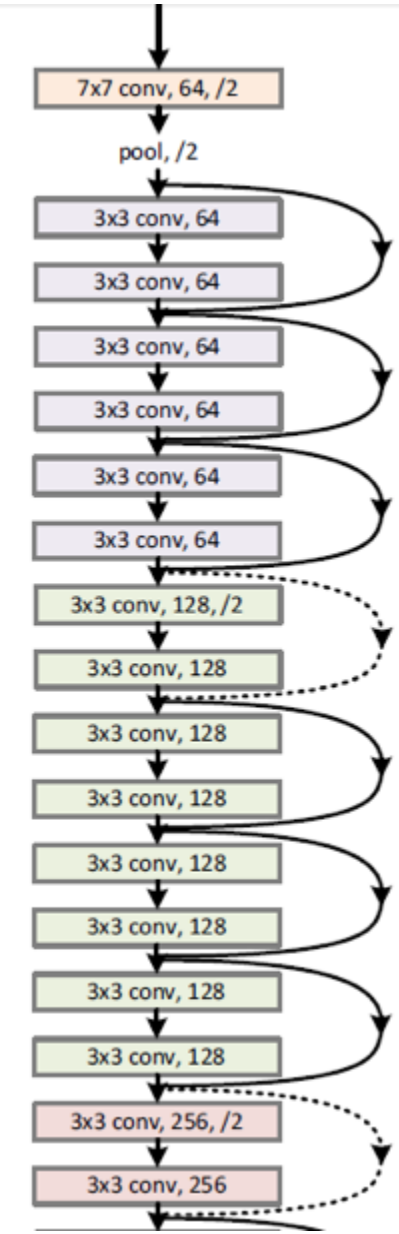
- AlphaGo Zero的深度神经网络架构，是一个“双头”架构。
- 它的前20层是常见的神经网络结构。这些层的后面接着是“两个头”：
  - 一个头利用前20层的输出，产生下一步落子获胜的概率，
  - 另一个头利用前20层的输出，推理出当前玩家在此情形下最终获胜的概率。



AlphaGo Zero：残差网络

- 神经网络采用微软的ResNet结构。利用两层的ResNet残差学习模块。
- 输入特征包括：历史特征+自己当前特征+对手当前特征





AlphaGo Zero

- 1.初始化神经网络。
- 2.进行自我对弈，每一步进行1600次MCTS模拟（大约需要0.4秒）。
- 3.随着自我对弈的进行，从最近的50万场比赛中抽取2048个落子位置以及比赛的输赢情况作为样本。
- 4.使用MCTS向前搜索，对着法进行评估，训练神经网络。
- 5.对步骤3、4每1000次进行迭代，评估当前的神经网络与以前的最佳版本;如果胜率达到55%，就使用新的神经网络生成游戏，摒弃以前的版本。
- 重复第3、4步70万次，而自我对弈游戏不断进行——三天后，AlphaGo Zero诞生！

相关论文

- 《Mastering the game of Go with deep neural networks and tree search》
- 《Mastering the game of Go without human knowledge》

参考资料

- https://zhuanlan.zhihu.com/p/28348110
- https://blog.csdn.net/liweibin1994/article/details/79348000
- http://wulc.me/2018/05/11/强化学习笔记(3)-%20从%20Policy%20Gradient%20到%20A3C/
- https://zhuanlan.zhihu.com/p/21725498
- https://www.sohu.com/a/219278000_610300

问答互动

在所报课的课程页面，
1、点击“全部问题”显示本课程所有学员提问的问题。
2、点击“提问”即可向该课程的老师和助教提问问题。

联系我们

小象学院：互联网新技术在线教育领航者

– 微信公众号：**小象学院**