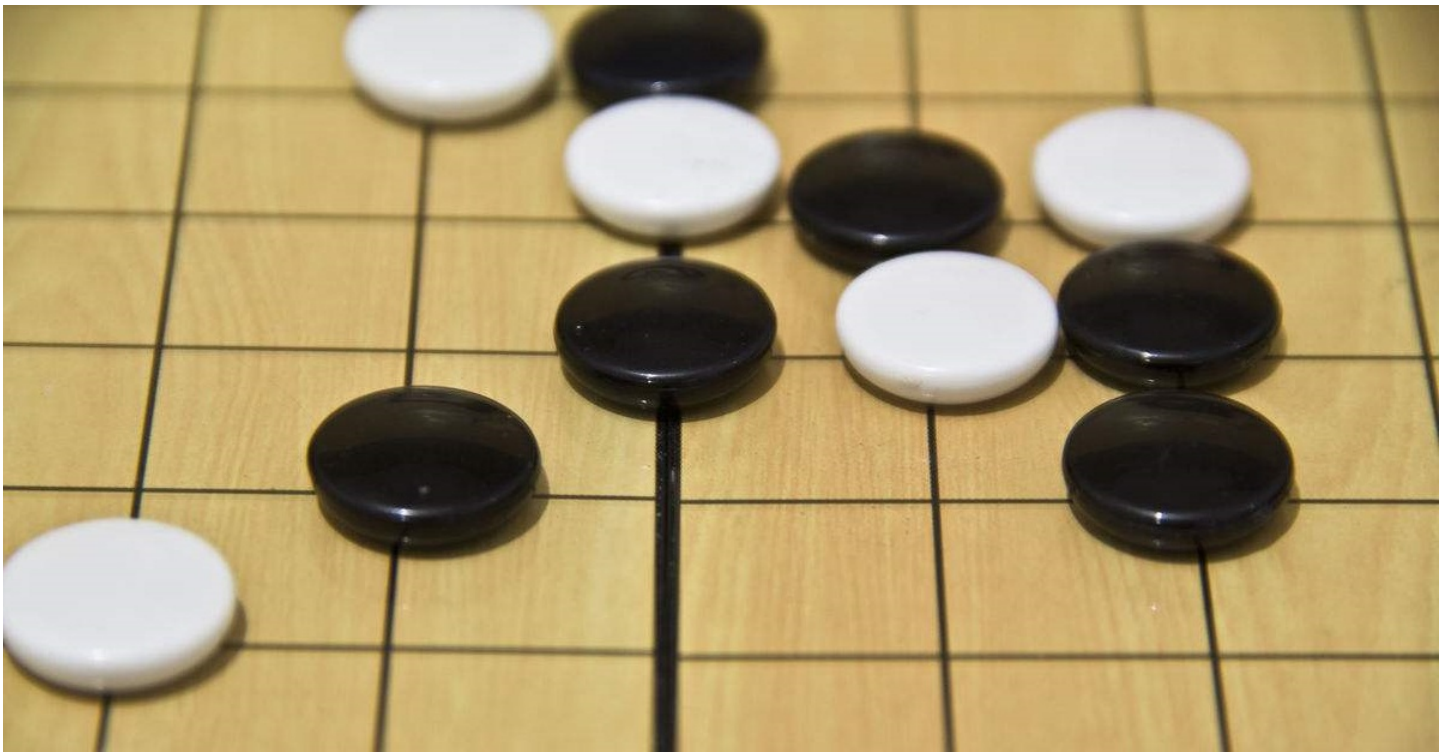
知乎 首发于 强化学习知识大讲堂



AlphaZero实战：从零学下五子棋（附代码）

一缕阳光
优化算法、机器学习

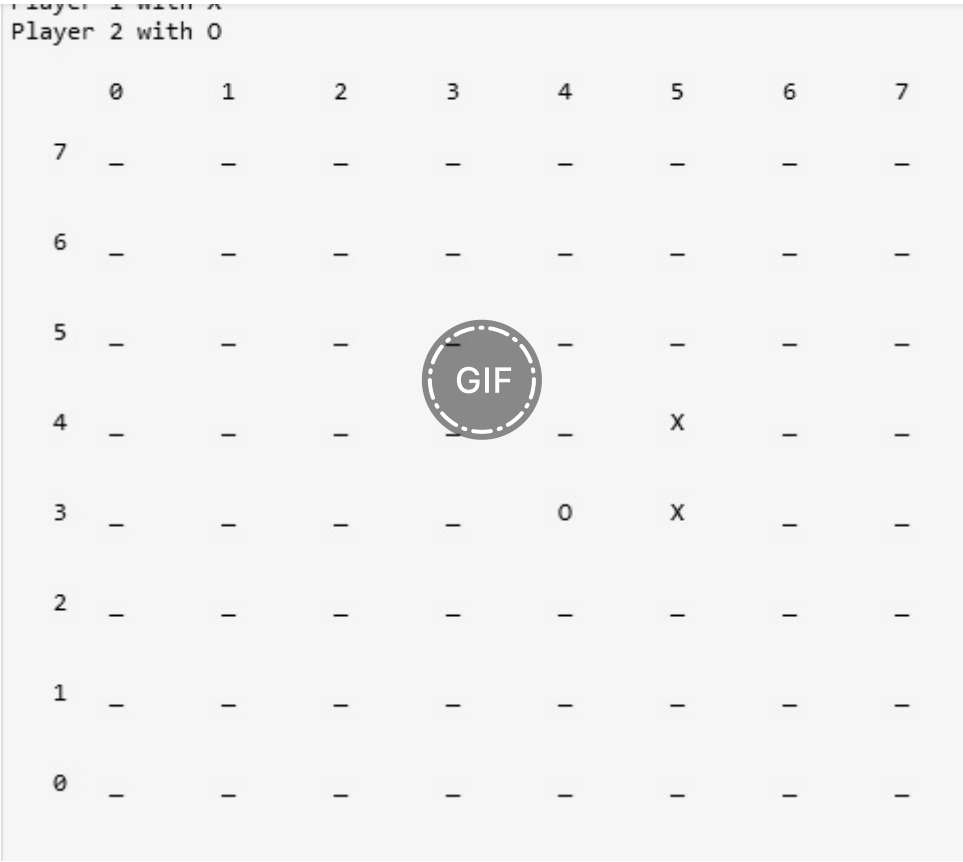
535 人赞了该文章

2017年10月，AlphaGo Zero横空出世，完全从零开始，仅通过自我对弈就能天下无敌，瞬间刷爆朋友圈，各路大神分分出来解读，惊叹于其思想的简单、效果的神奇。很快就有大神放出了开源版的AlphaGo Zero，但是只有代码，没有训练出来的模型，因为据大神推算，在普通消费级的电脑上想训练出AlphaGo Zero的模型需要**1700年！**然而DeepMind在AlphaGo Zero的论文里只强调运行的时候需要4个TPU，而完全没有提及训练过程的最大计算需求在于生成self-play数据，还引起了一点小争议。还好，过了不到两个月，在12月初，DeepMind就在Arxiv上低调放出了更加通用的AlphaZero的论文。AlphaZero几个小时就征服围棋、国际象棋和日本将棋的壮举再次惊叹世人，但同时DeepMind大方公开的self-play阶段使用的5000个TPU也让大家纷纷感叹，原来是**“贫穷限制了我们的想象力”！**

扯得有点远了，让我们回到这篇文章的正题：**AlphaZero实战**，通过自己动手从零训练一个AI，去体会AlphaZero自我对弈学习成功背后的关键思想和一些重要技术细节。这边选择了五子棋作为实践对象，因为五子棋相对比较简单，大家也都比较熟悉，这样我们能更专注于AlphaZero的训练过程，同时也能通过亲自对阵，来感受自己训练出来的AI慢慢变强的过程。经过实践发现，对于在6*6的棋盘上下4子棋这种情况，大约通过500~1000局的self-play训练（2小时），就能训练出比较靠谱的AI；对于在8*8的棋盘上下5子棋这种情况，通过大约2000~3000局自我对弈训练（2天），也能得到比较靠谱的AI。所以虽然贫穷，但我们还是可以去亲身感受最前沿成果的魅力！完整代码以及4个训练好的模型已经上传到了github：github.com/junxiaosong/...

```
AI move: 4,2

Player 1 with X
Player 2 with O

      0       1       2       3       4       5       6       7

   7  _       _       _       _       _       _       _       _

   6  _       _       _       _       _       _       _       _

   5  _       _       _       _       _       _       _       _

   4  _       _       X       _       _       _       _       _

   3  _       _       _       _       _       _       _       _

   2  _       _       _       _       _       _       _       _

   1  _       _       _       _       _       _       _       _

   0  _       _       _       _       _       _       _       _
```

每一步棋执行400次MCTS模拟

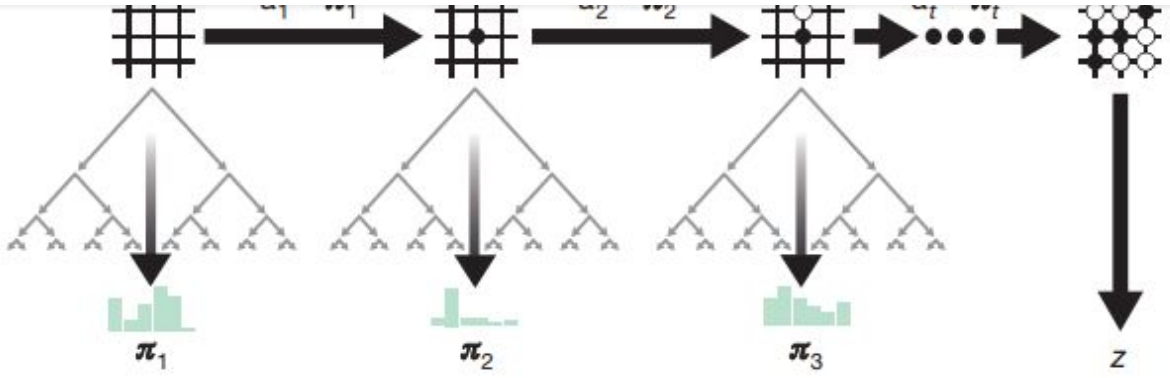


每一步棋执行800次MCTS模拟

从上面的对局样例可以看到，AI已经学会了怎么下五子棋，知道什么时候要去堵，怎么样才能赢，按我自己对阵AI的感受来说，要赢AI已经不容易了，经常会打平，有时候稍不留神就会输掉。这里有一点需要说明，上面展示的两局AI对弈中，AI执行每一步棋的时候分别只执行了400次和800次MCTS模拟，进一步增大模拟次数能够显著增强AI的实力，参见AlphaZero论文中的Figure 2（注：AlphaZero在训练的时候每一步只执行800次MCTS simulations，但在评估性能的时候每一步棋都会执行几十万甚至上百万次MCTS模拟）。

下面，我结合AlphaZero算法本身，以及github上的具体实现，从自我对局和策略价值网络训练两个方面来展开介绍一下整个训练过程，以及自己实验过程中的一些观察和体会。

自我对局（self-play）



self-play过程示意图

完全基于self-play来学习进化是AlphaZero的最大卖点，也是整个训练过程中最关键也是最耗时的环节。这里有几个关键点需要说明：

1. 使用哪个模型来生成self-play数据?

在AlphaGo Zero版本中，我们需要同时保存当前最新的模型和通过评估得到的历史最优的模型，self-play数据始终由最优模型生成，用于不断训练更新当前最新的模型，然后每隔一段时间评估当前最新模型和最优模型的优劣，决定是否更新历史最优模型。而到了AlphaZero版本中，这一过程得到简化，我们只保存当前最新模型，self-play数据直接由当前最新模型生成，并用于训练更新自身。直观上我们可能会感觉使用当前最优模型生成的self-play数据可能质量更高，收敛更好，但是在尝试过两种方案之后，我们发现，在6*6棋盘上下4子棋这种情况下，直接使用最新模型生成self-play数据训练的话大约500局之后就能得到比较好的模型了，而不断维护最优模型并由最优模型生成self-play数据的话大约需要1500局之后才能达到类似的效果，这和AlphaZero论文中训练34小时的AlphaZero胜过训练72小时的AlphaGo Zero的结果也是吻合的。个人猜测，不断使用最新模型来生成self-play数据可能也是一个比较有效的exploration手段，首先当前最新模型相比于历史最优模型一般不会差很多，所以对局数据的质量其实也是比较有保证的，同时模型的不断变化使得我们能覆盖到更多典型的数据，从而加快收敛。

2. 如何保证self-play生成的数据具有多样性?

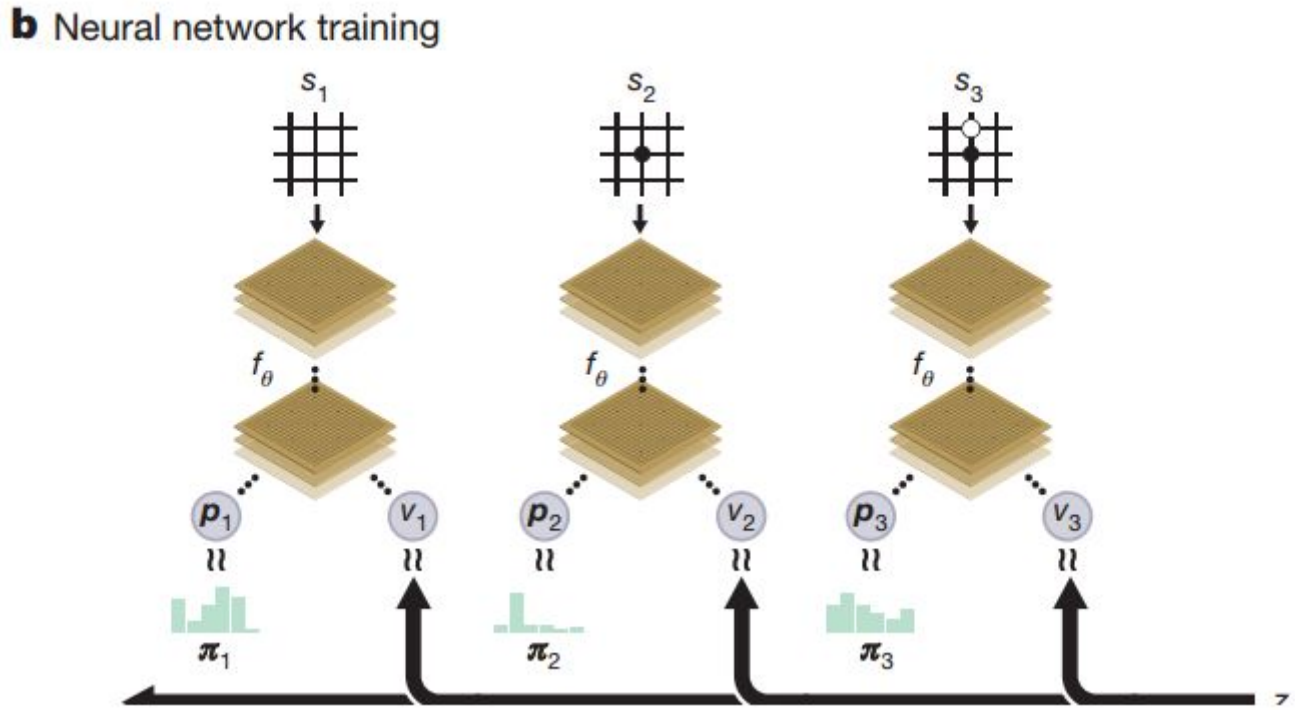
一个有效的策略价值模型，需要在各种局面下都能比较准确的评估当前局面的优劣以及当前局面下各个action的相对优劣，要训练出这样的策略价值模型，就需要在self-play的过程中尽可能的覆盖到各种各样的局面。前面提到，不断使用最新的模型来生成self-play数据可能在一定程度上有助于覆盖到更多的局面，但仅靠这么一点模型的差异是不够的，所以在强化学习算法中，一般都会有特意设计的exploration的手段，这是至关重要的。在AlphaGo Zero论文中，每一个self-play对局的前30步，action是根据正比于MCTS根节点处每个分支的访问次数的概率采样得到的（也就是上面Self-play示意图中的 $a_t \sim \pi_t$ ，有点类似于随机策略梯度方法中的探索方式），而之后的exploration则是通过直接加上Dirichlet noise的方式实现的（ $P(s,a) = (1-\varepsilon)P_a + \varepsilon\eta_a$,有点类似于确定性策略梯度方法中的探索方式）。在我们的实现中，self-play的每一步都同时使用了这两种exploration方式，Dirichlet noise的参数取的是0.3，即 $\eta \sim \mathbf{Dir}(0.3)$,同时 $\varepsilon = 0.25$.

在self-play过程中，我们会收集一系列的 (s, π, z) 数据， s 表示局面， π 是根据MCTS根节点处每个分支的访问次数计算的概率， z 是self-play对局的结果，其中 s 和 z 需要特别注意从每一步的当前player的视角去表示。比如 s 中用两个二值矩阵分别表示两个player的棋子的位置，那么可以是第一个矩阵表示当前player的棋子位置，第二个矩阵表示另一个player的棋子位置，也就是说第一个矩阵会交替表示先手和后手player的棋子位置，就看 s 局面下谁是当前player。 z 也类似，不过需要在一个完整的对局结束后才能确定这一局中每一个 (s, π, z) 中的 z ，如果最后的胜者是 s 局面下的当前player，则 $z = 1$,如果最后的败者是 s 局面下的当前player，则 $z = -1$,如果最后打平，则 $z = 0$.

4. self-play数据的扩充

围棋具有旋转和镜像翻转等价的性质，其实五子棋也具有同样的性质。在AlphaGo Zero中，这一性质被充分的利用来扩充self-play数据，以及在MCTS评估叶子节点的时候提高局面评估的可靠性。但是在AlphaZero中，因为要同时考虑国际象棋和将棋这两种不满足旋转等价性质的棋类，所以对于围棋也没有利用这个性质。而在我们的实现中，因为生成self-play数据本身就是计算的瓶颈，为了能够在算力非常弱的情况下尽快的收集数据训练模型，每一局self-play结束后，我们会把这一局的数据进行旋转和镜像翻转，将8种等价情况的数据全部存入self-play的data buffer中。这种旋转和翻转的数据扩充在一定程度上也能提高self-play数据的多样性和均衡性。

策略价值网络训练



策略价值网络训练示意图

所谓的策略价值网络，就是在给定当前局面 s 的情况下，返回当前局面下每一个可行action的概率以及当前局面评分的模型。前面self-play收集到的数据就是用来训练策略价值网络的，而训

1. 局面描述方式

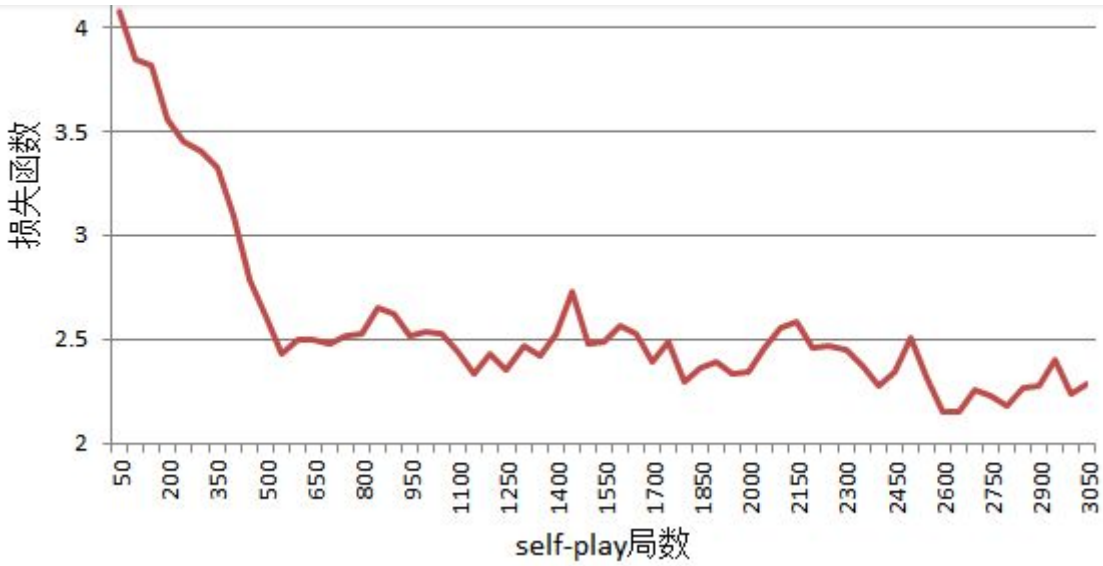
在AlphaGo Zero中，一共使用了17个 19×19 的二值特征平面来描述当前局面，其中前16个平面描述了最近8步对应的双方player的棋子位置，最后一个平面描述当前player对应的棋子颜色，其实也就是先后手。在我们的实现中，对局面的描述进行了极大的简化，以 8×8 的棋盘为例，我们只使用了4个 8×8 的二值特征平面，其中前两个平面分别表示当前player的棋子位置和对手player的棋子位置，有棋子的位置是1，没棋子的位置是0. 然后第三个平面表示对手player最近一步的落子位置，也就是整个平面只有一个位置是1，其余全部是0. 第四个平面，也就是最后一个平面表示的是当前player是不是先手player，如果是先手player则整个平面全部为1，否则全部为0. 其实在最开始尝试的时候，我只用了前两个平面，也就是双方的棋子的位置，因为直观感觉这两个平面已经足够表达整个完整的局面了。但是后来在增加了后两个特征平面之后，训练的效果有了比较明显的改善。个人猜想，因为在五子棋中，我方下一步的落子位置往往会在对手前一步落子位置的附近，所以加入的第三个平面对于策略网络确定哪些位置应该具有更高的落子概率具有比较大的指示意义，可能有助有训练。同时，因为先手在对弈中其实是很占优势的，所以在局面上棋子位置相似的情况下，当前局面的优劣和当前player到底是先手还是后手十分相关，所以第四个指示先后手的平面可能对于价值网络具有比较大的意义。

2. 网络结构

在AlphaGo Zero中，输入局面首先通过了20或40个基于卷积的残差网络模块，然后再分别接上2层或3层网络得到策略和价值输出，整个网络的层数有40多或80多层，训练和预测的时候都十分缓慢。所以在我们的实现中，对这个网络结构进行了极大的简化，最开始是公共的3层全卷积网络，分别使用32、64和128个 3×3 的filter，使用ReLu激活函数。然后再分成policy和value两个输出，在policy这一端，先使用4个 1×1 的filter进行降维，再接一个全连接层，使用softmax非线性函数直接输出棋盘上每个位置的落子概率；在value这一端，先使用2个 1×1 的filter进行降维，再接一个64个神经元的全连接层，最后再接一个全连接层，使用tanh非线性函数直接输出 $[-1, 1]$ 之间的局面评分。整个策略价值网络的深度只有5~6层，训练和预测都相对比较快。

3. 训练目标

前面提到，策略价值网络的输入是当前的局面描述 s ，输出是当前局面下每一个可行action的概率 $\mathbf{p}$ 以及当前局面的评分 v ，而用来训练策略价值网络的是我们在self-play过程中收集的一系列的 (s, π, z) 数据。根据上面的策略价值网络训练示意图，我们训练的目标是让策略价值网络输出的action概率 $\mathbf{p}$ 更加接近MCTS输出的概率 π ,让策略价值网络输出的局面评分 v 能更准确的预测真实的对局结果 z 。从优化的角度来说，我们是在self-play数据集上不断的最小化损失函数：$\ell = (z - v)^2 - \pi^T \log \mathbf{p} + c||\theta||^2$,其中第三项是用于防止过拟合的正则项。既然是在最小化损失函数，那么在训练的过程中，如果正常的话我们就会观察到损失函数在慢慢减小。下图展示的是一次在 8×8 棋盘上进行五子棋训练的过程中损失函数随着self-play局数变化的情况，这次实验一共进行了3050局对局，损失函数从最开始的4点几慢慢减小到了2.2左右。



在训练过程中，除了观察到损失函数在慢慢减小，我们一般还会关注策略价值网络输出的策略（输出的落子概率分布）的entropy的变化情况。正常来讲，最开始的时候，我们的策略网络基本上是均匀的随机输出落子的概率，所以entropy会比较大。随着训练过程的慢慢推进，策略网络会慢慢学会在不同的局面下哪些位置应该有更大的落子概率，也就是说落子概率的分布不再均匀，会有比较强的偏向，这样entropy就会变小。也正是由于策略网络输出概率的偏向，才能帮助MCTS在搜索过程中能够在更有潜力的位置进行更多的模拟，从而在比较少的模拟次数下达到比较好的性能。下图展示的是同一次训练过程中观察到的策略网络输出策略的entropy的变化情况。

另外，在漫长的训练过程中，我们最希望看到的当然是我们训练的AI正在慢慢变强。所以虽然在AlphaZero的算法流程中已经不再需要通过定期评估来更新最优策略，在我们的实现中还是每隔50次self-play对局就对当前的AI模型进行一次评估，评估的方式是使用当前最新的AI模型和纯的MCTS AI（基于随机rollout）对战10局。pure MCTS AI最开始每一步使用1000次模拟，当被我们训练的AI模型10:0打败时，pure MCTS AI就升级到每一步使用2000次模拟，以此类推，不断地

- 经过**550**局，AlphaZero VS pure_MCTS **1000** 首次达到**10:0**
- 经过**1300**局，AlphaZero VS pure_MCTS **2000** 首次达到**10:0**
- 经过**1750**局，AlphaZero VS pure_MCTS **3000** 首次达到**10:0**
- 经过**2450**局，AlphaZero VS pure_MCTS **4000** 取得**8胜1平1负**
- 经过**2850**局，AlphaZero VS pure_MCTS **4000** 取得**9胜1负**。

OK，到这里整个AlphaZero实战过程就基本介绍完了，感兴趣的小伙伴可以下载我github上的代码进行尝试。为了方便大家直接和已经训练好的模型进行对战体验，我专门实现了一个纯numpy版本的策略价值前向网络，所以只要装了python和numpy就可以直接进行人机对战啦，祝大家玩的愉快！^_^

参考文献：

1. AlphaZero: Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm
2. AlphaGo Zero: Mastering the game of Go without human knowledge

编辑于 2018-03-05

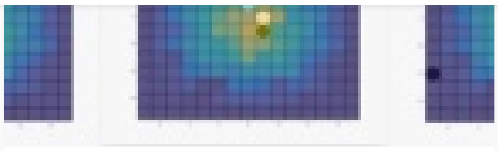
AlphaGo AlphaZero 强化学习 (Reinforcement Learning)

文章被以下专栏收录

强化学习知识大讲堂
让机器人学会思考
进入专栏

智能单元
面向通用人工智能和机器人学习，聚焦深度增强学习，可微神经计算机和生成对抗模…
进入专栏

推荐阅读



训练卷积神经网络下五子棋

MatthewYan

五子

王晓

183 条评论

切换为时间排序

写下你的评论...

huns 9 个月前

问一个问题。alphago zero进行自学习的时候。是如何自我改进的。我们知道，一盘围棋需要黑白双方交替进行各一百多手。影响最终胜负结果的可能只有那关键的几手棋。也就是说，如果单纯依据胜负结果来认定胜方的每一步棋都是更合理的显然不科学。

2

一缕阳光 (作者) 回复 huns 9 个月前

在AlphaGo zero自学习的过程中，会使用自我对局的胜负结果来学习一个价值网络，这个价值网络的作用是评估一个局面的好坏。如果只看一局比赛，认为胜方每一步对应的局面都是好的，那确实是不合理的，但实际训练是一个持续改进的过程，我们会慢慢的收集并使用很多局的信息。假设我们刚开始训练，现在的价值网络就是认为一个其实不怎么好的局面很好，那么在后续的自我对局中，如果执行某一步棋可以达到这个它认为很好的局面，它就会去执行这一步，但是这可能会导致它最终输掉这一局，当使用这一局对局的数据更新价值网络的时候，它就会学会那个局面其实没有那么好，这里面会有一个反馈校正的过程。随着对局数的慢慢增多，它就能渐渐学到一个局面的好的程度，这是一个连续的值，而不是离散的或好或坏。

前面说的都是用来评估局面的价值网络，它是直接根据胜负结果来训练并逐渐改进的。还有输出一个局面下不同位置落子概率的策略网络，它是使用MCTS搜索的结果来实现自我改进的，这边就不展开啦。

8 查看对话

Logow 9 个月前

损失函数中为什么不用KL来描述两个概率分布的差距?

赞

KL(pi, p)=cross_entropy(pi, p) - entropy(pi), 训练时pi是给定的数据，所以现在优化cross_entropy和优化KL divergence是等价的

3 查看对话

以上为精选评论

黄有 10 个月前

个人觉得可以用五子棋的vcf vct习题测试下智能程度

4

蜗niu2 10 个月前

可以说很棒了兄dei

赞

成卓轩 10 个月前

厉害厉害👍

赞

环城 10 个月前

很强，楼主是在Linux上实现的?

1

一缕阳光 (作者) 回复 环城 10 个月前

我是在Windows上实现的，不过用python也无所谓系统啦

3 查看对话

小小凡 10 个月前

厉害了！感觉能学到一万吨东西。想问一损失函数包含两个模块，策略和价值，再训练时候这两个全连接层是怎么训练的?

2

我是个世界的梦魇 10 个月前

十几年前手机里的内置五子棋游戏也是这个原理?

赞

一缕阳光 (作者) 回复 小小凡 10 个月前

有两部分输出的模型在训练时其实和只有一个输出的模型没有本质区别，你可以把策略价值网络的两部分输出p,v合起来看做模型的预测输出y_hat=(p,v)，然后把self-play数据中的pi和z看成是样

3 查看对话

一缕阳光 (作者) 回复 我是个世界的梦魇 10 个月前

我猜测之前的五子棋AI应该是用的minimax搜索结合alpha-beta剪枝的方式，然后剪枝的过程需要依赖很多人工总结的经验；

赞 查看对话

化鼠斯奎拉 10 个月前

能不能在15x15的棋盘上击败yixin?

2

一缕阳光 (作者) 回复 化鼠斯奎拉 10 个月前

方法本身应该有这样的潜力，但文中也提到了，在8*8的棋盘下训练3000局对局就需要2天左右了，所以想扩大到15*15同时达到很高实力的话已经不是个人电脑的计算能力能够支持的啦~

赞 查看对话

小小凡 回复 一缕阳光 (作者) 10 个月前

晓得了，谢谢！（都忘记了说谢谢，只能删除评论再写一半）还有一个问题，卷积自编码是否可以用于这里去提取前几层的特征，然后再利用真实数据去训练？

赞 查看对话

一缕阳光 (作者) 回复 小小凡 10 个月前

现在棋盘是直接用0,1二值矩阵表示的，下棋时关注的也就是这些1的位置，我们人类直接看这个二值矩阵的话也能直接理解当前的局面，所以我感觉这种0,1表示的局面本身已经有了比较高的抽象层次，比较直接的蕴含了重要信息。这和普通的图像很不一样，对于一张彩色图像，如果我们直接看那一堆RGB值的话几乎完全无法理解图像的内容，可能在那种情况下先用Autoencoder提取一些抽象的特征会更有帮助，而在下棋这个例子里可能就没那么必要了，个人见解

赞 查看对话

victor 10 个月前

和奕心还有妙手连珠对决一下，奕心能强一点。看看到什么程度了，五子棋一直以来没有一款王道的软件，是时候来一个了

1

victor 10 个月前

一个tpu大概什么算力啊，普通人配个好一点双路E5，E5也有很多型号嘛 算是到顶了

赞

根据谷歌公布的性能数据：“在推断任务中，TPU平均比英伟达的Tesla K80 GPU或英特尔至强E5-2699 v3 CPU速度快15至30倍左右”

赞 查看对话

1 2 3 4 ... 10 下一页