2019年9月18日 简单的Alpha Zero

一个简单的Alpha（Go）零教程

2017年12月29日

本教程将介绍同步单线程单GPU（阅读营养不良）游戏无关
最近执行 DeepMind的 AlphaGo Zero论文。培训代理人是一件很棒的工作
除了游戏规则之外，在没有*任何*人类知识的情况下进行纯粹的自我游戏。该
与DeepMind以前的论文相比，方法相当简单，AlphaGo Zero最终击败了AlphaGo
（令人信服地使用来自专家游戏的数据训练并击败最好的人类Go玩家）。最近，DeepMind
发表了预印本 arXiv上的 Alpha Zero将AlphaGo Zero方法扩展到Chess和Shogi。

这篇文章的目的是从AlphaGo Zero论文中提炼出关键思想并具体理解它们
通过代码。它假定基本熟悉机器学习和强化学习概念，并且
如果您了解神经网络基础知识和蒙特卡罗树搜索，则应该可以访问。在开始之前
（或在完成本教程后），我建议阅读原始论文。它写得很好，非常易读
有漂亮的插图！AlphaGo Zero通过自学强化学习进行训练。它结合了神经
网络和蒙特卡罗树搜索在优雅的策略迭代框架中实现稳定的学习。但
这只是单词 - 让我们直接深入细节。

神经网络

不出所料，有一个神经网络是事物的核心。神经网络 f_θ 由和参数化 θ
将输入板的状态视为输入。小号有两个输出：电路板状态的连续值 $v_\theta(s) \in [-1,1]$
从当前玩家的角度和政策 $\rightarrow p_\theta(s)$ 这是所有可能行动的概率向量。

在训练网络时，在每次自我游戏的结束时，神经网络都会提供训练
表格的例子 (s_t, π_t, z_t) 是对国家政策的估计（我们将了解如何达成政策）小号$_t$
下一节），和 $z_t \in \{-1, 1\}$ 从玩家的角度来看，这是游戏的最终结果（+1
如果玩家获胜，如果玩家输了则为-1）。然后训练神经网络以使随后的损失最小化
功能（不包括正规化条款）：

$$l = \sum_t (v_\theta(s_t) - z_t)^2 - \pi_t \cdot \log(p_\theta(s_t))$$

根本的想法是，随着时间的推移，网络将了解哪些州最终会导致胜利（或损失）。在
此外，学习该政策可以很好地估计某个州的最佳行动。神经
网络架构一般取决于游戏。大多数棋盘游戏如Go都可以使用多层游戏
CNN架构。在DeepMind的论文中，他们使用了20个残差块，每个块有2个卷积层。我曾是
能够得到一个4层CNN网络，然后是几个前馈层，可以用于6x6奥赛罗。

蒙特卡洛树寻求政策改进

给定状态，神经网络小号提供策略的估计。在培训阶段，我们希望 $p_\theta(s)$
改善这些估计。这是使用蒙特卡罗树搜索（MCTS）完成的。在搜索树中，每个
node表示板配置。两个节点之间存在有向边 $s_i \rightarrow s_j$ 如果有效的行动可以

导致状态从状态转变为。从空搜索树开始，我们将搜索树扩展为一个节点（州）。遇到新节点时，新节点的值为，而不是执行卷展栏
从神经网络本身获得。此值沿搜索路径向上传播。让我们再详细说明一下详情。

对于树搜索，我们保持以下内容：

Q （s，a）：从州采取行动的预期奖励，即Q值 一种 小号
N （s，a）：我们在模拟过程中从州采取行动的次数 一种 小号
P （小号，·）=：根据退回的政策从州政府采取行动的初步估计 小号
通过当前的神经网络。

从这些，我们可以计算出来 U （s，a）值的上限置信度为

$$U(s,a) = Q(s,a) + c_{puct} \cdot P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$

这里 c_{puct} 是一个控制探索程度的超参数。使用MCTS改进初始策略
由当前神经网络返回，我们以root作为初始化空搜索树。单一模拟 小号
进行如下。我们计算最大化置信上限的动作 一种 U （s，a）如果下一个
在我们的树中存在状态（通过对状态执行操作获得）我们以递归方式调用搜索。如果是的话 s'
不存在，我们将新状态添加到树中并初始化 P （小号'，·）=和价值（小号'）v （小号'）= v从
神经网络，并初始化 Q （s'，a）N （s'，全部为0。我们返回 v 而不是执行推广
传播 v （s）沿着当前模拟中看到的路径向上并更新全部 Q （s，值a）在另一
如果我们遇到终端状态，我们会传播实际奖励（如果玩家获胜则为+1，否则为-1）。

经过几次模拟，N （s，根a）的值为策略提供了更好的近似值。该
改进了随机政策 $\pi(s) \propto$ 只是标准化的计数 $N(s,\cdot)^{1/\tau} / \sum_b N(s,b)^{1/\tau}$ 在自我玩耍期间，我们
执行MCTS并通过从改进的策略中抽取一个移动来选择一个移动 $\pi(s) \propto$ 。以下是高级别的
实现一个搜索算法的模拟。

```
1  def 搜索（s，game，nnet）：
2      如果game.gameEnded（s）： return - game.gameReward（s）
3
4      如果s不是访问：
五          visited.add（S）
6          P [s]，v = nnet.predict（s）
7          回归 - 诉
8
9      max_u，best_a = - float（“inf”），- 1
10     对于一个在范围（game.getValidActions（S））：
11         u = Q [s] [a] + c_puct * P [s] [a] * sqrt（sum（N [s]））/（1 + N [s] [a]）
12         如果你 > max_u：
13             max_u = u
14             best_a = a
15     a = best_a
16
17     sp = game.nextState（s，a）
18     v =搜索（sp，game，nnet）
19
20     Q [s] [a] =（N [s] [a] * Q [s] [a] + v）/（N [s] [a] + 1）
21     N [s] [a] + = 1
22     回归 - 诉
```

mcts.py 托管与❤byGitHub上 查看原始

请注意，我们返回状态的负值。这是因为搜索树中的备用级别来自
不同球员的视角。以来 $v \in [-1,1]$ $-v$ 从角度来看是当前董事会的价值
其他球员

通过自我游戏进行政策迭代

第3页019年9月18日 简单的Alpha Zero

信不信由你，我们现在拥有训练我们无人监督的游戏代理所需的所有元素！学习
通过自我游戏本质上是一种策略迭代算法 - 我们使用我们的游戏来玩游戏和计算Q值
当前策略（在这种情况下是神经网络），然后使用计算的统计信息更新我们的策略。

这是完整的训练算法。我们用随机权重初始化我们的神经网络，从而开始
随机的政策和价值网络。在我们算法的每次迭代中，我们都会玩一些自我游戏的游戏。在
在游戏的每个回合中，我们从当前状态开始执行固定数量的MCTS模拟。我们挑选
从改进的政策中抽样采取行动。这给了我们一个训练的例子 s π 。
在游戏结束时填写：如果当前玩家最终赢得游戏，则为+1，否则为-1。搜索树是
在比赛中保存。

在迭代结束时，用获得的训练样例训练神经网络。旧的和
新网络互相攻击。如果新网络赢得超过设定的游戏阈值分数
（在DeepMind论文中为55％），网络将更新到新网络。否则，我们进行另一次
迭代来增强训练样例。

就是这样！奇怪的是，网络几乎每次迭代都会改进并且学会玩游戏
更好。下面提供了完整训练算法的高级代码。

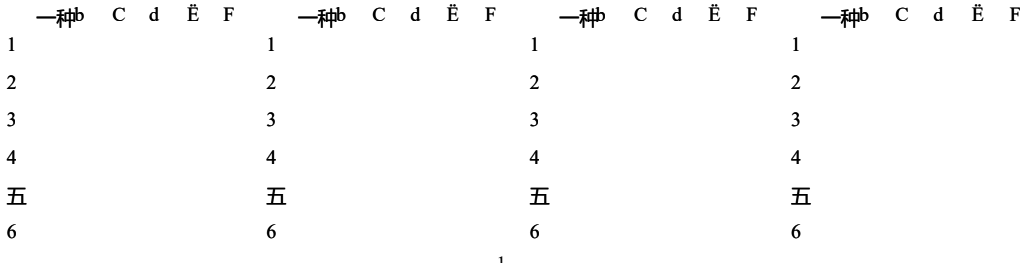
```
1  def policyIterSP（游戏）：
2      nnet = initNNet（）                                  #initialise随机神经网络
3      examples = []
4      对于我在范围（numIters）：
五          为ė在范围（numEps）：
6              examples + = executeEpisode（game，nnet）    # 从这个游戏收集例子
7          new_nnet = trainNNet（例子）
8          frac_win = pit（new_nnet，nnet）                 # 比较新网与之前的网
9          如果frac_win > 阈值：
10             nnet = new_nnet                              # 替换新网
11     返回nnet
12
13 def executeEpisode （game，nnet）：
14     examples = []
15     s = game.startState（）
16     mcts = MCTS（）                                      #initialise搜索树
17
18     而真：
19         for _ in range （numMCTSSims）：
20             mcts.search（s，game，nnet）
21         examples.append（[s，mcts.pi（s），None ]）       #rewards尚无法确定
22         a = random.choice（len（mcts.pi（s））， p = mcts.pi（s））来自改进政策的 #pample action
23         s = game.nextState（s，a）
24         如果game.gameEnded（s）：
25             examples = assignRewards（examples，game.gameReward（s））
26             返回示例
```

politer-selfplay.py 主持❤byGitHub上 查看原始

奥赛罗的实验

我们在一个GPU上训练了奥赛罗的游戏代理，用于6x6板。每次迭代由100组成
自我发挥的剧集和每个MCTS使用了25次模拟。请注意，这比数小几个数量级
AlphaGo论文中使用的计算（每次迭代25000集，每回合1600次模拟）。该模型
花了大约3天（80次迭代）进行训练，以便在NVIDIA Tesla K80 GPU上饱和。我们评估了模型

反对随机和贪婪的基线，以及极小极大的代理人和人类。它表现得相当好甚至
选择了人类使用的一些常用策略。



代理人（黑人）学会在早期游戏中捕捉墙壁和角落

经纪人（黑人）学会在比赛后期强行传球

关于细节的注释

这篇文章概述了AlphaGo Zero论文中的主要思想，并为此省去了更精细的细节
清晰。AlphaGo论文描述了其实现中的一些其他细节。他们之中有一些是：

国家历史：由于Go从目前的董事会状态，神经系统中无法完全观察到
网络还将过去7个步骤中的电路板作为输入。这是游戏本身的一个特点
Chess和Othello等其他游戏只需要当前的棋盘作为输入。
温度：执行MCTS后获得的随机策略使用指数计数，即
$\pi(a) = N(s,a)^{1/\tau} / \sum_b N(s,b)^{1/\tau}$，温度在哪里并控制探索的程度。
AlphaGo Zero使用$\tau = 1$（每个游戏的前30个动作的简单计数），然后
将其设置为无穷小的值（选择具有最大计数的移动）。
对称性：Go板对旋转和反射不变。当MCTS到达叶节点时，
使用反射或旋转版本的电路板调用当前神经网络以利用这种对称性。在
一般来说，这可以扩展到使用适合游戏的对称性的其他游戏。
异步MCTS：AlphaGo Zero使用MCTS的异步变体来执行模拟
在平行下。神经网络查询被批处理并且每个搜索线程被锁定直到评估
完成。另外，3个主要过程：自我发挥，神经网络训练和比较
新旧网络都是并行完成的。
计算能力：每个神经网络都使用64个GPU和19个CPU进行训练。使用的计算能力
从文件中不清楚地执行自我游戏。
神经网络设计：作者尝试了各种架构，包括有和没有网络
剩余网络，有和没有参数共享的价值和政策网络。他们最好的
架构使用剩余网络并共享价值和策略网络的参数。

本教程中提供的代码提供了所涉及算法的高级概述。一个完整的游戏

和框架无关的实现可以在这个GitHub仓库中找到。它包含一个例子
在PyTorch，Keras和TensorFlow中实施奥赛罗的比赛。

随意留下下面的问题/意见/建议:)。