人工智能导论作业 5：miniAlphaGo

石睿（211300024、211300024@smail.nju.edu.cn）

（南京大学 人工智能学院，南京 210093）

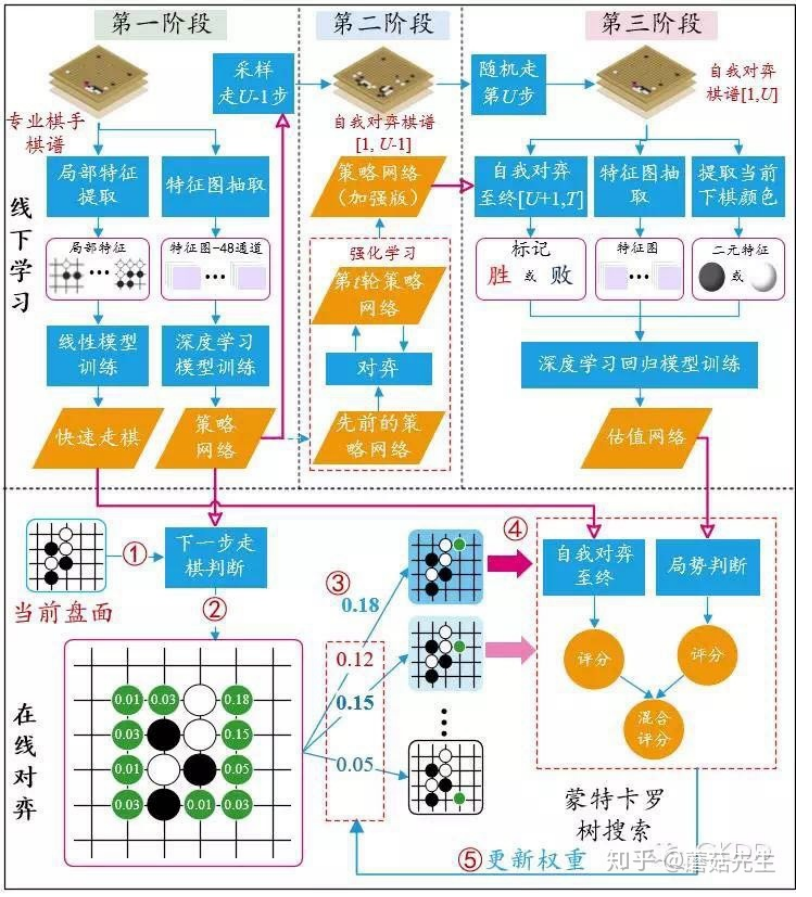
摘 要： 结合 MCTS、强化学习和 Self Play 等技术，实现一个简单的围棋 AI 程序

关键词： 蒙特卡洛树搜索（MCTS）； Alphago

中图法分类号： TP301 文献标识码：A

1、阅读 AlphaGo 原文，阅读本次作业代码，熟悉与围棋环境的数据交互方式及 RL 算法在提供的围棋环境下训练的方法，策略网络、值函数网络模型保存的方式

1.1 AlphaGo 实现[1][2]



AlphaGo 的学习在对弈之前就已经完成训练。对弈是依据离线学习拿到的快速走棋网络、策略网络、估值网络，对 MCTS 的搜索范围进行缩小，最终由 MCTS 来做出决策。

Part1：学习

第一阶段：训练监督学习策略网络。

Alphago 利用 3000 万多幅职业棋手对局的棋谱来有监督训练两个学习策略网络：

策略网络 P_σ(Policy Network)：基于全局特征和深度卷积网络(CNN)训练出来的，其主要作用是给定当前棋面状态作为输入，输出下一步棋在棋盘其它空地上的落子概率。**快速走子策略**(Fast Rollout Policy)：利用局部特征和线性神经网络模型训练出来的

策略网络速度较慢，但精度较高。其目的是为了对 MCTS 搜索空间做出有效限制；快速走子策略网络速度比较快，但精度比较低。其目的只是给 MCTS 对当前局面的评分提供一个参考，且只是多个参考中的一部分。综上，两个网络精度和速度的 trade-off 中选择了不同的方向。

第二阶段：训练强化学习策略网络。

利用先前训练好的监督学习策略网络初始化强化学习策略网络，然后随机挑选前面第 t 轮的策略网络和该强化学习网络互相对弈，利用增强式学习来修正强化学习策略网络的参数，最终得到**增强的策略网络**。

第三阶段：训练值网络。

利用增强学习策略网络自我对弈生成 3000 万个棋谱以及对应的输赢情况(实际过程如下图所示，每局先用监督学习策略网络生成 U 步，之后再使用强化学习策略网络自我对弈到决出胜负)，每个棋谱随机抽样棋面数据，构成 3000 万个棋面和实际输赢情况标签的训练数据。最后使用估值网络进行回归训练，目标是最小化输赢预测的误差损失。

Part2：对弈

核心思想是在蒙特卡罗搜索树中嵌入深度神经网络来减小搜索空间。

策略网络能够减小搜索的宽度，在扩展阶段发挥作用；值网络能够减小搜索的深度，在评估阶段发挥作用。

Step1：根据当前棋面提取相应特征。

Step2：蒙特卡洛树搜索

选择：选择下一步动作，进行棋面状态转移。$a_t=\text{argmax}(Q(s_t,a)+u(s_t,a))$

扩展：遇到叶子节点且该叶子节点访问次数超过给定的限制的时候进行扩展。使用监督学习策略网络估计该叶节点对应的棋面下，下一步走法其他动作的概率 p(a|s)，作为先验概率存放在对应的动作边上。

评估：使用价值网络和快速走子策略对当前叶子结点的状态进行评分。每次模拟走到叶子节点时，对叶子节点进行评估。使用价值网络和快速走子策略预测该叶子节点对应的棋面赢棋的可能性。其中值网络直接输入棋面特征，预测赢棋的可能性大小；快速走子策略进行多轮 rollouts 模拟，每轮 rollouts 从该叶子节点对应的棋面出发走棋，一直到决出胜负，平均多轮胜负情况作为赢棋可能性的预测值。

回溯：更新 Q 值和访问次数 N 值。上述评估阶段完成后，代表此次模拟结束，此时进行反向传播更新此次模拟到达该叶子节点所经过的所有边上的 Q 值和访问次数 N。更新完的 Q 值和 N 值用于下一轮模拟，下一轮模拟 Selection 阶段会使用新的 Q 值和 N 值选择下一步动作进行状态转移。

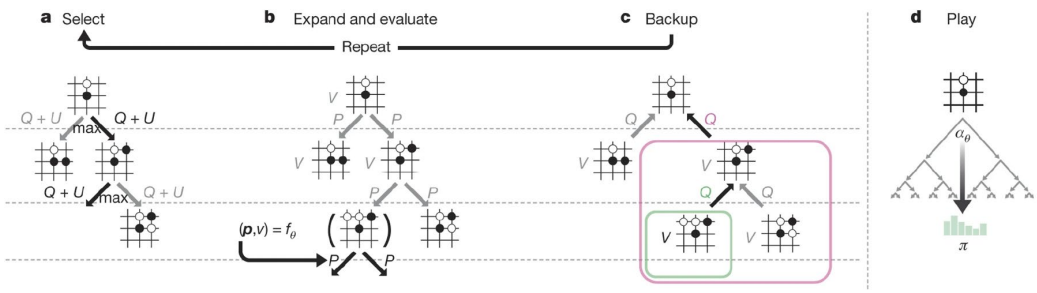
Step3：选择访问次数最多的子状态所对应的动作

所有轮次的模拟结束后，选择和根节点（当前棋面）相连接的、访问次数最多的边所对应的动作，作为下一步棋的走法。

2 在围棋环境中实现 MCTS 方法

Alphago 下棋的时候，做出决策的不是前文提到的深度学习、CNN 训练出的策略网络和价值网络，而是蒙特卡洛树搜索。训练好的策略网络和价值网络都可以单独地做出决策，而在 alphago 之中，策略网络和价值网络的目的是辅助 MCTS，降低 MCTS 搜索的深度和宽度。

以下简单对上文提到的 AlphaGo 所用到的 MCTS 进行扩展，并实现一个弱化版的 MCTS（不太会神经网络呀(；′⌒`)）



AlphaGo 原文中 MCTS 的过程

2.1 MCTS 方法介绍

MCTS 的基本原理就是模拟未来可能发生的情况，从而找出当前最优的动作。但这种模拟未来可能发生的情况不是遍历所有可能的情况，而是只遍历几种收益最高的走法（UCT 最大值）。

MCTS 的实现分为四个步骤

① 选择（select）

目的是找出胜算比较高的动作，只搜索这些好的动作而忽略其他的动作。而判断动作的好坏有两个指标：1、此动作的胜率 2、策略网络给动作的评分

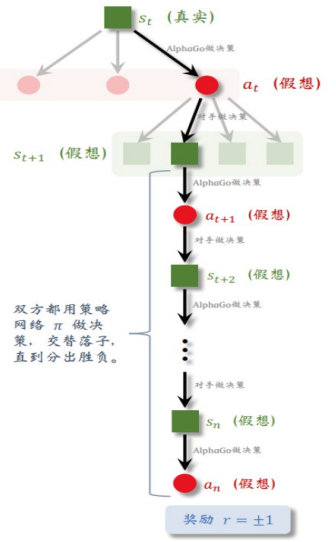
在选择阶段，从搜索树的根节点开始，不断选择 Q+U 值最大 $a_c = \underset{a}{argmax}\left(Q(s,a)+U(s,a)\right)$，其中 U（s,a）为 $U(s,a) = c_{puct}P(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$。直到搜索到对局结束或者达到搜索限度（本实验中采用搜索时间作为限度）。其中参数的解释如下 s：搜索树的一个节点代表的棋局状态；a：表示某一个可行的动作；N(s,a)：表示状态下可行动作被选中的次数；P(s,a)：为状态下的可行动作的先验概率；Q(s,a)：表示状态下可行动作的动作价值；c_{puct}：为一个决定探索程度超参数。

② 扩展（expand）

在第一阶段“选择”中拿到的动作，在 agent 脑海中假想执行过程，返回一个新的状态。MCTS 需要模拟 AlphaGo 和对手对局，AlphaGo 每执行一个动作，环境应该返回一个新的状态。因为围棋游戏具有对称性，AlphaGo 的策略，在对手看来是状态转移函数；对手的策略，在 AlphaGo 看来是状态转移函数。最理想情况下，模拟器的状态转移函数是对手的真实策略，然而 AlphaGo 并不知道对手的真实策略，所以只能用自己训练出的策略网络代替对手的策略，作为自己的状态转移函数。

③ 求值

选择阶段之后如果到了不是对弈结束的叶结点，对于对应的棋局状态，使用策略网络和价值网络对状态进行评估，得到对应棋局状态下一步各个可能动作的概率和的价值。为所有可能动作对应的棋局状态分别创建一个节点，将这些节点的先验概率设置为策略网络的输出概率值。



④ 回溯

进过上述扩展之后，之前的叶子节点，现在变成了内部节点。做完了扩展和求值后，从节点开始，逐层向搜索树根节点回溯，并依次更新搜索树当次被遍历的路径上各层节点的信息：N、Q 值。

2.2 MCTS 方法实现[3]

在 mini_go 文件夹中实现 MCTS，关于 MCTS 的节点和树的方法分别写在 agent.py 中的 Node 类和 MCTS 类中。对局的模拟的 Main 函数在 MCTS_vs_random.py 文件中（仿照已有的两个 demo 进行修改的）

Node 类的实现如下

```
class Node(object):
    def __init__(self, parent, env, time_step, action, P, Q):
        self.initial_state = time_step  # 初始状态,用来对局到最后评分的
        self.env = env#当前环境
        self.cur_id = self.initial_state.observations["current_player"]  # 是哪个玩家
        self.actions = self.initial_state.observations["legal_actions"][self.cur_id]#当前节点可以做出哪些动作
        #即有哪些子节点
        self.parent = parent  # 父节点-用来回溯更新的
        self.children = []#当前节点的子节点
        self.action = action  # 做了哪个动作到的当前节点
        self.Q = Q#mcts中对当前节点的估计
        self.N = 0#对当前节点的访问次数
        self.P = P#mcts中的先验概率
```

其中前四个属性是依据 mini_go 中特定的属性定义的。后六个属性是 MCTS 的四个执行过程中所必须的。属性的具体含义以及用处如图片所示。

MCTS 类的实现如下

```
class MCTS(object):
    def __init__(self, root, policy_net, value_fn, rollout_fn, env, time=10):
        self.policy_net = policy_net.policy_fn  # 深度学习拿到的策略网络
        self.value_net = value_fn  # 估值网络
        self.rollout_net = rollout_fn # 快速走棋的策略网络
        self.time_limit = time#运行的时间限制
        self.root = root
```

首先，MCTS 类的初始化接收定义变量时的策略网络、估值网络、快速走棋网络，辅助 MCTS 进行搜索。其次，接收运行时间限制，来作为 MCTS 搜索的限度。

第一步：选择。对每个子节点计算它的 Q+U 的值，并进行排序，拿到 Q+U 值最大的子节点作为即将访问并扩展的节点

```
def select(self, node):
    '''
    蒙特卡洛的第一步-选择
    选择最大置信度Q+U的子节点
    此时设定超参数c=1
    '''

    if node.cur_id == 0:
        node.children.sort(key=lambda child: child.Q + child.P * math.sqrt(node.N) / (1 + child.N))
    else:
        node.children.sort(key=lambda child: -child.Q + child.P * math.sqrt(node.N) / (1 + child.N))

    return node.children[-1]
```

第二步：扩展。对当前“根”节点的所有可能动作分别创建对应的子节点，在 agent 脑海中更新节点状态，并计算出子节点的先验概率和 Q 值，保存到节点的 children 数组之中。

```
def expand(self, node):
    '''
    select选择到一个节点之后，且该节点所代表的的游戏局面没有结束
    》》expand这个节点---创建一系列可能子节点，且对应的子节点之中存了从此节点做出的动作到子节点

    '''
    for action in node.actions:#所有的可选动作》》产生对应子节点
        envs = copy.deepcopy(node.env)
        timestep = envs.step(action)#从父节点依据可选动作产生新的局面
        xianyan = self.cul_P(timestep)
        P = xianyan[action]
        Q = self.cul_Q(timestep)
        child = Node(node, envs, timestep, action, P, Q)
        node.children.append(child)
```

第三步：求值。通过传入的估值网络计算 Q 值，策略网络计算 P 值

```
def cul_Q(self, timestep):#估值网络，计算Q值
    return self.value_net(timestep, timestep.observations["current_player"])

def cul_P(self, timestep):#深度学习拿到的策略网络，计算P
    return self.policy_net(timestep, timestep.observations["current_player"])
```

第四步：回溯。更新这条搜索路径上节点的 N 和 Q 值

```
def trace_back(self, node, value):
    while node.parent != None:
        node.N += 1
        node.Q = (node.N * node.Q + value) / node.N
        node = node.parent
```

最终，调用上面四个方法进行模拟。AlphaGo 原始的想法是选择访问节点次数最多的，其原因为在每一次模拟中，MCTS 找出所有可行的动作，计算他们的分数，然后选择其中分数最高的动作，并在 agent 脑海中执行。如果某个动作在模拟时胜率很大，那么它的价值就会很大，它的分数会很高，于是它被选中的几率就很大。也就是说如果某个动作很好，他被选中的次数就会大。

但因为本实验中 MCTS 并没有采用神经网络进行估值网络、快速走棋策略网络的训练，而是采用一种随机的方式。所以只通过访问次数来选择最终状态并不好，最终在尝试多种文章提到的选择方式后，采取 UCT（置信度上届）的方式得到的对局效果最好

$$\mathrm{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\log(N(v))}{N(v_i)}}$$

```
def simulate(self):
    start_time = time.time()
    while True:
        self.runing()
        if time.time() - start_time >= self.time_limit:#模拟的时间到了
            self.root.children.sort(key=lambda node: node.Q/(node.N+1)+
                                                    math.sqrt(math.log(self.root.N+1)/(node.N+1)))
            #选择UCT值最大的【平衡探索和利用】
            return self.root.children[0].action
```

```python
def runing(self):
    '''
    在simulate函数中调用，在规定的时间内进行搜索

    '''
    node = self.root
    while len(node.children) != 0:#有子状态可以选择
        node = self.select(node)
    if node.N > 1 or node.parent == None:
        self.expand(node)
        node = self.select(node)
    reward = self.rollout(node)
    self.trace_back(node, reward)
```

2.3 实验结果

重复试验约 10 次，得到的平均胜率约为 70%。其中快速走棋网络和估值网络可以优化，如果采用神经网络进行训练（得有数据）效果会更好

```
MCTS trainging episode  0
MCTS vs random
[1, -1]
MCTS trainging episode  1
MCTS vs random
[1, -1]
MCTS trainging episode  2
MCTS vs random
[1, -1]
MCTS trainging episode  3
MCTS vs random
[-1, 1]
MCTS trainging episode  4
MCTS vs random
[1, -1]
MCTS trainging episode  5
MCTS vs random
[-1, 1]
MCTS trainging episode  6
MCTS vs random
[1, -1]
MCTS trainging episode  7
MCTS vs random
[1, -1]
MCTS trainging episode  8
MCTS vs random
[1, -1]
MCTS trainging episode  9
MCTS vs random
[-1, 1]
Time elapsed: 931.0016179084778
```

因为对神经网络、深度学习以及库 tensorflow 不是很熟悉，再加上这两天阳了发高烧(╥﹏╥)，目前就只能完成 MCTS 的实现啦。关于 AlphaGo 中未了解的部分，在今后的学习过程中再认真学习吧┬┬﹏┬┬

参考文章：

1、 Alphago 的 nature 原文：https://www.lamda.nju.edu.cn/introAI/PDF/alphago2016.pdf

2、 Alphago 论文解读：https://zhuanlan.zhihu.com/p/310502314

3、 Csdn 上某一篇关于 MCTS 的讲解（上学期导出的 pdf，忘记网址啦）