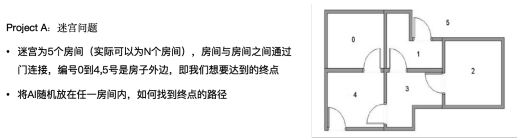
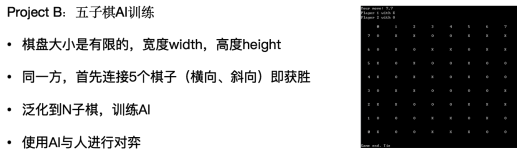
第十六节课 强化学习与推荐系统

https://github.com/cystanford 老师的GitHub

第十六节课笔记

讲义内容概括	知识点	Question?





强化学习（Reinforcement Learning）：
机器学习的一个分支：监督学习、无监督学习、强化学习

强化学习的特点：
没有监督数据、只有奖励信号
奖励信号不一定是实时的，很可能是延后的，甚至延后很多
时间（序列）是一个重要因素
当前的行为影响后续接收到的数据
强化学习有广泛的应用：游戏AI，推荐系统，机器人仿真，投资管理，发电站控制

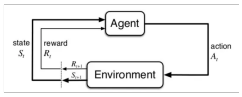
强化学习与机器学习：
强化学习没有教师信号，也没有label，即没有直接指令告诉机器该执行什么动作
反馈有延时，不能立即返回
输入数据是序列数据，是一个连续的决策过程

基本概念：
个体，Agent，学习器的角色，也称为智能体
环境，Environment，Agent之外一切组成的、与之交互的事物
动作，Action，Agent的行为
状态，State，Agent从环境获取的信息
奖励，Reward，环境对于动作的反馈
策略，Policy，Agent根据状态进行下一步动作的函数
状态转移概率，Agent做出动作后进入下一状态的概率

四个重要的要素：状态(state)、动作(action)、策略（policy）、奖励(reward)

强化学习：

- RL考虑的是个体（Agent）与环境（Environment）的交互问题
- 目标是找到一个最优策略，使Agent获得尽可能多的来自环境的奖励
- 比如赛车游戏，游戏场景是环境，赛车是Agent，赛车的位置是状态，对赛车的操作是动作，怎样操作赛车是策略，比赛得分是奖励
- 很多情况下，Agent无法获取全部的环境信息，而是通过观察(Observation)来表示环境（environment），也就是得到的是自身周围的信息



强化学习的基本结构

奖励（Reward）
信号的反馈，是一个标量，它反映个体在t时刻做得怎么样，个体的工作就是最大化累计奖励
强化学习假设是，所有问题解决都可以被描述成最大化累积奖励

序列决策（Sequential Decision Making）
目标：选择一定的行为系列以最大化未来的总体奖励

- 个体与环境的交互（Agent & Environment）

从个体的视角：

在 t时刻，Agent可以：

有一个对于环境的观察评估

做出一个行为

从环境得到一个奖励信号

环境可以：

接收个体的动作

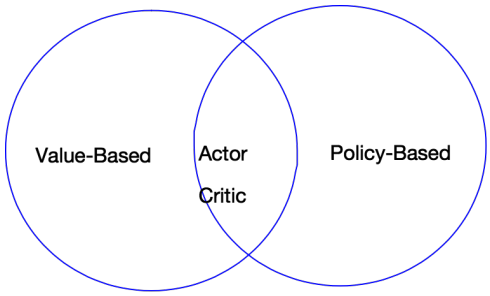
更新环境信息，同时使得个体可以得到下一个观测

给个体一个奖励信号

马尔可夫属性（Markov Property）：一个状态St是马尔可夫的，当且仅当

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, S_2, \ldots, S_t]$$

强化学习Agent：
基于价值的强化学习，Value-Based
通过学习价值函数指导策略制定（例如ε-greedy执行方法）
基于策略强化学习，Policy-Based
没有价值函数，直接学习策略
结合策略梯度以及价值函数的强化学习，Actor-Critic
既学习价值函数也学习策略的方法



策略网络：
任何游戏，玩家的输入被认为是行为a，每个输入（行为）导致一个不同的输出，这些输出称为游戏的状态s 可以得到一个不同状态–行动的配对的列表

$$S = \{(s_0, a_0), (s_1, a_1), \ldots, (s_n, a_n)\}$$

策略网络：

- S：状态集
- A：动作集
- R：奖励分布，给定（state，action）
- P：状态转移概率，对于给定的（state，action），下一个状态的概率分布
- ：贴现因子，为了防止奖励r达到无穷大的预防措施 => 无穷大的奖励会忽略掉智能体采取不同行动的区别
- ：最优策略

$$\pi = \arg\max_{\pi} E\left[\sum_{t\geq 0} \gamma^t r_t \mid \pi\right]$$

价值网络（数值网络）：

- 通过计算目前状态s的累积分数的期望，价值网络给游戏中的状态赋予一个分数（数值），每个状态都经历了整个数值网络
- 奖励更多的状态，会在数值网络中的数值Value更大
- 这里的奖励是奖励期望值，我们会从状态集合中选择最优的
- V：价值期望

$$V^{\pi}(s) = E\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, \pi\right]$$

策略网络与价值网络：
策略网络的输出，是一个落子的概率分布 比如，棋盘中现在轮到白棋走，蓝点代表成为下一手的可能性。
价值网络的输出，一个可能获胜的数值，即“价值”，这个价值训练是一种回归(regression)，即调整网络的权重来逼近每一种棋局真实的输赢预测
对于价值网络，当前局面的价值=对终局的估

计

QLearning：
Q函数，也称为动作值函数，有两个输入：「状态」和「动作」，它将返回在这个状态下执行该动作的未来奖励期望
将agent随机放在任一房间内，每打开一个房门返回一个reward
根据房间之间连通性的关系，可以得到Reward矩阵（R矩阵）
可以把 Q 函数视为一个在 Q-table 上滚动的读取器

使用QLearning：

- 初始化Q表，用于记录状态-动作对的值，每个episode中的每一步都会根据公式更新一次Q表

$$Q(S_t,A_t)\leftarrow Q(S_t,A_t)+\alpha[R_{t+1}+\gamma\max_a Q(S_{t+1},a)-Q(S_t,A_t)]$$

为了简便，将学习率 设为1，更新公式为：

$$Q(S_t,A_t)\leftarrow R_{t+1}+\gamma\max_a Q(S_{t+1},a)$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal,\cdot) = 0$
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$
  until S is terminal

随机策略有时候是最优的：
Policy-Based优势：
收敛性好，因为Policy-Based每次只改善一点点，但总是朝着好的方向在改善，而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
对于高维度或连续状态空间来说，更高效，使用基于价值函数的学习在得到价值函数后，制定策略时，需要比较各种行为对应的价值大小，这个比较过程就比较难，而采用Policy-Based会高效很多
能学到一些随机策略，而Value-Based通常是学不到随机策略
针对价值函数非常复杂的情况，采用Policy-Based更适合。比如当小球从从空中某个位置落下你需要左右移动接住时，计算小球在某一个位置时采取什么行为的价值是很难计算的，但基于策略就简单许多，只需要朝着小球落地的方向移动即可

AlphaGO Zero的强化学习
强化学习问题：
首先把要解决的问题转化成为一个环境

（environment）
状态空间（state space）：对于围棋来说，每一个棋盘布局（记为s）就是一个状态，所有可能的棋盘布局就是状态空间
动作空间（action space）：对于围棋来说，所有可能落子的位置就是一个动作空间
可行动作（available action）： 给定一个棋盘，哪里可以落子，哪里不可以
状态转化：下棋之后，对手可能会下的棋。如果是两个Alpha Zero对弈的话，相互是对方环境的一个部分
奖励函数：下棋之后得到的信号反馈。在围棋里面，就是胜率的一个正函数。胜率越大，奖励越大

AlphaGO Zero策略：
落子概率也称为策略（policy）
有了落子概率，简单的方式是直接按照这个概率进行落子 =>这会导致神经网络原地踏步，因为Policy Value Network的训练数据是自我对弈（self-play）
仅仅自己学习自己是不会有改进的，需要有一个办法来利用值函数的信息来优化这个策略
在AlphaGo系列算法里面是使用蒙特卡洛树搜索（MCTS）来进行策略优化的
MCTS的输出π是根据值函数V得到的一个更优策略，它将被用于通过self－play来生成数据供深度神经网络学习
MCTS是AlphaGo能够通过self－play不断变强的重要原因

初识强化学习

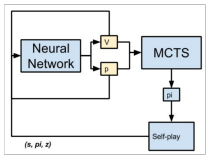
Project A：迷宫问题
Project B：五子棋AI训练
什么是强化学习
强化学习的基本概念
强化学习Agent分类
什么是策略网络
什么是价值网络
AlphaGo的强化学习原理
什么是蒙特卡洛树搜索MCTS

AlphaGo Step by Step

AlphaGo主逻辑
MCTS节点的定义
MCTS树的创建和使用
基于MCTS的AI Player
策略价值网络Policy Value Network实现
使用强化学习对五子棋AI进行训练
人机对弈
强化学习与推荐系统

AlphaGo Zero:

- Mastering the game of Go without human knowledge, 2017（DeepMind发表在Nature）
- https://www.gwern.net/docs/rl/2017-silver.pdf
- 使用一个参数为θ的深度神经网络 $f_\theta(s)$，将棋盘表示和历史记录作为输入，输出落子概率和价值 $(p, v) = f_\theta(s)$
- 落子概率向量p代表选择每个落子动作a（包括放弃行棋）的概率 $p_a = \Pr(a \mid s)$
- 价值v是标量评估，估计当前玩家在棋局状态为 s 时获胜的可能性
- AlphaGo Zero神经网络将策略网络和价值网络合并成一个单一的体系结构，其中神经网络由13层的卷积神经网络组成



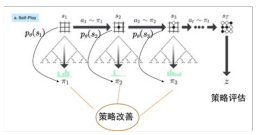
AlphaGo Zero训练流程图

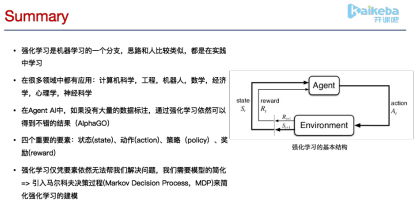
AlphaGo Zero:

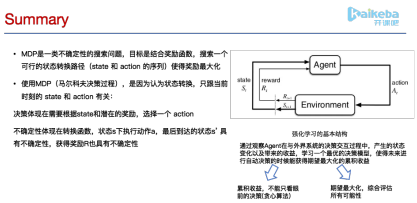
- 对于每个棋局s，通过神经网络 $f_\theta(s)$ 的指导来执行MCTS搜索，MCTS搜索输出每次落子的概率分布π。
- 经过搜索后的落子概率p 通常比神经网络 $f_\theta(s)$ 输出的落子概率p更强 => 将MCTS看作是一个强大的策略改进算法

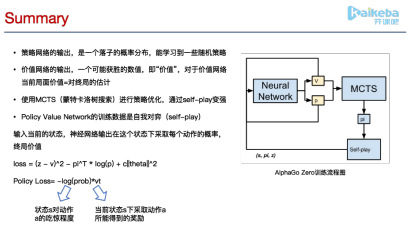
利用MCTS（蒙特卡洛搜索树）实现策略改善：

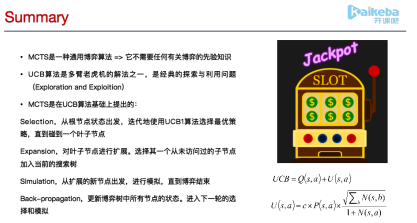
- 在每个状态s处，原神网络的策略为 $p_\theta(s)$ 经过蒙特卡洛搜索树得到的策略为 $\pi(s)$，因此在状态s处，策略改善为：$p_\theta(s) \to \pi(s)$

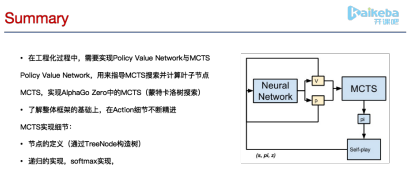


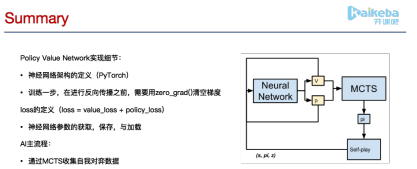










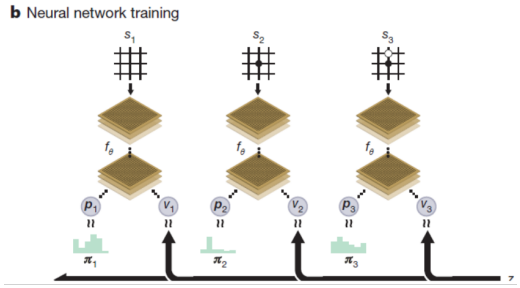


利用MCTS（蒙特卡洛搜索树）实现策略评估：

- 在每个状态s处，都会执行蒙特卡洛搜索树给出的策略，直到最终棋局结束
- 根据棋局的输赢会得到奖励 z 值
- 棋局的输赢是由当前策略决定的 => 将奖励值 z 当成是当前策略的评估

Loss Function

$$Loss = (z - v)^2 - \pi^T \log P + c\theta^2$$



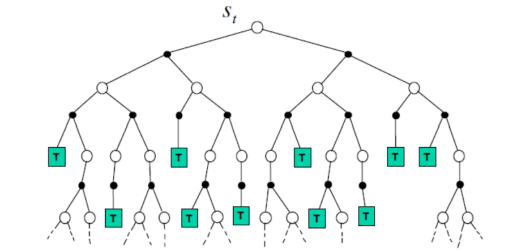
在每次迭代结束后，利用监督学习更新神经网络权值

蒙特卡洛算法

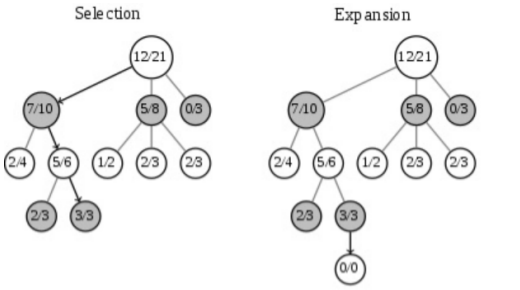
MCTS（Monte Carlo Tree Search）：
蒙特卡洛树搜索，结合了随机模拟的一般性和树搜索的准确性
MCTS是一个搜索算法，它采用的各种方法都是为了有效地减少搜索空间。在MCTS的每一个回合，起始内容是一个半展开的搜索树，目标是原先的半展开+再多展开一个/一层节点的搜索树
MCTS的作用是通过模拟来进行预测输出结果，理论上可以用于以{state,action}为定义的任何领域

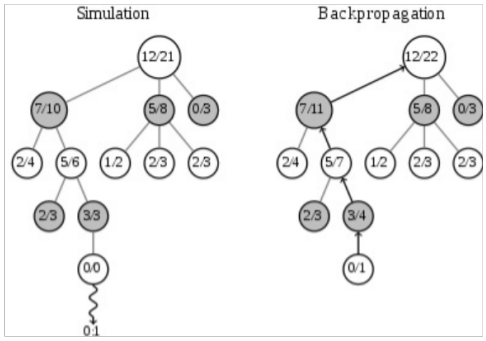
使用主要步骤：
选择，从根节点开始，按一定策略，搜索到叶子节点
扩展，对叶子节点扩展一个或多个合法的子节点
模拟，对子节点采用随机的方式（这也是为什么称之为蒙特卡洛的原因）模拟若干次实验。模拟到最终状态时即可得到当前模拟所得的分数
回传，根据子节点若干次模拟的得分，更新当前子节点的模拟次数与得分值。同时将模拟次数与得分值回传到其所有祖先节点并更新祖先

节点



MCTS原理：
每个节点代表一个局面，A/B代表被访问B次，黑棋赢了A次
我们将不断重复一个过程：
Step1，选择Select，从根节点往下走，每次都选一个“最有价值的子节点”，直到找到“存在未扩展的子节点”，即这个局面存在未走过的后续着法的节点，比如 3/3 节点
Step2，扩展Expansion，给这个节点加上一个 0/0 子节点，对应之前所说的“未扩展的子节点”
Step3，模拟Simluation，用快速走子策略（Rollout policy）走到底，得到一个胜负结果（Thinking：为什么不采用AlphaGo的策略价值网络走棋）
Step4，回传Backup，把模拟的结果加到它的所有父节点上，假设模拟的结果是 0/1，就把0/1加到所有父节点上





简单有效的选择公式

$$score = x_{child} + c \cdot \sqrt{\frac{\log(N_{parent})}{N_{child}}}$$

x是节点的当前胜率估计，N 是节点的访问次数,C是指定的常数
c越大就越偏向于广度搜索 => exploration
c越小就越偏向于深度搜索 => exploition

UCB算法（Upper Confidence Bound）：

UCB算法（Upper Confidence Bound）：

- 用于在Select过程中，判断节点的价值，每次选择选取最大的UCB节点：

$$UCB = Q(s,a) + U(s,a)$$

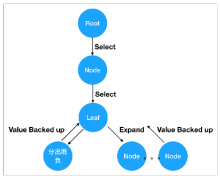
$$U(s,a) = c \times P(s,a) \times \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$

- Q(s,a)代表蒙特卡洛树中已探索的值，Q值的更新就是在之前蒙特卡洛树搜索中每一次backup反馈的值的平均值
- P值是先验概率，是由神经网络计算得到的值
- N是这个action的探索次数，c用于调节探索与利用的超参数
- 对于没有探索的action，和已经探索很多次但是探索反馈很高的action都会有较大的UCB值

MCTS蒙特卡洛树搜索

MCTS的流程理解：

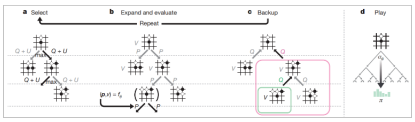
- 对于任意的局面State（就是节点），要么被展开过（expand），要么没有展开过（就是叶子节点）
- 展开过的节点可以通过Select步骤进入下一个局面State，下一个局面State仍然是这个过程……，一直持续下去直到这盘棋分出胜负，或者遇到某个局面没有被展开过为止
- 如果没有展开过，那么执行expand操作，通过神经网络得到每个动作的概率和胜率v，把这些动作添加到树上，最后把胜率v回传（backed up）
- Expand的同时就在做evaluate，这是为Backup做准备，因为在Backup步骤，我们要用v来更新Q值的，但是如果只做了一次Select，棋局还没有结束，此时无法得到v，必须要等到一盘棋下完才可以得到v（当前这步的好坏评估是由最终结果决定）



AlphaGO的MCTS

AlphaGo Zero中的蒙特卡洛树搜索：

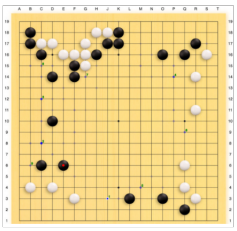
- Select，每次模拟通过选择具有最大UCT值的
- Expand&Evaluate，展开叶子节点，通过神经网络 $f_\theta(s) = (P(s), V(s))$ 来评估局面，P的值存储在叶子结点扩展边上
- Backup，更新行动价值Q等于在该行动下的子树中的所有评估值V的均值
- Play，当MCTS搜索完成时，返回局面s下的落子概率 π，与$N^{(1/\tau)}$成正比，其中N是从根状态每次移动的访问计数，τ是控制温度的参数



AlphaGo的策略价值网络

策略价值网络：

- AlphaGo 的策略网络，让我们更准确地选择需扩展的节点
- AlphaGo 的价值网络，可以与快速走子策略的模拟结果相结合，得到更准确的局面评估结果
- 策略网络可以让我们对动作使用输出概率来表示，对于离散动作空间，使用softmax动作的出现概率；对于连续空间，使用高斯分布来获取动作的概率

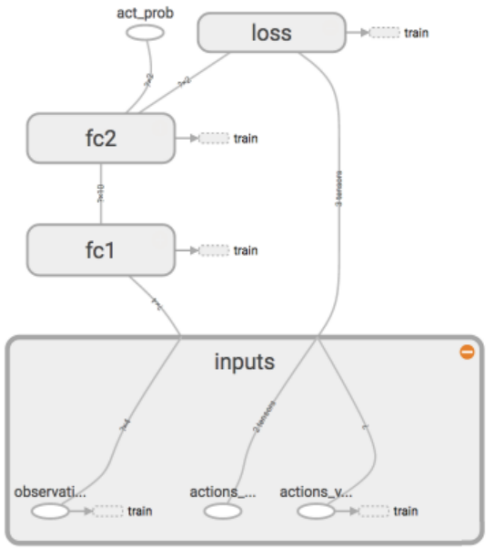


神经网络：

- 神经网络结构可以自行设计的
- 输入为当前棋盘，输出为双端口，分别表示当前棋盘的状态值（value）和当前棋盘各个位置的走子的概率，这里的棋盘状态值，即机器评价的当前局面的“好坏”

Policy Gradients

Main Graph



策略网络（Policy Gradients）

Policy Gradients:

- 基本思想，直接根据状态输出动作的概率 => 使用神经网络
- 作用，输入当前的状态，神经网络输出在这个状态下采取每个动作的概率
- 对于强化学习来说，我们不知道动作的正确与否，只能通过奖励值来判断这个动作的相对好坏
- 如果一个动作得到的reward多，那么就让它出现的概率增大，如果一个动作得到的reward少，就让它出现的概率减小
- loss= -log(prob)*vt

log(prob)表示状态s对动作a的吃惊程度，如果概率越小，-log(prob)越大，vt表示当前状态s下采取动作a所能得到的奖励（即当前的奖励和未来奖励的贴现值的求和）

- 因为要看到奖励，所以Policy Gradients算法必须通过完整的eposide才能进行参数更新，而不是像value-based方法那样，每一个(s,a,r,s')都可以进行参数更新
- 如果在prob很小的情况下，得到了很大的Reward，即vt数值高，那么-log(prob)*vt就更大，表示更吃惊，(选了一个不经常使用的动作，却发现能得到了一个很好的reward，那么就对这次的参数进行一个大幅调整)
- 如果这个回合选择某一动作，下一回合选择该动作的概率就会大一些，同时看奖惩值，如果奖惩是正的，那么会放大这个动作的概率，如果奖惩是负的，就会减小该动作的概率

策略网络（Policy Gradients）

Policy Gradients算法：

- 蒙特卡洛策略梯度reinforce算法，使用价值函数v(s)来近似
- 输入：N个蒙特卡洛完整序列,训练步长α
- 输出：策略函数的参数θ

for 每个蒙特卡洛序列:

a. 用蒙特卡洛法计算序列每个时间位置t的状态价值vt

b. 对序列每个时间位置t，使用梯度上升法，更新策略函数的参数θ:

$$\theta = \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) v_t$$

返回策略函数的参数θ

- 这里的策略函数可以是softmax策略，高斯策略或者其他策略

AlphaGo主逻辑

- AI = Policy Value Network + MCTS
- Policy Value Network，策略价值网络

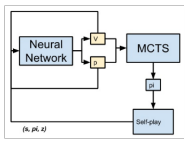
策略网络，输入当前的状态，神经网络输出在这个状态下采取每个动作的概率

价值网络，对于价值网络，当前局面的价值=对终局的估计

- MCTS，蒙特卡洛树搜索

可以提供很好的策略改善，它的输入是个普通的策略（normal policy），我们可以通过MCTS得到一个更好的策略（good policy）输出

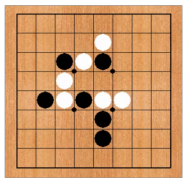
通过MCTS完成自我对弈，从而更新策略网络



AI实现

五子棋AI工程：

- game.py，定义了游戏的棋盘、获取棋盘状态，下棋（更新棋盘状态），判断是否有人获胜，绘制棋盘，两个player对弈，自我对弈
- human_play.py，人机对弈，人来输入下棋位置，调用AI进行对战
- mcts_alphaZero.py，实现AlphaGo Zero中的MCTS（蒙特卡洛树搜索），使用了策略网络来指导树搜索并计算叶节点
- mcts_pure.py，实现了随机走子策略的MCTS（蒙特卡洛树搜索）
- policy_value_net_pytorch.py，策略价值网络，用来指导MCTS搜索并计算叶子节点
- train.py，训练AI主程序



MCTS算法流程

MCTS算法：

- 树结构：树结构定义了一个可行解的解空间，每一个叶子节点到根节点的路径都对应了一个解（solution）
- 蒙特卡洛方法：MSTC不需要事先给定打标样本，随机统计方法充当了驱动力的作用，通过随机统计实验获取观测结果
- 损失评估函数：提供一个可量化的确定性反馈，用于评估解的优劣 => MCTS是通过随机模拟寻找损失函数代表的背后"真实函数"
- 反向传播线性优化：每次获得一条路径的损失结果后，采用反向传播（Backup）对整条路径上的所有节点进行整体优化
- 启发式搜索策略：算法遵循损失最小化的原则在整个搜索空间上进行启发式搜索，直到找到一组最优解或者提前终止

SoftMax计算

SoftMax：

- 机器学习中重要的计算工具，可以兼容logistics算法、可以独立作为机器学习的模型进行建模训练、还可以作为深度学习的激励函数
- 简单的说，就是计算一组数值中每个值的占比
- 假设一共有n个用数值表示的分类 $S_i, i \in (0, n$ ，n表示分类的个数，softmax计算公式为：

$$P(S_i) = \frac{e^{V_i}}{\sum_i e^{V_i}}$$

i表示k中的某个分类，gi表示该分类的值

- 假设有三个数值A=5,B=1,C=-1，那么他们的softmax占比为

$$P(A) = \frac{e^5}{e^5 + e + e^{-1}}$$

$$P(B) = \frac{e}{e^5 + e + e^{-1}}$$

$$P(C) = \frac{e^{-1}}{e^5 + e + e^{-1}}$$

计算结果为：P(A)=0.9817, P(B)=0.0180, P(C)=0.0003

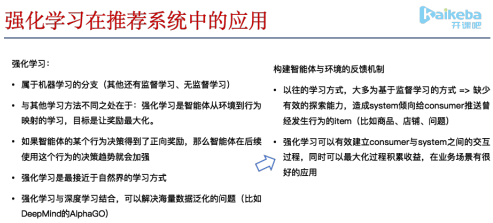
SoftMax计算

SoftMax特性：

- 归一化：最后的合计为1，即P(A)+P(B)+P(C)=1
- 放大效果：单纯从数值来看，5和1的差距并不大，但是通过指数运算有明显的放大效果，5的占比能到98%以上
- 散列性质，每一个比率虽然最后都会进行归一，但是他们放大之前的数值是可以相互不干扰的

```
# 定义SoftMax函数，求概率
def softmax(x):
    probs = np.exp(x - np.max(x))
    probs /= np.sum(probs)
    return probs
```

策略价值网络实现（基于PyTorch）



总结Summary

- 记录作业内容
  - Thinking1：机器学习中的监督学习、非监督学习、强化学习有何区别
  - Thinking2：什么是策略网络，价值网络，有何区别
  - Thinking3：请简述MCTS（蒙特卡洛树搜索）的原理，4个步骤Select, Expansion，Simluation，Backpropagation是如何操作的
  - Thinking4：假设你是抖音的技术负责人，强化学习在信息流推荐中会有怎样的作用，如果要进行使用强化学习，都有哪些要素需要考虑
  - Thinking5：在自动驾驶中，如何使用强化学习进行训练，请说明简要的思路
  - Action（五子棋）：
    - 棋盘大小 10 * 10
    - 采用强化学习（策略价值网络），用AI训练五子棋AI
    - 编写代码，说明神经网络（策略价值网络），MCTS原理
    - 生成五子棋AI模型 .model
    - 进行人机对弈
  -
-