

基于强化学习的黑白棋的设计与实现

Design and Implementation of Othello Based on Reinforcement Learning

Zhao Qian

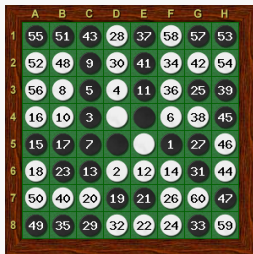
Adviser: Wang Jianhua

Yantai University
School of Computer and Control Engineering

2019.05.30

Why Reversi and Prior Work

Why Reversi?



- 以往算法的设计中大都使用博弈树搜索的方法
- 2017 年 AlphaGo Zero 出现了
- 打算将这种思路扩展到黑白棋中

Prior Work

Minimax Tree Search + Alpha-Beta pruning

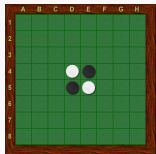
- 搜索空间巨大（随搜索层数指数级增加）
- 棋力受限于搜索的层数（理想时间内很难提高）
- 棋力受限于设计者的能力（如估值函数的设计）

Monte Carlo Tree Search

- 无需任何领域知识便可工作
- 非对称式增长，算法会频繁地访问“更感兴趣”的节点，并聚焦搜索空间于更加相关树的部分
- 算法可在任何时间终止，并返回当前最优的估计

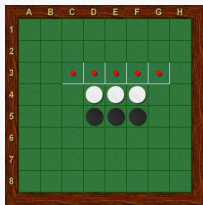
Reversi: Basic Rules

How to play

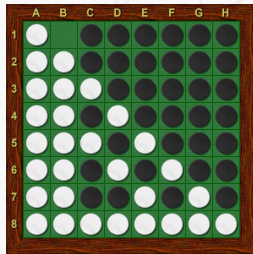


- 棋盘 8×8 大小
- E4、D5 为黑棋
- D4、E5 为白棋
- 执黑先手

- 落子必须在一空位
- 落子必须造成翻转
- 无子可走则本轮 pass
- 有子可走则必须走棋



How to win

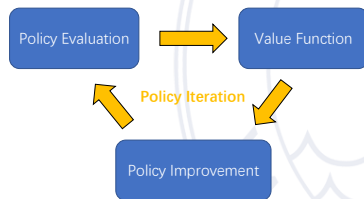
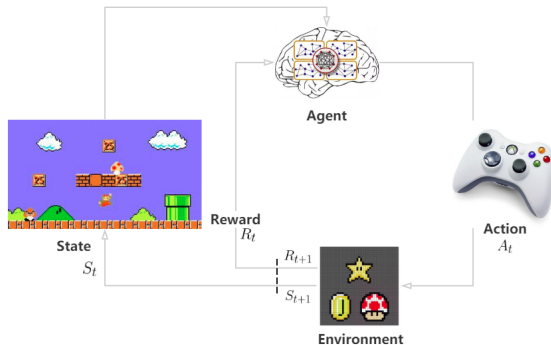


游戏结束:

- ① 双方都无子可下
- ② 棋子数目多者获胜，若相等为平局

Reinforcement Learning

强化学习解决一类序贯决策问题，它不关心输入的数据长什么样子，它只关心当前采取什么样的行动才能使得最终的回报最大

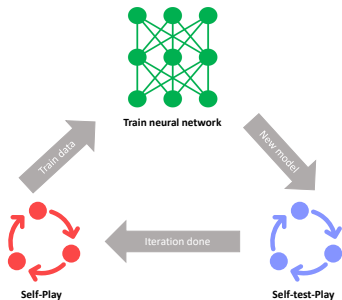


Structure Breakdown

神经网络 f_θ (predict)

$$(\vec{p}, v) = f_\theta(s)$$

- $\vec{p}_\theta(s_t)$: 当前玩家在状态 s_t 下选择每个动作的概率
- $v_\theta(s_t)$: 当前玩家在状态 s_t 下获胜的概率估计



Policy Evaluation

$f_\theta(s) \xrightarrow{(\vec{p}, v)} \text{MCTS} \xrightarrow{\vec{\pi}_t} \text{action}$
 到达终止态 s_T 获得本局胜利者 Z

Policy Improvement

通过 self play 得到一组 train data
 使用 train data 训练神经网络得出新 model
 新旧 model 进行对抗判断胜率是否超过阈值需要更新

Neural Policy and Value Network

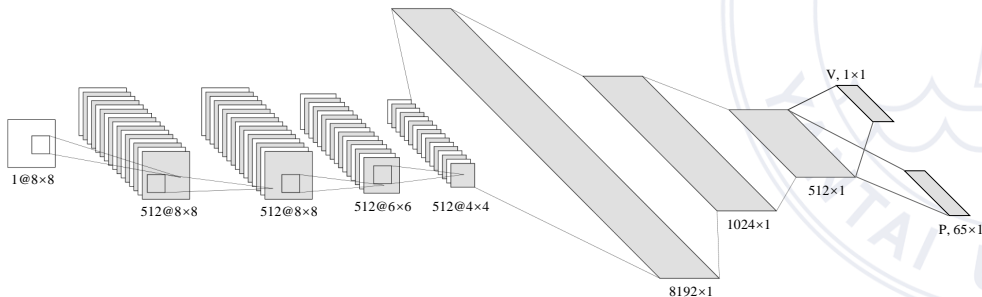
神经网络 f_θ (train)

Input: train data $(s_t, \vec{\pi}_t, z_t)$

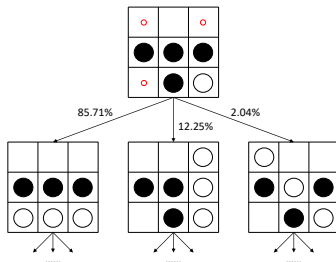
ToDo: 最小化 $\vec{p}_\theta(s_t)$ 与 $\vec{\pi}_t$, $v_\theta(s_t)$ 与 z_t 间的差距

$$Loss = \sum_t (v_\theta(s_t) - z_t)^2 - \vec{\pi}_t \cdot \log(\vec{p}_\theta(s_t))$$

NNet Structure (CNN)



Monte Carlo Tree Search for Policy Improvement



- $Q(s, a)$: 在状态 s 下采取动作 a 所得到的预期奖励
- $N(s, a)$: 所有模拟中在状态 s 下采取动作 a 的次数
- $P(s, \cdot) = \vec{p}_\theta(s)$: 在状态 s 下采取动作的先验概率
- $U(s, a)$: Q value 的置信区间上界

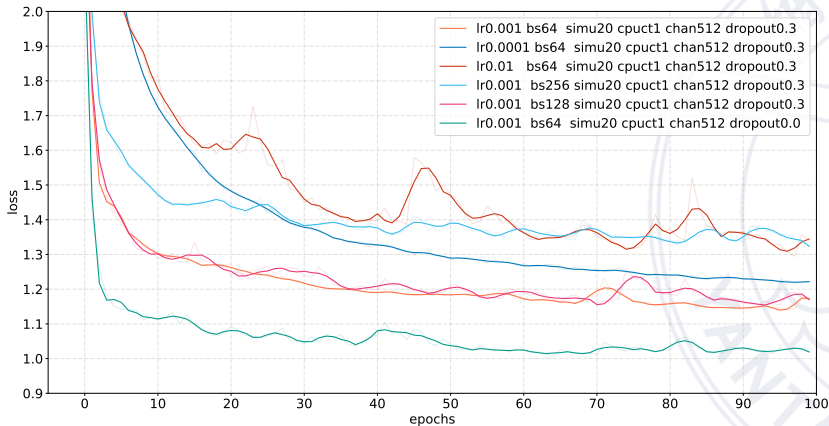
其中 $U(s, a) = Q(s, a) + c_{puct} \cdot P(s, a) \cdot \frac{\sqrt{\Sigma_b N(s, b)}}{1 + N(s, a)}$

```

1: procedure MCTS( $s, \theta$ )
2:   if  $s$  is terminal then
3:     return game_result
4:   end if
5:   if  $s \notin \text{Tree}$  then
6:      $\text{Tree} \leftarrow \text{Tree} \cup s$ 
7:      $Q(s, \cdot) \leftarrow 0$ 
8:      $N(s, \cdot) \leftarrow 0$ 
9:      $P(s, \cdot) \leftarrow \vec{p}_\theta(s)$ 
10:     $v \leftarrow v_\theta(s)$ 
11:    return  $v$ 
12:   else
13:      $a \leftarrow \arg \max_{a' \in A} U(s, a')$ 
14:      $s' \leftarrow \text{get\_next\_state}(s, a)$ 
15:      $v \leftarrow \text{MCTS}(s', \theta)$ 
16:      $Q(s, a) \leftarrow \frac{N(s, a) \times Q(s, a) + v}{1 + N(s, a)}$ 
17:      $N(s, a) \leftarrow N(s, a) + 1$ 
18:     return  $v$ 
19:   end if
20: end procedure
  
```

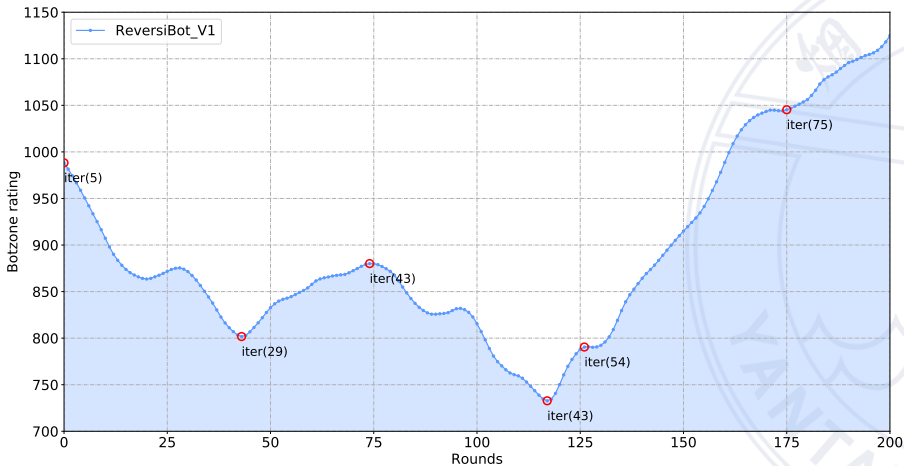
Experiments and Analysis

一块 Tesla T4 中训练了 150h, 共迭代 120 次, 其中成功 29 次, 平均每次成功迭代耗时 5.17h



不同超参数配置下的 LOSS 变化曲线

Experiments and Analysis



Elo rating 随迭代版本的变化曲线

Experiments and Analysis

不同算法实现的 AI 与本方对局结果

基准	本方胜场数 / 总场数	本方胜率
基于随机策略	80/80	100%
基于可转换棋子最大的贪心	80/80	100%
基于使对方行动力最小的贪心	80/80	100%
极大极小搜索 + Alpha-Beta 剪枝	56/80	70%

Conclusions and Prospect

一些结论

- 人们往往认为机器学习只有在大数据的作用下才能发挥作用，但这是不准确的
- 有些时候设计一个好的自学习算法比数据的输入更重要
- 不需要任何人类知识的输入，它总是和一个与自己棋力相当的 AI 对局并从中总结经验改进自己
- 使用强化学习方法相比于传统算法预测时间短（但需预先进行训练），可提升空间大

期间遇到的问题（已解决）

- Keras 多线程预测产生的线程安全问题（**预先将图处理为静态的**）
- 多进程 self-play fork 时产生的内存资源消耗严重的问题（**减少不必要的内存复制**）

展望未来

- 即使后来的 AlphaZero 可以挑战 Dota2 等相对于围棋更为复杂的游戏，但这些游戏相对于现实世界来说仍然属于简单问题
- 步入强人工智能领域还需要走很长的路，也希望我们未来的生活会因此变得更加美好

Acknowledgement

- 感谢王建华老师在毕设期间的指导
- 感谢 ACM 实验室卢云宏老师、周世平老师、封玮老师
- 感谢班主任毕远伟老师
- 感谢四年里遇到的各位老师和同学

请多提宝贵意见