课程名称: 算法设计基础

指导老师: 王万元

第一次报告

version 0.1

2017/3/18

GomokuAI设计

1.相关调查及初步设计

Environment: Python 2.7

3/5
首先想到的是利用机器学习来完成设计.
通过搜索,能看到现在有多种五子棋实现的思路,例如基于最小值最大值搜索和Alphabeta剪枝的五子棋程序. 五子棋和围棋有诸多相似之处,我想到了之前大致看过Nature上AlphaGo的论文,似乎有共通之处. 计划仿照其结构实现此AI.
计划
1.使用MCTS初步实现五子棋程序
2.在简单实现的基础上对算法进行改进(减少计算时间及计算资源,提高准确度等)
3.进一步学习卷积神经网络,策略网络,构建落子评估器.
4.尝试寻找棋谱资源(或者自我对弈?),进行训练.
5.学习构建价值网络(Value Network),将其与第一步的落子评估器线性耦合.
6.进行大量训练.
考虑的点:
1.如果要求botzone的话,内存等相关要求估计不能满足限制.
2.单纯从本课程来说,使用神经网络感觉并没有算法相关的东西.感觉像走一套流程
3.如果可能的话也希望能学习一些其他的相关算法

3/7
现在已经使用简单的MCTS制作了井字棋程序(类似于五子棋的简化版),计划下一步进行五子棋的实现,并构建效果更好的MCTS模型.由于有其他任务和CNN相关,决定再先完成另一任务,对CNN有进一步理解之后,再回来构建落子评估器并寻找棋谱进行训练.

3/8
又想到了提供棋谱(训练)的方法(但是就需要对棋谱做preprocessing,感觉会花些功夫)
或从Gomokucup AI比赛中找到一些强大的程序,来做'陪练',解决训练问题.

Preference:
Mastering the game of Go without human knowledge
Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

2.Code Structure

现在实现了基本的使用MCTS的程序,还没有到botzone测试,因为自己在评估的时候发现了诸多问题,详见后面分析部分

1.整体结构

```
1   # coding=utf-8
2
3   from math import *
4   import random
5
6
7   class GomokuState:...
205
206
207 class Node:...
265
266
267 def UCT(rootState, maxIter):...
306
307
308 def Play():...
335
336
337 if __name__ == "__main__":...
341
```

2.GomokuState

定义棋盘状态

```
class GomokuState:
    def __init__(self):...
    def Clone(self):...
    def DoMove(self, move):...
    def GetMoves(self):...
    def GetResult(self, playerjm):...
    def __repr__(self):...
```

分别用于复制,落子,返回剩余位置,判断是否结束,控制台调试

3.Node&UCT

```
class Node:
    def __init__(self, move=None, parent=None, state=None):...
    def UCTSelectChild(self):...
    def AddChild(self, m, s):...
    def Update(self, result):...
    def __repr__(self):...
```

```
245     def UCT(rootstate, itermax):
246
247         rootnode = Node(state=rootstate)
248
249         for i in range(itermax):
250             node = rootnode
251             state = rootstate.Clone()
252
253             # Select
254             while node.untriedMoves == [] and node.childNodes != []:...
257
258             # Expand
259             if node.untriedMoves != []:...
263
264             # Rollout
265             while state.GetMoves() != []:
266                 state.DoMove(random.choice(state.GetMoves()))
267
268             # Backpropagate
269             while node != None:...
273
```

```
def UCTPlayGame():

    state = GomokuState()
    breakFlag = False
    while (state.GetMoves() != []):
        print str(state)
        if state.playerJustMoved == 1:
            m = UCT(rootstate=state, itermax=8000)  # 一方的落子数据
        else:
            m = UCT(rootstate=state, itermax=8000)  # 另一方的落子数据,可以加个接口
        print "Move: " + str(m) + "\n"
        state.DoMove(m)
        if state.GetResult(state.playerJustMoved) == 1.0:
            print "Player " + str(state.playerJustMoved) + " wins!"
            breakFlag = True
            break
        elif state.GetResult(state.playerJustMoved) == 0.0:
            print "Player " + str(3 - state.playerJustMoved) + " wins!"
            breakFlag = True
            break
    if breakFlag == False:
        print "Nobody wins!"
```

3.Analysis

自己offline测试时,考虑了一些问题

问题:对内存和时间消耗大.

搜索空间很大,迭代次数很多,每一步消耗的时间很长.

对于8x8的棋盘,每步最多迭代4000次,每步大约消耗5s的时间,基本以和局结束.若调小一方的最大迭代次数,如4000次对3000次,则明显3000次输的多

改进:可以考虑加入剪枝策略

改进:改进里的一些函数,提高速度,如并行化

例:判断是否结束的函数中,写的是按不同方向遍历整个棋盘,效果很差.想的改进方向是以当前落子点为中心,遍历4个方向判断即可,应该能显著减少时间.

还有其中的一些for循环可以并行化,也能减少不少时间

问题:效果差

如下图,落子集中在上半区,反映的是到达的搜索空间很小,即速度,时间,空间都很差....

```
# up

for row in range(self.size):
    opWinPin = 0
    seWinPin = 0
    for col in range(self.size):
        if self.Board[row][col] == playerSelf:
            opWinPin = 0
            seWinPin += 1
            if seWinPin == 5:
                return 1.0
        elif self.Board[row][col] == playerOppe:
            seWinPin = 0
            opWinPin += 1
            if opWinPin == 5:
                return 0.0
        else:
            seWinPin = 0
            opWinPin = 0

# left and right

for col in range(self.size):
    opWinPin = 0
```

```
...0.X..OO.0...
..........X....
.........X.0...
....0X.........
...............
...............
...X...........
..X............
...............
...............
...............
...............
...............
....X..........
...............
```

问题:感觉考虑网站上的限制,MCTS并不是适合的做法,在规定时间内,迭代次数较少,连我都打不过...

MinMax搜索和AlphaBeta剪枝更适合一些.

但是还想继续按神经网络的思路做下去