本科生毕业论文（设计）

题　　目　基于深度增强学习的五子棋人工智能实现

学　　院　计算机学院

专　　业　计算机科学与技术

学生姓名　王点

学　　号　2013141462018　年级　2013

指导教师　潘薇

教务处制表

二O一七 年 五 月 二十 日

基于深度增强学习的五子棋人工智能实现

计算机科学与技术

学生 王点 指导老师 潘薇

摘要：深度增强学习自问世以来，受到了国内外学者的广泛关注与研究，一跃成为机器学习及人工智能领域的前沿研究方向之一。深度增强学习基于增强学习的核心思想，使用深度学习的方法进行值函数近似，已经被广泛用于包括计算机博弈在内的各个领域，国内对于深度增强学习的研究与应用尚处于初级阶段。五子棋是计算机博弈的重要分支之一，目前国内外主要使用树搜索加剪枝的技术完成五子棋的人工智能设计。本文基于深度增强学习，研究了增强学习与深度增强学习的关键技术，将深度增强学习应用于五子棋游戏环境，完成了基于深度增强学习的五子棋人工智能智能体。该智能体使用卷积神经网络对预处理之后的棋盘进行评估，计算各个落子动作的 Q 值，选取 Q 值最大的落子动作进行执行，随后根据计算 Q 值与期望 Q 值的差异进行学习。针对不同学习阶段，本文完成了快速落子算法以及ϵ-贪婪策略以增加智能体的学习效率。针对五子棋的游戏特征，本文设计了价值模型以在不同局面下返回给智能体不同的奖励值。在 50 万步训练之后，智能体达到了 55%的胜率。

关键词：增强学习，深度增强学习，DQN，卷积神经网络，五子棋

Playing Gomoku with Deep Reinforcement Learning

Computer Science and Technology

Student: Wang Dian Adviser: Pan Wei

Abstract: Since deep reinforcement learning has been widely concerned by scholars at home and abroad, it has become one of the leading research directions in the field of machine learning. Deep reinforcement learning based on the core idea of reinforcement learning, using deep learning method for value function approximation, has been widely used, including computer game strategy, while the domestic research and application of deep reinforcement learning is still in the early stage. Gomoku is one of the important branches of the computer game strategy. By now the tree search and pruning algorithm was mainly used to design Gomoku AI. Based on deep reinforcement learning, this paper studied the key technologies of reinforcement learning and deep reinforcement learning, and applied the deep reinforcement learning to the game environment of Gomoku, and completed the agent playing Gomoku based on deep reinforcement learning. The algorithm uses the convolution neural network to evaluate the chessboard after the preprocessing, calculates the Q value of each action, select the maximum and play, and then learn from the difference between the Q value it calculated and the expected Q value. For the different learning stages, this paper completed the fast-move algorithm and the ϵ-greedy strategy to increase the learning efficiency of the agent. For the game features of Gomoku, this paper designs a value model to return the different reward to the agent in different situations. After 500 million steps of training, the agent reaches win rate of 55%.

keyword: reinforcement learning, deep reinforcement learning, DQN, convolutional neural network, Gomoku

目录

1. 绪论

1.1. 课题背景

增强学习是一种多阶段接收环境反馈的机器学习方法，其学习目标是建立从环境状态到动作的映射关系，从而使得系统的一系列动作从环境中获得的累计奖励值最大。

深度增强学习是将深度学习中的神经网络与增强学习相结合的一个崭新领域，其起源于 DeepMind 小组所开发的 DQN 方法[1]。

五子棋是一种两人对弈的纯策略型棋类游戏，是世界智力运动会竞技项目之一，双方分别使用黑白两色的棋子，下在棋盘直线与横线的交叉点上，先形成 5 子连线者获胜。

1.2. 研究意义

传统的棋类游戏博弈算法主要采用树搜索与剪枝技术，其棋力在算法编写完成之时便已经确定，而本文研究的深度增强学习方法通过在不断的对局中进行学习，提升自己的棋力。深度增强学习相关算法的研究对机器学习及人工智能各个领域将有实际的应用价值，对博弈论、机器博弈等领域的发展会起到积极的推进作用。

自 DQN 算法问世至今已有四年的时间，其所代表的深度增强学习已经成为机器学习领域的前端方向，但国内对 DQN 算法的研究与应用尚处于起步阶段，因此本文也希望对 DQN 算法在国内的应用与发展做出贡献。

1.3. 国内外研究现状

当今主流的五子棋引擎以及五子棋人工智能文献主要采用树搜索与剪枝的技术，包括 Proof-number Search，Dependency-based Search[3]等算法。

DeepMind 小组在文献[1]中引入了 DQN 算法，随后在文献[8]中发表了 DQN2.0 版本。这两篇文献均采用 DQN 算法训练智能体玩 Atari 游戏。

国内的深度增强学习文献较少，文献[9]通过对 VDBE（Value-Difference Based Exploration）方法的拓展应用提出了一种基于自适应探索改进的深度增强学习算法。

1.4. 目前研究所存在问题

尽管国内外对于深度增强学习的研究与实践和对五子棋人工智能算法的研究已取得

了很大进展，但依然存在部分问题：

1. 主流五子棋 AI 算法依然停留在以树搜索与剪枝算法得出最优解的较为经典的思想上。
2. 深度增强学习算法的研究与应用在国内尚处于起步阶段。
3. DQN 算法的研究与应用在国内尚处于起步阶段。
4. 对于动作空间较大的智能体，其学习速度较慢，期望值与实际值损失的收敛速度较慢。

1.5.本文工作

根据上述研究现状，本文主要进行了以下几个方面的研究工作：

（1） 将深度增强学习的思想以及其代表算法 DQN 算法应用于五子棋中，实现了基于 DQN 算法的五子棋人工智能。

（2） 针对五子棋游戏特性，对棋盘特征进行预处理，使得神经网络能更准确地提取棋盘特征。

（3） 实现了ϵ-贪婪策略及快速落子算法，使智能体在探索阶段避免无意义的落子，加快学习速度。

2. 系统结构

本文实现的基于 DQN 的五子棋人工智能在实现上可分为环境部分与智能体部分，两部分的功能与训练流程如下：

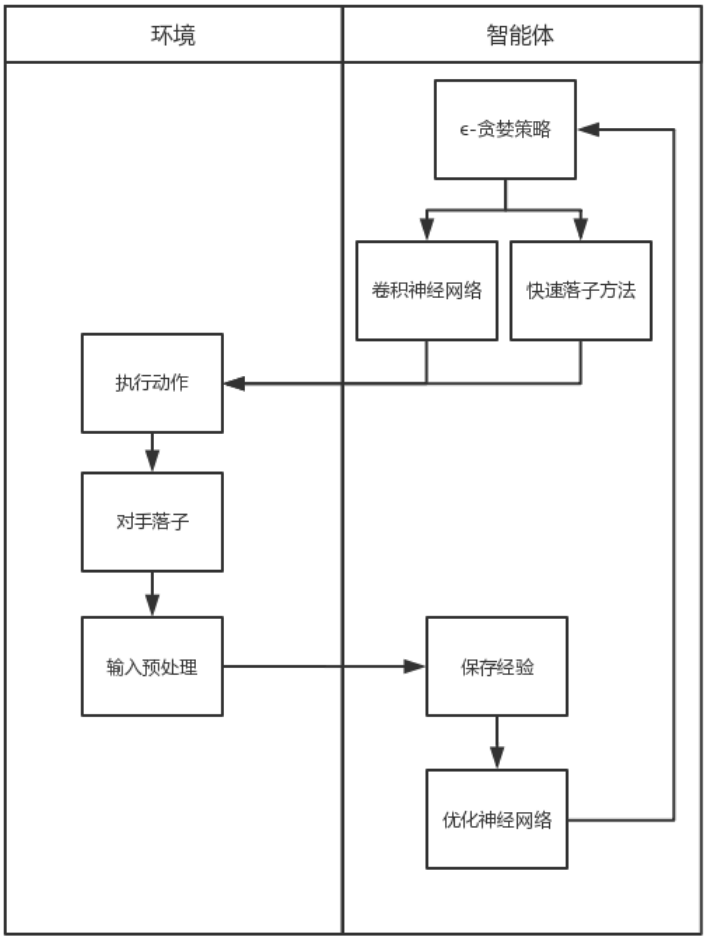


图 2-1 基于 DQN 的五子棋人工智能结构流程图

智能体部分首先使用ϵ-贪婪策略选择使用快速落子方法或卷积神经网络计算落子位置，随后将落子结果输入到环境中，环境完成我方落子和对方落子后对棋盘进行处理（即神经网络的输入预处理）获得观测结果，将奖励值、观测结果、游戏是否结束传回给智能体，智能体将此步经验进行保存，随后从经验池中选取经验对神经网络模型进行训练。

3. 环境

3.1. 价值模型

3.1.1. 棋型

五子棋中，重要的棋型有如下几种：

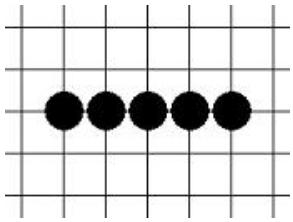


图 3-1 连 5

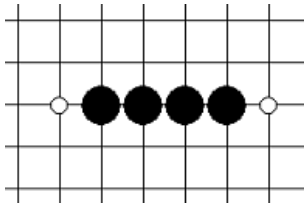


图 3-2 活 4

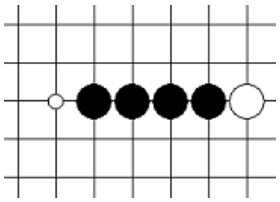


图 3-3 冲 4

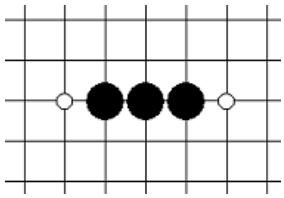


图 3-4 活 3

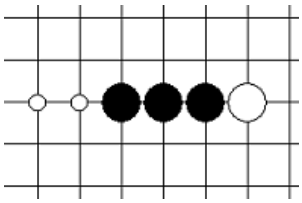


图 3-5 眠 3

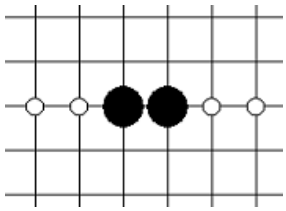


图 3-6 活 2

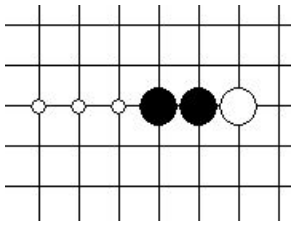


图 3-7 眠 2

3.1.2. 奖励值

表 3-1 奖励值

棋型	奖励值
连 5	±10
活 4	±5
冲 4	±2
活 3	±1

我方达成棋型获得正奖励值，敌方达成获得负奖励值，五子棋游戏进程中双方交替落子，因此除我方连 5 或敌方连 5 直接获得奖励值±10 以外，其余情况我方落子的奖励值为我方棋型奖励值+对方棋型奖励值。

3.2. 输入预处理

3.2.1. 进攻价值

表 3-2 进攻价值

进攻前的棋型	距离获胜的步数	进攻后的棋型	进攻价值
活 4	1	连 5	5
冲 4	1	连 5	5
活 3	2	活 4	4

表 3-2（续）

眠 3	2	冲 4	3
活 2	3	活 3	2

3.2.2. 防守价值

表 3-3 防守价值

防守前的棋型	距离获胜的步数	防守价值
冲 4	1	4
活 3	2	3

3.2.3. 卷积神经网络的输入

受文献[23]启发，本文对棋盘状态进行抽象，使用数个 13*13 大小的二进制数组作为输入：

表 3-4 卷积神经网络的输入

特征	层数	描述
黑/白	2	黑子和白子位置
落子范围	1	下一步的落子区域
进攻	4	具有进攻价值的区域（对应 4 种进攻价值）
防守	2	具有防守价值的区域（对应 2 种防守价值）
共计	9	

输入的前两层是全部黑子和白子的位置。第三层是落子范围，本文规定落子范围为当前有子的区域以外两格内，此限制可以减少智能体对无意义的位置进行尝试。

随后四层是具有进攻价值的区域，分别对应进攻价值 2、3、4、5。最后两层是具有防守价值的区域，分别对应防守价值 3、4。

此 9 层二进制的卷积神经网络输入便是智能体从环境获得的观测结果。

下面进行举例说明：

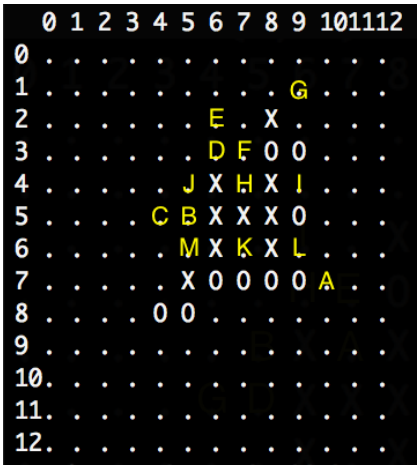


图 3-8 棋局示例

对于图 3-8 所示的棋局，我方为黑（X）棋，敌方为白（0）棋。

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	1	0	0	0	0
5	0	0	0	0	0	0	1	1	1	0	0	0	0
6	0	0	0	0	0	0	1	0	1	0	0	0	0
7	0	0	0	0	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0

(a)黑子的位置

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	1	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	1	1	1	1	0	0	0
8	0	0	0	0	1	1	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0

(b)白子的位置

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	1	1	1	1	1	0	0
1	0	0	0	0	0	0	1	1	1	1	1	1	0
2	0	0	0	0	1	1	1	1	0	1	1	1	0
3	0	0	0	0	1	1	1	1	0	0	1	1	0
4	0	0	0	0	1	1	0	1	0	1	1	1	0
5	0	0	0	1	1	1	0	0	0	0	1	1	0
6	0	0	0	1	1	1	0	1	0	1	1	1	0
7	0	0	1	1	1	0	0	0	0	0	1	1	0
8	0	0	1	1	0	0	1	1	1	1	1	1	0
9	0	0	1	1	1	1	1	1	1	1	1	1	0
10	0	0	1	1	1	1	1	1	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0

(c)落子区域

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	0	0	0	0
4	0	0	0	0	0	0	1	0	1	0	1	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	1	0	1	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0

(d)进攻价值为 2 的点

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	1	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0

(e)进攻价值为 3 的点

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0

(f)进攻价值为 4 的点

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0

(g)进攻价值为 5 的点

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0

(h)防守价值为 3 的点

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0

(i)防守价值为 4 的点

图 3-9 9 层输入

3.3. 对手

本文选用了 github 平台上的一个基于树搜索的五子棋 AI[24]作为对手进行训练。为避免训练样本的过度重复，本文规定以对手以 0.7 的概率使用此算法落子，0.3 的概率使用后文所述的快速落子方法落子。

4. 智能体

4.1. 主循环

主循环是智能体不断循环执行的代码。

主循环

```
while true:
    #ϵ-贪婪策略
    action = select_action(state)
    #执行动作 action
    next_state, reward, done = step(action)
    #若游戏结束
    if done:
        next_state = None
        #初始化环境
        env.restart
    #将四元组<state, action, next_state, reward>保存在经验池中
    memory.push([state, action, next_state, reward])
    #训练神经网络
    optimize_model
```

4.2. ϵ-贪婪策略

在智能体学习的初步阶段，智能体需要做足够多的随机动作尝试以获得训练样本，之后，智能体需要更多地遵循自己的计算结果选择动作以对神经网络进行更深层的训练。ϵ-贪婪策略的意义便是充分结合二者。

select_action(ϵ-贪婪策略)

```
#eps 是ϵ的简写
eps_threshold = EPS_END + (EPS_START - EPS_END) * e^(-1*(steps_done/EPS_DECAY))
random = random(0, 1)
if random > eps_threshold:
    计算 Q 值，选择 Q 值最大的动作
else:
    随机选择动作
```

本文规定，初始ϵ值（EPS_START）为 0.9，最小ϵ值(EPS_END)为 0.1，ϵ值衰退率（EPS_DECAY）为 2000。

通过ϵ-贪婪策略，智能体最初会以 0.9 的概率随机选择动作执行，之后随着执行步数的增加，随机选择动作的概率会对数减小，当步数达到一定程度时，最终智能体随机选择动作的概率将会无限趋近于 0.1。

4.3. 经验池

经验池保存数个经验单元，经验单元为一个四元组，包括当前状态，下一状态，动作，以及奖励值。智能体在进行神经网络训练时，从经验池中随机取样进行训练。经验池技术与普通的基于单个样本的训练方式的优点在于：

1. 经验的每一步都可能用于多次训练，提高了数据效率。
2. 由于样本之间的强相关性，直接从连续样本学习是低效的，随机化样本会破坏这些相关性。
3. 训练时，当前的神经网络参数确定下一个参数被训练的样本。例如，某一局游戏双方各下 20 步棋，那么如果使用基于单个样本的训练方式，智能体的第二步落子将影响随后 18 次训练的样本，这样参数可能会被困在局部极小的地方，甚至发生灾难性的发散。经验池将平滑学习并避免参数中的振荡或发散。

在本文中，经验池大小为 10000，每次训练随机采样 128 组。

4.4. Q 值计算

本文规定期望 Q 值的计算方式为：

$$Q_i(s_i, a_i) = r_i + \gamma \max_a Q_{i+1}(s_{i+1}, a) \tag{4-1}$$

其中，r_i为智能体第i步落子所获得的奖励值，γ为折扣系数，本文规定$\gamma = 0.9$，$\max_a Q_{i+1}(s_{i+1}, a)$为下一步所有可能落子的 Q 值的最大值。

4.5. 卷积神经网络的结构

本文将卷积神经网络用于值函数近似，即计算每个落子位置的 Q 值。在训练过程中，神经网络将不断调整参数以更加准确地计算 Q 值，在每个训练周期中，卷积神经网络将从经验池中取 128 组经验单元，并对这些经验的 Q 值计算结果与期望的 Q 值只差使用随机梯度下降方法（学习率=0.128）进行参数调整，以达到学习的目的。

卷积神经网络的输入是 3.2.3 节所述的预处理后的棋盘描述，输出 13*13 个 Q 值，随后智能体将选择 Q 值最大的位置进行落子。

在本文所实现的最成功的卷积神经网络中，首个卷积层的输入频道为 9，对应前文所述的 9 层输入。输出频道为 64，卷积核大小为 5，步长为 1，零填充为 2。第二个卷积层的输入频道为 64，输出频道为 128，卷积核大小为 3，步长为 1，零填充为 1。第三个卷积层的输入频道与输出频道均为 128，卷积核大小为 3，步长为 1，零填充为 1。每个卷积层之后配有与卷积层输出频道大小相等的 BatchNorm 层以规范化本层输出，随后是非线性处理层，之后连接到下一卷积层。最后一层是全连接层，输出大小为 13*13，对应 13*13 个落子空间。

除去 3 层，最大卷积核大小为 128 的卷积神经网络外，本文在调整参数的过程中还实现了包括 3 层，最大卷积核大小为 64；2 层，最大卷积核为 128；2 层，最大卷积核为 64 在内的三个额外的卷积神经网络以进行对比。

无意义或非法的落子区域的 Q 值如果过大，会干扰期望 Q 值的计算，使得神经网络的计算结果非常不稳定。因此本文在神经网络计算结果输出之前，对非落子区域的计算结果做-100 操作。

4.6. 快速落子方法

在建立神经网络后，智能体需要随机选择动作以产生训练样本，但由于五子棋的可选动作范围过大，采用完全随机动作产生训练样本的学习效率不佳。因此本文为提高智能体的学习效率，编写了快速落子方法。

快速落子方法

```
atk1 = (进攻价值=2 的区域)
atk2 = (进攻价值=3 的区域)
atk3 = (进攻价值=4 的区域)
atk4 = (进攻价值=5 的区域)
dfs1 = (防守价值=3 的区域)
dfs2 = (防守价值=4 的区域)
#若有活 4，直接落子获胜
if atk4 is not empty:
return random choose from atk4
#若敌方有冲 4，敌方将在一步内获胜，必须防守
else if dfs2 is not empty:
pool = dfs2
#若能产生活 4，我方将在两步内获胜
else if atk3 is not empty:
pool = dfs3
```

```
#若敌方有活 3，敌方将在两步内获胜，必须防守
else if dfs1 is not empty:
    #若能冲 4，敌方必须在下一步防守，我方可能双 4 获胜，因此以概率 0.3 选择冲 4
if atk2 is not empty and random(0,1) > 0.7:
    pool = atk2
#对敌方活 3 进行防守
else:
    pool = dfs1
#对冲 4 或活 3 进行进攻
else if atk2 is not empty or atk1 is not empty:
pool = atk2 + atk1
#在落子范围内随机落子
else:
pool = (落子范围)
#将落子范围控制在当前有子的矩形外围 1 格（注 1）
delete some actions from pool
return random choose from pool
```

注 1：此步操作的目的是避免产生过于孤立的我方棋子，例如：

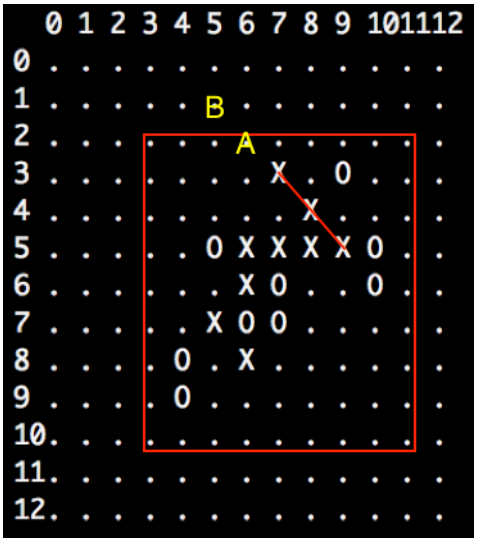


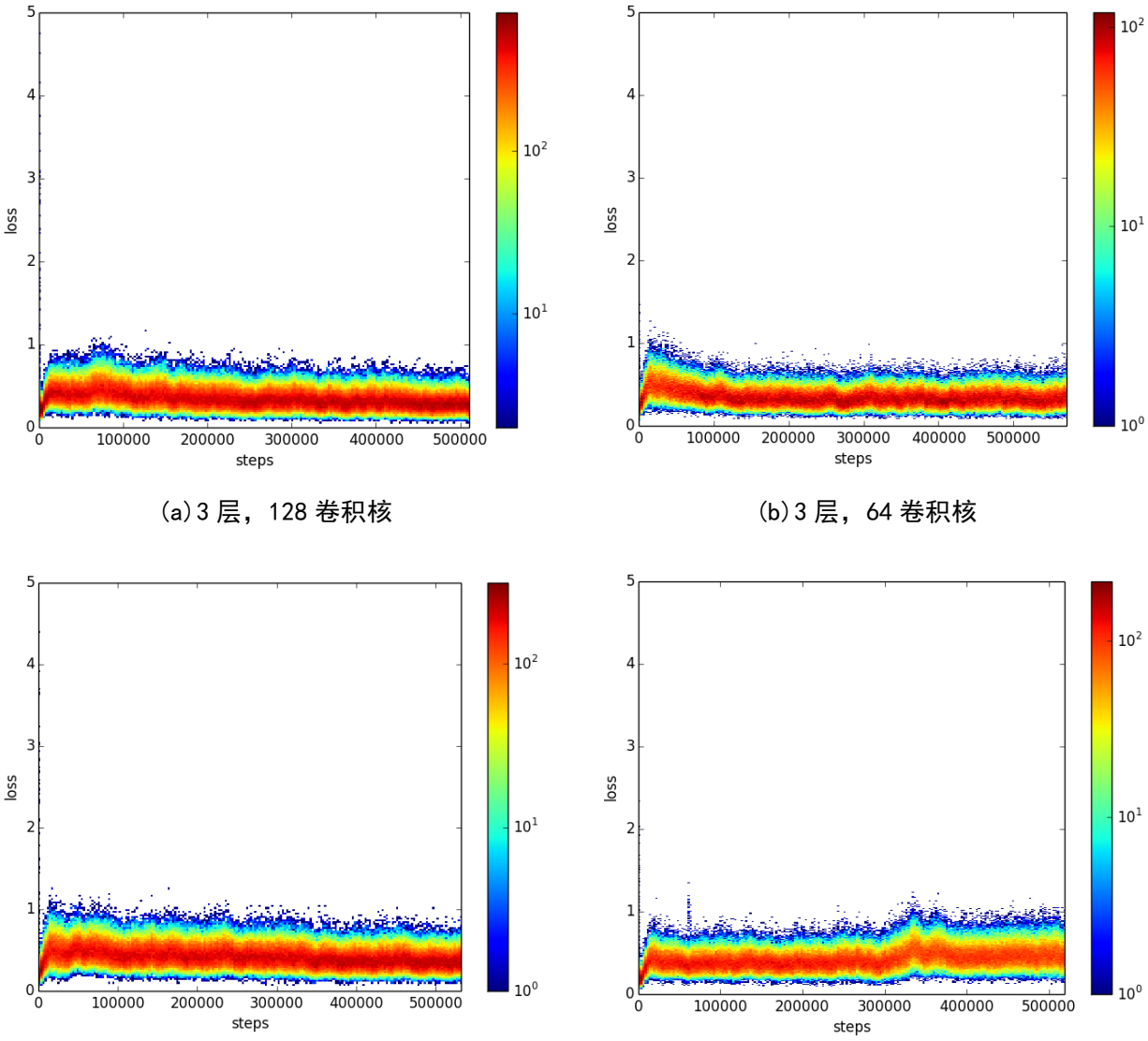
图 4-1 棋局举例

此时对于 X 子而言，落子 A 明显优于 B。虽然两步都可以产生冲 4，但 B 点过于孤立，相比之下 A 点更容易在未来产生价值。

5. 结果分析

本文的实验环境为 ubuntu16.04.1；macOS 10.12.3；Python2.7。本文使用同样的 DQN 算法，分别对 3 层 128 卷积核；3 层 64 卷积核；2 层 128 卷积核；2 层 64 卷积核四个卷积神经网络进行了训练，每个卷积神经网络都进行了超过 50 万次落子。表现最好的模型是 3 层 128 卷积核的卷积神经网络，其 loss 值更为收敛，平均 Q 值更大，胜率达到了 55%。

5.1. loss 值



(a)3 层，128 卷积核　　(b)3 层，64 卷积核

(c)2 层，128 卷积核　　(d)2 层，64 卷积核

图 5-1 loss 值

loss 值是期望 Q 值与神经网络计算出的 Q 值之差，体现了智能体对 Q 值计算的准确程度。若 loss 值收敛于 0，证明神经网络模型是有效的。图 5-1 中的四幅图分别是四种

结构的卷积神经网络的 loss 值随步数的密度变化图，横轴是智能体落子的总步数，纵轴是 loss 值大小，颜色代表了 loss 值的分布密度。除 2 层 64 卷积核的神经网络外，其余三幅图都有收敛趋势，同时左上图的收敛趋势最明显。

5.2. 平均 Q 值

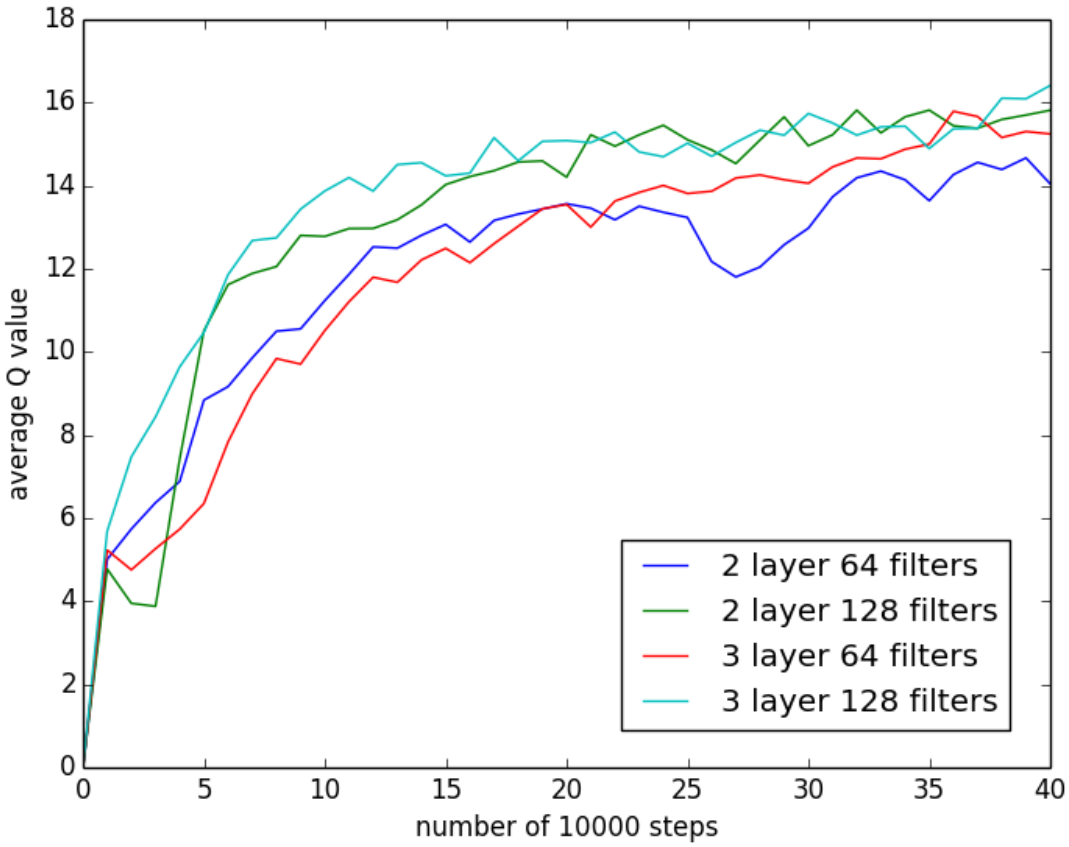


图 5-2 每 10000 步平均 Q 值

本图体现了智能体在 10000 步内的平均 Q 值随落子步数变化的趋势。横轴的单位是 10000 步，纵轴是 10000 步内 Q 值的算术平均值，如图可见，智能体计算出的 Q 值逐渐提升，体现出本文所实现的算法可以使智能体逐渐学会优秀的落子策略，同时 3 层最大卷积核为 128 的卷积神经网络的 Q 值计算总体大于其他三个神经网络。

5.3. 胜率

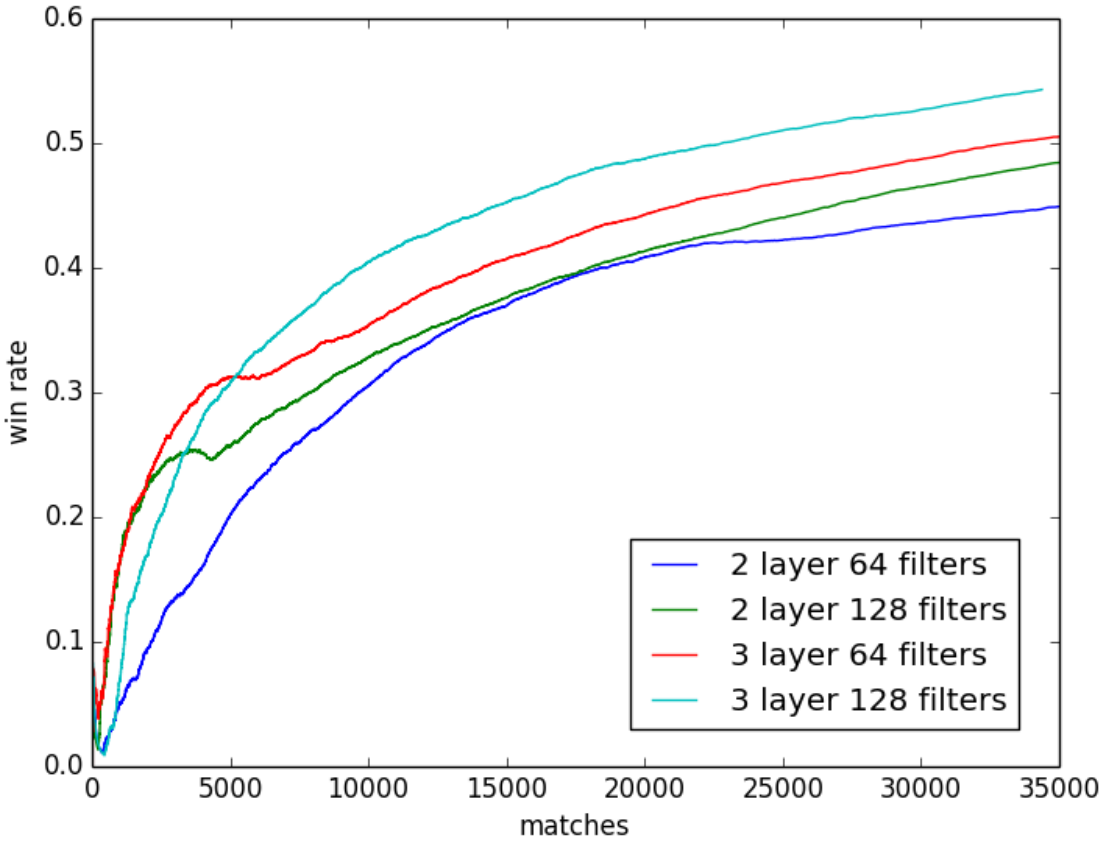


图 5-3 胜率

本图的横轴是对局数，纵轴是智能体的胜率。如图可见，3 层最大卷积核为 128 的卷积神经网络的胜率总体大于其他三个神经网络。此外，更高层、卷积核更大的卷积神经网络在本文所实现的算法中的表现更好。3 层 128 卷积核的卷积神经网络在约 50 万次落子（约 3.5 万次对局）的训练后胜率约为 55%。

6. 总结与展望

本文基于增强学习与深度增强学习的思想，完成了基于深度增强学习的五子棋人工智能。本文的具体成果如下：

1. 实现了基于 DQN 算法的五子棋人工智能智能体。
2. 实现了用于对智能体进行训练的五子棋游戏环境。
3. 针对五子棋的游戏特性，建立了奖励值模型和进攻防御棋型的价值模型。
4. 实现了ϵ-贪婪策略和快速落子方法，以加快智能体的学习速度。

6.1. 不足

本文目前阶段训练成果的智能体具备了一定的棋力，但还不足够完善：

1. 由于设备硬件条件限制，本文所实现的智能体的训练次数不足，智能体表现还未达到最优。
2. 本文只对智能体下黑子的情况进行了训练与分析。
3. 快速落子方法，非落子区域处理等方法在很大程度上增加了智能体的学习效率，但同时可能限制了智能体的落子选择。
4. 本文尝试实现了策略网络，但未成功。
5. 本文尝试了自我对弈，但效果极差。

6.2. 未来展望

本文展望从以下几个方向对研究进行进一步完善：

1. 将深度增强学习与监督学习相结合，在学习的起步阶段让智能体学习人类棋局，加快学习效率。
2. 将深度增强学习与树搜索和剪枝相结合，实现类似于 TD(λ)的算法。
3. 在智能体学习到一定程度后进行自我对弈，进一步提升棋力。
4. 尝试不同的攻防价值模型或神经网络结构或输入结构。

参考文献

[1] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv, 2013: 1312.5602.

[2] 孙锴, 弈心——最强的五子棋引擎[EB/OL]. http://www.aiexp.info/pages/yixin-cn.html. 2017.

[3] Allis L V. Searching for solutions in games and artificial intelligence[M]. Ponsen & Looijen, 1994.

[4] 朱全民, 陈松乔. 五子棋算法的研究与思考[J]. 计算技术与自动化, 2006, 25(2):71-74.

[5] 郑培铭, 何丽. 基于计算机博弈的五子棋 AI 设计[J]. 电脑知识与技术, 2016, 33: 036.

[6] 宫瑞敏. 基于增强学习的计算机博弈策略的研究与实现[D]. 沈阳理工大学, 2011.

[7] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587): 484-489.

[8] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529-533.

[9] 毛坚桓, 殷璐嘉. 基于自适应探索改进的深度增强学习算法[J]. 微电子学与计算机, 2016, 33(6):139-142.

[10]David Silver, RL Course by David Silver[EB/OL]. https://www.youtube.com/playlist?list=PLzuuYNsE1EZAXYR4FJ75jcJseBmo4KQ9-. 2015.

[11]Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. Cambridge: MIT press, 1998.

[12]Richard E. Bellman. Dynamic Programming[M]. Princeton, NJ: Princeton University Press, 1957.

[13]Watkins C J C H, Dayan P. Technical Note: Q-Learning[J]. Machine Learning, 1992, 8(3):279-292.

[14]Wikipedia, Q-learning[EB/OL]. https://en.wikipedia.org/wiki/Q-learning. 2017.

[15]Wikipedia, Convolution[EB/OL]. https://en.wikipedia.org/wiki/Convolution. 2017.

[16]LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

[17]ujjwalkarn, An Intuitive Explanation of Convolutional Neural

Networks[EB/OL]. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. 2016.

[18]影风 LEY, [翻译]神经网络的直观解释[EB/OL]. http://www.hackcv.com/index.php/archives/104/. 2016.

[19]Adit Deshpande. A Beginner's Guide To Understanding Convolutional Neural Networks[EB/OL]. https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/. 2016.

[20]Lin L J. Reinforcement learning for robots using neural networks[D]. Fujitsu Laboratories Ltd, 1993.

[21]Wikipedia, 五子棋[EB/OL]. https://zh.wikipedia.org/wiki/五子棋. 2016.

[22]Wagner J, Virag I. Solving renju[C]. ICGA journal. 2001: 30-35

[23]Maddison C J, Huang A, Sutskever I, et al. Move evaluation in go using deep convolutional neural networks[J]. arXiv preprint arXiv, 2014: 1412.6564.

[24]skywind3000, gobang[EB/OL]. https://github.com/skywind3000/gobang. 2017.

[25]Sutskever I, Martens J, Dahl G, et al. On the importance of initialization and momentum in deep learning[C]. International Conference on Machine Learning. 2013: 1139-1147.