南 开 大 学

本 科 生 毕 业 论 文（设 计）

中文题目：不完全信息下强化学习的研究与应用

外文题目：Research and Application of Reinforcement Learning under Incomplete Information

学　　号：1510109

姓　　名：张万鹏

年　　级：2015 级

专　　业：信息与计算科学

系　　别：信息与数据科学系

学　　院：数学科学学院

指导教师：阮吉寿

完成日期：2019 年 4 月 26 日

关于南开大学本科生毕业论文（设计）的声明

本人郑重声明：所呈交的学位论文，是本人在指导教师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或没有公开发表的作品内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：

年　　月　　日

本人声明：该学位论文是本人指导学生完成的研究成果，已经审阅过论文的全部内容，并能够保证题目、关键词、摘要部分中英文内容的一致性和准确性。

学位论文指导教师签名：

年　　月　　日

摘　要

传统的强化学习由于理论上的各种假设与限制，只适用于简单的“完全信息博弈问题”，为了能够解决实际中更常见但也更复杂的“不完全信息博弈问题”，本文首先基于模拟采样的思路对传统的理论算法进行改进，解决了不完全信息下难以建模的根本问题。又基于统计学中的 Bootstrap 思想，将模拟采样与 Bootstrap 估计值相结合，改善数据的使用效率，提升模拟近似的准确性。最后再与梯度下降法和深度神经网络相结合，进一步提升算法的运算效率，提出了针对多人不完全信息博弈的“自适应 Deep Q-Learning 算法”。实验表明，“自适应 Deep Q-Learning 算法”能够通过强化学习有效探索出多人博弈技巧，还能高效快速处理较大数量级的实时数据流。

关键词： 强化学习；不完全信息博弈；模拟采样；Bootstrap 方法；自适应算法；神经网络

Abstract

Traditional reinforcement learning is only applicable to simple "complete information game problems" due to various theoretical assumptions and limitations. In order to solve the more common but more complex "incomplete information game problem" in practice, this paper is based on simulation first. The sampling idea improves the traditional theoretical algorithm and solves the fundamental problem which is difficult to model under incomplete information. Based on the Bootstrap idea in statistics, the analog sampling is combined with the Bootstrap estimation to improve the efficiency of data usage and improve the accuracy of the analog approximation. Finally, combined with the gradient descent method and the deep neural network, the computational efficiency of the algorithm is further improved, and an "Adaptive Deep Q-Learning Algorithm" for multi-person incomplete information game is proposed. Experiments show that the "Adaptive Deep Q-Learning Algorithm" can effectively explore multiplayer game skills through reinforcement learning, and also indicates that the algorithm can process large-scale real-time data streams efficiently and quickly.

Key Words: Reinforcement Learning; Incomplete Information Game; Analog Sampling; Bootstrap Method; Adaptive Algorithm; Neural Network

目 录

一、 绪论

（一） 课题背景

在传统的机器学习方法中，常见的主要有监督学习和无监督学习[1]，其中，监督学习是指给定输入 $\boldsymbol{x}$ 和输出 $\boldsymbol{y}$ 的训练集，通过学习输入 $\boldsymbol{x}$ 和输出 $\boldsymbol{y}$ 之间的对应关系的算法；而无监督学习则尝试从不带标签的训练集中推断结论，找到数据之间的隐藏结构[1][2][3]。

强化学习则是机器学习方法中有别于监督学习和无监督学习的另一类算法，在给定环境下模拟各种行为和动作，接收环境传递的激励和惩罚反馈，自行学习如何行动才能使长期收益最大化。强化学习方法与其他机器学习方法最大的区别，在于强化学习重点关注评价性反馈，而非指导性反馈。其中，评价性反馈是对样本的一个客观评分，而指导性反馈则是明确根据问题背景提供最优解信息[4]。

在传统的强化学习问题中，环境信息会明确给出，如围棋这种双人博弈游戏，由于棋盘盘面有限，且规则简单清晰，对手的当前状态和下一步状态之间的转移概率分布可以得到准确的表示[5]。通过得知这些信息，能够清晰建立准确的环境模型，进而做到通过具体的数学分析来求得最优解。

一场博弈中，如果玩家完美掌握了对手的策略、特征、回报函数等信息，称玩家掌握**完全信息**，并称这样的博弈为**完全信息博弈**。前面提到的双人棋类游戏就是完全信息博弈[6]。反之，这样的信息称为**不完全信息**，对应的博弈场景称为**不完全信息博弈**或者**动态博弈**[6]。

在不完全信息下，由于难以重建环境模型，传统的强化学习算法很难用于处理这类问题[7]。如在纸牌游戏中，若要建立环境模型，首先需要得到对手可

能的手牌概率分布，才能在此基础上建立状态转移概率分布，而不像完全信息下只需建立状态转移概率模型。

不完全信息博弈的这一特点导致其解空间非常大[8]，传统的强化学习算法难以克服这一问题，因此需要改进传统的强化学习算法，使得强化学习能够得到更广泛的应用。

（二） 论文研究目标和内容

本文将先介绍传统强化学习的核心思想，然后介绍传统的“基于 Bellman 迭代求解的强化学习算法”，但由于传统强化学习算法只适合处理完全信息博弈问题，因此提出了“基于 Monte Carlo 模拟的强化学习算法”，通过模拟采样来解决不完全信息问题下不能精确建立模型的问题。

进一步地，若只是简单地进行 Monte Carlo 模拟，又将会带来数据效率上的问题，因此基于统计学中的 Bootstrap 思想[9][10] 提出了“Q-Learning 算法”。

最后，为了提升运算上的效率，将 Q-Learning 算法与梯度下降法以及深度神经网络相结合[11]，然后针对多模型博弈问题背景做出相应优化，提出了“自适应 Deep Q-Learning 算法”。

为了验证本文所提出的“自适应 Deep Q-Learning 算法”能够有效地处理不完全信息下的强化学习问题，选取了一种民间纸牌游戏为仿真实验载体[12]，在其规则下进行仿真模拟实验，运用“自适应 Deep Q-Learning 算法”进行训练学习，来验证这一强化学习算法的学习效果。

二、 相关研究综述

（一） 强化学习基本框架

在强化学习中，将问题背景划分为“决策者”和“环境”两个部分，其中，“决策者”指算法模型本身，“环境”指决策者以外的信息集合。

环境会随时间发生改变，每个时间下都有特定的“状态”，所有可能出现的状态所组成的集合称为“状态空间”，记作 $\mathcal{S}$，而在时刻 t 对应的状态记作 $S_t \in \mathcal{S}$。在每个状态下，决策者可以根据环境状态的信息来采取“行动”，将某个状态 s 下所有可采取的行动组成的集合称为“状态 s 下的行动空间”，记作 $\mathcal{A}(s)$，特别地，将 t 时刻 s 状态下的特定行动记作 $A_t \in \mathcal{A}(s)$。

决策者采取行动 A_t 后，会对环境产生影响，环境则会因此发生改变，由本时刻的状态 S_t 进入下一时刻的状态 S_{t+1}，同时对决策者当前时刻的行为 A_t 产生一个评价性反馈并传递给决策者，将这样的评价性反馈称为“奖励”，记作 $R_t \in \mathcal{R} \subset \mathbb{R}$，其中 $\mathcal{R}$ 为“奖励空间”，表示所有可能的奖励值组成的集合。

决策者收到环境传来的奖励 R_t 后，将能得知环境对刚才的行动 A_t 的客观评价，从而根据该信息来调整自己的“决策策略”，并用于进行下一轮行为决策。而决策的调整，则是强化学习里的重要研究对象。简单而言，调整策略的核心思想是要最大化总收益，即将奖励值的累积求和最大化。

这里引入“回报值”的概念，它是总收益值的一个更规范的描述。定义 t 时刻之后的回报值为 G_t，其表达式为

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots \tag{2.1}$$

特别地，强化学习问题可被分为两种类型：片段型和连续型。片段型问题

指整体问题能够分解为存在起始状态和终止状态、步骤有限的片段，这些片段均有着相似结构，但不一定完全相同。比如纸牌游戏的一轮牌局就是一个片段，多轮牌局全体构成整体问题。相反，连续型问题则指那些不能分解为子片段的，持续连贯的问题。

在两种问题类型下，回报值 G_t 的公式需做一些调整才能合理。对于片段型问题，从 t 时刻开始，设其所处片段的终止状态对应的时刻为 T ，则应定义

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{2.2}$$

而对于连续型问题，若简单地将奖励值直接相加，显然会使 G_t 趋于无穷，这将导致无法比较不同行动 A_t 的效果。考虑将 G_t 定义为

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

其中 $\gamma(0 \le \gamma < 1)$ 为“削减率”，其直观效果是将观测信息的重点集中在较近时刻的奖励值上。

（二） Markov 决策过程

强化学习的目标是：学习出能由状态信息决定最佳行动的决策策略[4]。状态信息包含了实时信息和一定的历史信息，而显然历史信息不能太过庞大，否则会严重影响学习效率，此时考虑引入 Markov 性质，将问题的模型表示为 Markov 决策过程[13]，这样便能将“历史状态信息”通过状态的转移概率分布体现出来[14]。

1. Markov 性质

记全部历史信息的联合概率分布为

$$\Pr\{S_{t+1} = s', R_{t+1} = r | S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\} \tag{2.4}$$

记环境的状态转移概率分布为

$$p(s',r|,s,a)=\Pr\{S_{t+1}=s',R_{t+1}=r|S_t=s,A_t=a\} \tag{2.5}$$

定义 2.1 若有 $p(s',r|s,a)=\Pr\{S_{t+1}=s',R_{t+1}=r|S_0,A_0,R_1,\ldots,S_t,A_t\}$，则称这样的问题具有 ***Markov* 性质**。

通过引入 Markov 性质，决策者每次进行决策时只需考虑当前状态，因为历史信息也包含在了当前状态之中。

定义 2.2 称一个强化学习问题为 *Markov* 决策过程，当前仅当这个强化学习问题满足 *Markov* 性质[4][13]。

根据之前的定义，$p(s',r|s,a)=\Pr\{S_t=s',R_t=r|S_{t-1}=s,A_{t-1}=a\}$，且有 $\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r|s,a)=1$。

其中，$p(s',r|s,a)$ 表示决策者在处于状态 s 时做出行动 a 的条件下，进入下一个状态 s' 并收到奖励值反馈 r 的概率分布。

2. 状态价值函数与行为价值函数

强化学习的核心是通过对行动的评价来调整决策策略，需要将行动的评价以及决策策略都定义为函数，才能进行进一步的具体分析。

记概率空间为 $\mathcal{P}$，将策略函数定义为映射：$\pi:\mathcal{S}\times\mathcal{A}\to\mathcal{P}$。特别地，条件概率 $\pi(a|s)$ 表示决策者在状态 s 时选择行动 a 的概率分布。

确定了策略 π 后，可以定义状态值函数和行动值函数：

$$\begin{aligned}
v_\pi(s)&=\mathbb{E}_\pi[G_t|S_t=s]=\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\mid S_t=s\right],\forall s\in\mathcal{S}\\
q_\pi(s,a)&=\mathbb{E}_\pi[G_t|S_t=s,A_t=a]=\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\mid S_t=s,A_t=a\right]
\end{aligned} \tag{2.6}$$

其中，$v_\pi(s)$ 表示从状态 s 开始，决策者之后若完全遵守策略 π 来采取行

动，所能得到的期望回报值。而 $q_\pi(s,a)$ 表示在状态 s 下，已经采取了某个行动 a，之后若完全遵守策略 π 采取行动，所能得到的期望回报值。

（三） 基于 Bellman 最优方程的强化学习算法

1. Bellman 最优方程

定理 2.3 *在有限马尔可夫决策过程中，当前状态 s 与未来所有可能状态 s' 满足 Bellman 方程关系，即*

$$v_\pi(s)=\sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)\left[r+\gamma v_\pi(s')\right] \tag{2.7}$$

证明.

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t\,|\,S_t=s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\,|\,S_t=s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}+\gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}\,|\,S_t=s\right] \\
&= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)\left[r+\gamma\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}\,|\,S_{t+1}=s'\right]\right] \\
&= \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)\left[r+\gamma v_\pi(s')\right], \qquad \forall s\in\mathcal{S}
\end{aligned}$$

得证。 □

强化学习问题的目标是找到解决问题的最优策略，这一目标的前提是得到最优策略对应的价值函数，才能根据这个价值函数推出最优策略。

定义 2.4 *在策略空间 $\mathcal{P}$ 中存在且至少存在一个策略优于其他所有策略，称满足这一条件的策略为最优策略，其对应的价值函数 $v_*(s)$ 和 $q_*(s,a)$ 分别称为最优状态值函数、最优行动值函数。*

在该定义的基础上，可以得到 Bellman 最优方程[13]。

定理 2.5 *v_* 作为价值函数时，状态 s 与未来所有可能状态 s' 满足 Bellman 最优方程关系，即*

$$v_*(s) = \max_a \sum_{s',r} p(s',r \mid s,a)[r + \gamma v_*(s')] \tag{2.8}$$

证明. 在最优策略下，显然有

$$\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s',r \mid s,a)[r + \gamma v_*(s')]
\end{aligned}$$

得证。 □

Bellman 最优方程的行动价值函数形式为

$$q_*(s,a) = \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right] \tag{2.9}$$

对于 q_*，同理可证得类似定理2.5的结论，不再赘述。

在求得 v_*, q_* 的基础上，即可得到**最优策略**，最优策略的定义如下：

定义 2.6 *显然在行动空间 $\mathcal{A}(\mathcal{S})$ 中存在且至少存在一个行动，其行动价值函数值能取到 v_* 或 q_*，记该行动为 a_*，如果一个策略只将非零概率分配给这些行动，称这个策略是最优策略。*

至此，已经得到一个最基本的强化学习算法：求解 Bellman 最优方程得到最优价值函数 v_* 和 q_*，根据最优价值函数确定出最优策略。但由于 Bellman 方

程较为复杂，直接求解计算难度较大，所以在实际中并不实用，需要对其进行一定的改进。

2. 迭代法近似求解 Bellman 方程

为了提升求解 Bellman 方程的效率，考虑采用 Jocobi 迭代法[15]。Jacobi 迭代法的收敛条件为

定理 2.7 对于方程组 $\boldsymbol{x}_{k+1}=\boldsymbol{C}\boldsymbol{x}_k+\boldsymbol{b}$，给定方程组的一个初始近似解 $\boldsymbol{x}_0$，由前式迭代产生的序列 $\{\boldsymbol{x}_0,\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_k,\cdots\}$ 收敛的充分必要条件是矩阵 $\boldsymbol{C}$ 满足条件 $\lim\limits_{k\to\infty}\boldsymbol{C}^k=\boldsymbol{O}$。

定理 2.8 *Bellman* 方程可由 *Jacobi* 迭代法求得收敛解。

证明.

将 Bellman 方程的迭代形式为

$$v_{k+1}(s)=\sum_a \pi(a\mid s)\sum_{s',r} p(s',r\mid s,a)[r+\gamma v_k(s')] \tag{2.10}$$

设状态空间 $\mathcal{S}$ 中有 n 个不同的状态，则可将该 Bellman 迭代式具体展开为

$$\begin{aligned}
v_{k+1}(s_1) =& c_{11}v_k(s_1)+c_{12}v_k(s_2)+\cdots+c_{1n}v_k(s_n)+b_1\\
v_{k+1}(s_2) =& c_{21}v_k(s_1)+c_{22}v_k(s_2)+\cdots+c_{2n}v_k(s_n)+b_2\\
&\vdots\\
v_{k+1}(s_n) =& c_{n1}v_k(s_1)+c_{n2}v_k(s_2)+\cdots+c_{nn}v_k(s_n)+b_n
\end{aligned} \tag{2.11}$$

其中

$$\begin{aligned}
c_{ij} &= \sum_a \pi(a\mid s_i)\sum_r p(s_j,r\mid s_i,a)\gamma\\
&\le \sum_a \pi(a\mid s_i)\sum_{s_j,r} p(s_j,r\mid s_i,a)\gamma=\gamma
\end{aligned} \tag{2.12}$$

$$b_i = \sum_a \pi(a \mid s_i) \sum_r p(r \mid s_i, a) r \tag{2.13}$$

记

$$\boldsymbol{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{pmatrix} \tag{2.14}$$

则可简记 Bellman 迭代式的矩阵形式为

$$\boldsymbol{v}_{k+1} = \boldsymbol{C}\boldsymbol{v}_k + \boldsymbol{b} \tag{2.15}$$

对于 Bellman 迭代式 2.15 ，由于 $0 \leq \gamma < 1$ ，显然有

$$\lim_{k\to\infty} \boldsymbol{C}^k = \lim_{k\to\infty} \begin{pmatrix} \gamma & \gamma & \cdots & \gamma \\ \gamma & \gamma & \cdots & \gamma \\ \vdots & \vdots & \ddots & \vdots \\ \gamma & \gamma & \cdots & \gamma \end{pmatrix}^k = \boldsymbol{O} \tag{2.16}$$

因此由定理 2.7 可知，Bellman 迭代式 2.10 可由 Jacobi 迭代法快速求解，并且解收敛，定理得证。

□

根据定理2.8的结论，对于任意强化学习问题，都可以给出一个基于 Bellman 迭代求解的强化学习算法：

算法 1 基于 Bellman 迭代求解的强化学习算法

初始化向量 $V(s), \forall s \in \mathcal{S}$
repeat
    $\Delta \leftarrow 0$
    for 每个 $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$

$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
输出策略 π

三、 基于模拟采样的强化学习算法

基于 Markov 性质的前提，即在 Markov 决策过程中，之前给出的基于迭代求解 Bellman 方程的强化学习算法非常简洁清晰，但该算法对环境条件的依赖性较强，并且还需要显示地确定环境的动态分布 $p(s',r|s,a)$ ，这正是阻碍不完全信息博弈问题直接采用传统强化学习方法的最大难题。

在传统的“完全信息博弈”中，如围棋这类双人博弈问题，由于棋盘盘面有限，且规则简单清晰，对手的当前状态和下一步状态之间的转移概率分布可以得到准确的表示[5]，因此可以很方便地确定环境的动态分布 $p(s',r|s,a)$，但对于实际中更常见且更复杂的“不完全信息博弈”，如纸牌类游戏，若要建立环境模型，首先需要得到对手可能的手牌概率分布，才能在此基础上建立状态转移概率分布，然而对手的手牌概率分布是随着游戏进行而动态变化的，显然并不能准确得知 $p(s',r|s,a)$ ，传统的强化学习算法很难克服这一问题[4]，所以需要在其基础上做进一步改进，通过模拟采样的方法来获取和近似估计回报值，避免对环境进行模型重建。

（一） Monte Carlo 模拟

Monte Carlo 方法[1][16] 也称统计模拟方法，上一章的基于解 Bellman 方程的强化学习算法，由于依赖环境动态信息 $p(s',r|s,a)$ ，在不完全信息下无法得到此式，仅适合处理传统的完全信息问题，但借由 Monte Carlo 模拟方法，可以通过模拟和采样来近似地获得环境信息，进而也能使用强化学习来解决不完全信息问题。

在未知 $p(s',r|a,s)$ 的情况下，算法不能使用的根本原因在于无法计算 Bellman 方程 2.15 中的 $\sum_{s',r} p(s',r|s,a)[r+\gamma v(s')]$ ，即无法清晰地预知未来的状

态与反馈信息，关键原因在于每一步的奖励值 r 无法取得，而如果采用 Monte Carlo 方法进行模拟，可以进行大量模拟实验，通过计算样本回报值的均值，便能有效解决这一点。

记某个回合中通过模拟得到的反馈数据序列为 $\{R_0, R_1, R_2, \ldots, R_T\}$，在这一回合中，可计算出时刻 t 对应的回报值

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T \tag{3.1}$$

计算 Monte Carlo 模拟获得的回报值均值，可以作为价值函数的估计，即可得到基于 Monte Carlo 模拟的强化学习算法：

算法 2 基于 Monte Carlo 模拟的强化学习算法

将 π 初始化为一个随机策略
初始化向量 $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
将 $Returns(s,a)$ 初始化为一个空数组，$\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
loop
    随机选取初始状态 $S_0 \in \mathcal{S}$ 和初始行动 $A_0 \in \mathcal{A}(S_0)$
    服从现有策略 π 与环境交互生成一个回合片段
    得到 $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for 回合中的每一步 $t = T-1, T-2, \ldots, 0,$ **do**
        $G \leftarrow \gamma G + R_{t+1}$
        if S_t, A_t 在 $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$ 中出现过 **then**
            将 G 追加进数组 $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
            跳出本层循环
        end if
    end for
end loop

输出最终策略 π

（二） Bootstrap 自助学习

Monte Carlo 算法的价值函数更新式为

$$V(S_t) \leftarrow V(S_t) + \alpha\left[G_t - V(S_t)\right] \tag{3.2}$$

在 Monte Carlo 算法中，由于需要等待一个完整的片段结束，得到每个时间点的奖励值 R_t，才能返回到之前各时间点求得对应的回报值 $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$ 。这一缺点限制了算法的使用场景，其高延迟使得算法无法处理实时学习问题，而 Monte Carlo 本身需要等待一个完整回合结束的特点也导致无法处理连续型问题。

为了能够做到实时更新，即需要在每个时间点 t 得到当前反馈奖励值 R_t 时能够即使更新，而非累积足够多反馈奖励值才去更新，故考虑结合统计学中的 Bootstrap 思想[9][10]，利用“重抽样”的思想来尽可能利用信息价值，根据当前时刻 t 下的信息进行一定程度的“再利用”，使用估计值来替换 G_t 。

具体地，记 $\widehat{V}(S_t)$ 表示 $V(S_t)$ 的一个估计值，则有

$$\widehat{V}(S_{t+1}) \approx R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \tag{3.3}$$

从而可得到

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma\left[R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right] \end{aligned} \tag{3.4}$$

将3.3代入3.4可得

$$G_t \approx R_{t+1} + \gamma \widehat{V}(S_{t+1}) \tag{3.5}$$

将3.5式代入更新式3.2中，可得到改进后的更新式

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma \widehat{V}(S_{t+1}) - V(S_t) \right] \tag{3.6}$$

由于上述更新式使用前后两个时刻的 Bootstrap 估计值之差来作为更新量，故称使用上述更新式来学习价值函数的方法为**时序差分学习**。显然可见，时序差分学习可以在每一时间点 t 进行更新和学习，无需等待完整的片段结束得到 G_t 才去更新，其效率要比普通的 Monte Carlo 方法更高，且因此特性而适用于连续型的非片段问题。

1. Q-Learning 算法

将上述的时序差分算法更新式3.6中的状态价值函数 $V(s)$ 改为行动价值函数 $Q(s,a)$ ，写作

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \left[R_{t+1} + \gamma \widehat{Q}(S_{t+1},A_{t+1}) - Q(S_t,A_t) \right] \tag{3.7}$$

在强化学习的实际应用中，需要为行动价值函数 $\widehat{Q}(S_t,A_t)$ 指定一个具体的估计值。由于在更新过程中，后一次行动的决策更倾向于选择前一次价值 Q 更大的行动，因此最直观的考虑是选取 $\max_a Q_{old}(S_t,a)$ 作为 $Q_{new}(S_t,A_t)$ 的 Bootstrap 估计值，此时的算法更新式为

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t) \right] \tag{3.8}$$

由于该算法以行动价值函数 $Q(s,a)$ 为核心，基于这一评估指标进行各种决策学习，故称使用了该更新式的强化学习算法为 **Q-Learning 算法**。Q-Learning 算法的详细流程如下：

算法 3 Q-Learning 算法

将 π 初始化为一个随机策略
初始化向量 $Q(s,a),\forall s\in\mathcal{S},a\in\mathcal{A}(s)$
loop
    随机选取初始状态，记作 S
    repeat
        根据价值函数 Q 选取状态 S 下的最优行动 A
        采取行动 A，观测环境反馈 R 以及后续状态 S'
        $Q(S,A)\leftarrow Q(S,A)+\alpha\left[R+\gamma\max_a Q(S',a)-Q(S,A)\right]$
        $S\leftarrow S'$
    until S 为终止状态
end loop

根据学得的 Q 函数决定并输出最终策略 π

2. ε-贪心 Q-Learning 算法

前面的算法中，在得到价值函数之后，都是直接对其取 $\arg\max$ 来作为最优策略，这是完全的贪心学习[17][18]，初期如果过于贪心去信赖尚不完善的价值函数，容易导致策略收敛在局部最优值，所以需要将贪心的程度适当降低，于是提出下面的 ε-贪心学习算法

定义 3.1 在强化学习中，若决策者以 $1-\varepsilon(0<\varepsilon<1)$ 的概率采取贪心行动，以 ε 的概率随机选择一个行动 a，称这样的学习方法为 ε-贪心学习算法。

在强化学习中，当决策者采取 ε 的概率随机采取行动而非贪心选择当前价值最大的行动时，认为这是在**探索**，反之则是在**利用**。适当地进行探索，可以加速对环境未知信息的学习，同时也有利于及时跳出局部最优点，同时，只要 ε 不设太大，以 $1-\varepsilon$ 的概率来利用已学习的知识，在其所在时刻下相比纯贪心学习也几乎不会有损失，能够兼顾信息探索率和信息的充分利用率。

具体地，给出贪心学习下的策略函数定义

定义 3.2 设

$$\pi(a|s)=\begin{cases}\frac{\varepsilon}{|\mathcal{A}(s)|} & ,a\text{为非贪心行动}\\ 1-\varepsilon-\frac{\varepsilon}{|\mathcal{A}(s)|} & ,a\text{为贪心行动}\end{cases} \tag{3.9}$$

其中 $|\mathcal{A}(s)|$ 表示行动空间 $\mathcal{A}(s)$ 中的元素个数。称这样的策略 π 是 ε-贪心策略。

下面证明，将非贪心行动的概率逐渐降低、贪心行动的概率逐渐提高，能够提高期望价值函数。

证明. 记 π' 为贪心改进后的策略，其表达式为

$$\pi'(a|s)=\begin{cases}\frac{\varepsilon}{|\mathcal{A}(s)|} & ,a\text{为非贪心行动}\\ 1-\varepsilon-\frac{\varepsilon}{|\mathcal{A}(s)|} & ,a\text{为贪心行动}\end{cases} \tag{3.10}$$

而改进前的策略为

$$\pi(a_i|s)=\begin{cases}\frac{\varepsilon}{|\mathcal{A}(s)|}+\delta_i & ,a_i\neq a_*\\ 1-\frac{(|\mathcal{A}(s)|-1)\varepsilon}{|\mathcal{A}(s)|}-\sum_{a_i\neq a_*}\delta_i & ,a_i=a_*\end{cases} \tag{3.11}$$

其中 $\delta_i>0$，表示策略改进的差异量，则有

$$\begin{aligned}v_{\pi'}(s)&=\sum_a\pi'(a|s)q_\pi(s,a)\\&=\frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a)+(1-\varepsilon)\max_a q_\pi(s,a)\end{aligned} \tag{3.12}$$

记 $M=\max\limits_a q_\pi(s,a)$，首先证明不等式

$$\max_a q_\pi(s,a)\geq\sum_a\frac{\pi(a|s)-\frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon}q_\pi(s,a) \tag{3.13}$$

该不等式证明如下：

$$
\begin{aligned}
&\sum_a \frac{\pi(a|s)-\frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a) \\
&= \frac{\pi(a_*|s)-\frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} M + \sum_{a_i \neq a_*} \frac{\pi(a_i|s)-\frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a_i) \\
&= \frac{1-\frac{(|\mathcal{A}(s)|-1)\varepsilon}{|\mathcal{A}(s)|}-\sum_{a_i \neq a_*}\delta_i-\frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} M + \sum_{a_i \neq a_*} \frac{\delta_i}{1-\varepsilon} q_\pi(s,a_i) \\
&= \frac{1-\varepsilon}{1-\varepsilon} M - \frac{1}{1-\varepsilon}\sum_{a_i \neq a_*}\delta_i M + \frac{1}{1-\varepsilon}\sum_{a_i \neq a_*}\delta_i q_\pi(s,a_i) \\
&= M - \frac{\sum_{a_i \neq a_*}\delta_i}{1-\varepsilon}(M - q_\pi(s,a_i)) \\
&\leq M = \max_a q_\pi(s,a)
\end{aligned}
$$

不等式3.13得证，将其代回3.12中，则有

$$
\begin{aligned}
v_{\pi'}(s) &\geq \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + (1-\varepsilon)\sum_a \frac{\pi(a|s)-\frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a) \\
&= \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) - \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + \sum_a \pi(a|s) q_\pi(s,a) \\
&= v_\pi(s)
\end{aligned}
$$

证毕。 □

因此证得，将非贪心行动的概率逐渐降低、贪心行动的概率逐渐提高，能够提高期望价值函数，从而若将 ε-贪心策略与 Q-Learning 算法结合，由于 ε-贪心策略有着较好的跳出局部最优点的性质，且因 Q-Learning 算法下期望价值函数的逐渐提高能够确保算法的收敛性，便能做到既提升算法跳出局部最优点的鲁棒性，同时也促进算法的收敛性。两种方法相结合后的算法流程如下：

算法 4 基于 Q-Learning 的 ε-贪心强化学习算法

将 π 初始化为一个随机策略
初始化向量 $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
loop
    随机选取初始状态，记作 S
    repeat
        在 S 状态下，根据策略 π 选择最优行动 A
        采取行动 A，观测环境反馈 R 以及后续状态 S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha \left[R + \gamma \max_a Q(S',a) - Q(S,A) \right]$
        $A^* \leftarrow \arg\max_a Q(S_t,a)$
        for $\forall a \in \mathcal{A}(S)$ **do**
            if $a \neq A^*$ **then**
                $\pi(a|s) \leftarrow \frac{\varepsilon}{|\mathcal{A}(s)|}$
            else
                $\pi(a|s) \leftarrow 1 - \varepsilon - \frac{\varepsilon}{|\mathcal{A}(s)|}$
            end if
        end for
        $S \leftarrow S'$
    until S 为终止状态
end loop

输出最终策略 π

四、 基于 Q-Learning 的机器学习方法

在 Q-Learning 算法中，主要目的是通过更新式 3.8 来逼近最优行动值函数 $Q^*(s,a)$ 。具体地，在每一步中用新的估计值 $r+\gamma\max_a Q(s,a)$ 来逼近 $Q(s,a)$ ，注意到，Q 函数的参数 $s\in\mathcal{S},a\in\mathcal{A}$ 所在的参数空间都很大，对参数进行逐个更新的效率很低，使得该算法实用性并不高。为了提高算法效率，提升算法的实用性，应当将函数参数化后，使用机器学习方法进行更新和训练，结合深度神经网络[1][2][19] 来训练 Q 函数。

（一） 梯度下降 Q-Learning 算法

首先将价值函数 Q 参数化，设其参数为 $\boldsymbol{w}$ ，记参数化的价值函数为 $\widehat{Q}(s,a,\boldsymbol{w})$。为了使参数化的价值函数 $\widehat{Q}$ 能逼近真实价值函数 Q ，仍以 Q-Learning 算法的更新公式为核心，在其基础上使用梯度下降法[20][21][22]。

在 Q-Learning 梯度下降法中，以更新前后的行动价值函数均方误差为损失函数，定义为

$$J(\boldsymbol{w})=\frac{1}{2}\left[R_{t+1}+\gamma\max_a\widehat{Q}(S_{t+1},a,\boldsymbol{w})-\widehat{Q}(S_t,A_t,\boldsymbol{w})\right]^2 \tag{4.1}$$

根据梯度下降法，以优化降低损失函数 $J(\boldsymbol{w})$ 为目标进行更新，其更新式为

$$\begin{aligned}\boldsymbol{w}_{t+1}=&\boldsymbol{w}_t-\alpha\nabla J(\boldsymbol{w})\\ =&\boldsymbol{w}_t+\alpha\left[R_{t+1}-\gamma\max_a\widehat{Q}(S_{t+1},a,\boldsymbol{w}_t)-\widehat{Q}(S_t,A_t,\boldsymbol{w}_t)\right]\\ &\times\left[\gamma\nabla\max_a\widehat{Q}(S_{t+1},a,\boldsymbol{w}_t)-\nabla\widehat{Q}(S_t,A_t,\boldsymbol{w}_t)\right]\end{aligned} \tag{4.2}$$

其中，α 是梯度下降法中的更新速率，用来控制梯度更新的速度。最终更新停止后得到的 $\boldsymbol{w}^*$ 即可用以还原出参数化的价值函数 $\widehat{Q}^*(s,a,\boldsymbol{w}^*)$ ，进而可以

给出决策函数 π 。可将 ε-贪心学习与梯度下降 Q-Learning 算法相结合，详细的算法流程如下：

算法 5 ε-贪心梯度下降 Q-Learning 算法

将 π 初始化为一个随机策略
初始化参数函数 $\widehat{Q}: \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
设定超参数：步长 $\alpha > 0$，贪心率 $\varepsilon > 0$
将价值函数的权重 $\boldsymbol{w} \in \mathbb{R}^d$ 初始化
loop
    随机选取初始状态，记作 S
    repeat
        在 S 状态下，根据策略 π 选择最优行动 A
        采取行动 A ，观测环境反馈 R 以及后续状态 S'
        if S' 不是终止状态 **then**
            $\nabla \boldsymbol{J} \leftarrow \gamma \nabla \max_a \widehat{Q}(S', a, \boldsymbol{w}) - \nabla \widehat{Q}(S, A, \boldsymbol{w})$
            $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \left[R + \gamma \max_a \widehat{Q}(S', a, \boldsymbol{w}) - \widehat{Q}(S, A, \boldsymbol{w}) \right] \nabla \boldsymbol{J}$
        else
            $\nabla \boldsymbol{J} \leftarrow -\nabla \widehat{Q}(S, A, \boldsymbol{w})$
            $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \left[R - \widehat{Q}(S, A, \boldsymbol{w}) \right] \nabla \boldsymbol{J}$
        end if
        $A^* \leftarrow \arg\max_a Q(S_t, a)$
        for $\forall a \in \mathcal{A}(S)$ **do**
            if $a \neq A^*$ **then**
                $\pi(a|s) \leftarrow \frac{\varepsilon}{|\mathcal{A}(s)|}$
            else
                $\pi(a|s) \leftarrow 1 - \varepsilon - \frac{\varepsilon}{|\mathcal{A}(s)|}$
            end if
        end for
        $S \leftarrow S'$
    until S 为终止状态
end loop

输出最终策略 π

（二） 自适应 Deep Q-Learning 算法

上一节已将 Q-Learning 算法与机器学习中的梯度下降算法相结合，在此基础上，可以通过建立神经网络来实现算法的梯度更新部分[23]，便能进一步提高行动价值函数 $\widehat{Q}$ 的收敛速度。

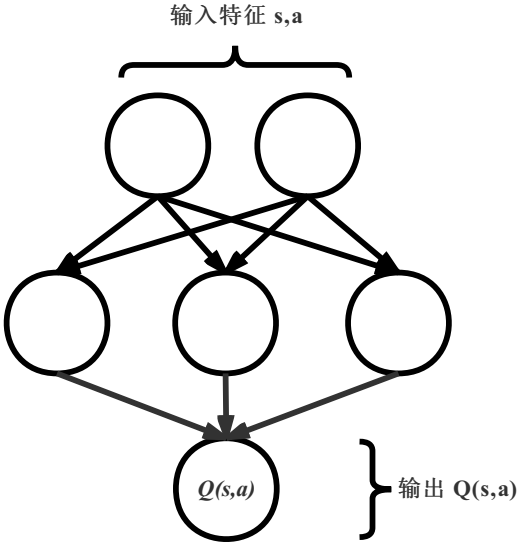


图 4.1: Deep Q-Network 示意图

如上图所示，将实时数据流中的状态和行动作为特征向量传入神经网络的输入层，基于参数化的 Q-Learning 公式

$$\begin{aligned}\boldsymbol{w}_{t+1} =& \boldsymbol{w}_t - \alpha \nabla J(\boldsymbol{w}) \\ =& \boldsymbol{w}_t + \alpha \left[R_{t+1} - \gamma \max_a \widehat{Q}(S_{t+1}, a, \boldsymbol{w}_t) - \widehat{Q}(S_t, A_t, \boldsymbol{w}_t) \right] \\ & \times \left[\gamma \nabla \max_a \widehat{Q}(S_{t+1}, a, \boldsymbol{w}_t) - \nabla \widehat{Q}(S_t, A_t, \boldsymbol{w}_t) \right]\end{aligned} \tag{4.3}$$

通过神经网络的反向梯度传递来更新权值参数，进而实现梯度下降算法[23]。称使用深度神经网络进行梯度下降的 Q-Learning 算法为 Deep Q-Learning 算法，将该神经网络结构称为 Deep Q-Network 。

在实际的应用场景中，不完全信息博弈常为多人对战类游戏，因此需要针对性地优化前面所描述的 Q-Learning 算法。

在 Q-Learning 中，基于 Bootstrap 思想，采用了 $\max_a Q(s,a)$ 作为 $Q(s,a)$ 的更新估计值，但在神经网络中大量使用 max 函数会带来一定的偏差值，带来决策误差。为了解决这一问题，需要使用两个神经网络交替使用来抵消这一偏差值。

定理 4.1 记两个独立的 *Deep Q-Network* 分别为 Q_1, Q_2，并假定他们能无偏估计真实值，即满足条件 $\mathbb{E}[Q_1(s,a)] = q(s,a)$，$\mathbb{E}[Q_2(s,a)] = q(s,a)$。若使用 Q_1 来参与行为决策，使用 Q_2 来为相应的行为进行评估，最终所得到的评价值则为真实值的无偏估计。

证明. 由题设，Q_1 用于行为决策，设 A^* 为 Q_1 决策出的最优策略，即有 $A^* = \arg\max_a Q_1(a)$，此时 A^* 的评价指标为

$$Q_2(S, \arg\max_a Q_1(a)) = Q_2(S, A^*) \tag{4.4}$$

根据题设条件

$$\mathbb{E}[Q_2(s,a)] = q(s,a) \tag{4.5}$$

以及 Q_1 与 Q_2 互相独立，可知

$$\mathbb{E}\left[Q_2(S, \arg\max_a Q_1(a))\right] = \mathbb{E}[Q_2(s,A^*)] = q(s,A^*) \tag{4.6}$$

故行动 A^* 的评价值 $\mathbb{E}[Q_2(s,A^*)]$ 是无偏估计值，证毕。 □

因此根据定理 4.1，可以通过构建两个神经网络交替使用来抵消偏差值，使评估指标为真实值的无偏估计，从而可以确保在该评估体系下能够训练和学习出恰当的博弈策略。

在不完全信息博弈的实际应用场景中，会有 N 名不同风格的选手对战，为了加强算法的自适应性和鲁棒性，需要进一步将 Deep Q-Network 模型分离为 $2N$ 个不同的神经网络，通过共同学习来消除因不同对手风格差异带来的影响，如下图所示。

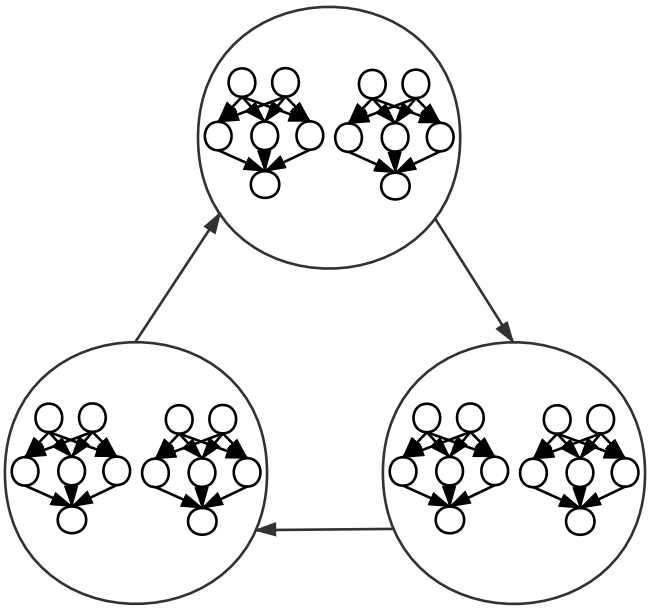


图 4.2: 自适应 Deep Q-Network (N=3)

如上图所示（图中 $N=3$），通过为每位选手分配一对 Deep Q-Network 之后，依次进行模拟博弈训练，基于训练数据执行 Deep Q-Learning 算法，通过前面的证明可知，这样的算法能够收敛得到 $Q(s,a)$ 函数的无偏估计值，便可基于估计函数 $\widehat{Q}(s,a)$ 推出最优策略 π 。由于算法能够自适应地根据玩家数量调整网络参数，且每个玩家学习模型下的双神经网络能够自适应地准确消除各自的偏差，故称这样的算法为**自适应 Deep Q-Learning 算法**。

五、 仿真实验与评估

本文选择一种简单的民间纸牌游戏，来验证前面所提出的“自适应 Deep Q-Learning 算法”是否能够较为适应地处理现实中的实际问题。

（一） 实验环境建立

1. 纸牌游戏的背景与规则介绍

在本游戏中共有三位玩家进行对战博弈，发牌前会从初始牌面中去掉“红桃2”、“梅花2”、“方块2”、“黑桃 A”这 4 张牌，共计 48 张牌，每人发牌 16 张。发牌后，第一回合由持手牌“黑桃3”的玩家出牌，随后依逆时针顺序跟牌，且要求在能出牌的情况下必须出牌，不可直接过牌。

作为第一位玩家出牌时，可任意选择牌型出牌，而跟牌阶段则只能按规则在指定牌型下出牌，最先将手牌出完的玩家为胜者。

该游戏规则简单，共有以下几种牌型：

表 5.1 牌型介绍

牌型简称	牌型描述	牌型简称	牌型描述
对子	两张相同的牌	三条	三张相同的牌
炸弹	四张相同牌	三带一	三条+单牌
三带二	三条+任意两张牌	单顺	至少5张的连续牌
双顺	至少2组连续对子	三顺	至少2组连续三条
飞机	至少2组连续三带二	四带二	炸弹+任意两张牌

该纸牌游戏规则下，由于每轮出牌后具有动态不确定性，因此该游戏属于不完全信息博弈。

2. 模拟建立的纸牌游戏环境

基于游戏的基本规则，使用 Python 编程语言构建了一个简单的游戏模拟环境。为了能够进行数值化的模型训练，需要将不同纸牌转化为对应的数值编号，具体地，将游戏规则下一副牌（该规则下共 48 张牌）从“黑桃3”到“方块2”由小到大对应为一个长度为 48 的一维向量，每张牌都有一个唯一的 ID ，例如 ['黑桃3', '黑桃5', '红桃5', '梅花5', '黑桃7'] 转化后应为 [0, 8, 9, 10, 16]。后面的实验介绍中，模型内部结构均基于此前提。

在 utils.py 文件中，定义了 divide_cards() 函数来实现发牌功能，能够随机将 48 张牌分为三份，发放给三位玩家，每位玩家各自分配到长度为 16 的一维向量，代表自己的手牌。

同时还定义了 calculate_score() 函数，它能够根据游戏规则来计分，起到向模型传递反馈奖励值的作用。

在 RunFastGame.py 中，定义了一个完整的 RunFastGameEnv 类，它能够完整地模拟卡牌游戏对局，其中 RunFastGameEnv.get_state() 函数能够获得当前所处的状态 s，RunFastGameEnv.play_cards() 能够接收策略传入的指令来采取相应的行动 a。

具体地，RunFastGameEnv 能够模拟卡牌游戏对局中的每位玩家，它能基于环境信息以及模型的策略决策情况来模拟完整的游戏对局，从而产生了真实的经验数据，提供给模型进行 Monte Carlo 模拟，从而得以实现之前提出的算法。

（二） 算法仿真与评估

1. Deep Q-Network 算法实现细节

在 DQN.py 中，定义一个 QNet 类，建立一个全连接神经网络，其深度为 4 层。输入层有 209 个神经元，对应输入特征向量的 209 个元素；隐藏层 1 有 512 个神经元；隐藏层 2 有 256 个神经元；隐藏层 3 有 128 个神经元；隐藏层 4 有 64 个神经元；输出层只有一个神经元，其输出值为 Q 函数所对应的值。

在输入层和三个隐藏层之间都加入了 ReLU 线性整流单元[24] 作为激励函数，其表达式为 $\mathrm{ReLU}(x) = \max(0, x)$。

同时也在这些连接层中加入了 DropOut 机制，它可以通过随机丢弃一些中间结果来有效防止过拟合[25]。

完整的神经网络结构如下图所示：

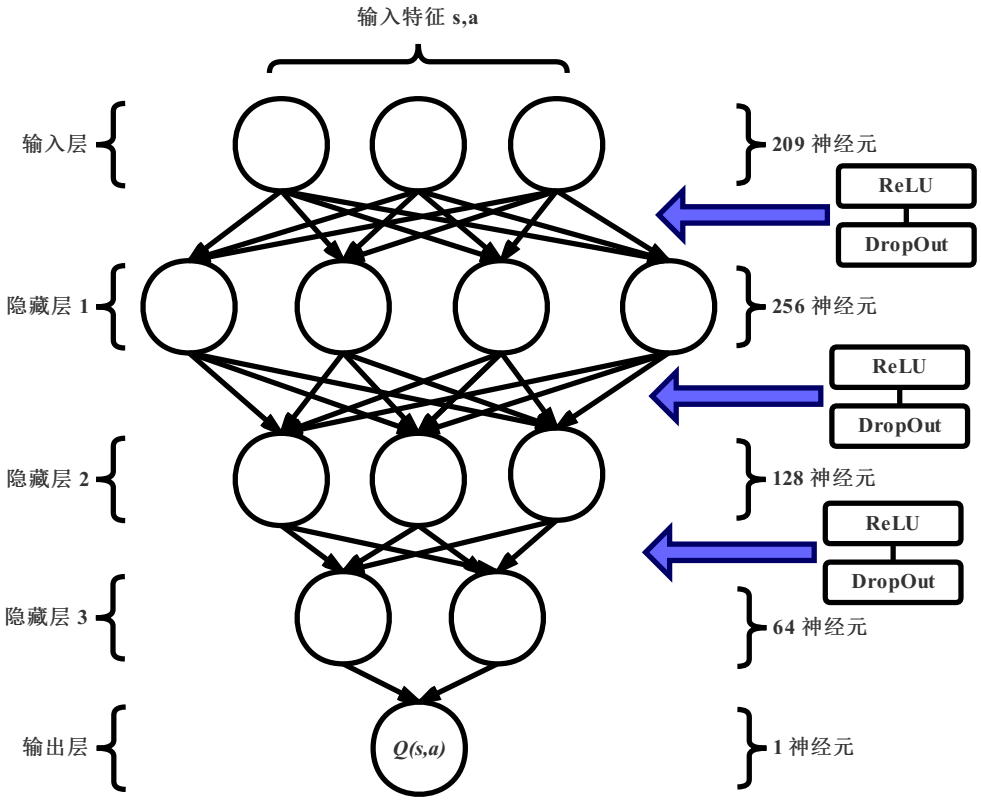


图 5.1: Deep Q-Network 实现细节

在 `DQN.py` 中还定义了一个 `DQN` 类，用以实现 Deep Q-Learning 的细节。其中，`DQN.choose_action()` 函数代表策略函数 π，输入的状态 s 后，根据 Q 函数大小决定的策略概率分布 $\pi(a|s)$ 来决定下一步应采取的行动 a，其中

$$\pi(a|s) = \begin{cases} \frac{\varepsilon}{|\mathcal{A}(s)|} & , a \neq \arg\max_a Q(s,a) \\ 1 - \varepsilon - \frac{\varepsilon}{|\mathcal{A}(s)|} & , a = \arg\max_a Q(s,a) \end{cases} \tag{5.1}$$

`DQN.store_transition()` 函数和 `DQN.add_reward()` 函数用于将模拟过程中的一些关键信息存储于缓存中.

`DQN.learn()` 函数的作用是将缓存的信息提取出来，通过神经网络进行 Q-Learning 更新，更新公式为

$$\widehat{Q}(S_t,A_t,\boldsymbol{w}) \leftarrow \widehat{Q}(S_t,A_t,\boldsymbol{w}) + \alpha\left[R_{t+1} + \gamma \max_a \widehat{Q}(S_{t+1},a,\boldsymbol{w}) - \widehat{Q}(S_t,A_t,\boldsymbol{w})\right] \tag{5.2}$$

最后，将 Deep Q-Network 模型接入进前面构建好的 `RunFastGameEnv` 环境下，即可开始模拟和训练的过程。

2. 仿真实验细节

在前面所建立的仿真实验环境 `RunFastGameEnv` 中，由于是三人不完全信息博弈，即有 $N=3$，故总共需要建立六个 Deep Q-Network，每位玩家分配两个神经网络。

由于自适应 Deep Q-Learning 算法能够抵消估计偏差值，因此可以使用风格不同的三位玩家模型进行博弈训练，而风格差异化能够增加训练模型的鲁棒性，增强模型处理不确定性状态的能力。记三名玩家为分别为 A,B,C ，其中 A 玩家对应的模型为主训练模型，玩家 B,C 为辅助学习模型，通过调整贪心率 ε 使得辅助学习模型的决策水平与主训练模型形成差异化。其中 $\varepsilon_B = \varepsilon_A/2$ ，$\varepsilon_C = \varepsilon_A/3$。

此外，为了加速强化学习的学习效率，将算法中的贪心学习率 ε 进行了动态调整，从初始值 $\varepsilon_0 = 0.4$ 随训练次数线性下降，每个 epoch 下降 0.01，达到预设的下限阈值 $\varepsilon_{stop} = 0.08$ 后停止动态调整。

基于前面所描述的细节，本实验的具体流程如下：

算法 6 训练过程

初始化三名水平不同的玩家 A,B,C
repeat
    开始新一局游戏，并为三名玩家发牌
    repeat
        根据游戏规则决定当前回合应当出牌的玩家
        进入当前玩家的视角，根据场面信息获取当前所处状态 S
        将状态 S 传入 Q 函数和 π 策略，决策下一步所要采取的行动 a
        实际采取行动 a，接收环境传递的反馈值，并存入缓存
    until 执行行动 a 后本局游戏分出胜负
    根据游戏规则，统计该局游戏各玩家得分，并追加存储至缓存
    将缓存的信息，作为输入参数传入神经网络
    经由神经网络完成一次训练学习，得到新的 Q 函数和 π 策略，用于下一次决策
until 达到预设的训练次数
结束训练，输出训练所得到的模型和参数

实验的完整流程图如下所示：

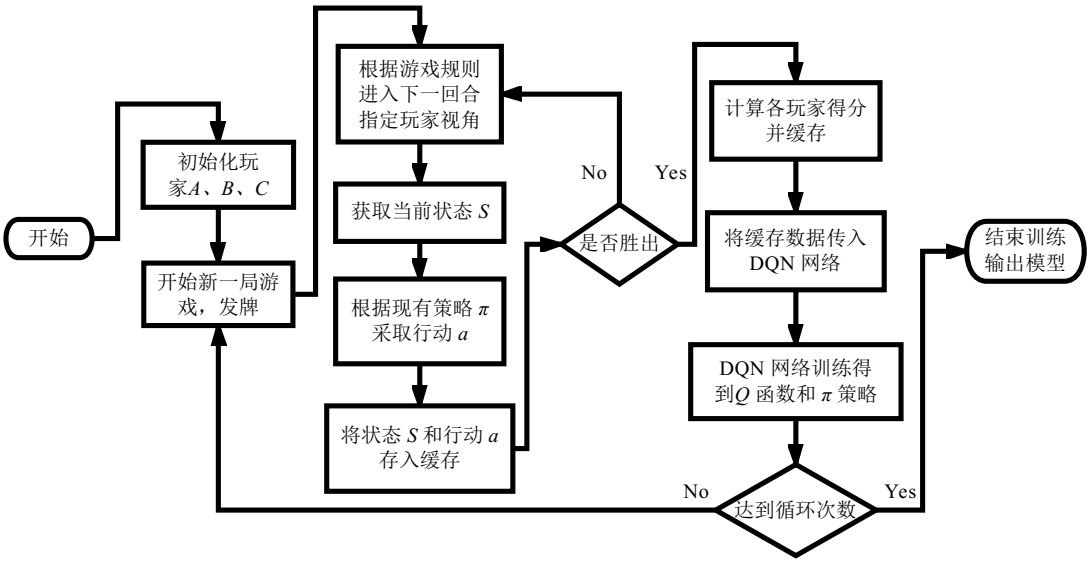
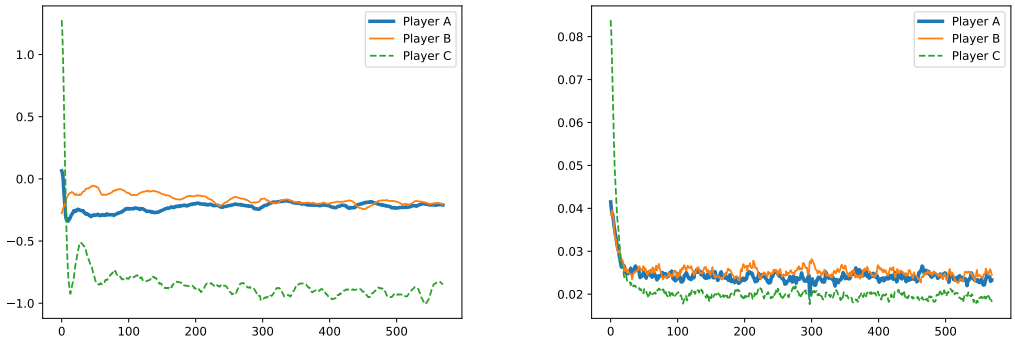


图 5.2: 实验流程图

（三） 仿真实验结果与分析

在 NVIDIA® GTX 1080 GPU 的硬件条件下，经过 100 万余局自我对战学习（其中每 2000 局作为一个 epoch 集中学习），最终决策函数 Q 得到如下图所示的收敛情况。



(a) Q 函数训练均值　　(b) Q-Learning 训练误差值

图 5.3: 仿真实验的训练结果

图5.3a刻画的是 Q 函数随训练次数的均值，观察可知经过该算法训练可使行动价值函数 $Q(s,a)$ 收敛，进而可以得到 ε-贪心策略：

$$\pi(a|s) = \begin{cases} \frac{\varepsilon}{|\mathcal{A}(s)|} & ,a \neq \arg\max_a Q(s,a) \\ 1-\varepsilon-\frac{\varepsilon}{|\mathcal{A}(s)|} & ,a = \arg\max_a Q(s,a) \end{cases} \tag{5.3}$$

其中策略 $\pi(a|s)$ 是一个条件概率分布，其含义是模型处于状态 s 时，决定采取行动 a 的概率。

另外，也可根据训练中的损失函数观察其收敛效果，图5.3b所示的即为自适应 Deep Q-Learning 算法的梯度损失项：

$$J(\boldsymbol{w}) = \frac{1}{2}\left[R_{t+1} + \gamma \max_a \widehat{Q}(S_{t+1}, a, \boldsymbol{w}) - \widehat{Q}(S_t, A_t, \boldsymbol{w})\right]^2 \tag{5.4}$$

从图中可观察知，损失项最后下降至一个较小值，Q 函数逐渐收敛。

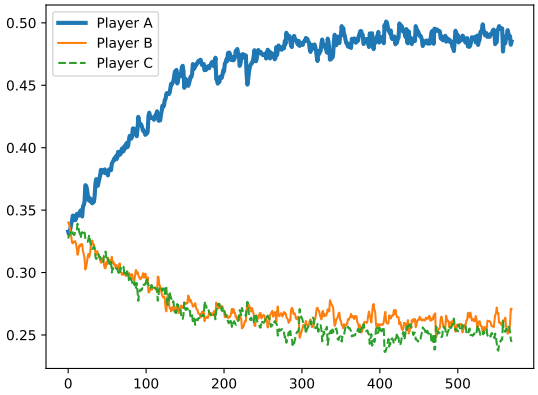


图 5.4: Q 函数与实时训练胜率

图5.4展现了三个玩家模型在训练中的实时胜率，观察发现其在训练过程中，自适应 Deep Q-Learning 算法的主训练模型胜率不断上升，远优于另外两个辅助学习模型，显然自适应 Deep Q-Learning 算法通过强化学习的思想学习和掌握了一定程度的游戏技巧。

为了验证自适应 Deep Q-Learning 的实际学习效果，使用训练出的策略模型进行了独立测试，通过调整算法中的贪心率 ε ，从而影响到最终的实际决策策略 π ，如5.3式所示。独立测试的结果如下：

表 5.2: 独立测试结果

主模型 A		辅助模型 B		辅助模型 C	
$1-\varepsilon_A$	胜率	$1-\varepsilon_B$	胜率	$1-\varepsilon_C$	胜率
0.99	0.662	0.99	0.165	0.99	0.173
0.99	0.685	0.90	0.158	0.90	0.157
0.95	0.621	0.80	0.191	0.80	0.188
0.95	0.646	0.70	0.173	0.70	0.181
0.90	0.558	0.60	0.221	0.60	0.221
0.90	0.583	0.50	0.207	0.50	0.210

基于独立测试结果，发现主训练模型在各种贪心率下相比于辅助训练模型均有着明显的策略优势，因此可以得出结论：自适应 Deep Q-Learning 算法能够有效处理不完全信息博弈问题。

六、 总结与展望

论文提出了能够解决多人博弈问题的“自适应 Deep Q-Learning 算法”。该方法将传统的理论强化学习方法与新兴的深度学习技术相结合，其创新点在于使用了双神经网络来抵消偏差，以及分离神经网络进行交互博弈，来进行自适应学习。

实验结果表明，该算法能够较好地解决不完全信息博弈问题，收敛速度快同时收敛效果好，且该算法短时间内能进行大量级的数据训练，算法效率高。但同时也还有着较大提升空间，有待进一步地加深研究。

随着机器学习理论的不断发展，其在实际中的应用也渐渐受到重视。传统的针对数据集内部关系问题的监督学习和无监督学习已经得到了广泛的研究，而新兴的实时反馈数据型博弈问题则渐渐成为一个关注的重点，这使得强化学习在机器学习中变得越发重要。

目前，对于简单的完全信息博弈问题，已经提出了许多有着不错成果的强化学习理论，如结合了 Monte Carlo 搜索树的 AlphaGo 算法[5]，但对于更实际且更复杂的不完全信息博弈问题，还尚未得到充分的研究，因此，针对不完全信息博弈问题的强化学习将会是未来的一个重要研究方向，有待进一步地充分研究。

参考文献

[1] GOODFELLOW I, BENGIO Y, COURVILLE A. Deep learning[M]. [S.l.]: MIT press, 2016.

[2] BISHOP C M. Pattern recognition and machine learning[M]. [S.l.]: springer, 2006.

[3] 李航. 统计学习方法[M]. 北京: 清华大学出版社, 2012.

[4] SUTTON R S, BARTO A G. Reinforcement learning: An introduction[M]. [S.l.]: MIT press, 2018.

[5] SILVER D, SCHRITTWIESER J, SIMONYAN K, et al. Mastering the game of go without human knowledge[J]. Nature, 2017, 550(7676): 354.

[6] MARINATTO L, WEBER T. A quantum approach to static games of complete information[J]. Physics Letters A, 2000, 272(5-6): 291-303.

[7] MACDERMED L, ISBELL C, WEISS L. Markov games of incomplete information for multi-agent reinforcement learning[C]// Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence. [S.l.]: [s.n.], 2011.

[8] SANDHOLM T. The state of solving large incomplete-information games, and application to poker[J]. AI Magazine, 2010, 31(4): 13-32.

[9] EFRON B, TIBSHIRANI R J. An introduction to the bootstrap[M]. [S.l.]: CRC press, 1994.

[10] 王兆军, 邹长亮. 数理统计教程[M]. 北京: 高等教育出版社, 2014.

[11] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing atari with deep reinforcement learning[J]. ArXiv preprint arXiv:1312.5602, 2013.

[12] DAHL F A. A reinforcement learning algorithm applied to simplified two-player Texas Hold’em poker[C]// European Conference on Machine Learning. [S.l.]: [s.n.], 2001: 85-96.

[13] HOWARD R A. Dynamic programming and markov processes.[J]., 1960.

[14] ROSS E A Sheldon M. Stochastic processes[M]. Vol. 2. [S.l.]: Wiley New York, 1996.

[15] 林成森. 数值分析[M]. 北京: 科学出版社, 2007.

[16] ROBERT C, CASELLA G. Monte Carlo statistical methods[M]. [S.l.]: Springer Science & Business Media, 2013.

[17] EDMONDS J. Matroids and the greedy algorithm[J]. Mathematical programming, 1971, 1(1): 127-136.

[18] CORMEN T H, LEISERSON C E, RIVEST R L, et al. Introduction to algorithms[M]. [S.l.]: MIT press, 2009.

[19] MURPHY K P. Machine learning: a probabilistic perspective[M]. [S.l.]: MIT press, 2012.

[20] TSITSIKLIS J N, VAN ROY B. Analysis of temporal-diffference learning with function approximation[C]// Advances in neural information processing systems. [S.l.]: [s.n.], 1997: 1075-1081.

[21] BOTTOU L. Large-scale machine learning with stochastic gradient descent[G]// Proceedings of COMPSTAT'2010. [S.l.]: Springer, 2010: 177-186.

[22] SUTTON E A Richard S. Fast gradient-descent methods for temporal-difference learning with linear function approximation[C]// Proceedings of the 26th Annual International Conference on Machine Learning. [S.l.]: [s.n.], 2009: 993-1000.

[23] HECHT-NIELSEN R. Theory of the backpropagation neural network[G]// Neural networks for perception. [S.l.]: Elsevier, 1992: 65-93.

[24] LI Y, YUAN Y. Convergence analysis of two-layer neural networks with relu activation[C]// Advances in Neural Information Processing Systems. [S.l.]: [s.n.], 2017: 597-607.

[25] SRIVASTAVA N, HINTON G, KRIZHEVSKY A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. The Journal of Machine Learning Research, 2014, 15(1): 1929-1958.

致　谢

首先，我要感谢阮吉寿教授在我本科学习期间对我的悉心关照和指导，让我学到了良多的理论知识，收获了丰富的科研经历。也要感谢肖喜教授在毕业设计的课题研究和论文撰写上的各种帮助与指导。同时也感谢父母、家人在背后一直默默地支持与鼓励，感谢朋友们在身边的陪伴与帮助。

四年的大学生活已经接近尾声，在这宝贵的四年时光中，我收获了知识，锻炼了能力，建立了友谊，热爱上了生活，这将是我人生中最宝贵、最浓重的一笔记录。回首这四年，有过挫折与磨难，也有欣喜与快乐，身边的每一个人都在不断地帮助和鼓励我前进，感谢你们，感谢这段时间所有的磨练和收获，让我更加成熟、坚强与自信，谢谢!