深 圳 大 学

本 科 毕 业 论 文（设计）

题目: 基于双层网络结构 DQN 的游戏 AI 实现与改进

姓名: 陈晔

专业: 网络工程

学院: 计算机与软件学院

学号: 2015080129

指导教师: 张昊迪

职称: 教授

2019 年 4 月 5 日

深圳大学本科毕业论文（设计）诚信声明

本人郑重声明：所呈交的毕业论文（设计），题目《基于双层网络结构 DQN 的游戏 AI 实现与改进》是本人在指导教师的指导下，独立进行研究工作所取得的成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式注明。除此之外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。本人完全意识到本声明的法律结果。

毕业论文（设计）作者签名：

日期：　　年　　月　　日

目　录

基于双层网络结构 DQN 的游戏 AI 实现与改进

计算机与软件学院网络工程专业　陈晔

学号：2015080129

【摘要】作为人工智能研究的重要应用领域，游戏 AI 为各类算法与模型研究，提供了良好的应用与测试平台。与现实世界相比，游戏环境相对比较简单，条件可控，又一定程度上满足现实仿真要求。所以近年来以 DeepMind、OpenAI 等为代表的众多科研团队，在游戏 AI 领域进行了许多开创性的探索与研究。而当前主要方法为数据驱动的机器学习相关方法，如深度强化学习等。少量方法基于专家规则描述，在某些场景中也取得不俗表现。本文结合深度强化学习与规则描述等多种方法，以基于 python 的开源游戏《Flappy Bird》为测试平台，研究深度神经网络结构、超参数以及规则判断引入对学习效率的影响。

【关键词】机器学习；数据驱动；深度强化学习；规则描述

1 引言

1.1 研究背景及意义

深度 Q 网络，简称 DQN，是一种深度强化学习理论模型[2]，其所构建的能够“学习”的游戏 AI，是从实践经验中总结出来的产物。学习这一概念来自于人类，人类因学习而拥有智慧，也因学习能够不断产生新的智慧和经验，用于解决过去以及将来出现的难题。最早的游戏 AI，可以追溯到经典游戏《吃豆人》。在《吃豆人》中，对于四种不同颜色的怪物都设计了不同的追击算法。早期的游戏环境与互动动作相对简单有限，因此可以使用预设的资料库来构建游戏 AI。到了近代，出现了以 MOBA 游戏（以动作自由度高与实时状态变化快而著称的游戏类型）为基础构建的游戏 AI。同时在 2017 年，出现了全球瞩目的柯洁与围棋 AI ALPHA GO 对垒的事件。围棋是一项具有庞大可能性的游戏。通过预设知识库构建出以上两种表现良好的游戏 AI 几乎成为不可能的任务。

通过预设的知识资料库来构建 AI，容易随着环境事件的复杂性上升而遇到性能瓶颈。而强化学习强调的是代理模型在环境中的表现，寻求在探索环境和利用当前的知识之间找到平衡[3]。通过强化学习构建 AI，能不断从环境中“学习新的知识”，即通过不断的试错，调整行为策略提升模型表现，最后取得最优行为策略。DQN 理论结合了深度卷积神经网络结构，构建了一种能不断“学习的游戏”AI，并且大幅提升了模型的训练速度与环境表现。

因此，在本次毕业设计项目中，我们通过学习 Q-learning 原理，复现基于游戏 Flappy Bird 的 DQN 模型，并在此基础上采取更进一步的神经网络结构优化（双层网络结构的实现）、算法优化（Double DQN）与加入针对本次毕业设计项目选取的游戏环境特定的干预规则，以寻求训练效率最大化，对我们的强化学习的研究之路具有重要的实用经验和研究意义。

1.2 基本流程与模型构成

强化学习被认为是三种机器学习范式之一，其余两种分别是监督学习和无监督学习。监督学习的目的是减少实际输入与预期输出之间的误差，其中存在一个“导师”，可以对给定的输入提供应有的输出。而无监督学习则是在不存在导师的情况下构建内部表征。强化学习与监督学习的不同之处在于其不需要通过给定的输入获取对应的输出对。

强化学习的流程组成结构有：

Agent。代表强化学习算法模型，本文简称为代理模型。

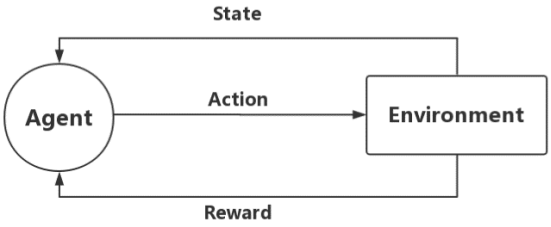
Environment。代表游戏环境。

State。环境当前返回的情况，代表当前环境状态。

Reward。环境的即时返回值，本文简称奖励值。

Action。代理模型在每个状态所有可能采取的动作。

整个流程为：代理（Agent）处于一个环境（Environment）中。代理在环境中执行某些动作（Action）。这些动作有时候会导致奖励值（Reward）增加，并且环境会根据选择的动作返回下一个状态（State）。流程如图：



作图：陈晔

图 1 强化学习流程图

以上的流程在强化学习的过程中不断循环进行。强化学习的目的就是寻找一个最优的策略，在环境中获取尽可能高的奖励值。

除了流程结构以外，强化学习系统有四个组成要素：

Model。代表环境的模型。

Policy。代理模型根据当前状态决定下一个动作的策略，即状态到动作的映射，本文简称策略。

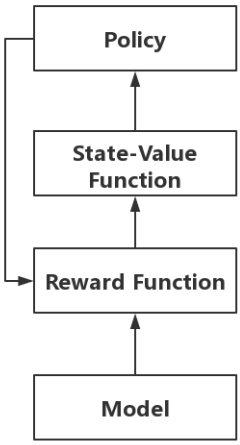
Reward Function。瞬时奖惩函数，获取代理模型与环境交互中产生的奖励信号。奖励信号是代理模型调整行为策略的基础。函数返回值通常是一个标量，正数表示正面效果，负数表示负面效果。因此该信号也可以用于代理模型对动作的评估。

State-Value Function。状态值函数。强化学习的目标可以总结为学习最优策略来最大化奖励。但是这里并不是统计瞬时奖励，否则就变成简单的贪心策略了。我们需要刻画包括未来的奖励，如公式：

$$V_\pi(s) = E_\pi[R] = E[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]$$

$V_\pi(s)$ 被定义为状态 s（比如 $s_0 = s$）作为输入的返回值，并遵循策略 π 。因此简单来说，价值函数可以评估输入的状态“有多好”。R 定义为未来折扣奖励的总和，r_t 代表第 t 步的奖励。$\gamma \in [0,1]$ ，代表折扣因子。

元素组成图如下：



作图：陈晔

图 2 强化学习系统

1.3 马尔科夫决策过程（MDP）

了解马尔科夫决策过程，首先要了解马尔科夫性。马尔科夫性的数学描述为：

$$\mathrm{P}(s_{t+1}|s_t, s_{t-1} \dots s_1) = \mathrm{P}(s_{t+1}|s_t)$$

即在一个随机过程中，下一个时刻的状态只与当前状态有关，与之前的状态无关。

一个马尔科夫随机过程指的是过程中的任意两个状态之间都满足马尔科夫性。一个马尔科夫随机过程用一个二元组 (S, P) 表示，S 是随机过程的有限状态集合，P 是状态的概率转移矩阵。

而马尔科夫决策过程是以马尔科夫随机过程为基础的。马尔科夫决策过程用一个四元组定义 $(\mathrm{S}, \mathrm{A}, P_a, R_a)$ 。

S 是一组有限的状态。

A 是动作的集合（比如，A_s 是状态 S 提供的可用动作的集合）。

$P_a(s, s') = P_r(s_{t+1} = s'|s_t = s, a)$ 是状态 s 在时刻 t 采取动作 a 导致当前状态向下个时刻 $\mathrm{t}+1$ 的状态 s' 转换的概率。

$R_a(s, s')$ 是采取动作 a 后状态 s 转换到状态 s' 收到的即时奖励（又称期望的即时奖励）。

马尔科夫决策过程里的一次回合（比如一次游戏）形成一系列有限的状态，动作和奖励：

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, \dots, s_{n-1}, a_{n-1}, r_n, s_n$$

这里 s_i 代表状态，a_i 代表动作，r_{i+1} 代表实施动作后的奖励。这一集在在状态 s_n 时结束

（比如，s_n 代表游戏结束时的状态）。

简而言之，状态和动作的集合从一个状态转换到另一个状态的规则即构成了马尔科夫决策的过程。典型的马尔科夫决策过程如下图：

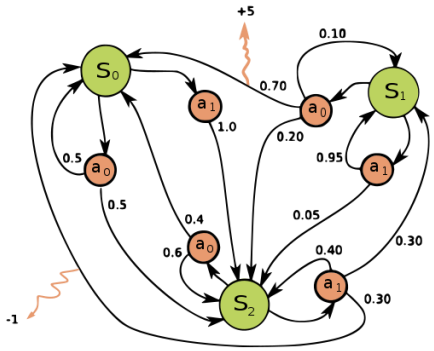


图 3 马尔科夫决策过程

图中有三个状态S_0, S_1, S_2（绿色）、两种动作a_0, a_1（橙色）、两种奖励$+5, -1$（橙色箭头）

1.4 基于 MDP 框架的算法模型分类

马尔科夫决策过程是大多数强化学习算法的理论基础。对于强化学习而言，获得最优策略的方法是迭代更新值函数的估计值。若当前状态为 s，当前选择的动作为 a_t，状态为 s_{t+1}，奖励为 r_{t+1}，则 s 向 s_{t+1} 转移的概率 P 与奖励 r_{t+1} 只取决于 s 和 a_t，与之前的状态和动作无关。

基于模型法（model-base）[6]，是指代理模型在训练过程中需要学习马尔科夫决策过程的知识。基于模型的优化策略过程，对于策略 π 和值函数 V(s)，第一步是利用策略估算函数值 V；然后通过函数值 V 更新策略 π。如此反复，最后得到最优策略和最优价值函数。

无模型（model-free）[6]，是相对于模型算法而言的说法。它不使用与马尔科夫决策过程相关的转移概率分布和奖励函数的算法。转移概率分布和奖励函数通常被称为环境的模型。无模型的优化策略过程，对于策略 π 和初始的 Q 值 Q(s,a)，第一步是利用策略 π 估算 Q 值；然后是通过 Q 值更新策略 π。如此反复，最后得到最优策略和最优价值函数。

当没有模型的时候，当前状态所有可能的后续状态不可知，无法确定当前状态下的最优动作。所以我们使用 Q 值 Q(s,a) 来代替 V(s)。这样当前状态的后续状态就不再是先决条件。对于使用 Q 值的无模型强化学习，只基于 Q 值选择当前的最优动作可能会陷入局部最优，所以需要偶尔不遵循 Q 值选取动作，也就是需要一定的探索。

1.5 本课题主要工作

本文主要的工作是通过学习 Google Deep mind 团队关于 DQN 的研究论文与成果[2][4][5]，实现 DQN 在游戏 Flappy Bird 上的训练模型，并可视化模型训练过程中的效果评估数据指标，加以观察分析。然后加入基于先验知识的启发式规则对训练过程进行干预，观察对训练的影响，期望得出正面的提升效果。

2 强化学习模型算法原理

2.1 Q-learning

2.1.1 原理介绍

在 1989 年，Watkins 第一次在论文中介绍了 Q-learning 算法[1]。Q-learning 是一种无模型的强化学习算法。

Q-learning 的目标是学习一种行为策略，能告知代理模型在不同的情况下采取不同的行动。对于任何有限的马尔科夫过程，它从当前的状态开始，在所有后续的步骤中找到最大化预计总奖励的最优行为策略[7]。理论上，在给定无限探索时间和部分随机策略的情况下，Q-learning 能为任意的有限马尔科夫决策过程找到最优行为策略。Q 值函数返回强化学习提供的奖励值，并且可以作为评估当前状态采取的动作的标准[8]。

Q-learning 的算法为：

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from $Q(e.g.,\varepsilon-greedy)$
    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

即：

（1） 采用随机值初始化 Q 表

（2） 进入循环（对每一回合）

（3） 初始化状态 S

（4） 进入循环（对于这一回合的每一步）

（5） 从当前状态 S 选取动作 A 并使用 Q 值的策略（比如 $\varepsilon-greedy$）

（6） 实施动作 A ，观察奖励 R 与下一个状态 S'

（7） 使用贝尔曼等式更新 Q 表

（8） 把当前状态 S 更新为下一个状态 S'

（9） 直到抵达终止状态，开始一个回合

Q 表是一个记录每个状态对应不同动作下的 Q 值表格，训练过程中通过不断更新 Q 表来获取最优策略。一个简单的 Q 表例子如下表：

表 1 简单的 Q 表实例

	A_1	A_2
S_1	1	−2
S_2	−3	4

S_1、S_2代表状态，A_1、A_2代表当前状态可选择的动作。$Q(S_1, A_1)$即代表在 S_1状态下选择动作 A_1的 Q 值。Q 值通常是一个标量，通过比较两个 Q 值的大小来选取动作（行为策略会选择 Q 值大的那个动作） 。

算法中的 Q 值评估组合了状态-动作的质量，即：

$$Q: S \times A \rightarrow R$$

在开始训练模型的时候，Q 表中的值是随机化的（也可以设置为全 0）。代理模型在每 t 回合选择一个动作 a_t，观察奖励 r_t，并进入下一个状态 s_{t+1}，更新 Q 值。Q 值的更新使用了旧的 Q 值以及加权的新值，具体数学公式如：

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{old\ value} + \underbrace{\alpha}_{learning\ rate} \cdot \overbrace{\left(\underbrace{r_t}_{reward} + \underbrace{\gamma}_{dicount\ factor} \cdot max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right)}^{learned\ value}$$

α 是学习率，$\alpha \in (0,1)$。γ 是对未来奖励的折扣因子，$\gamma \in (0,1)$。

为了方便说明，我们继续引入一个简单的表格：

表 2 未来状态 Q 表

	A0	A1
…	…	…
St	a	b
St+1	c	d

对于 $r_t + \gamma \cdot max_a Q(s_{t+1}, a_{t+1})$，我们称为 Q 现实值，也称为 $Q(s_t, a_t)$的现实值；

对于 $Q(s_t, a_t)$，我们称为 Q 估计值，也称为 $Q(s_t, a_t)$ 的估计值。

我们可以看出，在 $Q(s_t, a_t)$ 的现实值里包含了 $Q(s_{t+1}, a_{t+1})$ 的估计值。也就是说，当前状态的 Q 现实值里包含了对下一个状态的最大 Q 估计值。简而言之，Q 现实值等于对下一步状态的衰减的最大 Q 估计值与当前奖励值之和。而 Q 现实值与 Q 估计值之差我们称为差距，也称为 loss（或者 cost）。在下文即将提到 DQN 算法里 loss 值是用来优化行为策略的一个重要标签。

上文提到，如果一直依据 Q 值选取动作，容易陷入局部最优解。所以在算法里我们也提到了 $\varepsilon - greedy$ 策略。举例来说，当 $\varepsilon = 0.9$ 的时候，代理模型有 90%的概率代理会依照 Q 值选择最佳动作，另外有 10%的概率会使用随机动作。这样就保证了训练过程中存在一定

的探索性。ε 的值也不一定是恒定的，可以设置随着训练回合的增多而衰减，即随着训练回合的次数的增多，我们随之降低使用随机动作的频率，更多遵照 Q 值选取动作。尽管如此，在前期因为代理模型需要尽可能地探索环境的可能性，因此对于得到表现良好的训练模型而言选取随机动作依然是十分必要的步骤。

2.1.2 存在的缺点

Q-learning 的算法核心就是 Q 表的更新迭代，所以也就面临着一个非常直接的问题。如果我们的环境是一个较为复杂的环境，同时每一个状态有多个可选的动作，那么就会造成 Q 表的维度爆炸。对于一个非常庞大的 Q 表，每次的检索以及更新会带来严重的效率问题，这显然是不可接受的。为了解决这个问题，下文即将提到 DQN 算法展示了一种新的思路。

2.2 深度 Q 网络（Deep Q Network）

2.2.1 单层网络结构的 DQN

DQN 是由 2013 年 Google Deep mind 团队首次提出的融合了 Q-learning 原理的强化学习算法[2]。Deep mind 团队运用 DQN 算法在雅达利 2600 模拟器上游玩了 49 个游戏，在多达一半的游戏（29 个游戏）中超越人类玩家得分的 75%，可谓效果惊人。

DQN 算法对于 Q-learning 算法做出的最大改进是利用神经网络近似耗时且低效的大型 Q 表检索更新。DQN 在仅接受非常少的先验知识的情况下使用高维信息作为输入（具体来说，是游戏的截图），所以运用了卷积神经网络来处理图像。DQN 模型不必再存储每个状态的所有可能动作对应的 Q 值，只需要把代表当前状态的图像帧输入神经网络，就能获取所有可能动作对应的 Q 值。如图：

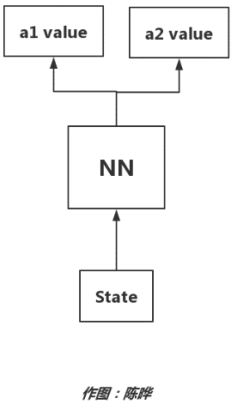


图 4 DQN 模型的输入与输出

对于雅达利 2600 模拟器上的游戏环境，Deep mind 团队采取的输入是经过处理的 4 个连续 84*84 的截图（对图像尺寸做了缩减，原始的游戏图像截图对神经网络计算要求太高），然后输入到两个卷积层，之后是两个全连接层，最后输出是包含每个可能动作的 Q 值的向量。神经网络的结构如图：

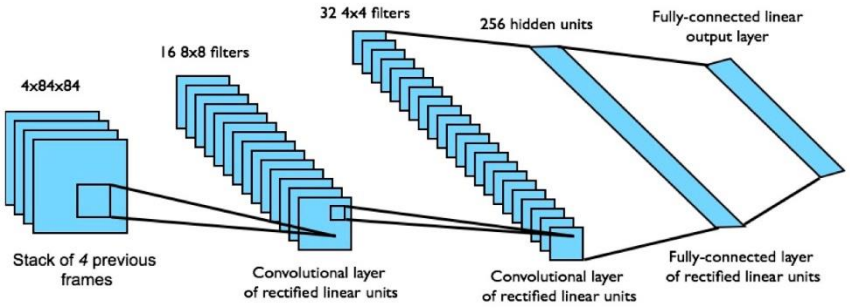


图 5　Deep mind 团队采取 DQN 模型神经网络结构

除了利用神经网络近似 Q 表之外，DQN 模型另外一个重大的改进是使用了经验回放算法（Experience Replay）。经验回放机制本质是一种受生物启发的机制。即使用先前行动的随机样本而不是最近行动的样本进行训练[8]。从游戏的第一个回合开始，如果代理模型得到探索环境的数据组为 s_1, a_1, r_2, s_2，那么第 t 回合的数据组即为 $s_t, a_t, r_{t+1}, s_{t+1}$，我们把从第 1 回合到第 t 回合的经验数据都存进经验池里，然后从中随机采样出 $\mathrm{s, a, r}, s'$ 的数据组进行训练以更新神经网络参数。如图：

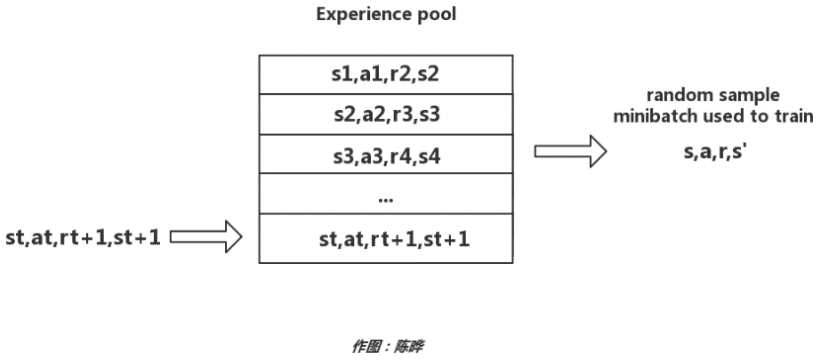


图 6　DQN 模型的经验回放机制

之所以要采用经验回放机制，是因为 DQN 作为有监督的学习模型，对数据要求其满足独立同分布，但是 Q-learning 算法得到的数据样本是前后有联系的，因此把训练过程的数据打乱成碎片化存储，就能很好的打破数据之间的关联性。

DQN 模型算法的最后一步是构造一个标签来优化神经网络参数。DQN 模型采用的损失函数（loss function）是沿用 Q-learning 算法里的 Q 现实值与 Q 估计值之差，然后进行平方处理。

$$\text{Q 现实：} r_j + \gamma max_{a'} Q(\emptyset_{j+1}, a'; \theta)$$
$$\text{Q 估计：} \mathrm{Q}(\emptyset_j, a_j; \theta)$$

损失函数的数学公式为：

$$L_i(\theta_i) = E_{s,a\sim\rho(\cdot)}[(y_i - Q(s, a; \theta))^2]$$

模型算法的目标是使用梯度下降的方式，让目标函数最小化，这个过程也是反向传播更新神

经网络参数的过程。DQN 模型通过多次训练，逐渐调整神经网络参数，获得最优行为策略。

2013 NIPS DQN 的完整算法：

Initialize replay memory D to capacity N
Initialize action − value function Q with random weights
for episode = 1, M do
Initialize sequence $s_1 = \{x_1\}$ $and\ preprocessed\ sequenced\ \emptyset_1 = \emptyset(s_1)$
$for\ t = 1, T\ do$
$With\ probability\ \epsilon\ select\ a\ random\ action\ a_t$
otherwise select $a_t = max_a Q^*(\emptyset(s_t), a; \theta)$
Execute action $a_t\ in\ emulator\ and\ observe\ reward\ r_t\ and\ image\ x_{t+1}$
Set $s_{t+1} = s_t, a_t, x_{t+1}\ and\ preprocess\ \emptyset_{t+1} = \emptyset(s_{t+1})$
Store transition $(\emptyset_t, a_t, r_t, \emptyset_{t+1})\ in\ D$
Sample random minibatch of transitions $(\emptyset_j, a_j, r_j, \emptyset_{j+1})$from D
Set $y_j = \begin{cases} r_j & for\ terminal\ \emptyset_{j+1} \\ r_j + \gamma max_{a'} Q(\emptyset_{j+1}, a'; \theta) & for\ non-terminal\ \emptyset_{j+1} \end{cases}$
Perform a gradient descent step on $\left(y_j - \mathrm{Q}(\emptyset_j, a_j; \theta)\right)^2$
end for
end for

即：

（1） 初始化回放经验池 D ，容量设置为 N

（2） 用随机权重初始化动作-值函数 Q（即神经网络，为 Q 值网络）

（3） 设定游戏总片段数为 M 。初始化网络的输入，并计算网络的输出

（4） 在概率为 ϵ 的情况下选择随机动作，或者通过网络的最大 Q 估计值选择动作

（5） 在模拟器里执行动作 a_t 得到奖励 r_t 和下个状态的图像 x_{t+1}

（6） 计算下一个时刻网络的输出

（7） 把当前状态、当前动作、当前奖励值、下个状态作为四元组存储进经验池里

（8） 从经验池里随机采样

（9） 利用随机采样的样本计算状态的目标值，如果在这一步游戏结束，该步的 Q 现实值即为奖励值；如果没有结束，则遵照公式计算。

（10） 以 loss 函数为标签，利用梯度下降进行反向传播调整神经网络参数

2.2.2 双层网络结构的 DQN

双层网络结构 DQN 算法模型是由 Deep mind 团队在 2015 Nature 上首次提出。双层网络 DQN 与单层网络 DQN 的基本算法原理相同。最大的区别是双层网络结构的 DQN 算法模型加入了额外的一层神经网络用于计算 Q 现实值，这层神经网络我们称为目标网络（Target_net）。Q 估计值交由另外一层神经网络负责计算，这层神经网络我们称为估计网络（Eval_net）。两个网络结构相同，使用同样的随机化参数初始化网络。估计网络的参数与单层网络一样，保持每步更新，而目标网络的参数则有一定的延迟，在每 N 轮迭代后使用估计网络的参数更新。由于目标网络的参数在一段时间内是不变的，所以在这一段时间里计算的目标 Q 值也是保持不变的。引入目标网络对于降低 Q 估计值和 Q 现实值的相关性有一定程度的帮助，提高了算法的稳定性。

本文后续实验主要基于双层网络结构的 DQN 进行，神经网络结构示意图如下：

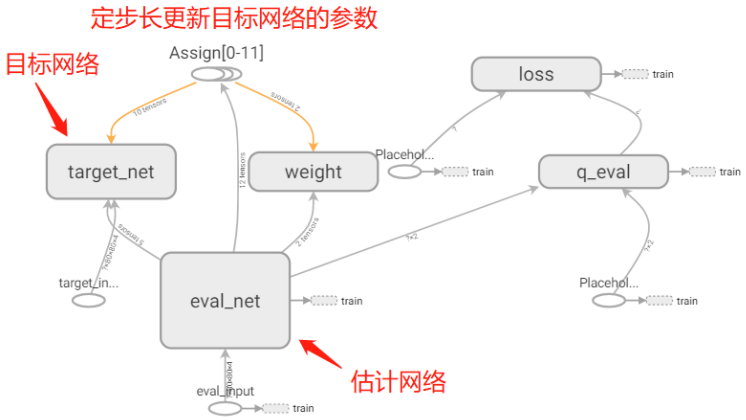


图 7 目标网络与估计网络的参数更新

双层网络 DQN 的完整算法如：

Initialize replay memory D to capacity N
Initialize action − value function Q with random weights θ
Initialize target action − value function $\hat{Q}$ *with weights* $\theta^{-} = \theta$
For episode $= 1,$ M do
  Initialize sequence $s_1 = \{x_1\}$ *and preprocessed sequence* $\emptyset_1 = \emptyset(s_1)$
  For t $= 1,$ T do
    With probability ε select a random action a_t
    otherwise select $a_t = argmax_a Q(\emptyset(s_t), a; \theta)$
    Execute action a_t *in emulator and observe reward* r_t *and image* x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ *and preprocess* $\emptyset_{t+1} = \emptyset(s_{t+1})$
    Store transition $(\emptyset_t, a_t, r_t, \emptyset_{t+1})$ *in D*
    Sample random minibatch of transitions $(\emptyset_j, a_j, r_j, \emptyset_{j+1})$from D
    Set $y_j = \begin{cases} r_j & if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma max_{a'} \hat{Q}(\emptyset_{j+1}, a'; \theta^{-}) & otherwise \end{cases}$
    Perform a gradient descent step on $\left(y_j - \mathrm{Q}(\emptyset_j, a_j; \theta)\right)^2$ *with respect to the*

network parameters θ
Every C steps reset $\hat{Q} = Q$
End For
End For

即：

（1） 初始化回放经验池 D ，容量设置为 N

（2） 用随机权重初始化估计神经网络 Q

（3） 用同样的随机权重初始化目标神经网络 $\hat{Q}$

（4） 设定游戏总片段数为 M ，初始化网络的输入，并计算网络的输出

（5） 在概率为 ϵ 的情况下选择随机动作，或者通过估计网络的最大 Q 估计值选择动作

（6） 在模拟器里执行动作 a_t 得到奖励 r_t 和下个状态的图像 x_{t+1}

（7） 计算下一个时刻网络的输出

（8） 把当前状态、当前动作、当前奖励值、下个状态作为四元组存储进经验池里

（9） 从经验池里随机采样

（10） 利用随机采样的样本计算状态的目标值，如果在这一步游戏结束，该步的 Q 现实值即为奖励值；如果没有结束，则遵照公式计算，并使用目标网络计算 Q 估计值。

（11） 以 loss 函数为标签，利用梯度下降进行反向传播调整神经网络参数。其中的 Q 估计值使用估计网络计算。

（12） 每 C 步用估计网络的参数更新目标网络的参数

2.2.3 双 Q 学习算法（Double DQN）

Double DQN 是基于双层网络 DQN 的训练算法优化，由 Deep mind 团队于 2016 年 AAAI 上首次提出。

Double DQN 优化算法旨在减少过估计（Overestimate）的问题。造成过估计的问题是是由于在计算 Q 现实值的时候的取最大值操作，双层网络结构 DQN 的 Q 现实值公式为：

$$Y_t^{DQN} = R_{t+1} + \gamma \underbrace{max}_{a} Q(S_{t+1}, a; \theta_t^-)$$

算法每次计算 Q 现实值都是通过取下个状态的最大值 Q 估计值得到的，但是神经网络计算的 Q 估计值是有误差的，所以每次在获取 Q 现实值获得的误差也是最大的，而在通过作为标签的损失函数改进神经网络的时候，也会受到误差最大化的影响，进而导致过估计。

Double DQN 对 Q 现实值的计算做出了改进，公式为：

$$Y_t^{DoubleDQN} = R_{t+1} + \gamma Q(S_{t+1}, \underbrace{argmax}_{a} Q(S_{t+1}, a; \theta_t), \theta_t^-)$$

Double DQN 算法对于 Q 现实值里下个状态的 Q 估计值动作的选取，不再是直接选取动作对应的最大 Q 值，而是利用另外一个拥有较新参数的估计网络获取 $Q(S_{t+1}, a)$ 中具有较大值的动作，然后再用这个动作决定 Q 现实值里的 $Q(S_{t+1}, a)$ 选取哪个动作对应的 Q 值。

Double DQN 能有效减小训练过程中的 Q 现实值误差，也是本文将要运用到的优化算法。

3 实验游戏模型

3.1 模型背景知识

3.1.1 模型简介

本文采用的游戏环境模型为基于 python 语言编写的开源游戏 Flappy Bird[9]。在 Flappy Bird 游戏中我们需要控制小鸟进行跳跃，通过柱子。小鸟每通过一根柱子，得分加一。小鸟碰到柱子或者地面，本回合游戏结束，开始下一回合。游戏没有终点，若一直通过柱子，本回合就能一直进行。原始游戏如下图：

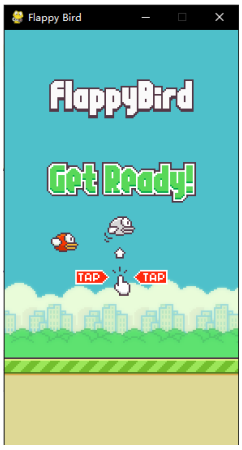


图 8 原始游戏

小鸟共有三种颜色，柱子有两种的颜色，背景图有两种，游戏中随机切换。如下图：

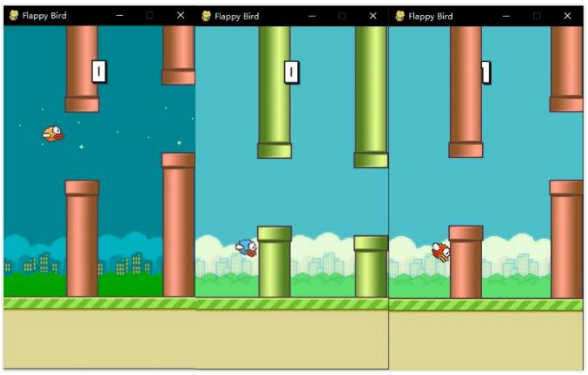


图 9 游戏中的随机素材切换

3.1.2 模型预处理

为了降低卷积神经网络处理输入截图的压力以及方便后续工作的展开。我们对游戏图像素材进行替换预处理：

（1） 把小鸟的颜色限制为红色。

（2） 把柱子和地面的素材替换成简单的单色素材。

（3） 把背景换成全黑。

（4） 去掉游戏开始准备界面与得分计数。

经过预处理的游戏图像如下图：

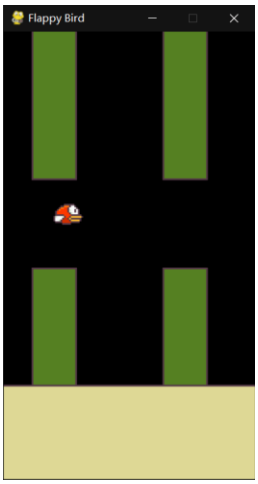


图 10 经过预处理的游戏模型

3.1.3 模型交互设置

对于本次构建 DQN 模型代理来说，从零开始学习如何游玩 Flappy Bird，接受的输入只有高维信息，即游戏的图像。可选的动作有两种，跳跃与不跳（不跳就是什么都不做）。奖励值是通过的柱子，通过一根柱子奖励值加 1，平时没有通过柱子正常飞行时每回合奖励值为 0.1。如果碰到柱子或者地面，小鸟死亡，该回合结束，奖励值为-1，游戏重新开始下一回合。

在早期阶段，神经网络的参数都是随机化的，对于输入的状态计算的 Q 估计值对于动作质量的评估并没有太大的意义。这个阶段，也相当于模型的探索阶段，小鸟会不断碰到柱子，游戏回合会不断结束又重新开始，这也是典型的“试错”阶段。神经网络的参数也在这个过程中被不断调整，最终学会如何“游玩”Flappy Bird，也就是获得最优动作策略。

对于本模型的 $\varepsilon-greedy$ 策略，ε 设置的初始值为 0.1，然后逐渐衰减为 0.0001。而 Deep mind 团队在雅达利 2600 游戏上使用 ε 初始值为 1，然后逐渐衰减为 0.1。相比之下，本次实验模型似乎使用随机动作的概率要小很多。但是因为 Flappy bird 游戏大约以每秒 30 帧的速度显示画面，代理模型选择一个动作只要 0.03 秒，实验结果表示过高的 ε 会使得小鸟选择跳跃动作的频率过高，从而一直飞到屏幕的最上方碰到柱子，因此我们需要降低使用随机动作的概率，以保证正常的训练过程。

3.2 模型的神经网络结构

本次实验模型采用 tensorflow 来构建模型的神经网络结构。流程：

（1） 每步的最后四帧处理成 $80\times80\times4$ 作为神经网络的输入

（2） 输入到第一层卷积层进行处理

（3） 进行池化层处理

（4） 输入到第二层卷积层进行处理

（5） 输入到第三层卷积层进行处理

（6） 第三层卷积层的输出进行维度处理之后输入到第一层全连接层

（7） 第一层全连接的输出再输入到第二层全连接层，第二层全连接层也是我们的 Q 值函数层。因为我们的游戏模型只有两个可选动作（跳与不跳），所以最后输出为包含两个可能动作的 Q 值向量

模型流程结构图如：

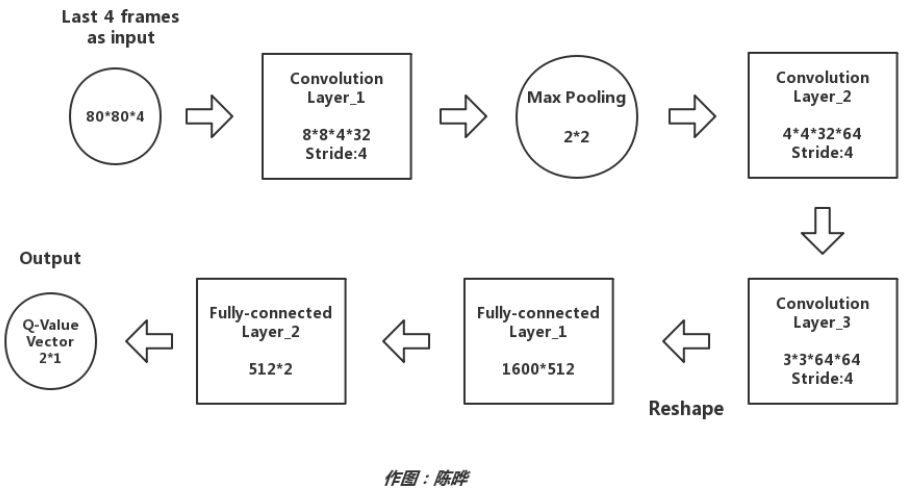


图 11 神经网络结构流程

本次实验模型主要构建两个结构相同并如同上图的神经网络，一个作为目标网络（target_net），一个作为估计网络（eval_net），以实现双层网络的 DQN 结构。双层网络结构的 DQN 完整模型如下图：

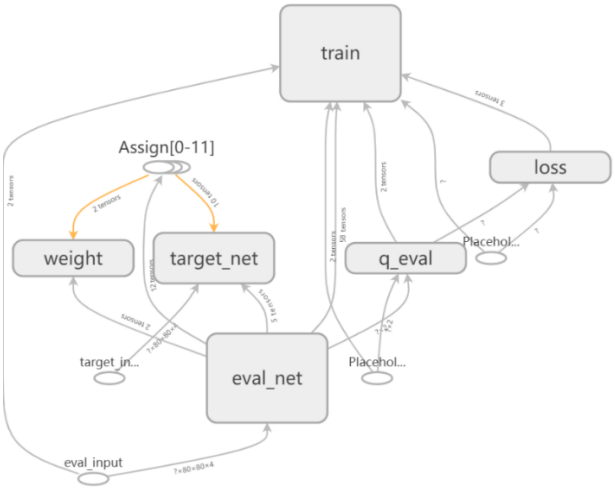


图 12 双层网络结构的 DQN 完整模型

3.3 基于模型先验知识的干预原理

3.3.1 基于预测帧的 Q 值干预

针对本次实验采用的游戏模型，我们设置了一套基于模型先验知识的人为干预训练过程的规则机制。对于游戏中小鸟，每一个状态下都有两个可选动作（跳与不跳），选择不同的动作，会进入不同的下一个状态。

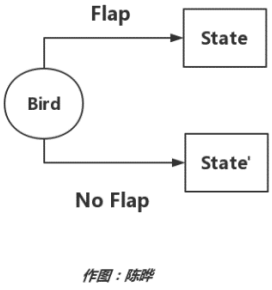


图 13 小鸟动作选择

我们希望加入一种规则，来辅助小鸟进行当前动作的选择。在游戏环境中，小鸟横向的速度是固定值，纵向的跳跃加速度与下降加速度也是固定值，并且存在最大与最小速度的限制速度。因此我们可以把以上信息作为先验知识输入模型，以同时计算当前状态小鸟选择两种不同动作的下一帧的位置，并画出下一帧的图像，然后把图像输入进神经网络，得到基于预测帧的动作 Q 估计值向量。我们从中选择较大的那一个 Q 估计值，然后与当前状态的 Q 估计值相加。进行完以上步骤以后，小鸟再通过当前 Q 估计值进行实际动作选择。流程如图：

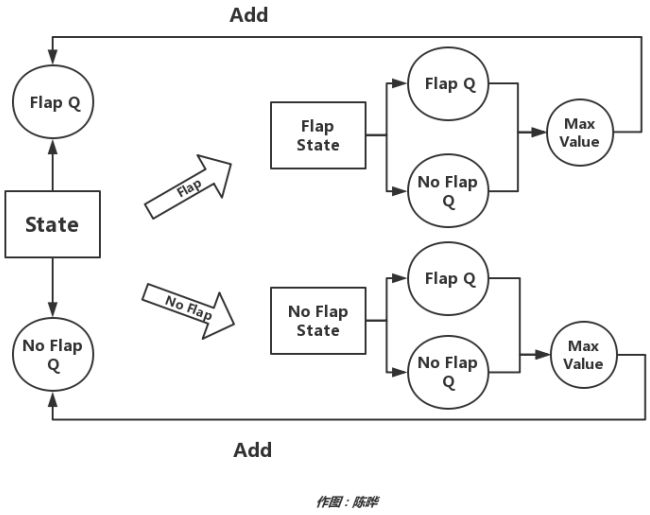


图 14 基于预测帧的 Q 值干预流程图

3.3.2 基于预测帧的 Q 值决策

基于预测帧的 Q 值决策机制是上一种干预机制的变种。我们不再通过预测帧的 Q 估计值加入到当前状态的 Q 估计值，而是直接通过预测帧的最大 Q 估计值来选取动作，流程如图：

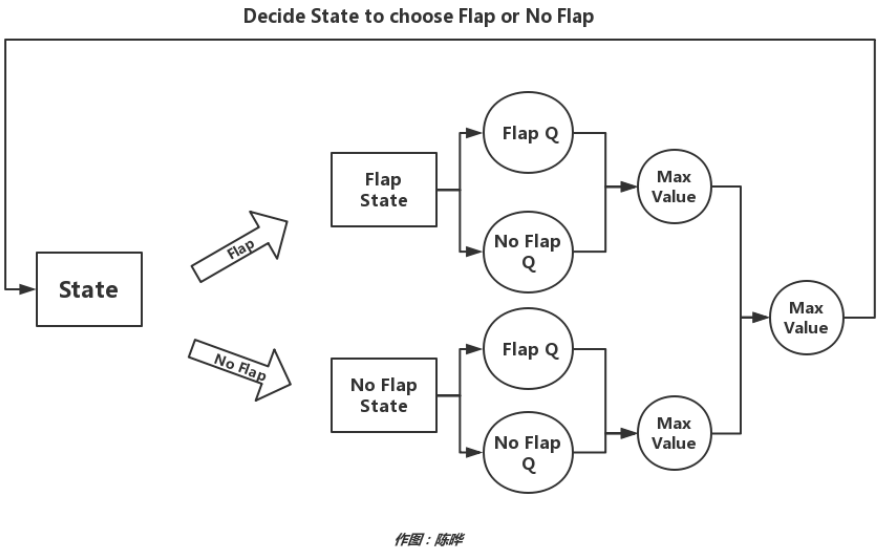


图 15 基于预测帧的 Q 值决策

3.3.3 基于图像识别的动作决策

前两种机制是基于已经高度抽象化的未来 Q 估计值对当前状态进行干预。因此我们希望尝试加入一种更为直观的干预机制。

人类玩家玩游戏的时候，是通过视觉信息判断小鸟此刻应该选择哪一个动作的，如果小鸟此刻高于柱子，则选择不跳；如果此刻低于柱子，则选择跳。那么我们也可以将这套识别系统加入环境模型中。具体原理是是通过识别小鸟的质心 Y 轴坐标与将要通过的第一组柱子的坐标，如果低于下柱子，则选择跳；如果高于上柱子，则选择不跳。机制在小鸟飞过柱子之前给出决策动作。在小鸟飞过柱子的间隔过程中以及其他情况则交给神经网络参数进行动作调节。识别原理如图：

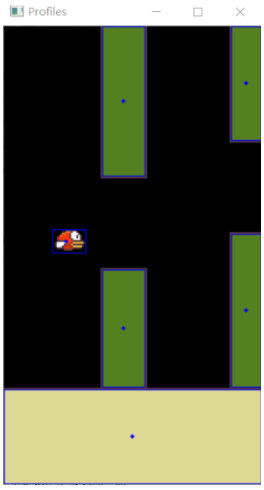


图 16 游戏模型的图像识别

另外需要特别说明的是，我们希望在早期让训练模型更多地探索游戏环境以调整神经网络参数，所以在使用该机制时我们借鉴了模型选择动作的 $\varepsilon-greedy$ 机制，设置了一个 β 值，初值设置为 0.1，然后在两百万步之内上升到 1。也就是按照一定的递增概率来触发这个基于图像识别的动作决策机制。

3.4 模型的评估指标

3.4.1 每五万步的平均 Q 估计值

得益于 tensorflow 自带的 tensorboard 记录终端，我们能轻松记录下模型训练过程中的数据指标，并且不用当心数据读写对程序运行的负面效率影响。

我们首先记录的是每五万个回合的平均 Q 估计值，横轴为训练步长，纵轴为 Q 估计值的大小。例图如：

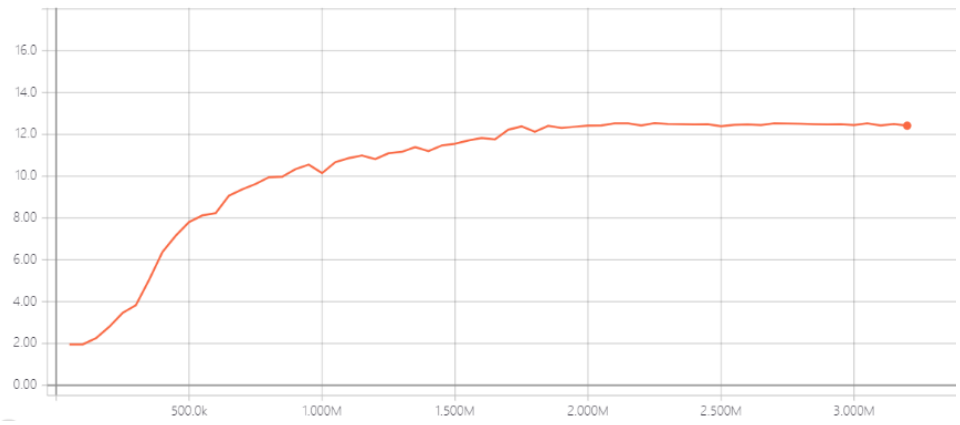


图 17 每五万步的平均 Q 估计值

这个数据指标也是 Deep mind 团队在 2013 年 NIPS 发表的初次介绍 DQN 算法模型使用的训练指标之一。根据实验数据表明，Flappy Bird 的 DQN 模型表现出呈现收敛状态的游戏水平时，平均 Q 估计值的值域大概保持在 12 到 13 之间。但是我们并没有直接的数学证据表明平均 Q 估计值呈现稳定水平的时候能够说明模型收敛，我们只能把这个指标作为观察训练过程是否正常进行的数据表征。

3.4.2 损失函数（loss function）

第二个记录的数据为损失函数的返回值。数学公式为：

$$L_i(\theta_i) = E_{s,a\sim\rho(\cdot)}[(y_i - Q(s,a;\theta))^2]$$

从公式以及之前的定义可以的得知，函数的实际意义为 Q 现实值与 Q 估计值的差的平方。这也是神经网络通过反向传播调整网络参数的重要指标。网络参数是朝着使得损失函数值越来越小的趋势进行调整的。横轴为训练步长，纵轴为函数值。例图：

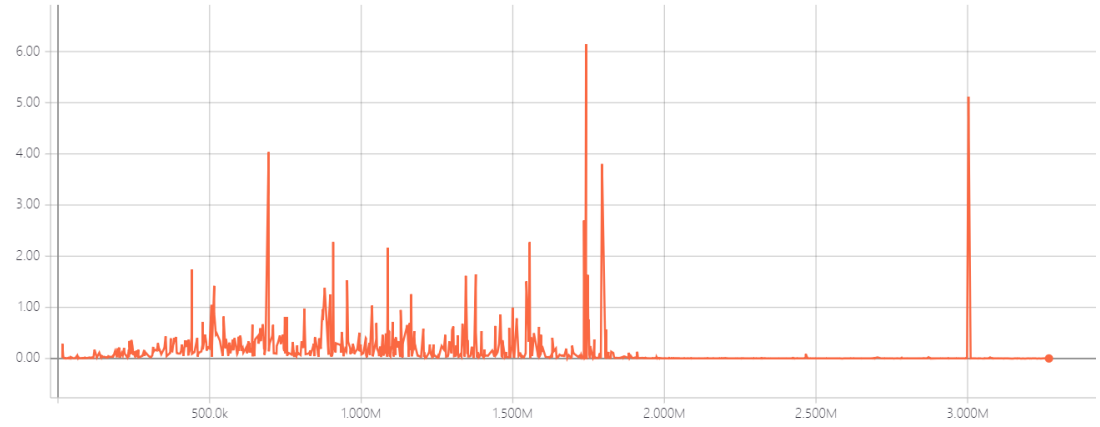


图 18 损失函数

在模型游戏表现呈现收敛的时候，损失函数也会呈现收敛的情况，虽然后面也会出现较大的波动值，但是可以视为探索到了极少数没有探索到的状态，因此导致数值出现波动，并不影响整个收敛趋势。但是损失函数呈现收敛状态并不能作为直接推导模型收敛的证据，很多时候只是作为训练模型收敛的必要条件，但是不能作为充分条件。并且会存在一种特殊情

况使得损失函数模型表现未收敛的情况下呈现收敛状态。本次毕业设计项目不以损失函数作为主要观察指标，关于这一点本文后面的工作会详细说明。

3.4.3 每回合累计奖励值

我们以每次小鸟重生开始至撞到柱子或者地面为止计数为一回合，统计每回合小鸟通过的柱子数。即横轴为小鸟生命数（也是游戏回合数），纵轴为通过的柱子数。例图：

图 19 每回合累计奖励值

对于强化学习而言，最能体现模型收敛的证据就是代理模型在游戏环境中的表现。代理模型训练的最终目的是获取最优行为策略，从而在游戏环境中得到更好的表现。Deep mind 团队在雅达利 2600 游戏中也使用了环境的奖励值作为数据指标，但是他们统计的为每五万步的平均奖励，即与统计平均 Q 估计值数据的横轴坐标相同。而对于我们本次的实际游戏环境来说，以训练步长为横坐标并不是一个好的选择，因为小鸟通过柱子的步长数对于整个回合过程中的步长数占比太小。如果以时间单位度量，在飞行的 10 秒过程内，飞过柱子的时间可能只占 1 到 2 秒，以训练步长为横坐标，则会统计大量无意义的零数据。

而我们选择统计每回合的累计游戏奖励而不是统计一定数量回合的平均累计奖励，是因为在后期模型收敛的时候，由于训练模型对于最优策略已经达到足够的逼近效果，小鸟会在很长一段时间内一直游玩，能够飞过非常多的柱子。此时如果统计一定数量回合的平均奖励值，我们得到图表数据的时间成本会变得难以承受。因为我们以单个回合计数，所以即便模型表现趋于收敛，但是还是会出现奖励值较小的波动回合，但是只要出现明显的累计奖励值波峰（如示例图后期通过了几百根柱子），我们便可认为模型对于最优策略已经达到足够的逼近。

因此，我们认为该数据指标能作为较为有效的推出模型收敛的证据。

3.4.4 全连接层的历史参数差值

我们想尝试是否能通过直接的数据表现判断模型是否收敛，这是我们做的一个尝试。Deep mind 团队曾在发表的论文中指出[2]，目前没有明确的方法能在数学上确定模型已经训练收敛了，最直接而明确的方法依旧是观察代理模型在环境中的游戏表现。因此我们提出一个猜想，如果一个 DQN 模型已经被训练成几乎收敛的状态，那么它的神经网络的参数可能会存在变化减小的趋势。基于之前我们统计的平均 Q 估计值与损失函数在训练模型表现近乎收敛时表现出的稳定的状态，我们决定尝试记录与平均 Q 估计值输出有较为密切关系的最后两层全连接层的参数状况。由于神经网络参数是以矩阵的方式存在的，所以我们对神经

网络参数进行历史版本的记录，然后与最新的网络参数作差得出参数的差值矩阵，对差值矩阵进行 L2 正则化的处理，求出一个标量以画出折线图。横轴为训练步长，纵轴为记录的标量。

我们首先测试了每隔 100 步训练步长的历史参数差值矩阵，记录如图 16,、17：

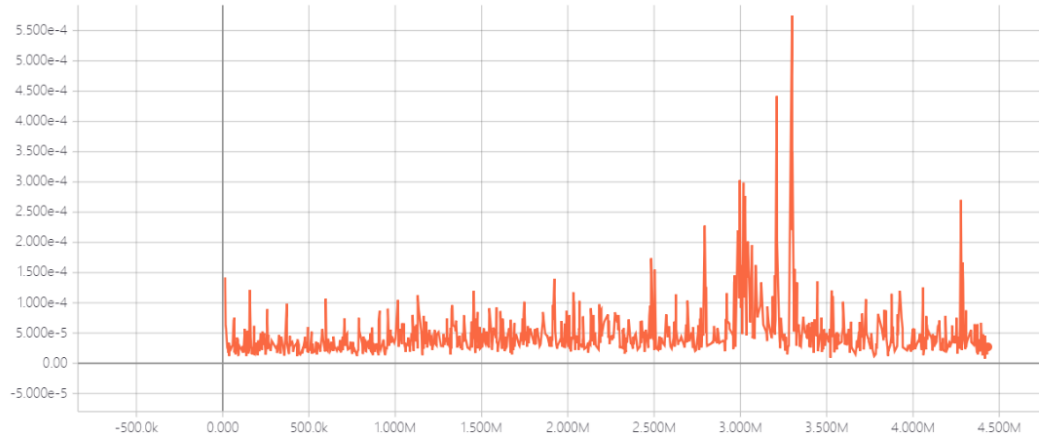


图 20 第一层全连接层 差值矩阵步长为 100

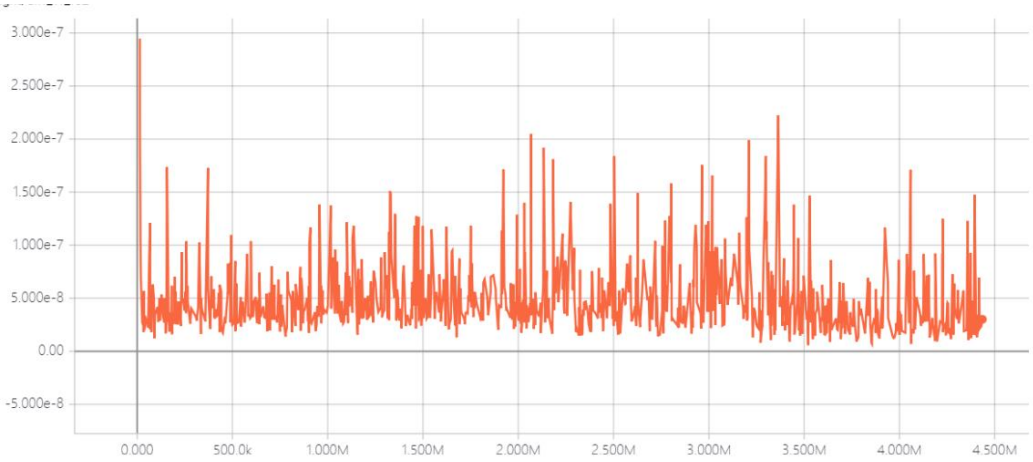


图 21 第二层全连接层 差值矩阵步长为 100

观察趋势，我们可以看到，记录的关于两层全连接层的标量绝对值都处于一个较小的值域区间，但是比起数值绝对值大小我们更看重数值变化的趋势。对于第二层全连接层来说，标量除了在训练开始呈现大幅下降的趋势，其余时间都保持反复的震荡。对于第一层全连接层来说，除去有部分时期出现较大波动趋势，其它大部分时期同样保持反复的震荡。也就是说，从开始训练到模型收敛，神经网络的参数一直处于一种迭代更新的状态，不会出现我们设想中的参数变化趋势减小的情况。

对于实验结果，我们进行了进一步的思考。我们一开始设置步长为 100，是基于让参数之间的变化差异更明显的考虑而设置的。但是基于目前的实验结果，我们的另一个猜想是，过长的差值矩阵步长也许会导致差值矩阵差异过大，导致微小的收敛的趋势被掩盖，所以我们缩减差值矩阵的步长为 1，结果如图 22、23：

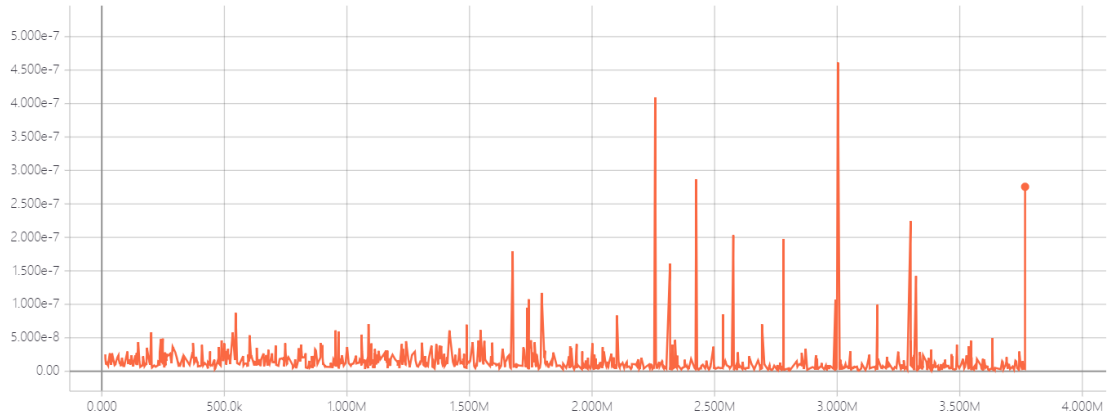


图 22 第一层全连接层 差值矩阵步长为 1

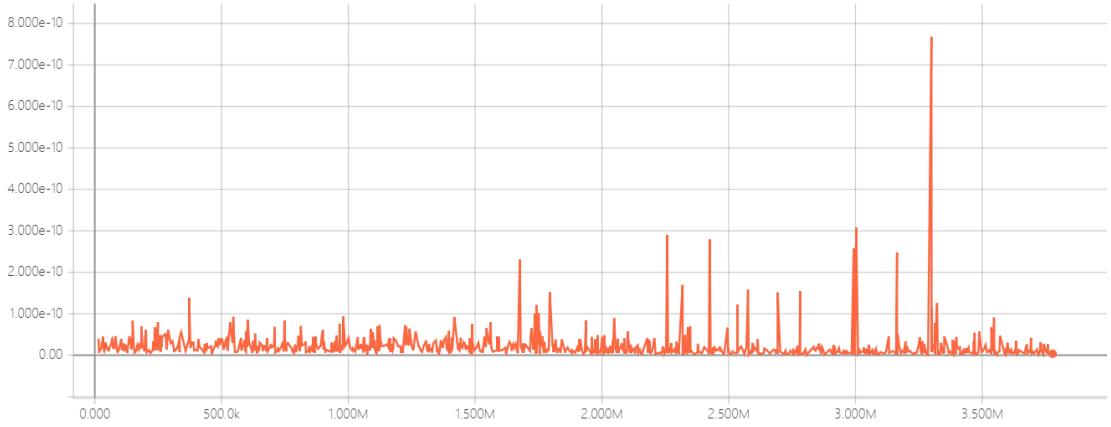


图 23 第二层全连接层 差值矩阵步长为 1

比较遗憾的是，即便我们减小差值矩阵的记录步长，依旧不能从图像观察到较为明显的神经网络参数收敛趋势，即通过记录神经网络参数的历史变化以寻求明确的数据证据证明训练模型收敛在本次项目中不是一种可行的解决方案。

4 双层网络 DQN 模型的超参数优化与优化算法测试

4.1 双网络间参数较优更新步长

4.1.1 实验测试

双网络结构 DQN 的原理就是利用目标网络参数的延迟降低 Q 估计值与 Q 现实值的相关性，从而提高算法的稳定性。虽然我们的直觉是目标网络参数延迟越大，Q 估计值与 Q 现实值的相关性越低。但是参数延迟具体对训练模型的影响需要通过实际的实验来验证。

我们设置了五组实验，分别是：

（1） 单网络模型。简称为 s 模型。作为评估模型性能的基准值。图中为绿色线。

（2） 双网络模型，目标网络参数更新步长为 50。简称为 α 模型。图中为灰色线。

（3） 双网络模型，目标网络参数更新步长为 100。简称为 β 模型。图中为天蓝色线。

（4） 双网络模型，目标网络参数更新步长为 200。简称为 γ 模型。图中为深蓝色线。

（5） 双网络模型，目标网络参数更新步长为 300。简称为 δ 模型。图中为棕色线。

本次实验超参数说明：

（1） 所有实验均使用同一随机种子值 26，确保实验可重复并具有可对比性。

（2） $\varepsilon-greedy$ 的 ε 值均在 300 万步内从 0.1 衰减到 0.0001。

首先我们观察平均 Q 估计值的统计指标。以 α 模型与 s 模型作为对比，如下图：

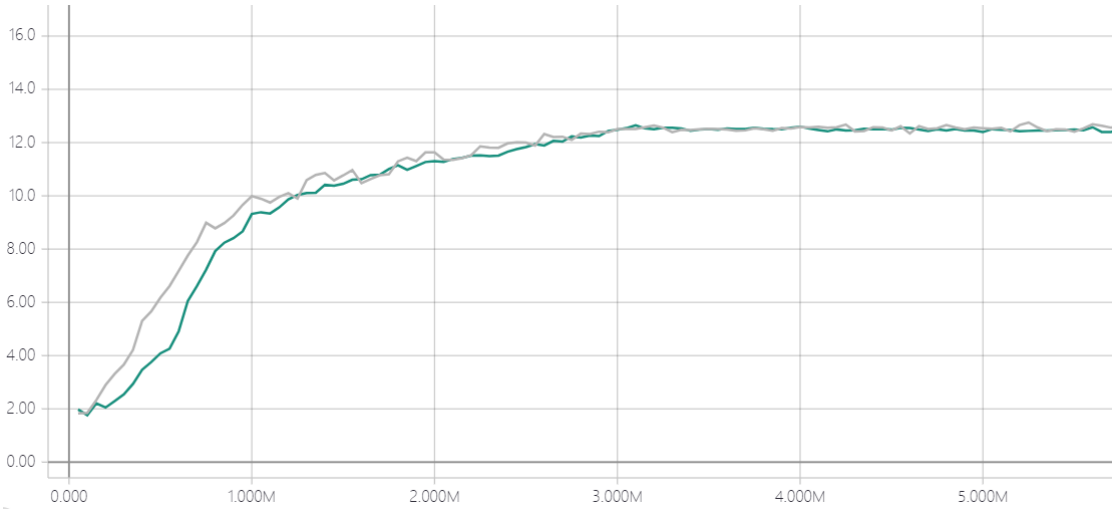


图 24 绿色 s 模型，灰色 α 模型

我们可以看到两个模型的平均 Q 估计值均在大约 300 万训练步长时达到稳定的值域，但是在前 100 万训练步长，α 模型的 Q 估计值增长速率明显高于 s 模型。我们把其他模型都加入到对比中，如图：

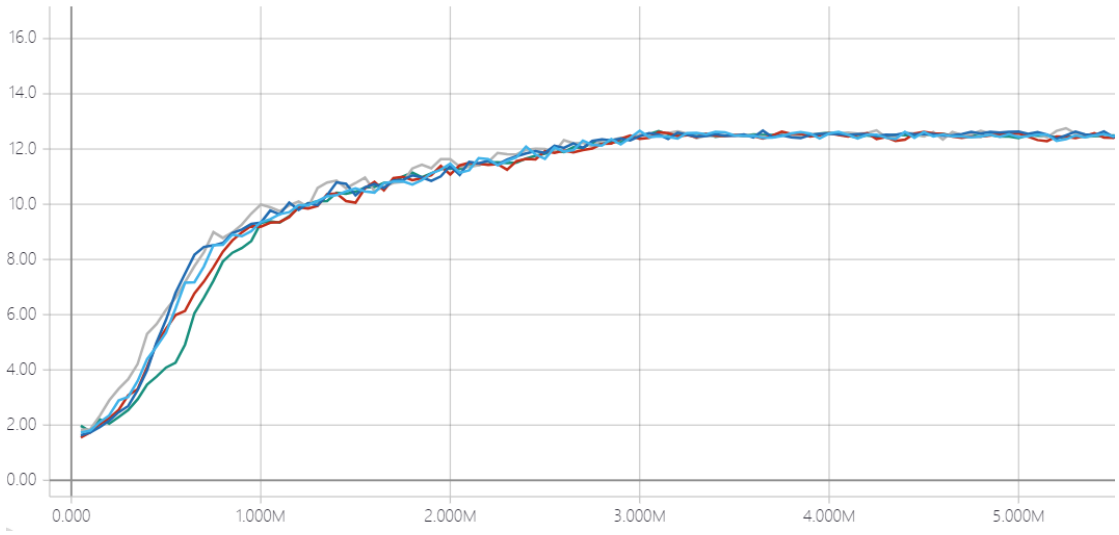


图 25 绿色 s 模型，灰色 α 模型，天蓝色 β 模型，深蓝色 γ 模型，棕色 δ 模型

我们可以看到所有模型均在大概 300 万训练步长达到稳定的平均 Q 估计值。在前 100 万训练步长中，单网络模型的平均 Q 估计值增长速度最慢，其余双网络模型的平均 Q 估计值增长速度虽然各异，但差距都不明显。对于实验结果，我们的猜想是双网络结构在前期有助于神经根网络的参数调整，故使得平均 Q 估计值增长较快。

对于损失函数的记录，s 模型大约是在 300 万时开始收敛，如图：

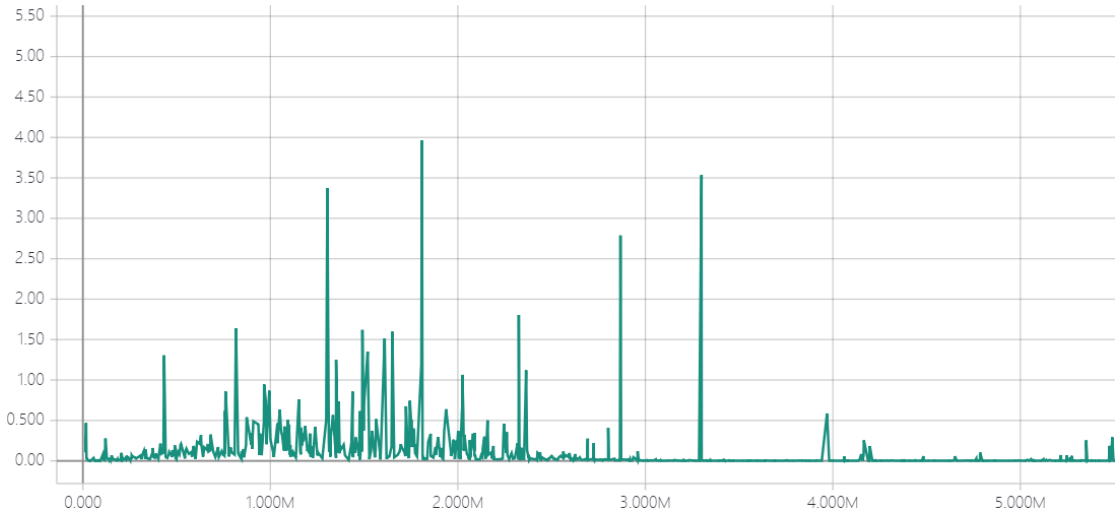


图 26 s 模型损失函数

其余四个模型的损失函数收敛情况如图：

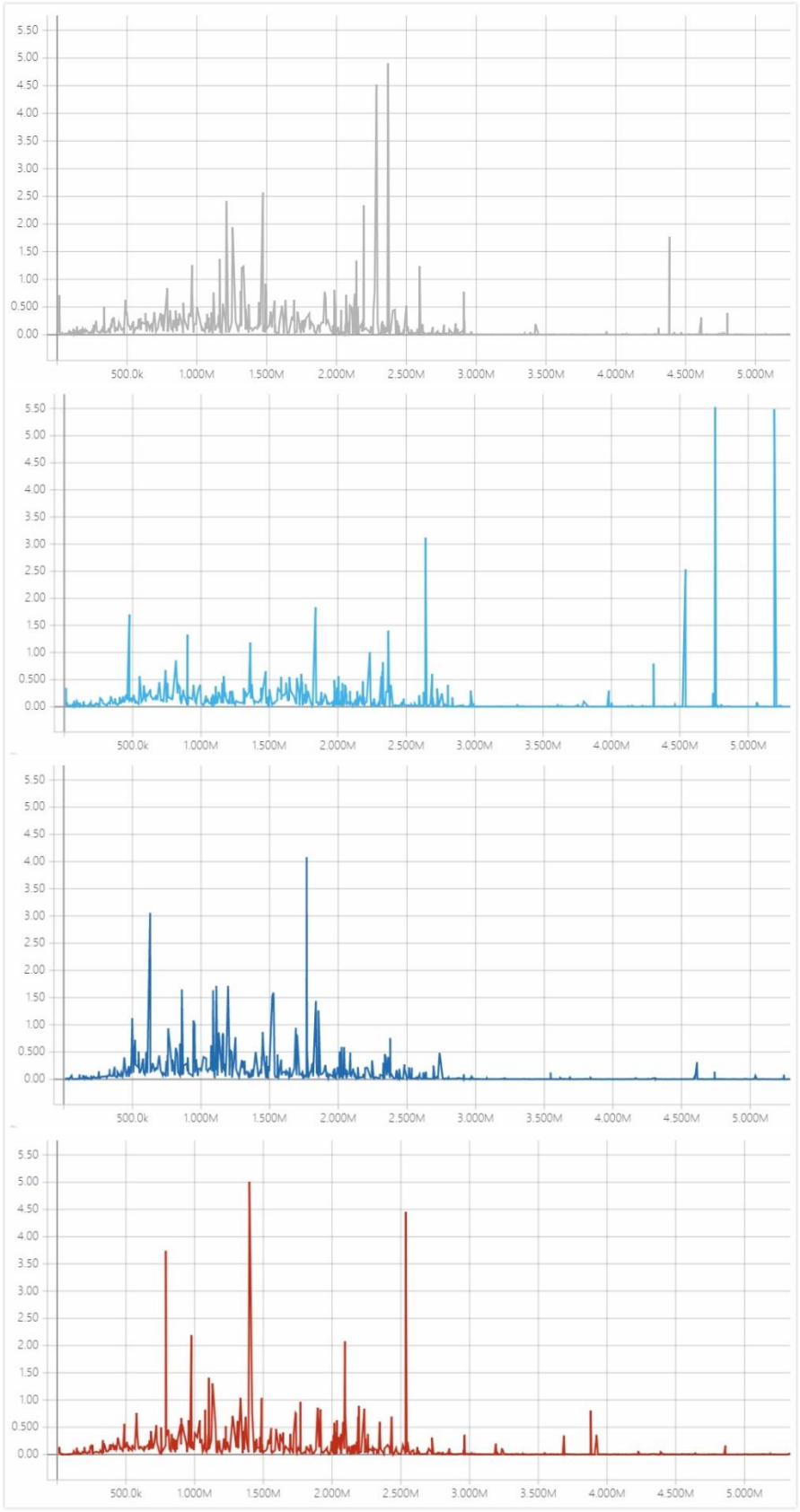


图 27 α 模型， β 模型， γ 模型， δ 模型

可以看出，其余四个双网络模型的损失函数会略微早于 300 万之前收敛，但提前量并不算太多。前两个数据表征不能带来较为直观的收敛情况，而观察累计奖励值的数据能给出较为明显的对比情况。

以 δ 模型与 s 模型作对比，可以看到达到奖励值波峰所用的回合数基本相同。如图：

图 28 棕色 δ 模型，绿色 s 模型

可以看出在参数更新步长为 300 的时候，双网络结构与单网络结构的训练效率并没有什么区别。我们再对比参数更新步长为 200,100 的模型，如图：

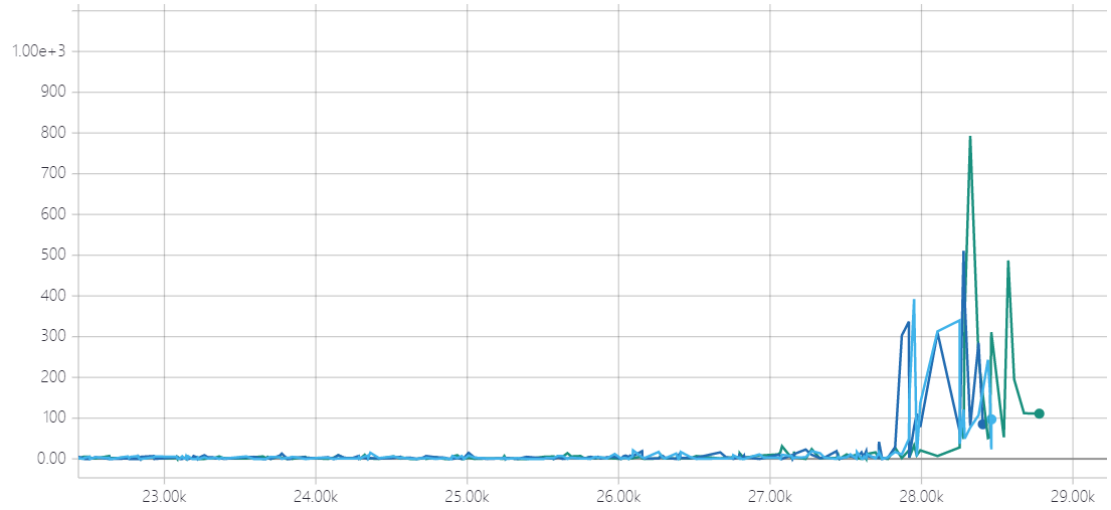


图 29 天蓝色 β 模型，深蓝色 γ 模型，绿色 s 模型

当我们把参数更新步长缩短之后，可以看到双网络模型训练效率相比单网络模型训练效率有了一定的提升，在更早的回合达到了累计奖励波峰。而参数更新步长为 200 与 100 之间的差别非常小，可以认为曲线基本处于重合趋势。最后我们来看参数更新步长为 50 的模型对比，如图：

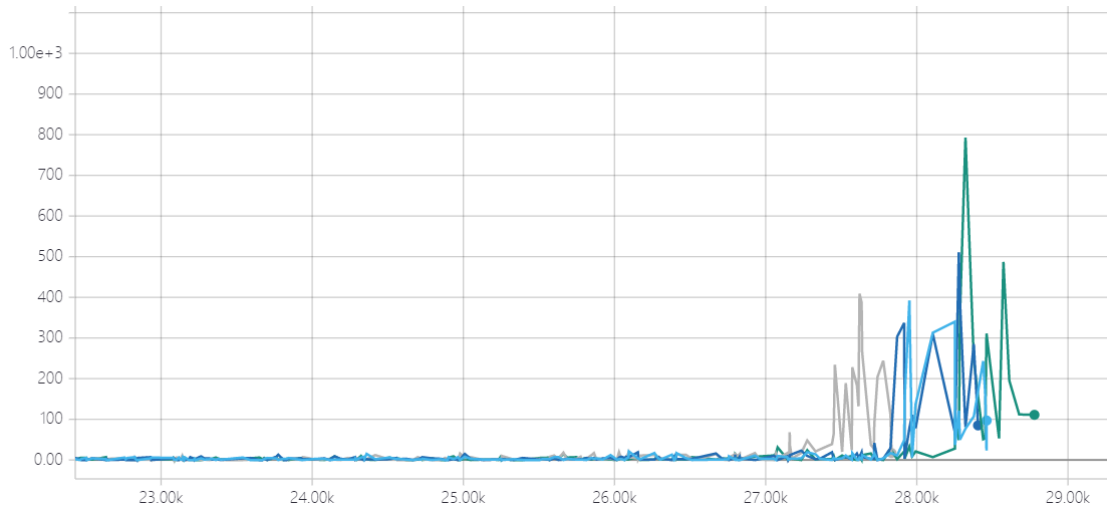


图 30 灰色 α 模型，天蓝色 β 模型，深蓝色 γ 模型，绿色 s 模型

可以很明显看出，网络参数更新步长为 50 在这几个模型中的训练效率是最高的。

4.1.2 实验总结

通过以上实验结果，可以得出的结论为：在本次实验环境中，所有双网络结构 DQN 模型相比于单网络结构 DQN 模型在前期的平均 Q 估计值上升速度更快。而对于训练效率而言，网络参数的更新步长并不是越长越好，过长的参数更新步长反而会导致训练效率下降。本组实验训练效率最高的为采用网络参数更新步长为 50 的 α 模型。

4.2 Double DQN 算法优化

4.2.1 实验测试

我们在双网络结构的 DQN 模型基础上加入 Double DQN 算法进行优化。我们设置了两组新实验，并与之前做的实验进行对比，进行观察的实验模型有：

（1） 单网络模型。简称为 s 模型。图表为绿色线。

（2） 双网络模型，目标网络参数更新步长为 50。简称为 α 模型。图表为灰色线。

（3） 双网络模型，目标网络参数更新步长为 100。简称为 β 模型。图表为天蓝色线。

（4） 双网络模型，目标网络参数更新步长为 50，使用 Double DQN 优化算法。简称为 α_1 模型。图表为粉红色线。

（5） 双网络模型，目标网络参数更新步长为 100，使用 Double DQN 优化算法。简称为 β_1 模型。图表为橙色线。

本次实验超参数说明：

（1） 所有实验均使用同一随机种子值 26，确保实验可重复并具有可对比性。

（2） $\varepsilon - greedy$ 的 ε 值均在 300 万步内从 0.1 衰减到 0.0001。

对于平均 Q 估计值的观察，如图：

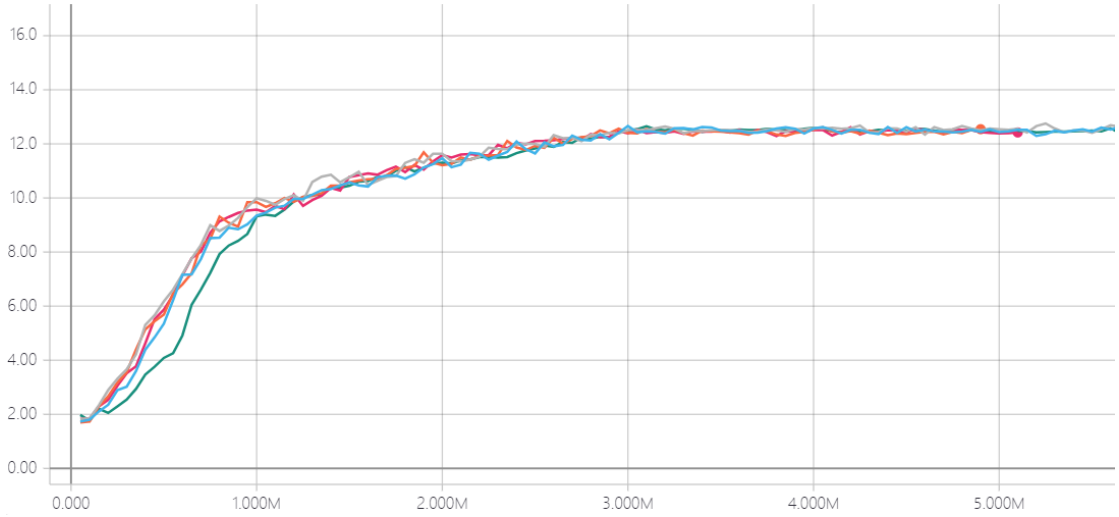


图 31 灰色 α 模型，天蓝色 β 模型，粉红色 α_1 模型，橘色 β_1 模型，绿色 s 模型

我们得到的结论是类似的：双网络模型在前 100 万训练步长的平均 Q 估计值增长速度均比单网络模型快，而互相之间则区别不大。所有模型的平均 Q 估计值均在 300 万训练步长达到稳定的状态。

对于累计奖励值指标，我们先进行两两比较，即 α 模型与 α_1 模型进行比较，β 模型与 β_1 模型进行比较。如下图：

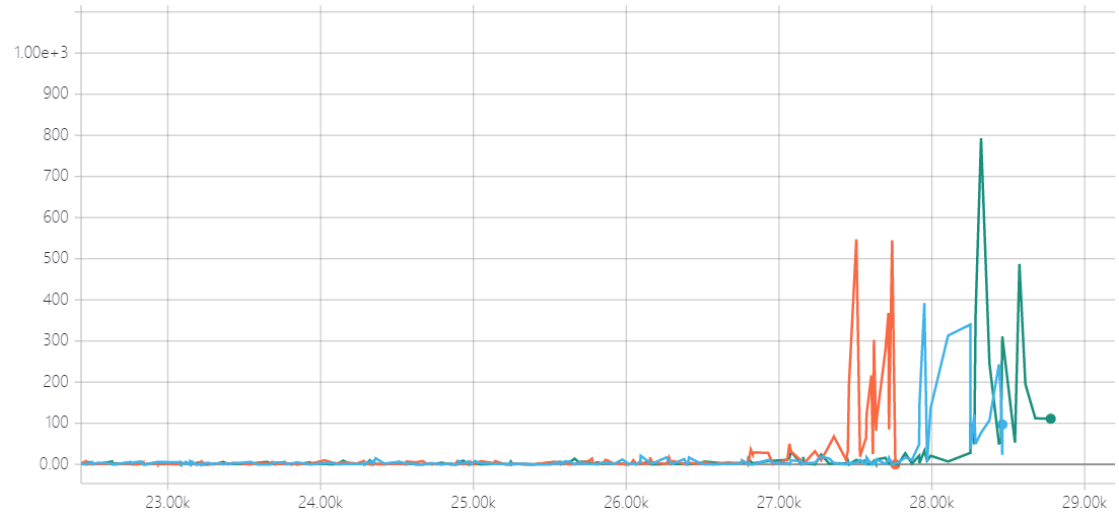


图 32 橘色 β_1 模型，天蓝色 β 模型，绿色 s 模型

由图可知，在网络参数更新步长为 100 的情况下，运用了 Double DQN 算法优化的双网络结构模型相较于原本的双网络结构模型训练效率再次得到提升。

在网络参数更新步长为 50 的情况下，训练数据如图：

图 33 粉红色 α_1 模型，灰色 α 模型，绿色 s 模型

我们可以发现，应用 Double DQN 优化算法的双网络模型的奖励值波峰会较大一些，但是总体而言与原版双网络模型到达奖励值波峰所用的回合数大体是一致的，因此我们不能认为在网络参数更新步长为 50 的情况应用 Double DQN 优化算法对训练效率有明确的提高，但是综合网络参数更新步长为 100 的情况下的所进行的实验，我们有理由认为使用 Double DQN 算法对训练模型有正面影响。

4.2.2 实验总结

使用 Double DQN 优化算法对于使用较长网络参数更新步长的双网络结构 DQN 模型有较明显的提升；在使用较短网络参数更新步长的情况下，即使对训练效率没有明显提升，也对模型取得较大奖励值具有正面效果。综合以上结论，使用 Double DQN 优化算法对我们训练过程有明显帮助。

5 基于先验知识的启发式干预训练

5.1 基于预测帧的 Q 值干预

5.1.1 实验测试

上文提到的实验是关于整个通用模型的结构与训练算法调整的实验。现在我们来观察加入干预机制后的实验情况。

我们的设想是，如果 Q 估计值的评估动作质量的作用具有全局性，或者是在一定的回合内具有部分全局性，那么我们使用未来 Q 估计值改进当前 Q 估计值，也许能够辅助代理模型游玩游戏。

为了控制训练中的变量影响，准确确定我们的干预机制是否生效。我们对于加入此规则的模型统一设置原始的双网络结构，不采用 Double DQN 优化算法。

第一组对照实验：

（1） 单网络结构。作为基准线，简称 α 模型。图中以绿色线表示。

（2） 双网络结构，参数更新步长为 100，全程加入为未来 Q 估计值干预，简称 P_1 模型。图中以橘色线表示。

每五万步平均 Q 估计值如图：

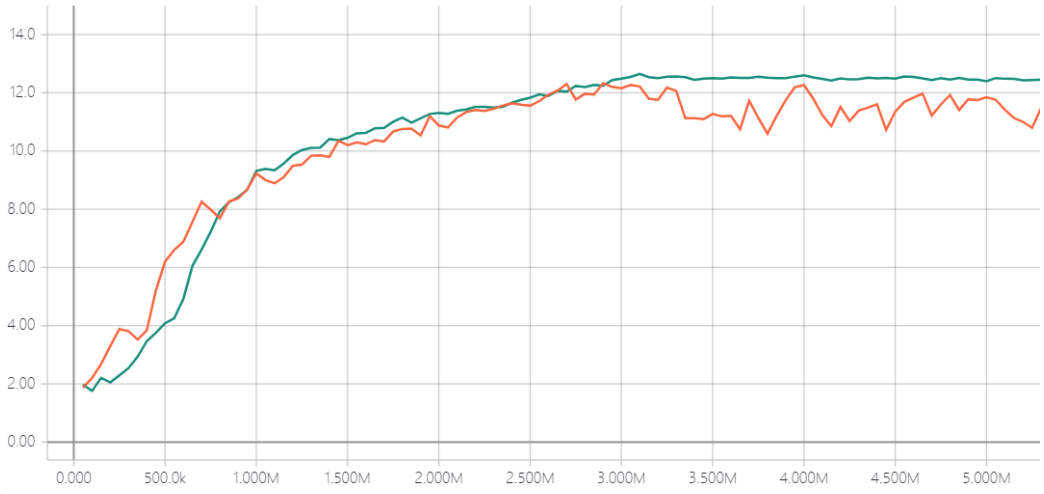


图 34 橘色 P_1 模型，绿色α 模型

我们可以看到，虽然在较早期 P_1 模型的平均 Q 估计值增长速度较快，但是在 300 万训练步长之后 P_1 模型的平均 Q 估计值出现了波动的情况，并没有保持较为稳定的状态。

累计奖励值如下图：

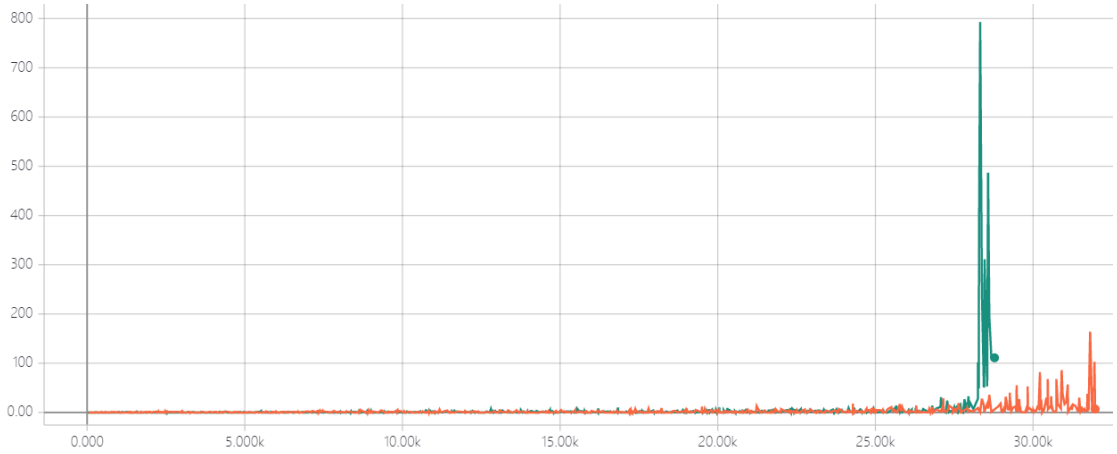


图 35 橘色 P_1 模型，绿色 α 模型

从图中我们可以看到，P_1 模型的训练效率是不如 α 模型的，但是从之前的实验评估中我们的得出的结论是双网络结构模型的训练效率是要明显优于单网络结构的，也就是说，这里加入的新机制对训练起到了负面效果。

与此同时，我们也在思考，早期的神经网络的参数较为随机化，计算得出的 Q 估计值对动作质量的评估可能不存在太大的意义，如果过早加入基于未来 Q 估计值的干预机制可能会对训练造成负面效果。因此我们添加了两组新的实验，总体实验组为：

（1） 单网络结构。作为基准线，简称 α 模型。图中以棕色线表示。

（2） 双网络结构，参数更新步长为 100，全程加入为未来 Q 估计值干预，简称 P_1 模型。图中以橘色线表示。

（3） 双网络结构，参数更新步长为 100，100 万以后加入为未来 Q 估计值干预，简称 P_2 模型。图中以天蓝色线表示。

（4） 双网络结构，参数更新步长为 100，200 万以后加入为未来 Q 估计值干预，简称 P_3 模型。图中以粉红色线表示。

300 万训练步长以后模型已经趋于收敛了，所以往后更长的训练步长我们就不设置实验了。

每五万步平均 Q 估计值如图：

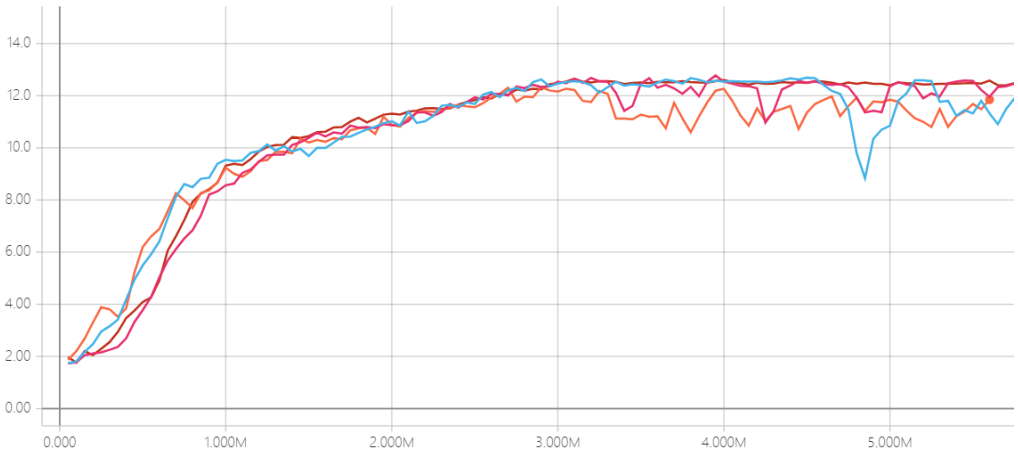


图 36 棕色为 α 模型，橘色为 P_1 模型，天蓝色为 P_2 模型，粉红色为 P_3 模型

由图可以看出，P_2 模型与 P_1 模型在前期较 α 模型的平均 Q 估计值增长速度较快，而 P_3 模型较 α 模型的平均 Q 估计值增长速度要稍慢。但是所有加入基于未来 Q 估计值干预机制的双网络模型的平均 Q 估计值在 300 万训练步长以后都出现了震荡的情况，无法保持稳定的状态。

累计奖励值如图：

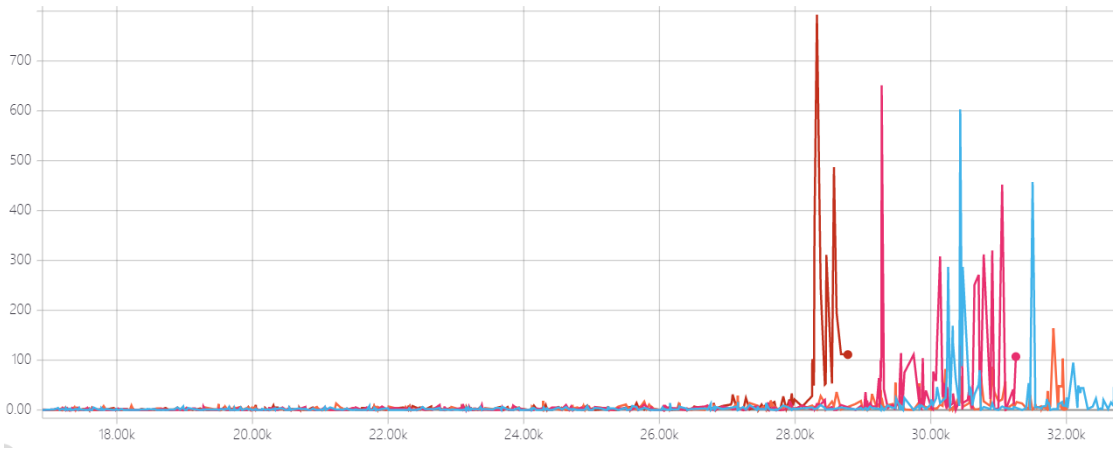


图 37 棕色为 α 模型，橘色为 P_1 模型，天蓝色为 P_2 模型，粉红色为 P_3 模型

根据图可知，在所有加入干预机制的双网络模型中， P_3 模型的训练效率是最高的，而 P_3 模型加入干预机制的时机是最晚的。所有双网络模型的训练效率与加入干预机制的训练步长的时机相关，加入得越早，则训练效率越低，并且均低于我们作为基准线的单网络模型的训练效率。

5.1.2 实验总结

我们使用未来 Q 估计值加入当前 Q 估计值进行 Q 值干预的实验结果并没有如预期中提升模型的训练效率。使用基于预测帧的 Q 估计值干预决策似乎不是一种可行的干预方式，但是我们仍然需要对“Q 值评估动作质量具有一定的全局性”这一设想进行进一步的确认，也就带来了下面的实验。

5.2 基于预测帧的 Q 值决策

5.2.1 实验测试

通过上文我们可以得知，通过预测帧的 Q 估计值加入到当前的状态的 Q 估计值这干预机制对训练并没有产生预想中的正面影响，甚至还降低了训练模型收敛的速度。那么我们之前的关于 Q 估计值评估动作质量具有一定的全局性的设想是否正确呢？我们通过采取更加激进的干预规则来验证我们的设想。简单来说，我们不再通过预测帧的 Q 估计值加入到当前状态的 Q 估计值，而是直接通过预测帧的最大 Q 估计值来选取动作。

我们希望神经网络经过一定的训练之后再使用这个决策系统，因为在我们的设想中，基于预测帧的 Q 值有评估动作质量的意义是建立在有一定训练度的神经网络的基础上的。所以我们先从 100 万训练步长以后加入这个决策系统。

平均 Q 估计值图像：

图 39 训练 100 万后加入预测决策系统

这是非常出乎我们意料之外的情况。在 100 万的训练步长的时候，也是我们加入这个干预机制的时候，原本处于上升趋势的平均 Q 估计值出现快速下降，之后一直保持一种不规律的震荡趋势，无法像正常收敛模型那样保持一种稳定值域。

累计奖励值如图：

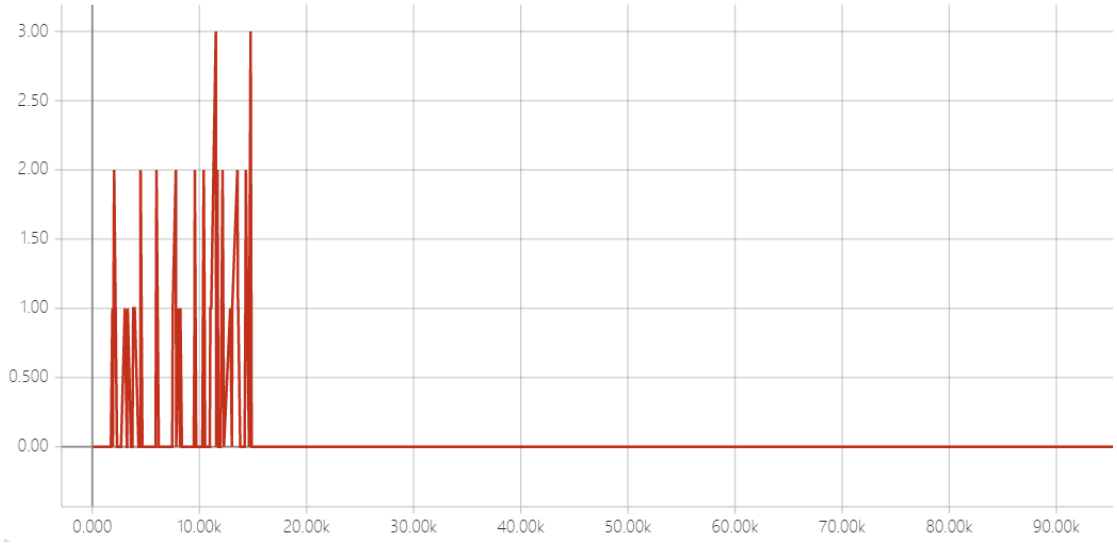


图 40 训练 100 万后加入预测决策系统

在我们加入预测决策系统之后，原本在早期训练过程中还能通过几根柱子的游戏表现也迅速下降，甚至在之后小鸟连一根柱子都不能通过了。几乎可以说，这个决策系统对训练模型的游戏表现造成了毁灭性影响。

损失函数如图：

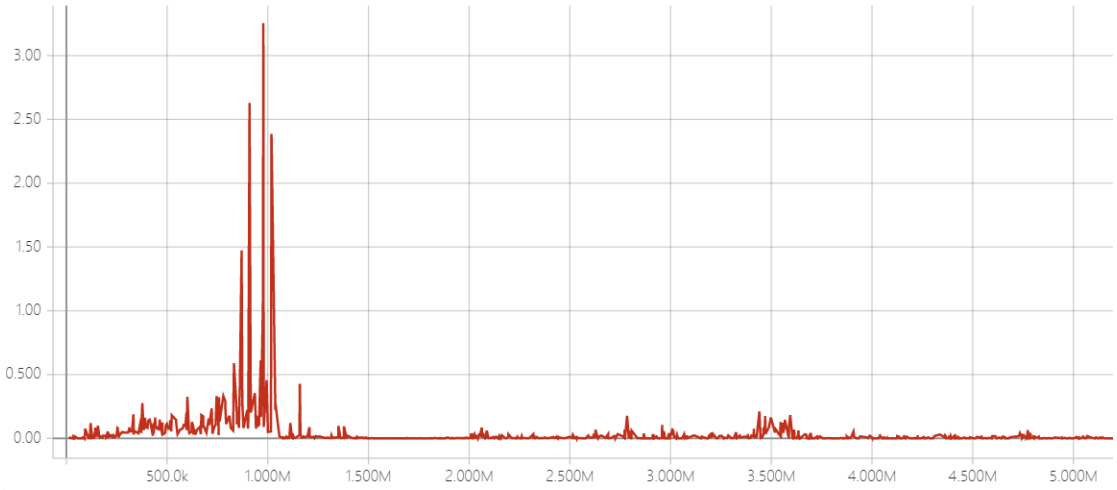


图 41 训练 100 万后加入预测决策系统

在之前我们提到损失函数只能作为辅助观察模型收敛的表征，而不能作为直接推断模型收敛的证据。这里图像也是一个很好的例证。在我们加入预测决策系统后，我们可以看到模型的游戏表现完全失败了，然而损失函数却呈现快速收敛的趋势。这也是从反面证明了损失函数的收敛不能作为说明模型收敛的直接证据。

作为补充实验，我们在 200 万、300 万的训练步长后也加入同样的预测决策机制。并且按照正常的训练速度，300 万之后训练模型的游戏表现已经开始收敛了。

平均 Q 估计值图像：

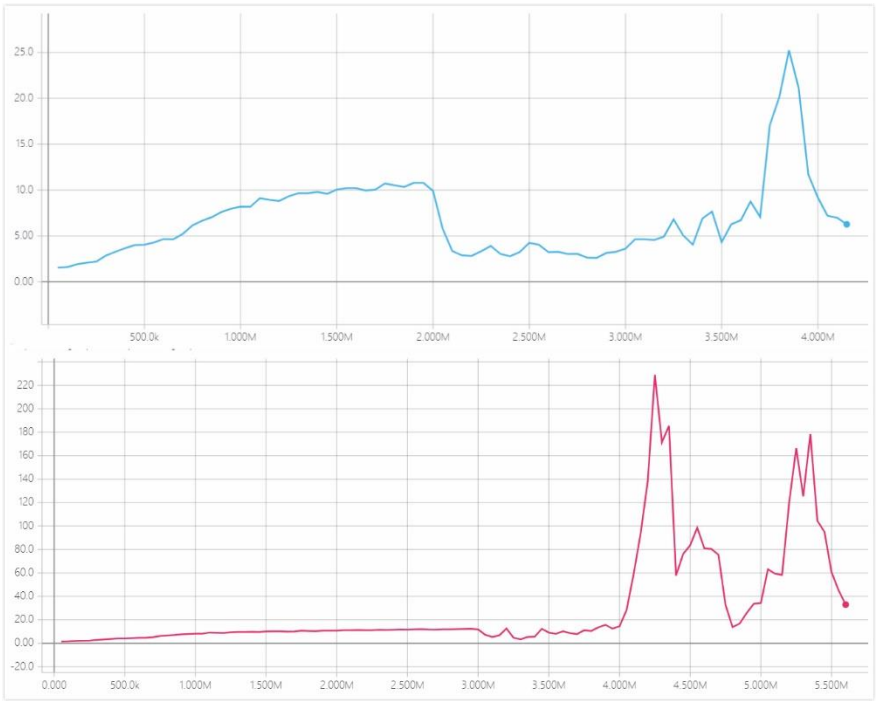


图 42 上图 200 万后加入机制，下图 300 万后加入机制

累计奖励值如图：

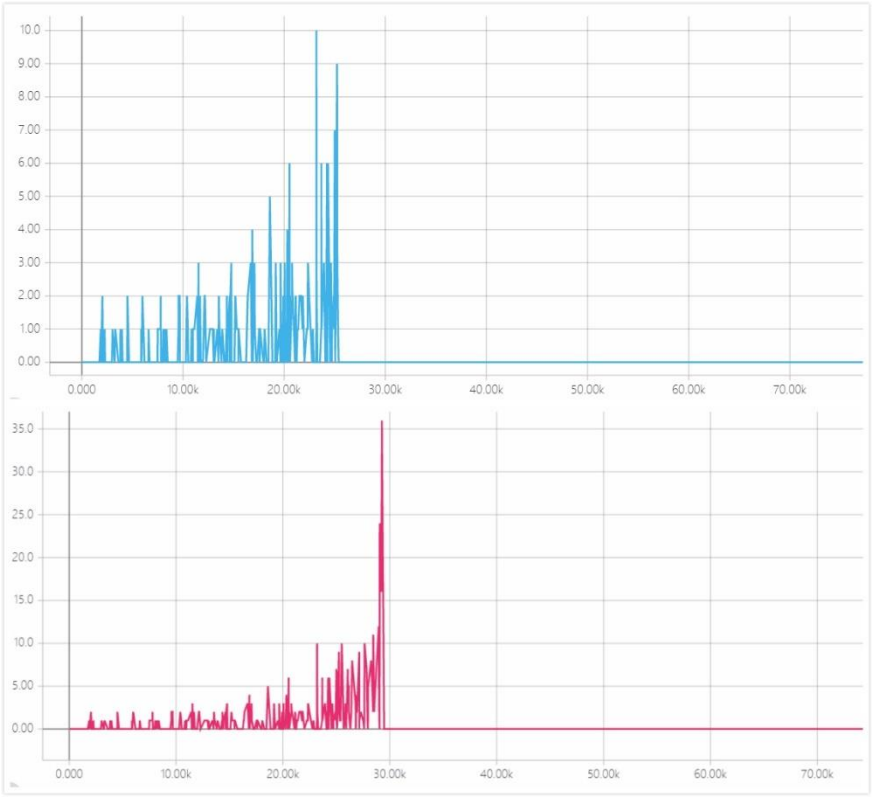


图 43 上图 200 万加入机制，下图 300 万加入机制

损失函数如图：

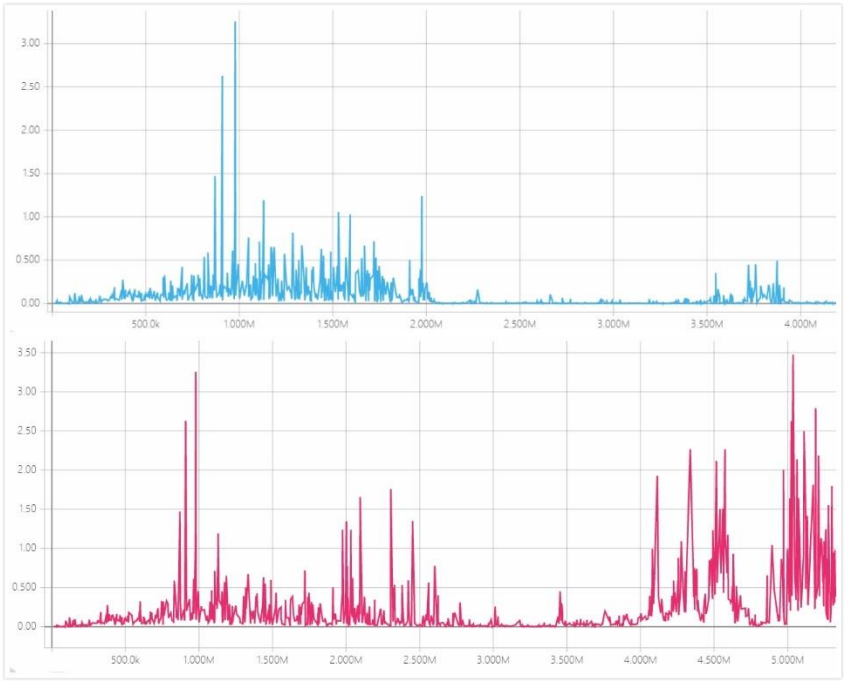


图 44 上图 200 万后加入机制，下图 300 万后加入机制

可以看到，我们所得到的实验结果与 100 万后加入预测决策机制的结果是类似的。甚至是在模型趋于收敛后加入预测决策机制，也会对模型的游戏表现造成毁灭性影响。因此担心预测决策效果不佳是因为神经网络还未达到一定的训练度的设想应该是不正确的。

5.2.2 实验总结

基于预测帧的 Q 估计值加入当前 Q 估计值机制以及基于预测帧的 Q 估计值决策都是一种基于神经网络的超前调节的思想设计的干预规则，但是都没有取得好的效果，甚至还会像本次实验一样完全破坏模型的游戏表现。不成功的原因也许是因为 Q 估计值评估动作质量这一性质并不具有全局性，但是我们没有办法从数学上完全确认这一点。DQN 模型说到底还是一种通过每回合迭代进行试错并调整网络参数的算法模型，是一种着重于当前状态的强化学习模型。如果用已经进行高度抽象化的未来 Q 估计值来干预现在状态的动作选择，可能会带来极大的未知风险。基于实验结果，基于预测帧最大 Q 估计值进行动作决策来提高模型训练效率是一种不可行的干预手段。

5.3 基于图像识别的动作决策

5.3.1 实验测试

前两种干预机制取得的效果并不理想。Deep mind 团队曾提到过，Q 值的微小变化可能对于长期的行为策略有着重大影响，这也许是前两个实验加入的干预规则难以取得成功的潜在原因之一。因此我们希望使用更为直观的干预机制来查看干预效果。

决策机制的触发借鉴了选择动作的 $\varepsilon-$greedy 机制，触发概率初始设定为 0.1，然后在两百万步之内上升到 1。也就是按照一定的递增概率来触发这个基于图像识别的动作决策机制。

依照之前的实验结果，我们选取之前的最优超参数条件以及优化算法。

设置的第一组实验模型为：

（1） 单网络模型，简称 s 模型，图中以绿色线表示。

（2） 双网络模型，网络参数 50 步更替，使用 Double DQN 优化算法，$\varepsilon-$greedy 的使用随机动作的概率值在 300 万内从 0.1 衰减至 0.0001，加入按递增概率触发的基于图像识别的动作决策机制。简称 α_p 模型。图中以深蓝色线显示。

另外再加入一组横向对照组：

（3） 双网络模型，目标网络参数更新步长为 50，使用 Double DQN 优化算法，$\varepsilon-$greedy 的使用随机动作的概率值在 300 万内从 0.1 衰减至 0.0001。简称为 α_1 模型。图中为粉红色线显示。

每五万步的平均 Q 估计值如图：

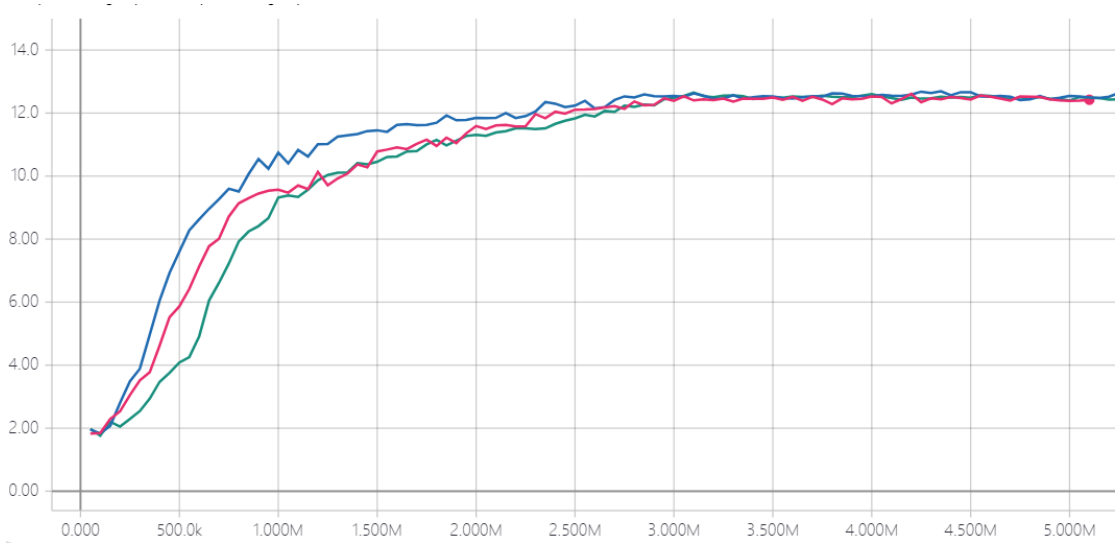


图 46 深蓝色 α_p 模型，粉红色 α_1 模型，绿色 s 模型

我们可以看到，全部模型几乎都在 300 万达到了稳定的平均 Q 估计值。但是在整个 Q 值上升阶段，α_p 模型，即使用了基于图像识别的动作决策的模型的平均 Q 估计值增长速度是最快的。

累计奖励值如图：

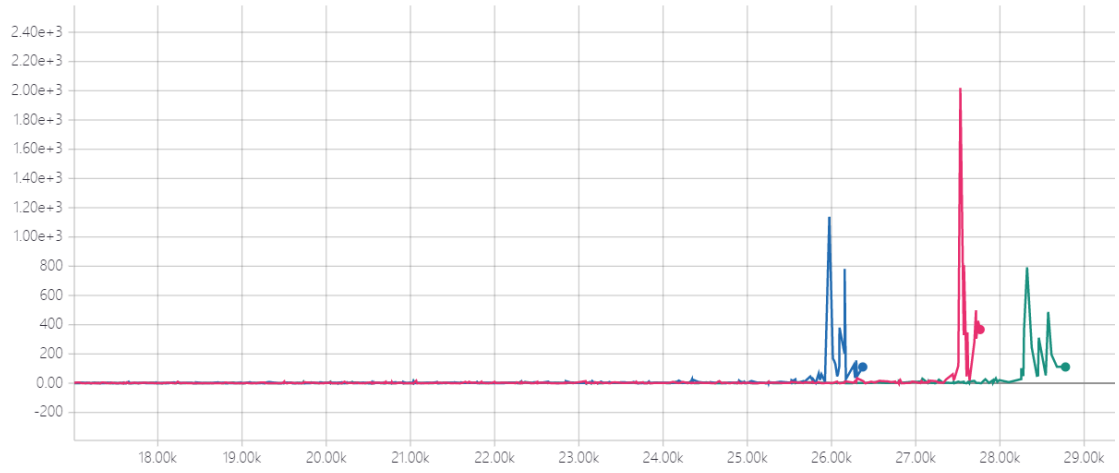


图 47 深蓝色 α_p 模型，粉红色 α_1 模型，绿色 s 模型

通过累计奖励值我们也可看出，α_p 模型达到累计奖励值波峰使用的回合数是最少的，体现了明显的效率提升。

5.3.2 实验总结

从实验结果可以得知，按递增概率触发的基于图像识别的动作决策机制取得了明显正面效果。与以往的干预机制不同，我们使用的是当前状态的信息，针对的是当前状态的动作决策，而不使用来着未来的状态信息进行干预，从而取得了训练效率的较大提升。

5.4 降低随机动作频率

5.4.1 实验原理与测试

这是我们做完以上实验后发现的补充实验。即使是在 $\varepsilon-greedy$ 策略的 ε 初始值为 0.1，并在 300 万步之内衰减到至 0.0001 的情况下，提高 ε 的衰减速度，也就是进一步降低模型选取随机动作的概率，让模型更多依照神经网络输出的 Q 估计值选择动作，能有效提高模型训练效率。

我们把 ε 的初始值设置 0.1，然后在 200 万步内置衰减至 0.0001。

第一组对照实验：

（1） 双网络结构，参数更新步长为 50。图中为灰色线。简称 D_1 模型。

（2） 双网络结构，参数更新步长为 50，依照上述说明提高 ε 的衰减速度。图中为绿色线。简称 D_2 模型。

每五万步平均 Q 估计值如图：

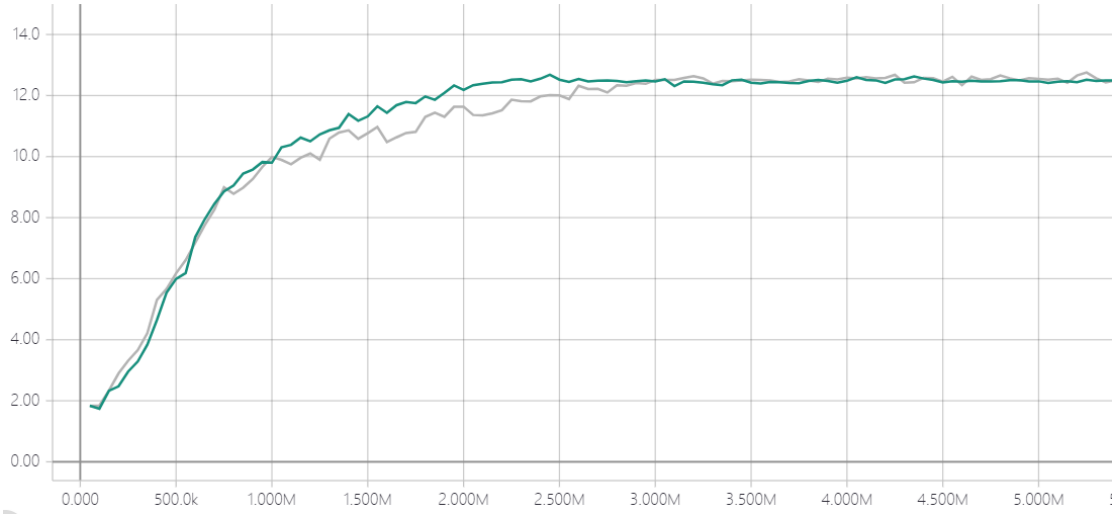


图 48 灰色 D_1 模型，绿色 D_2 模型

我们可以看到，在前 100 万训练步长中两个模型的平均 Q 估计值增长速度几乎是一致的，而提高了 ε 的衰减速度的 D_2 模型的平均 Q 估计值在 100 万与 300 万训练步长之间增长速度要更快一些，也更早达到稳定值。

累计奖励值如图：

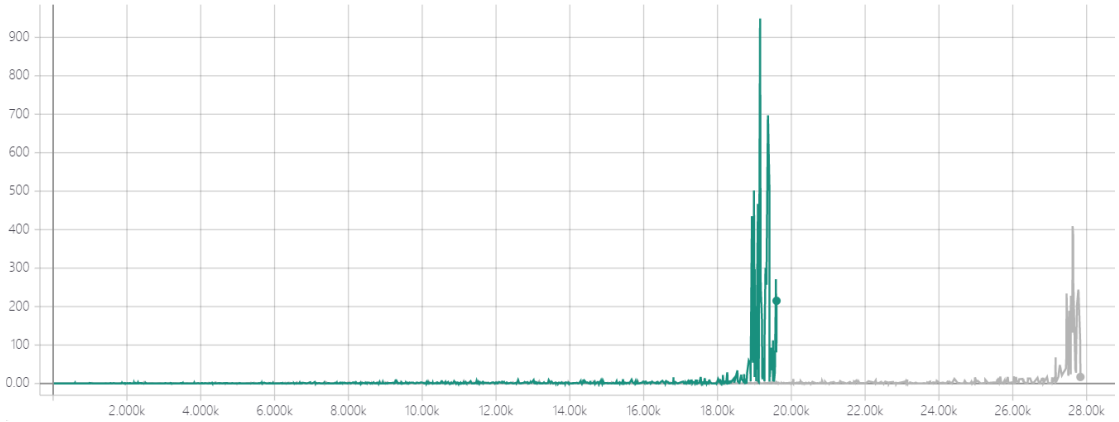


图 49 灰色 D_1 模型，绿色 D_2 模型

累计奖励值的数据进一步体现了提高了 ε 的衰减速度的 D_2 模型的训练效率较 D_1 模型有了非常明显的提升。

第二组对照实验：

（1） 双网络结构，参数更新步长为 50，使用 Double DQN 优化算法。图中为粉红色线。简称 D_3 模型。

（2） 双网络结构，参数更新步长为 50，使用 Double DQN 优化算法，依照上述说明提高 ε 的衰减速度。图中为橘色线。简称 D_4 模型。

每五万步平均 Q 估计值如图：

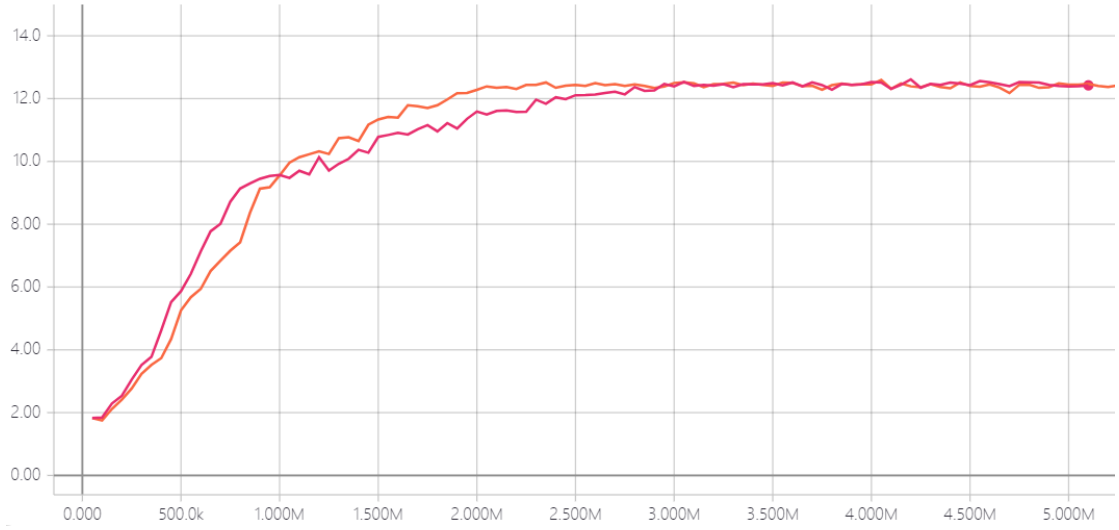


图 50 粉红色 D_3 模型，橘色 D_4 模型

可以看到提高 ε 的衰减速度的 D_4 模型在前 100 万训练步长的平均 Q 估计值的增长速度不如使用正常 ε 的衰减速度的 D_3 模型，但是在 100 万与 300 之间的训练步长中完成了反超，并更早达到稳定状态值。

累计奖励值如图：

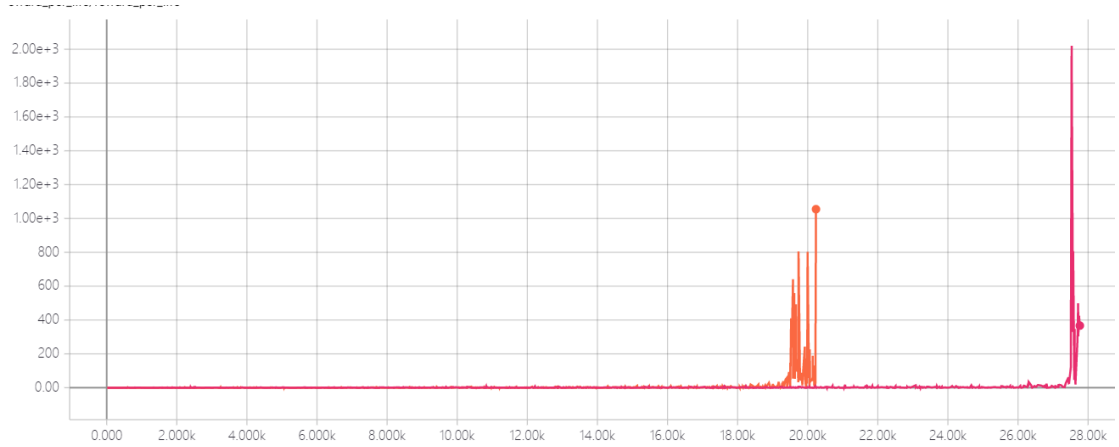


图 51 粉红色 D_3 模型，橘色 D_4 模型

与上一组对照实验类似，提高 ε 的衰减速度有助于明显提高模型的训练效率。

第三组对照实验：

（1）双网络结构，参数更新步长为 50，使用 Double DQN 优化算法，加入依照递增概率触发的基于图像识别的动作决策。图中为橘色线。简称 D_5 模型。

（2）双网络结构，参数更新步长为 50，使用 Double DQN 优化算法，加入依照递增概率触发的基于图像识别的动作决策，依照上述说明提高 ε 的衰减速度。图中为天蓝色线。简称 D_6 模型。

每五万步平均 Q 估计值如图：

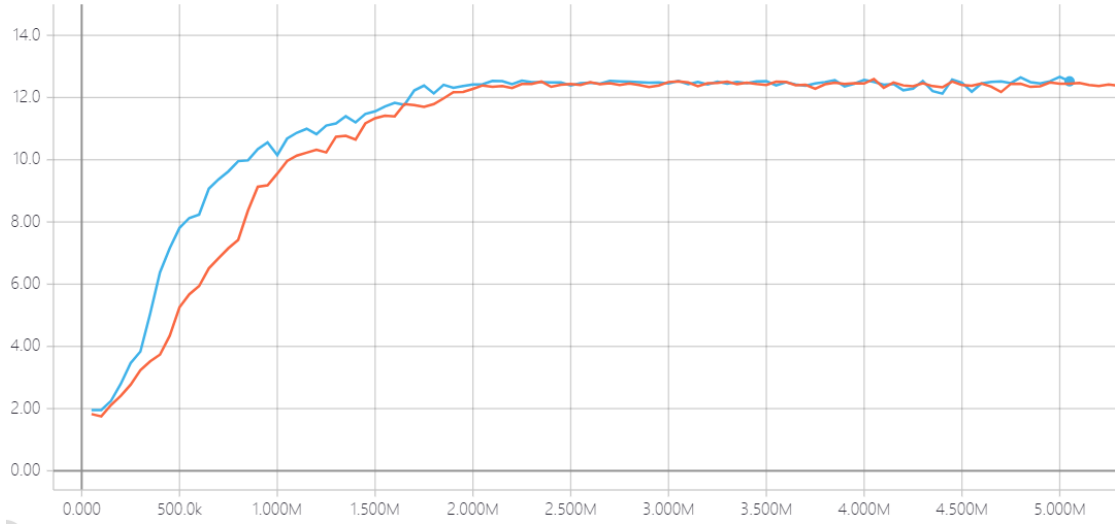


图 52 天蓝色 D_6 模型，橘色 D_5 模型

从图可以得知，提高 ε 的衰减速度 D_6 模型的平均 Q 估计值在训练阶段的增长速度一直大于 D_5 模型

累计奖励值如图：

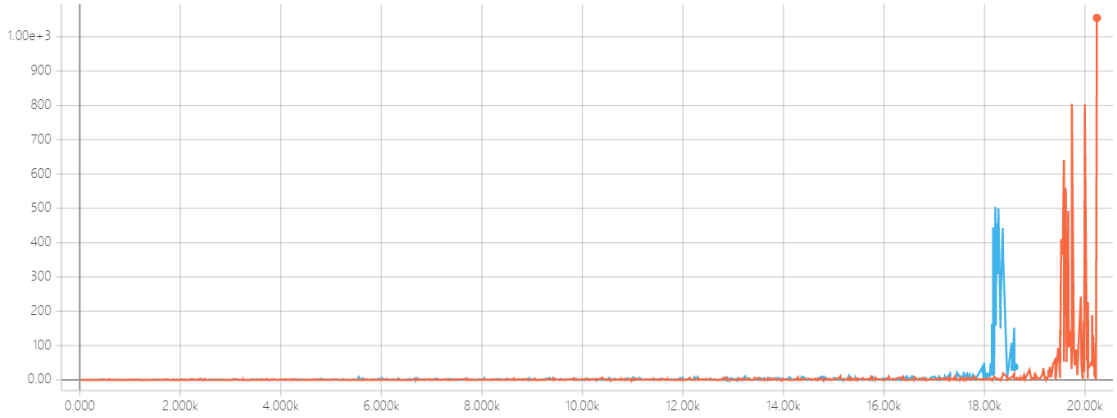


图 53 天蓝色 D_6 模型，橘色 D_5 模型

从图可以看出，提高 ε 的衰减速度 D_6 模型的训练效率较于 D_5 模型也存在着一定程度的提升。

5.4.2 实验总结

从实验结果可以得知，提高 ε 的衰减速度对于提升训练模型的训练效率有相当明显的提升。对于 DQN 模型来说，需要大量的探索环境的回合以不断调整神经网络参数，所以我们的训练回合都是以百万位单位计数。但是这次实验结果表明，DQN 模型的训练回合并不是越多越好，过多的训练回合可能会造成探索冗余的状况发生。适度缩减训练回合与使用随机动作的频率反而有助于提升模型的训练效率。

6 总结与展望

6.1 工作总结与实验结论

本次毕业设计项目首先通过双层网络结构 DQN 的实现以及对 Double DQN 优化训练算法的使用，对单网络结构的 DQN 的训练效率进行初步提升。然后对双网络模型的最优超参数进行测试确定，对双网络模型自身的训练效率进行二次提升。最后基于先验知识，对本次特定的游戏环境进行规制设定，以探索进一步提高训练效率的可能性，虽然其中两次尝试都不成功，但是最后还是确立了一种有效的基于图像识别的干预机制，这种机制较为简单直观。也进一步说明对于 DQN 模型的改进，不太适合对其中的基础步骤进行过大修改，否则难以得到较为良好的结果。应该以每步为中心，优化当前状态，加入尽可能直观的干预机制，会比较容易取得明显的正面效果。

最后是实验的最终奖励值累积图：

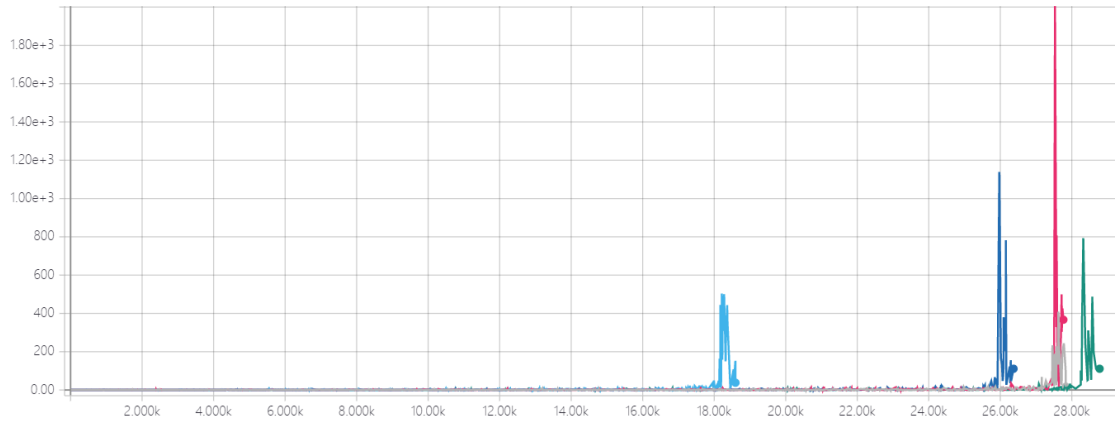


图 54 模型效率递增展示图

绿色线为单网络模型；

灰色线为双网络模型，参数更新步长为 50，ε 的衰减速度为 300 万内从 0.1 衰减至 0.0001；

粉红色线为双网络模型，参数更新步长为 50，使用 Double DQN 优化算法，ε 的衰减速度为 300 万内从 0.1 衰减至 0.0001；

深蓝色线为双网络模型，参数更新步长为 50，使用 Double DQN 优化算法，ε 的衰减速度为 300 万内从 0.1 衰减至 0.0001，使用概率递增触发的基于图像识别的动作决策；

天蓝色线为双网络模型，参数更新步长为 50，使用 Double DQN 优化算法，ε 的衰减速度为 200 万内从 0.1 衰减至 0.0001，使用概率递增触发的基于图像识别的动作决策；

天蓝色线代表的训练模型为本次实验项目的最佳训练模型。

6.2 不足与未来的改进

本次毕业设计项目虽然进行了多组实验以及多种机制的测试，依然存在很多不足之处，也存在很多可以进一步改进的地方。强化学习与游戏 AI 是非常具有深度的课题，本次毕业设计项目仅针对 Flappy Bird 游戏环境，尝试通过各种方法提升在这一特定环境下的模型训练效率。虽然本次毕业设计的总体实验经验以及干预规则制定的思路对我们日后更深一步的研究依然有帮助，但是对于大的游戏 AI 这一命题而言还是缺乏一定的泛用性，针对不同的游戏环境，还是需要对各种方法和机制进行针对性的调整 。

本次毕业设计中进行的实验均进行了随机值种子的控制，为的是保持可复现性，方便我们进行数据图像分析，以及在多组实验中保持较高的可比较性，但是我们还是应当尝试更多的随机种子数，更深一步确认机制与方法的在本次游戏环境中的稳定性，但是由于训练要求的巨额时间成本，没能进行更多的测试，这是未来我们进行更深一步探索的时候需要注意到的问题。

此外，Deep mind 团队提出的 DQN 强化学习框架近年来其实也一直在不断的发展。在 Double DQN 之后，针对神经网络的随机采样算法又进行了进一步的优化，优化过后的算法被称为有优先级的经验回放机制（Prioritized Experience Replay）[10]，主要思想是针对经验池里的训练样本，进行优先级的划分，优先训练那些价值高的样本，以提高模型的训练效果。受于时间限制于工作量限制，我们在本次毕设实验项目也未能进行进一步的尝试，有待我们日后深入探究进行尝试。

【参考文献】

[1] Watkins, C.J.C.H. Learning from Delayed Rewards (Ph.D. thesis)[D]. Cambridge University (1989)

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves,and Ioannis Antonoglou.Playing Atari with Deep Reinforcement Learning[C]. NIPS Deep Learning Workshop (2013)

[3] Kaelbling, Leslie P, Littman, Michael L, Moore, Andrew W. Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research[J](1996).

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis. Human-level control through deep reinforcement learning[J]. Nature 518,529-533(26 February 2015)

[5] Hado van Hasselt, Arthur Guez, David Silver. Deep Reinforcement Learning with Double Q-learning[C]. AAAI(2016)

[6] Sutton Richard S, Barto Andrew G. Reinforcement Learning: An Introduction (Second ed.)[M] (A Bradford Book.MIT Press, 2018).

[7] Melo,Francisco S."Convergence of Q-learning: a simple proof"[R].

[8] Matiisen,Tambet."Demystifying Deep Reinforcement Learning"[M].Computational Neuroscience Lab.(December 19, 2015)

[9] Kevin Chen. Deep Reinforcement Learning for Flappy Bird[R](2018)

[10] Tom Schaul, John Quan, Ioannis Antonoglou, David Silver.Prioritized Experience Replay[C].ICLR(2016)

致谢

首先由衷地感谢本次毕业设计我的指导老师张昊迪，在整个大四的学习生活中，给予了我很多的帮助和指导。从开题报告，到论文撰写，都耐心地给予我许多反馈意见，并教导我如何修改不足之处。在整个毕业设计的实验项目过程中，也给出了许多宝贵的建议，并提出了很多解决问题的新思路，帮助我解决了许多难题，使得项目充实而有意义。张老师渊博的学识，认真负责的治学态度使我受益匪浅。在此向张老师表示崇高的敬意和衷心的感谢。

感谢郭铠滨同学。作为一同在张老师带领下完成毕业设计的同学，我在同郭同学的沟通和交流中学习到了许多宝贵的知识，对毕业设计项目的完成非常有帮助。在此对郭铠滨同学表示衷心的感谢。

感谢四年以来老师们的辛勤授课，丰富了我们的知识，拓宽了我们的视野，提高了我们发现问题、解决问题的能力，使我的思想产生了质的飞跃。

感谢开源社区，提供了很多开放源码与类库，为我提供了庞大的优秀代码资源，使我在模型开发过程中得到很多启发。

感谢一直关心我、支持我的父母和我的朋友廖俊宇、朱欣、蒲靖怡、张学为、李立宜、郑国宇、潘天惠和欧凌修。是你们的陪伴和支持，让我顺利走到今天，并走向另外一个新的开始。

Implementation and Improvement of Game AI Based on Double Layer Network Structure DQN

【**Abstract**】As an important application field of artificial intelligence research, Game AI provides a good application and testing platform for various algorithms and model research. Compared with the real world, the game environment is relatively simple, the conditions are controllable, and the realistic simulation requirements are met to some extent. Therefore, in recent years, many research teams represented by DeepMind and OpenAI have carried out many pioneering explorations and researches in the field of game AI. The current main methods are data-driven machine learning related methods, such as deep reinforcement learning. A small number of methods are based on expert rule descriptions and have achieved good performance in some scenarios. This paper combines deep reinforcement learning and rule description and other methods, based on Python open source game "Flappy Bird" as a test platform to study the impact of deep neural network structure, hyperparameters and rule judgment on learning efficiency.

【**keywords**】 Machine learning；Data-driven；Deep Reinforcement learning；Rule description

指导教师：张昊迪