

IM&IS应用实践

深度强化学习解2048

设计题目

沈凯丽

学生姓名

2014213442

学 号

信息管理与信息系统14-02

专业班级

胡笑旋

指导教师

**2018**年 **1**月 **11** 日

深度强化学习解2048

**摘要：**AlphaGo打败世界顶尖棋手的李世石九段，人们深刻的感受到了人工智能的巨大潜力，学术界和民间由此掀起了人工智能的热潮。人工智能是让机器具有人的思维与环境交互实现目标，而游戏就是人与环境交互的一种模型化表示，所以很多人工智能的算法都会先实践于游戏场景，本文实验基于2048小游戏，采用深度强化学习DQN算法实现机器自动完成2048游戏目标。

本文首先阐述了DQN深度强化学习解2048游戏的相关理论。首先通过马尔科夫决策对问题的建模，基于此问题具有强化学习“免模式（model-Free）学习”和延迟回报的特征，选择强化学习Q-learning去求解。Q-learning决策过程的行为值函数（Q表格）进行迭代计算，针对Q-learning用Q表格记录值在面对高数量级的状态空间和动作空间时也会变得非常巨大而不可能完全遍历的问题，我们采用神经网络来计算Q-value的值，将Q表格的迭代更新转化为神经网络参数的更新，由此得到DQN（Deep Q-NetWork）算法。

接着介绍了该试验模型的建立。建模的关键是状态空间、动作空间、奖惩函数的设定，状态空间采用直接输入、人工提取特征（归一处理和未归一处理两种）和通过卷积神经网络（CNN）提取特征三种方法，共建立四个有代表意义的模型，进行迭代测试。模型是实现是基于python语言和tensorflow机器学习框架，DQN网络的搭建部分详细阐述了如何通过创建网络、记忆存储、行为选择、学习训练等方法进行实现。最后是实验结果分析，最终可能由于奖惩函数设置不合理，迭代次数不够、参数设置不合理等各种原因模型未能收敛，所以提出了新的奖惩函数的二代模型，时间原因还没能训练完。文末，也提出自己的反思和和未完成的调参计划。

**关键字**：DQN，Q-learning，强化学习，神经网络

目 录

[1、研究背景及意义 1](#_Toc504138187)

[2、DQN深度强化学习相关理论 2](#_Toc504138188)

[2.1马尔科夫决策与强化学习 2](#_Toc504138189)

[2.2 Q-learning 4](#_Toc504138190)

[2.3深度神经网络（DNN）及卷积神经网络（RNN） 5](#_Toc504138191)

[2.3.1神经网络结构 5](#_Toc504138192)

[2.3.2神经网络的训练 6](#_Toc504138193)

[2.3.3卷积神经网络（CNN） 7](#_Toc504138194)

[2.4深度强化学习（DRL）及DQN 8](#_Toc504138195)

[2.4.1深度神经网络思想 8](#_Toc504138196)

[2.4.2深度神经网络对Q-learning改进 9](#_Toc504138197)

[3、模型构建 10](#_Toc504138198)

[3.1环境搭建 10](#_Toc504138199)

[3.2模块设计 12](#_Toc504138200)

[3.3算法伪代码 14](#_Toc504138201)

[3.4Deep Q-Network搭建 15](#_Toc504138202)

[3.4.1初始化 16](#_Toc504138203)

[3.4.2创建神经网络 16](#_Toc504138204)

[3.4.3记忆存储 18](#_Toc504138205)

[3.4.4选择行为 18](#_Toc504138206)

[3.4.5学习 19](#_Toc504138207)

[4、实验环境及结果分析 20](#_Toc504138208)

[4.1实验环境 20](#_Toc504138209)

[4.1.1软件环境 20](#_Toc504138210)

[4.1.2硬件环境 20](#_Toc504138211)

[4.2可视化DQN结构 20](#_Toc504138212)

[4.3训练结果评估 22](#_Toc504138213)

[4.3.1第一代实验 22](#_Toc504138214)

[4.3.1第二代实验 24](#_Toc504138215)

[4.3.2调参： 25](#_Toc504138216)

[4.4实验反思和展望 26](#_Toc504138217)

[5、参考资料 27](#_Toc504138218)

# 1、研究背景及意义

2016年，DeepMind公司研发的计算机围棋程序AlphaGo成功地打败了近15 年来一直被认为是世界顶尖棋手的李世石九段，这距IBM的深蓝(Deep Blue)程序击败国际象棋棋王卡斯帕罗夫正好二十年，也再一次在学术界和民间掀起了人工智能的热潮。

人工智能是研究使计算机来模拟人的某些思维过程和智能行为（如学习、推理、思考、规划等）的学科。借助人工智能新技术实现自动化，将极大提高生产率，节省劳动成本，促进社会的繁荣与发展。Google的人工智能走在最前沿，但是国内各大高校、科研机构、企业也在抓紧研究，研究其理论、算法与应用，并不断从实验室走向现实。

AlphaGo的成功极大程度上归功于其采用了深度强化学习的算法。深度增强学习Deep Reinforcement Learning是将深度学习与增强学习结合起来从而实现从Perception感知到Action动作的端对端学习的一种全新的算法。简单的说，就是和人类一样，输入感知信息比如视觉，然后通过深度神经网络，直接输出动作，中间没有人工手动的工作。深度增强学习具备使机器人实现完全自主的学习一种甚至多种技能的潜力。

本实验所研究的是通过深度强化算法让机器自己玩2048的游戏。本质上，游戏就是人与环境交互的一种模型化表示，而且相对于社会环境，游戏具有非常明确的动作集合，状态转移规则、目标以及奖惩，反馈十分明确，因此以游戏作为解决问题，为研究方法是测试提供了理想化环境。2048的游戏除了具有明确的规则的反馈，每次移动又会有新的数字方块产生的不确定性，并且每次动作的选择对于最终结果是否有影响，会有多大影响，都是不确定的，而面对这样的不确定性，DQN能否像人一样找到合适的策略进行动作选择呢？

在本实验研究中，有一定规则和不确定的游戏环境实际上也是不确定的现实环境的映射，通过对于解2048游戏的探索，对深度强化学习方法的学习和探索有更加深刻的理解，为将深度强化学习的方法运用于解决其他现实问题奠定了基础。

# 2、DQN深度强化学习相关理论

## 2.1马尔科夫决策与强化学习

在人工智能领域，一般用agent来表示一个具备行为能力的物体，agent与环境发生交互，举玩游戏的例子，屏幕上游戏界面的情况就是环境，我们看到的环境是观测值（observation，因为并不是环境的所有情况都能被观测到，agent得到的只是它的观测值），然后我们根据观测值输出动作（鼠标或键盘操作）改变环境的状态，同时环境会给我们一个得分的反馈reward，反应环境变化的好坏，也可以作为这个动作的回报。

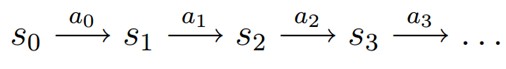


**图2-1 智能体agent与环境的交互图**

agent和环境的相互作用可以建模为马尔可夫决策问题，一个马尔可夫决策过程由一个五元组构成

* : 表示状态集(states)，有，表示第i步的状态。
* :表示一组动作(actions)，有，表示第i步的动作。
* 𝑃sa: 表示状态转移概率。𝑃s𝑎 表示的是在当前状态下，经过作用后，会转移到的其他状态的概率分布情况。比如，在状态下执行动作，转移到的概率可以表示为。
* ，是回报函数(reward function)。有些回报函数状态的函数，可以简化为。如果一组转移到了下个状态，那么回报函数可记为。如果对应的下个状态是唯一的，那么回报函数也可以记为。
* :折扣率，表示未来和现在的奖励的对agent的重要程度

MDP 的动态过程如下：某个智能体(agent)的初始状态为，然后从 A 中挑选一个动作执行，执行后，agent 按概率随机转移到了下一个状态，。然后再执行一个动作，就转移到了，接下来再执行，我们可以用下面的图表示状态转移的过程。



在该state到如何选择action的过程就称之为一个策略Policy，Agent要做的是找到一个策略Policy，使得在这样的状态下通过policy选择的action，可以得到更多的reward。马尔科夫决策的优化问题一般可用动态规划（Dynamic Programming）和强化学习（Reinforcement Learning）

智能体（agent）根据当前状态来采取动作，获得相应的奖赏之后，再去改进这些动作，使得下次再到相同状态时，智能体能做出更优的动作,，这种试错迭代的过程就是强化学习。



**图2-2强化学习基本算法框架**

## 2.2 Q-learning

Q-learning学习算法由C.Watkins于1989年在其博士学论文“Learning from the delayed rewards”中首次提出,属于强化学习中无模型学习中利用时序差分学习的方法，该算法是动态规划的有关理论及蒙特卡罗思想相互结合,以求解具有延迟回报的序贯优化决策问题为目标。在学习算法中根据算法对决策过程的行为值函数（Q表格）进行迭代计算,其迭代计算公式为

**（1）**

* :更新后新的*Q*值
* : 更新前即上一时刻*Q*值
* ：记忆中的最大利益，此时状态下对应的最大值
* : 在时执行后获得的奖惩，眼前利益
* ：学习速率
* ：折扣率

根据公式可以看出，学习速率越大，保留之前训练的效果就越少。折扣因子越大，记忆中的利益（）所起到的作用就越大，agent就会越重视以往经验，γ越小，agent只重视眼前利益。

Q-learning根据值函数计算出目标Q值，但是这里并没有直接将这个Q值（是估计值）直接赋予新的Q，而是采用渐进的方式类似梯度下降，朝目标Q值迈近一小步，取决于,这就能够减少估计误差造成的影响。类似随机梯度下降，最后可以收敛到最优的Q值。

**具体的算法如下：**

首先需给定有限离散状态和行为空间即马尔科夫决策过程状态集S和行为集A，状态转规则，奖惩函数，折扣因子，以表格形式存储行为值函数估计值Q（s，a），以及策略选择（这里采用贪心算法）

初始化马尔科夫决策过程的状态，行为值函数估计Q表格  
重复（对每一节episode）:   
  初始化状态*s*   
  重复（对episode中的每一步）：   
    使用某一个policy比如（）根据状态*s*选取一个动作执行   
    执行完动作后，观察reward和新的状态

  循环直到*s*终止

## 2.3深度神经网络（DNN）及卷积神经网络（RNN）

### 2.3.1神经网络结构

神经网络是由具有适应性的简单单元组成的广泛并行互联的网络，它的组织能够模拟生物神经系统对真实世界物体所做出的交互反应。神经网络技术起源于上世纪五、六十年代，当时叫感知机（perceptron），拥有输入层、输出层和一个隐含层。输入的特征向量通过隐含层变换达到输出层，在输出层得到分类结果。感知器只能做简单的线性分类任务，全连接的多层神经网络类似于多层感知机，但是学习效果更加强大。随着神经网络的发展，其表示性能越来越强。

神经网络的每个单元如下：



**图2-3 单个神经元结构**

其对应的公式如下：

（2）

其中，该单元也可以被称作是Logistic回归模型。当将多个单元组合起来并具有分层结构时，就形成了神经网络模型，一层输入层(input layer), 一层或多层隐藏层 (hidden layers), 一层输入层 (output layers)在神经网络中前一层输出的结果经过激活函数（activation function）处理后，将作为后一层的输入。

激活函数，链接函数，把我们定义的线性函数的输出转换成我们想要的格式。激励函数运行时，激活神经网络中某一部分神经元，将激活信息向后传入下一层的神经系统。满足两个条件：一是处处可微，（才能实现误差反向传播）；二是要非线性的。



**图2-4 多层神经网络结构**

神经网络即可以用来做分类(classification）问题，也可以解决回归(regression)问题。解决的是回归问题上，由于加权求和之后以后计算的方程是非线性的，所以理论上讲有一个很强大的功能，只要训练集足够大，隐藏层足够多，可以用神经网络模拟出任何方程。

神经网络没有明确的规则来设计最好有多少个隐藏层，根据实验测试和误差，以及准确度来实验并改进。

### 2.3.2神经网络的训练

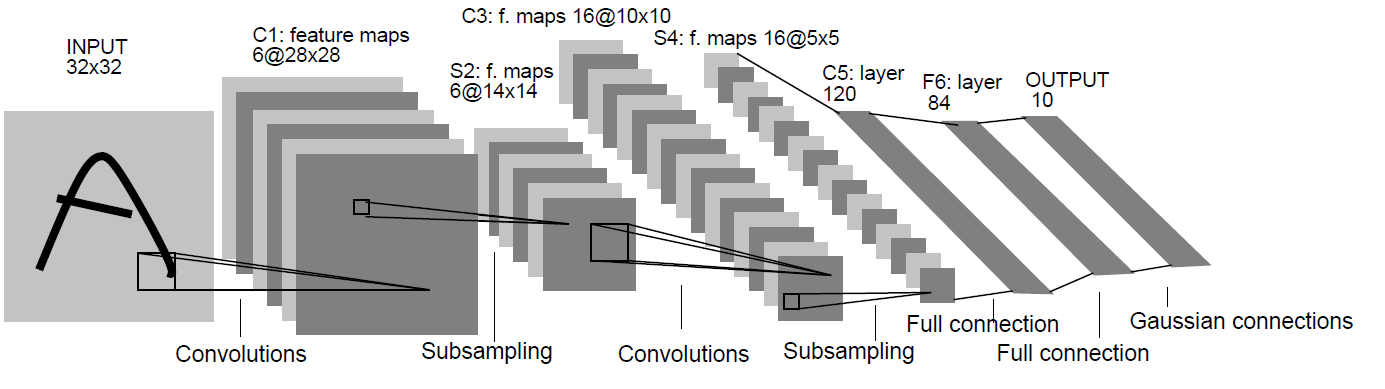
神经网络训练的目的，就是使得参数尽可能的与真实的模型逼近。具体做法是如下

1. 首先给所有参数赋上随机值。
2. 我们使用这些随机生成的参数值，来预测训练数据中的样本。
3. 计算经过神经网络后输入层预测值(predicted value)与真实值(target value)之间的方差（即损失，loss），我们的目标就是使对所有训练数据的损失和尽可能的小。
4. 解决方法是梯度下降，递归的逼近最小偏差模型，利用链式求导法则对隐含层的节点进行求导，即每次计算参数在当前的梯度，然后反方向移动（从输出层不断向前到输入层），将每个参数一点点地往使cost不断降低的方向移动。也称为误差反向传播，反方向（从输出层到隐藏层到输入层）来以最小化损失来更新每个连接的权重(weight）和误差（bias）。可用优化器（optimizers）类实现。

### 2.3.3卷积神经网络（CNN）

卷积神经网络由一个或多个卷积层和神经网络全连接层组成，可直接输入图像等原始信息，通过卷积层的特征提取和全连接层的拟合，进行分类和预测。CNN以其局部感知和权值共享的特殊结构在语音识别和图像处理方面有着独特的优越性。

通过卷积进行局部感知，也就是说神经网络不再是对每个像素的输入信息做处理了,而是图片上每一小块像素区域进行处理。因此对于CNN来说，并不是所有上下层神经元都能直接相连，而是通过“卷积核”作为中介。卷积核作为一个批量过滤器, 持续不断的在图片上滚动收集图片里的信息,每一次收集的时候都只是收集一小块像素区域, 然后把收集来的信息进行整理, 这时候整理出来的信息有了一些实际上的呈现。比如在识别脸的时候，进行局部感知识别出眼睛,鼻子等等. 再经过一次过滤, 脸部的信息也从这些眼睛鼻子的信息中被总结出来。



**图2-5卷积神经网络运作示意图**

卷积神经网络是一种特殊的深层的神经网络模型，它的非全连接和权值共享的网络结构使之更类似于生物神经网络，还有池化（Pooling），对“卷积核”提取进行进一步整理和压缩，降低了网络模型的复杂度，减少了权值的数量。可以识别位移、缩放及其他形式扭曲不变性的二维图形。

## 2.4深度强化学习（DRL）及DQN

Q-Learning可以解决一些增强学习的任务，但是依然有个根本的问题就是状态空间和动作空间的量级问题。现实情况往往比较复杂，如果状态空间特别大，那么Q(s,a)这个矩阵就会非常大，仅通过反复的测试无法获得足够的样本来遍历每个状态，这必然导致算法失败。联系到人类的行为特点，我们也不可能遍历所有的情况，更多的是将新的情况与记忆进行比对，如果相似，那么就可以采用相似的做法。因此，自然而然的，如果能用一个函数来表示value function，输入任意的状态都能输出结果，那么就可以把Q矩阵的更新问题变成一个函数拟合问题，相近的状态也就可以得到相近的输出动作。

深度神经网络可以自动提取复杂特征，因此用深度神经网络来进行值估计，就得到Deep Q-Learning，也就是Deep Reinforcement Learning的基本思想。

### 2.4.1深度神经网络思想

**1) value function approximator（值估计）**

用一个深度神经网络来作为*Q*值的网络，参数为*w*，我们需要更新参数w来使得*Q*函数逼近与最优的*Q*值。：

（3）

**2) loss function**

在*Q*值中使用均方差mean-square error 来定义目标函数objective function也就是loss function

（4）

可以看到，这里就是使用了Q-Learning要更新的*Q*值作为目标值。有了目标值，又有当前值，那么偏差就能通过均方差来进行计算。

**3) 计算参数w 关于loss function的梯度,求偏导**

（5）

**4)使用SGD实现End-to-end的优化目标**

有了上面的梯度，而可以从深度神经网络中进行计算，因此，就可以使用SGD （随机梯度下降）来更新参数，从而得到最优的*Q*值。

### 2.4.2深度神经网络对Q-learning改进

**一、经验池（experience replay）**

经验池的功能主要是克服经验数据的相关性（correlated data）和非平稳分布（non-stationary distribution）问题。具体做法是把每个时间步agent与环境交互得到的转移样本 (*st*,*at*,*rt*,*st*+1) 储存到回放记忆单元，要训练时就随机拿出一些（minibatch）来训练。（其实就是将游戏的过程打成碎片存储，训练时随机抽取就避免了相关性问题）。这样有两个好处：1. 数据利用率高，因为一个样本被多次使用。2. 连续样本的相关性会使参数更新的方差比较大，该机制可减少这种相关性。

**二、两个神经网络，参数冻结**

创建两个神经网络，参数冻结解决延迟更新。研究者在实践中发现当使用如NN这样的非线性函数逼近器逼近Q函数时RL学习过程并不稳定。这种不稳定有几种原因：经验数据（即观察序列）具有相关性。Q函数的微小改变会引起策略（policy）的巨大改变，进而改变训练数据分布，以及Q函数与Q函数目标之间的差值。前者可以用经验池解决。后者可采用创建两个神经网络，参数冻结解决延迟更新，即targetQ网络的参数只在一定步数后才更新，相当于延迟更新来减少Q函数和Q函数目标间的相关性。直观上，整个训练学习过程其实就是Q函数向Q函数目标逼近的过程，因为如果目标也跟着学习体一起变，那势必对收敛性造成影响。

你

# 3、模型构建

## 3.1环境搭建

**一、游戏目标**

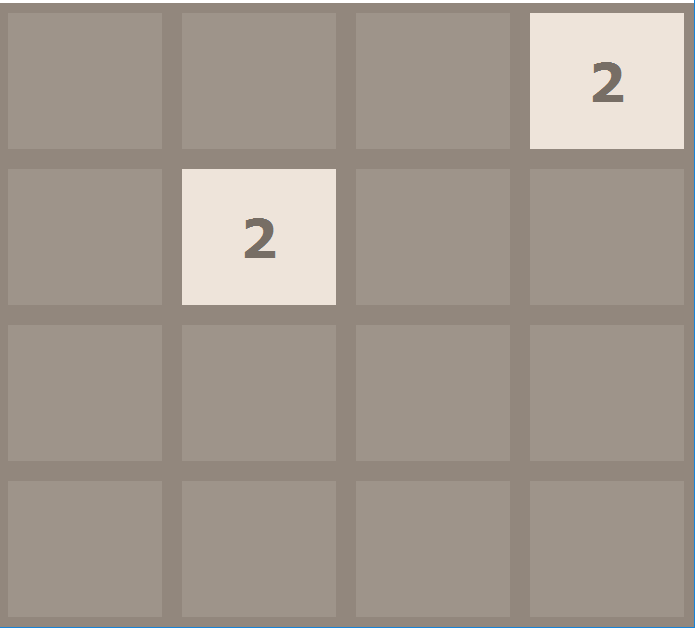
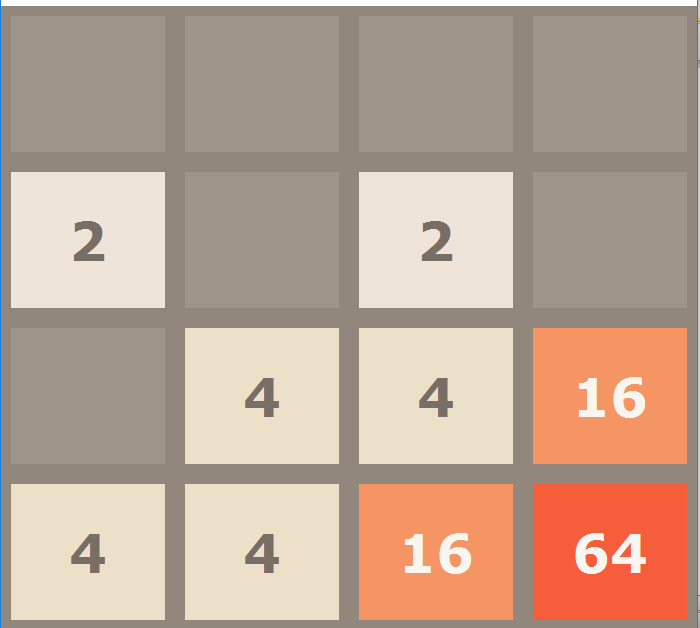
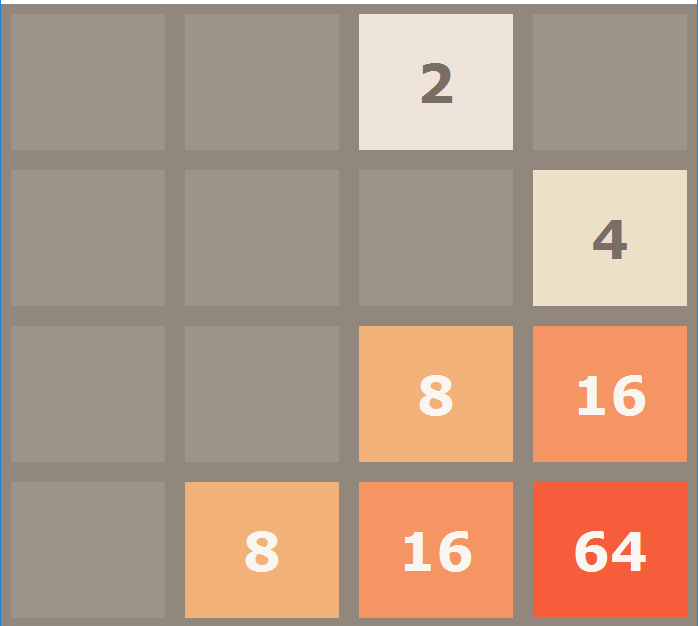
使用强化学习DQN的方法，让计算机能够自动完成2048游戏。

**二、游戏规则：**

**初始化：**初始在4\*4个格子中会随机生成两个2，游戏玩家通过按键进行动作选择，进行上下左右移动。

**动作选择及状态转移：**

玩家选择一个动作向上下左右中的一个方向移动，则所有数字均往该个方向移动，同时如果该方向上存在相同相邻的数字，则这两个数字叠加并加在移动方向所指末端的那个数，另一个数的位置的另一边若没有数字则空出来，有则继续往这个方向聚拢。同时每次动作移动之后，系统随机在空位置产生出一个2。如选择向上移动，一定若上下（或者左右）相邻相同，如两个2在竖直（或左右）方向上相同，则上下（或者左右）移动时可叠加生成4，最后到。

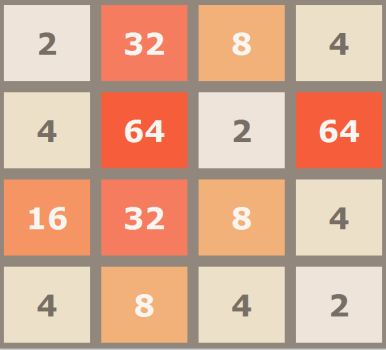
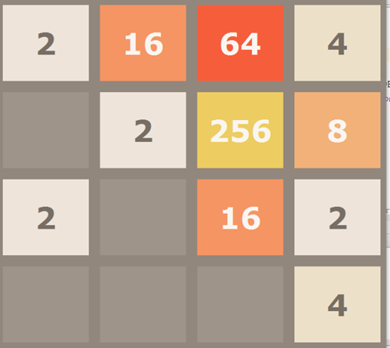
  

**图3-1：初始化界面 图3-2：游戏界面 图3-3：执行动作右移后界面**

**结果判定：**

如果游戏页面最终显示2048，则游戏胜利。如果在实现2048之前，游戏界面数字充满，且无法再移动合并，则游戏失败。

在实验环境下，一次性实现2048有难度，因此暂定于将目标设置为256，再慢慢提高实现2048。

**图3-4：游戏失败界面 图3-5：游戏成功界面**

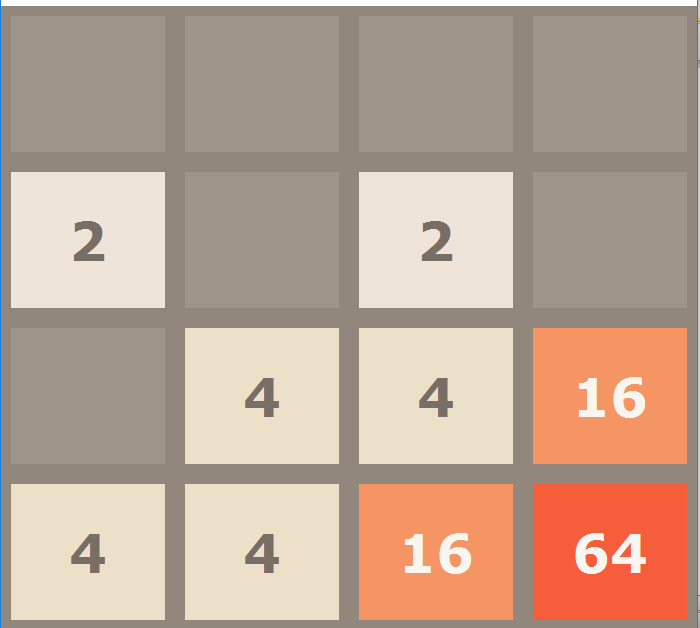
**接口：**

此环境由puzzle、logic两个类中相关方法实现，可下载直接得到，不是本实验的重点，因此在此不再叙述。本实验所需要的是得到游戏的状态变量，以及将DQN计算得到的action传递给游戏环境，使游戏基于此动作进行状态转移。因此，与环境的交互关键是编写接口的方法observation\_, done,reward,is\_end= env.step(action)，将神经网络计算得到的action传入环境，得到此action下的状态observation\_（环境观测值）、reward（奖惩），以及done（动作是否完成）， is\_end（游戏是否结束）

## 3.2模块设计

#### 3.2.1、状态空间（state）设计：

对应与游戏的图形界面的4\*4方格，将每个方格用相对应的数字表示（空格出标为0），可转化为图二的矩阵M4x4（Python语言中称为list列表）



**图3-6：游戏界面**

**状态空间一：**

由于全连接神经网络输入的都是一维列表，故第一种特征状态空间设计为：将矩阵M4x4进行降维扁平化处理，直接使其变成16个数字顺序排列的一维列表

（*x*1，*x*2，*x*3，…….*x*15）

这样输入的状态空间信息保留了较多原始信息，但是位置信息，尤其是相对位置信息可能有所损失。

**状态空间二：**

将矩阵进行预处理，人工提取特征得到一维S列表

（*m*1,*m*2,*m*3,*m*4,*l*1,*l*2,*l*3,*l*4 ,*max*,*x*1,*y*1）

*mi*代表第i行能合并的数之和，图3-5中第二行有两个2相邻可合并，则*m*2=4，*l*i代表第i列能合并的数之和，*max*代表矩阵中最大的数，（*x*1,y1）是其位置，图一中最大数为64，出现在第4行第4列，故用（64，4，4）表示在。图3-5可表示为（0，4，8，8，0，8，0，0，64，4，4）。

**状态空间三：**

将矩阵进行预处理，人工提取特征得到一维S列表，并进行归一化处理（S列表中每个数除以最大的数）

（*m*1,*m*2,*m*3,*m*4,*l*1,*l*2,*l*3,*l*4）

特征向量在被传入输入层时归一化处理，先标准化到0和1之间，是为了加速学习过程。而最大数及其位置坐标与每行每列能合并的数并不是同一类型的数，无法进行归一化处理，故此状态空间将其舍去，只留下*mi，l*i。。

无疑的是，这样做相比于状态空间四，必然损失了更多信息（人工提取信息必然是有限的有损失的，状态空间四也会有一定的信息损失），S列表所含的信息并不能完全表示实际环境的信息，但有可能能代表类似情况的信息，但是类似情况可做类似的选择，通过实验了可验证其是否合适，以及归一化处理之后能否加速学习。

**状态空间四**：

卷积神经网络（CNN）可直接对图片信息进行处理，即可直接处理二维或者三维列表，故将矩阵先进入两层卷积层进行特征提取，卷积层的结果再输入全连接的神经网络。卷积神经网络可以提取位置信息，尤其是相对位置，本实验的游戏就是根据相对位置进行动作选择的，因此，预期卷积神经网络会达到很好的效果，但是由于多加了两层卷积层，神经网络的深度和复杂性增大，参数增加，所需要的训练次数随之也将增大。

#### 3.2.2、动作（action）的选择：

**四种动作（action）的表示：**

上下左右四种动作，分别对应于0，1，2，3。*action*=0，代表执行动作“上”，将所有数字向上移动和合并。其他同理可得。

**动作（action）的选择：**

动作的选择通过神经网络计算得到，并且通过ε-greedy的策略选择。具体实现和步骤见下文DQN中的“选择行为”的方法讲到。

#### 3.2.3、奖惩函数（reward）的设立

**奖惩函数一：**

每一个动作之后状态改变，都会反馈一个*reward*反应这一步的好坏，具体数值为，每次移动合并多少数字就*reward*奖励的数值就是多少，2、2合并成4得4分则*reward*=4，4、4合并成8得8分则*reward*=8。游戏目标设为256时，最终合并两个128得到256，则*reward*=256。

移动不能合并就扣2分*reward*=-2，游戏失败扣50分*reward*=-50。

**奖惩函数二：**

有实验第一代数据可见奖惩函数一失败了，尤其是第二个模型店结果可见，训练结果没有变好反而往更差的结果发展了。其原因可能是太关注与每一步是否能合并而且尽可能多的合并，很容易陷入局部最优。

奖惩函数二按每一个episode设置奖惩函数。游戏成功则*reward*=10，游戏失败则*reward*=-10，利用最终结果进行反推每一步的效用和进行DQN参数更新，这是全局最优的思想进行的。

## 3.3算法伪代码

初始化记忆池，它的容量为*N*

初始化*Q*网络，随机生成权重*w*和偏差*b，*初始化目标*Q*网络*w\_=w*

循环遍历episode=1，2，3………,*M*

初始化游戏，得到

循环遍历step=1，2，3……*.T*

用策略生成

执行动作,接收到奖惩以及新的状态

将（）作为一个记录存在记忆池中

从记忆池中抽取minibatch条记录（）用来学习和训练

根据用梯度下降进行误差反向传递

每隔*C* steps更新目标*Q*网络的参数*w\_=w*

## 3.4Deep Q-Network搭建



**图3-7搭建含三层隐层的神经网络计算Q值**

前文所说，用神经网络替代*Q*值表，*Q*值将由神经网络计算观测值输入得到，我们用Tensorflow 来实现 DQN,我们需要搭建两个神经网络：target\_net、eval\_net。 target\_net 用于计算 q\_target 值, eval\_net 用于计算 q\_eval值。这两个神经网络结构是完全一样的, 只是里面的参数不一样，eval\_net拥有最新的神经网络参数，并及时更新，target\_net 是 eval\_net 的一个历史版本，拥有 eval\_net 很久之前的一组参数，而且这组参数被固定一段时间, 然后再被 eval\_net 的新参数所替换.两个神经网络的目的是通过暂时冻结 q\_target 参数 ，一定程度降低了当前*Q*值和目标*Q*值的相关性，提高了算法稳定性。



**图3-8两个神经网络下的参数传递**

**class DeepQNetwork主要包括4个方法：**

1. def \_init\_(self): #初始化
2. def \_build\_net(self):#建神经网络
3. def store\_transition(self, s, a, r, s\_):# 存储记忆
4. def choose\_action(self, observation): # 选行为
5. def learn(self): # 学习

### 3.4.1初始化

初始化参数：

* 学习效率：learning\_rate=0.01,
* 折扣率：reward\_decay=0.9,
* 使用的方法选择行为时：e\_greedy=0.9，初始为0，随后每100次学习增加0.001，90000次学习之后，开始以e\_greedy进行行为选择
* 每学习200步，将eval\_net的参数替换给target\_net：replace\_target\_iter=200,
* 记忆池容量：memory\_size=500

### 3.4.2创建神经网络

**全连接神经网络创建：**

创建的有三层全连接层的神经网络，输入层输入的参数个数由状态空间输入的特征值决定，上一层神经网络层的输出作为下一层网络层的输入，最终将输出每个action对应的*Q*值。



**图3-9含有三层隐层的全连接神经网络**

第一层隐层神经元输入个数为特征变量的个数，输出为32，第二层神经元输入个数32（第一层输出），输出为32，第三层神经元输入个数为32，输出个数由输出层决定，由于要输出四个动作下的q，因此为4.

神经网络第一层的输入为状态变量（或经过特征提取之后）的一维列表，将每一个权重乘以其对应的输入神经元，加上偏差，再经过激活函数（本实验激活函数为relu（）），然后将结果保存在相应的隐藏神经元。y= f（Weights \* x + biases）

第一层代码如下：

**with** tf.variable\_scope(**'l1'**):  
 w1 = tf.get\_variable(**'w1'**, [self.n\_features, n\_l1], initializer=w\_initializer, collections=c\_names)  
 b1 = tf.get\_variable(**'b1'**, [1, n\_l1], initializer=b\_initializer, collections=c\_names)  
 l1 = tf.nn.relu(tf.matmul(self.s, w1) + b1)

神经网络第二层中的权重和阈值与第一层设置的一样，只是进行矩阵相乘时输入层不同，此时的输入层为第一层伸进网络的输出。

神经网络第三层权重和阈值设置同上，输入层位第二层输出，输出层即为四个动作所对应的Q。

**含有卷积层的神经网络：**

卷积神经网络输入的为矩阵，预处理之后变为4\*4\*1的三维列表，2\*2的卷积核作为一个批量过滤器,在三维列表上收集信息,每一次收集的时候都只是收集一小块像素区域（区域大小为卷积核大小，为2\*2），每块区域所隔步数为1，16个卷积核去收集，因此第一层卷积后输出的为3\*3\*16的三维列表（结果依然需要经过激活函数处理，才能进行误差反向传递时参数更新）



**图3-10含有两层卷积层和三层隐层的全连接神经网络**

第二层卷积核大小依然是2\*2，每次移动步数为1，32个卷积核，输出2\*2\*32的三维列表（需经过激活函数处理）。进行扁平化处理之后输入全连接的神经网络。

**with** tf.variable\_scope(**'conv\_l1'**):  
 W\_conv1 = weight\_variable([2, 2, 1, 16]) *# patch 2\*2, in size 1, out size 16* b\_conv1 = bias\_variable([16])  
 h\_conv1\_eval = tf.nn.relu(conv2d(sx, W\_conv1) + b\_conv1)  
 h\_conv1\_target = tf.nn.relu(conv2d(s\_x, W\_conv1) + b\_conv1)*# output size 3\*3\*16*

### 3.4.3记忆存储

DeepQNetwork 类中的store\_transition(self, s, a, r, s\_)方法，将每次行动作为一条数据存储，表示action前状态，s\_表示action后的状态，a为action，r为reward。

记忆池大小设为500，即存储的数据不超过记忆池的大小，超过之后采用先进先出的方法，通过索引存储，不断用新的记忆取代旧的记忆。

### 3.4.4选择行为

使用的方法，以概率选择最优的，以的概率随机探索，防止陷入局部最优解。

（6）

流程如下：

1. 初始化 epsilon 变量，并设置它的最小值0.5，与最大值0.9，并将 epsilon 的初始值设置成最小值。
2. 随机生成一个数0-1的数p。
3. 判断 *p*是否小于 epsilon，如果 *p*小于 epsilon 则转到 4，否则转到 5。
4. 使用神经网络直接计算出结果

神经网络会输出在当前状态下所有动作的 *Q* 值，用argmax(actions\_value)函数选择其中最有价值（*Q* 值最大）的动作返回。

1. 使用随机策略（增加探索机会）。

随机选择一个在 Agent 当前状态下可以执行的动作。

1. 逐步增大epsilon 来减少探索机会。每隔100次，epsilon增加0.001，一直增加到最大值。

### 3.4.5学习

记忆池里随机抽取出一小批的数据当做训练样本，并计算出目标 Q 值来训练神经网络。

1. 从记忆池里随机抽取出一批样本。
2. 由于每条样本中，都保存有当时的数据（state, action, reward, state\_）所以，state传入eval\_net 计算得到q\_eval，state \_传入target\_net得到q\_next。
3. 基于公式q\_target = reward + reward\_decay \* Max{q\_next}计算得到q\_target
4. 计算损失值，q\_eval和q\_target的均方差

self.loss = tf.reduce\_mean(tf.squared\_difference(self.q\_target, self.q\_eval))

1. 通过优化器（optimizers）类，通过梯度下降进行神经网络参数优化

self.\_train\_op =tf.train.RMSPropOptimizer(self.lr).minimize(self.loss)

# 4、实验环境及结果分析

## 4.1实验环境

### 4.1.1软件环境

**编程语言：**Python3.5

实验采用的编程语言为Python3.5，Python是一种解释型、面向对象、动态数据类型的高级程序设计语言，由于其简单、易学、运行速度快（底层是C）、易扩展等优势应用广泛，尤其是近些年兴起的机器学习、数据分析等都将Python作为首选。 Python有很多科学计算扩展库，调用相关函数实现需要的功能，在本实验中就用到了NumPy、Pandas、Tkinter和Matplotlib。

**开发环境：**PyCharm

Python开发环境有很多，本实验采用PyCharm，其带有一整套可以帮助用户在使用Python语言开发时提高其效率的工具，比如调试、语法高亮、Project管理、代码跳转、智能提示等。

**机器学习框架：**tensorflow

TensorFlow是Google开发的一款神经网络的Python外部的结构包, 也是一个采用数据流图来进行数值计算的开源软件库.TensorFlow 让我们可以先绘制计算结构图, 也可以称是一系列可人机交互的计算操作, 然后把编辑好的Python文件 转换成 更高效的C++, 并在后端进行计算.

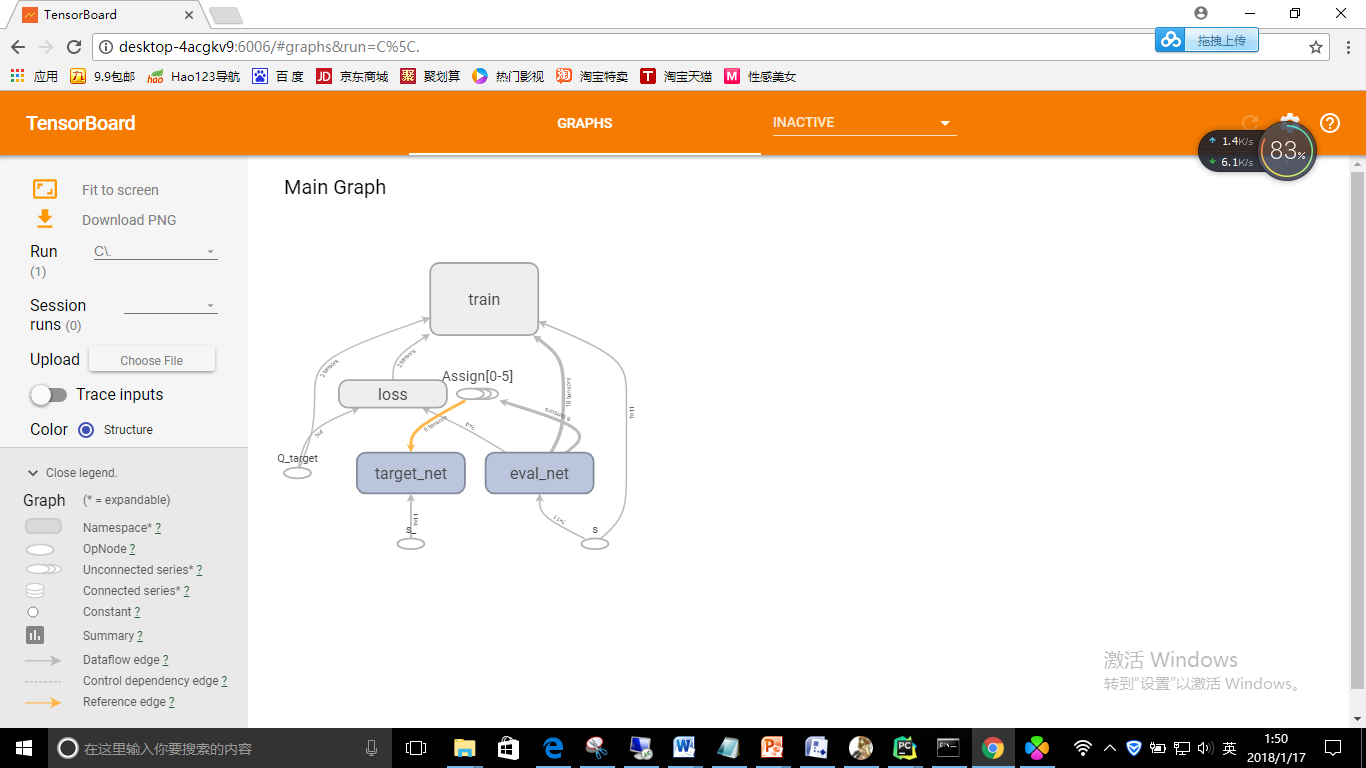
### 4.1.2硬件环境

**处理器：**Intel(R) Xeon(R) CPU E7-4820 @2,00GHz 2,00GHz(2处理器)

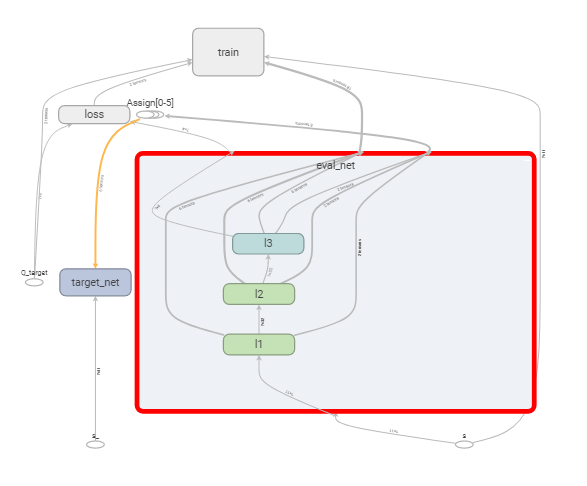
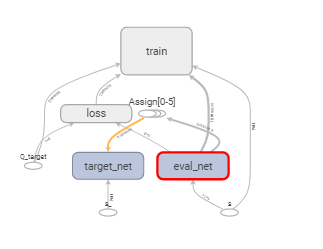
**系统类型：**64位操作系统，基于x86的处理器

## 4.2可视化DQN结构

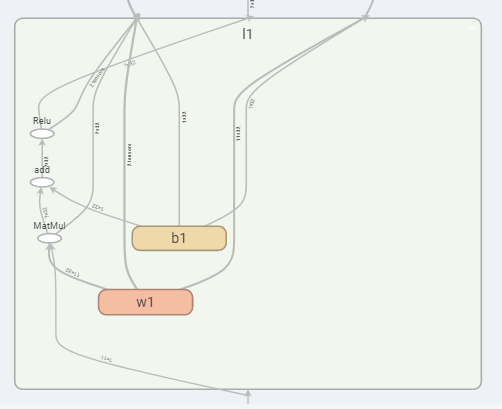
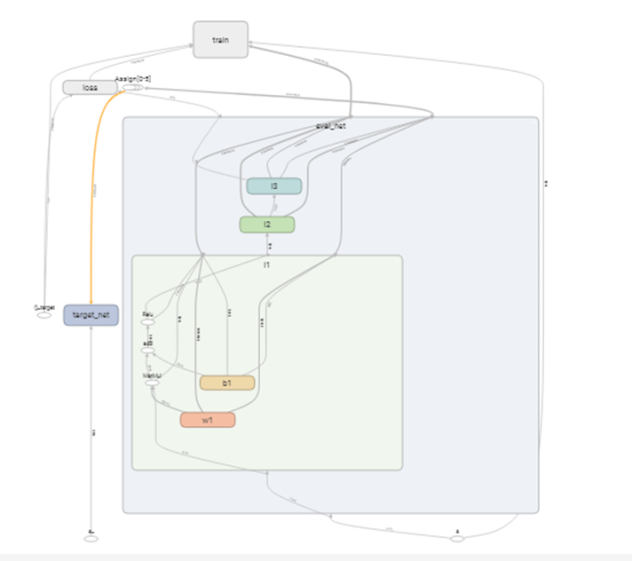
本实验神经网络的构建是基于tensorflow。同时，为了我们更直观的看到神经网络的内部构造以及数据运行时的流动，其团队开发了tensorboard，可使神经网络可视化，看到内部参数信。并且神经网络可分级展开，也可将部分结构分离开方便我们理解网络内部构造。



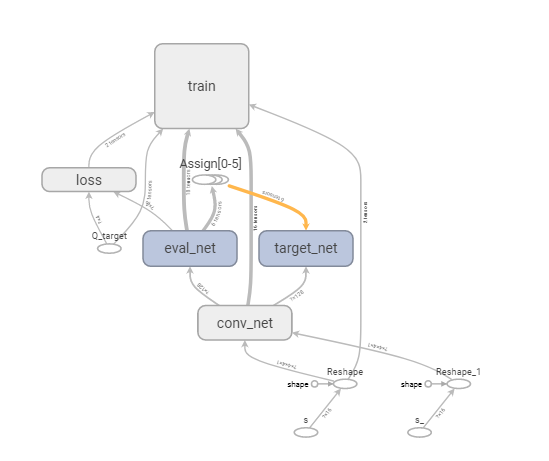
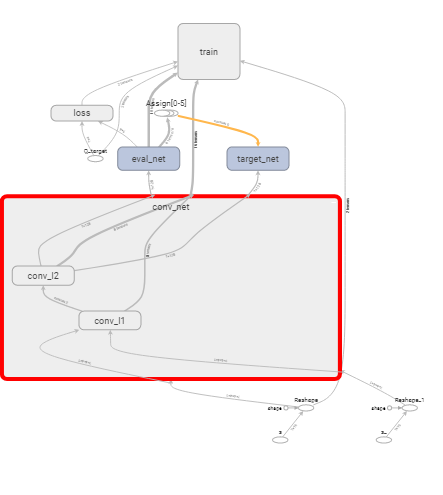
**图4-1 tensorboard可视化看神经网络**



**图4-2实验生成的全连接神经网络 图4-3神经网络展开可见三层隐层**



**图4-4继续展开可看到参数w,b 图4-5可看到参数w,b及运算**

**图4-6卷积神经网络 图4-7展开可看到两层卷积层**

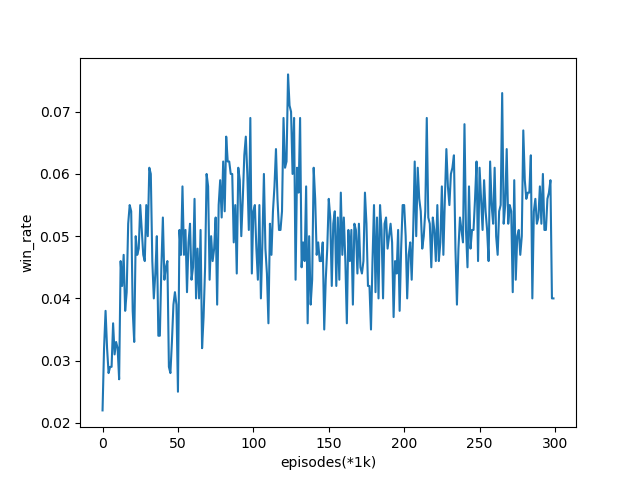
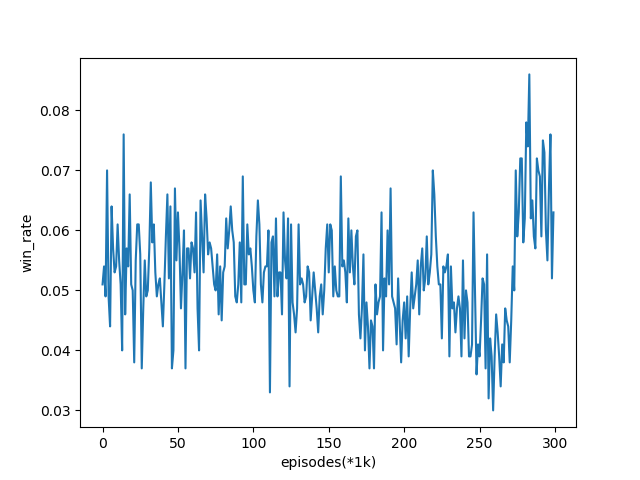
## 4.3训练结果评估

### 4.3.1第一代实验

在训练阶段，第一代的模型运行了500,000个episode，计算每1000次中赢的概率（暂时设定的目标为256）。由于神经网络初始参数是随机生成的，为避免由于初始参数原因造成的不能收敛或很快收敛的情况，每个模型都训练了两次。

**模型一实验结果分析：**

结果很遗憾没有收敛，有可能是实验训练次数不够，或者是某些参数设置不合理学习效率，折扣率等，贪婪算法的，记忆池容量等，但是本实验涉及参数过多，目前无法判断原因。

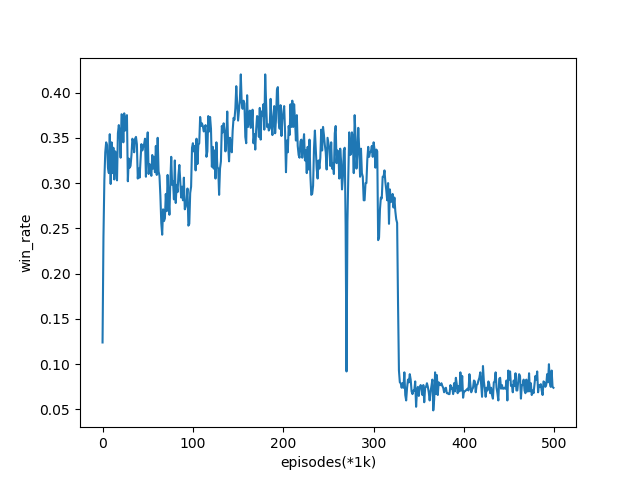
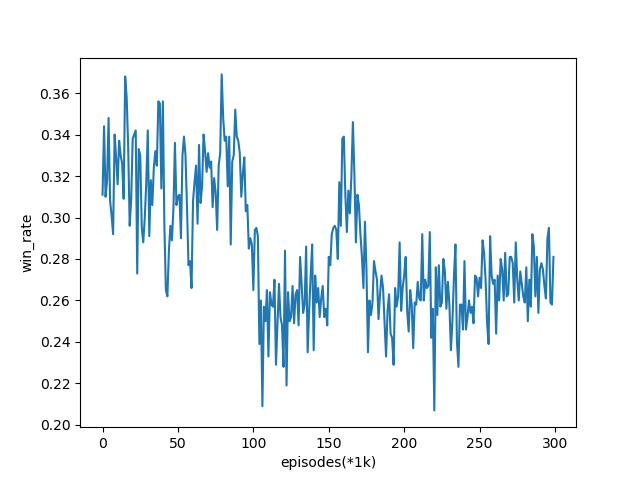
 

**图4-8实验一模型一结果（1） 图4-9实验一模型二结果（2）**

**模型二结果分析：**

模型三对应状态空间第三种，由结果可见，在一段运行范围内，训练结果慢慢呈现变好的趋势，赢的概率在变大，但是之后就开始下降，并且在突然的转折点之后，一下子降得很低，重复的两次实验都有这样的状况存在，可猜测可能是此状态空间设立不合理。

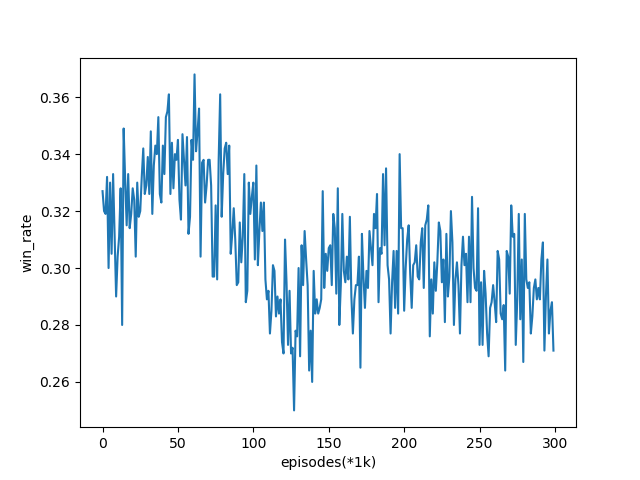
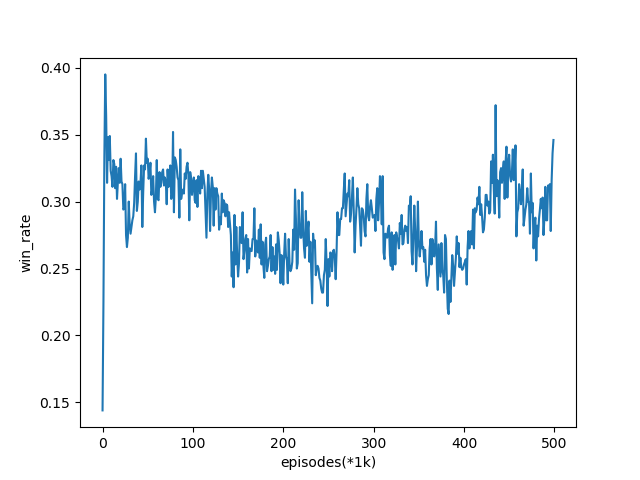
模型三我们加入了矩阵最大值极其位置，但是很多神经网络都要求最好先做归一化处理，防止极端值的影响，由此可以猜测最大值及其位置这些特征并没有被DQN理解。

**图4-10实验一模型二结果（1） 图4-11实验一模型二结果（2）**

**模型三结果分析**

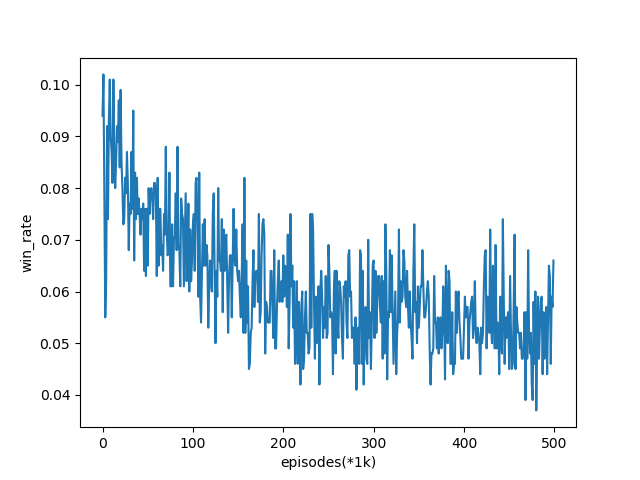
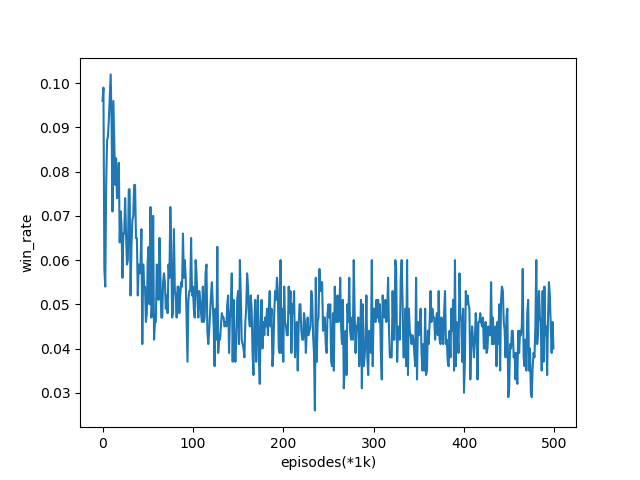
模型四结果两次结果都非常动荡，和模型一一样，无法判断是因为训练次数不够还是因为参数设置不合理。



**图4-12实验一模型三结果（1） 图4-13实验一模型三结果（2）**

**模型四结果分析：**

在所有的实验结果中，只有模型二是收敛了的。虽然与我们想要的赢的概率越来越大的目标相反了，但是收敛收敛才能去判断分析实验没有成功的原因。网络训练结果越来越差可以判断是奖励没有设置好，让DQN没有越学越好，反而越学越坏了。基于此结果，我们的第二代将采用第二种奖惩函数。

****

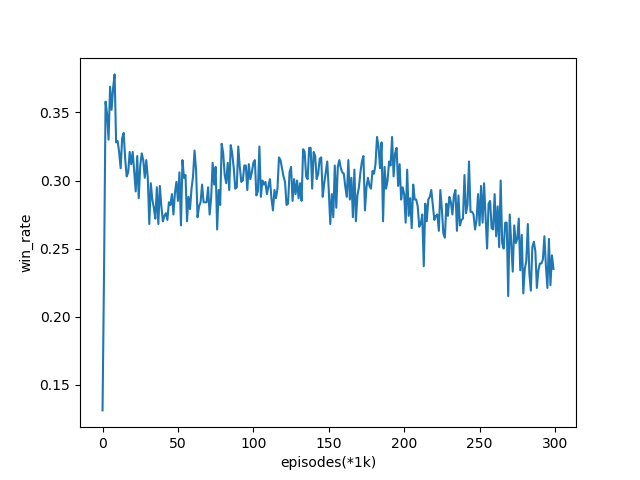
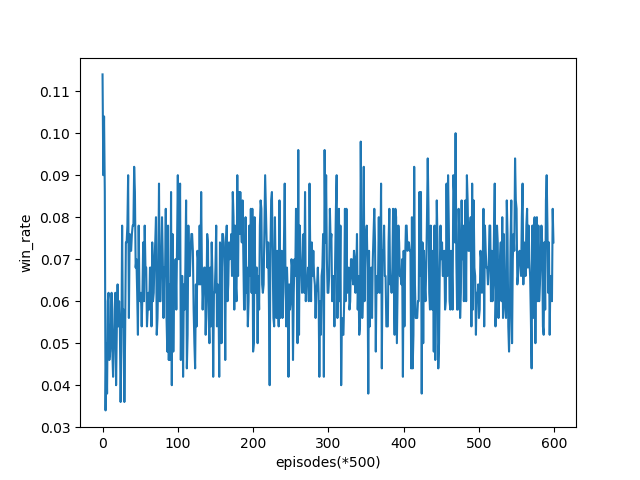
**图4-14实验一模型四结果（1） 图4-15实验一模型四结果（2）**

### 4.3.1第二代实验

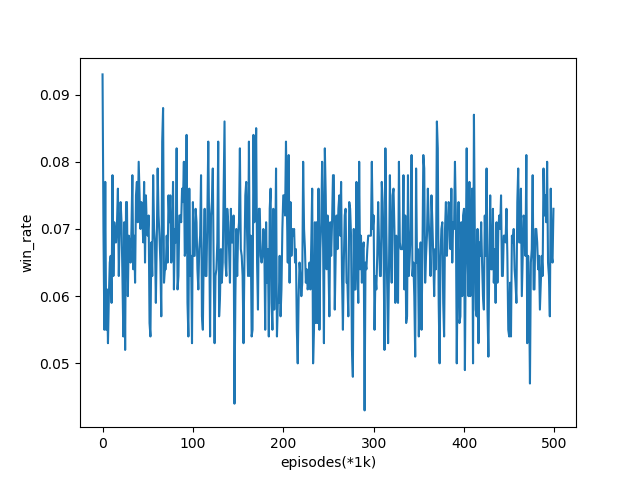
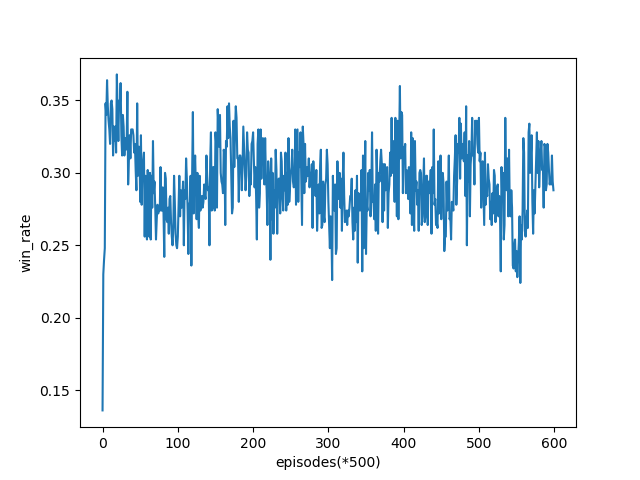
第二代实验的四个模型将采用奖惩函数二。

第二代实验由于时间原因卷全连接神经网络只运行了300万个episode（模型一有模型三每隔500episode统计赢的概率），积神经网络运行了50万个episode，且每个模型只运行了一次。（50万个episode在全连接是神经网络训练需要两夜一天约36小时，50万个episode在含有两个卷积层的神经网络中训练需要三夜两天约60个小时。）

实验结果如下：



**图4-16实验二模型一结果 图4-17实验二模型二结果**



**图4-18实验二模型三结果 图4-19实验二模型四结果**

然而，实验效果也不是很好可见，似乎并没有训练变好的效果。再次猜测，可能此奖惩函数可能造成记忆池大部分数据的reward都是空的，因此可能学不到什么。故想着接下来的尝试可以做，在最后一个episode结束得到一个reward之后再乘以一个折扣反向传递给之前的每一步。

### 4.3.2调参：

上述模型未能正向收敛，如果能收敛，则可以基于一个收敛较稳定的模型进行调参，基于如下公式（1），对和*α，*取不同的值，看收敛效果，具体如下：

（1）在其他超参数都相同的条件下，分别测试了值 = 0.9，0.95，0.99。

（2）在其他超参数都相同的条件下，分别测试了学习率= 0.002, 0.001, 0.0005, 0.0001

## 4.4实验反思和展望

本实验属于验证性实验，但是没有得到预期的效果，一部分原因是计算资源问题。人工智能之所以现在才发展起来，很大的原因是，如今计算机的计算能力、存储能力等有了极大的提高，能满足人工智能巨大的计算量。网上的FlappyBird据说使用GPU训练了一周但是，事实上本实验的状态空间比它还要大。Alpha Go更是用了一千多的CPU，一百多GPU来支持计算，Alpha Zero更是在用专门支持人工智能的TPU进行计算。但是，实验室可能只能使用cpu，因此对于简单的调参可能需要还几天把实验跑一遍才能出结果。

这次试验是从0开始的自主学习，深刻明白了互联网的巨大力量，以及科学合理借助互联网力量的重要性。我发现很多理论的学习还是需要借助具体的实验例子才能真正弄懂，一开始看论文会不理解，因为论文是知识的凝练，反而博客写的非常深入浅出，适合初学者去掌握一个概念。还有就是耐心，理论学习过程、编程过程、甚至最后实验测试阶段都遇到了各种各样的问题，放弃很简单，难的是思考和解决，甚至是寻找他人的帮助去解决这个问题。解决问题的能力远比学会一个新知识更加重要，因为知识本质上也是为解决问题服务的。

值得思考的是，2048是不是没有我想象中那么简单，直接用马尔科夫建模，是否Q-learning算法真的能顾实现，预期迭代次数也没有实际计算过。在原理上似乎我没有探究的那么深入。老师推荐了南大周志华《机器学习》这本书，对于原理的讲解写的由浅到深，还包括算法的进化过程及原因，很值得接下来继续学习。

本实验最大的意义应该是对于人工智能从浅层感知到深入实践，发现人工智能并没有那么神秘，但也并不简单。通过实验，我对于强化学习、神经网络等有了初步了解，并基于对2048游戏的解决，基本知道其算法和运用。同时，我学习了python语言，基于之前学过的JAVA，C语言，python的入门其实是很快的，但是Python有很多科学计算扩展库，调用相关函数实现需要的功能，这些事不能一下子弄懂的，不过可以在今后学习中边用边学，这次的实验也是一个边学边用的过程，这样的学习效果是最好的。

# 5、参考资料

1. Wikipadia（维基百科）AlphaGo [ EB/OL ] https://en.wikipedia.org/wiki/AlphaGo#Hardware
2. DeepMind [Human-level control through deep reinforcement learning](http://www.nature.com/nature/journal/v518/n7540/abs/nature14236.html) [ J ] . nature ，2015
3. [Playing Atari with Deep Reinforcement Learning](https://pdfs.semanticscholar.org/667f/b84bfca10bec165f9e2cca3e21f5e4829ca7.pdf) [ J ] . NIPS ，2013
4. Wikipadia（维基百科）Markov\_decision\_process 马尔科夫决策[ EB/OL ] .<https://en.wikipedia.org/wiki/Markov_decision_process>
5. CSDN.深度强化学习——DQN [ EB/OL ] <http://blog.csdn.net/u013236946/article/details/72871858>
6. Deep Reinforcement Learning 基础知识（DQN方面）[ EB/OL ]

<http://blog.csdn.net/songrotek/article/details/50580904>

1. Github. DQN经典算法案例[ EB/OL ]

<https://github.com/yenchenlin/DeepLearningFlappyBird>

1. CSDN.Q-learning算法详解.Q-learning经典案例 [ EB/OL ]

<http://blog.csdn.net/lwb102063/article/details/52734861>

1. CSDN.强化学习及其python代码展示[ EB/OL ]

<http://blog.csdn.net/u011649885/article/details/75276392>

1. mnemstudio .Q-learning example1 [ EB/OL ]

<http://mnemstudio.org/path-finding-q-learning-example-1.htm>

1. CSDN.用Tensorflow基于Deep Q Learning DQN 玩Flappy Bird [ EB/OL ]

http://blog.csdn.net/songrotek/article/details/50951537