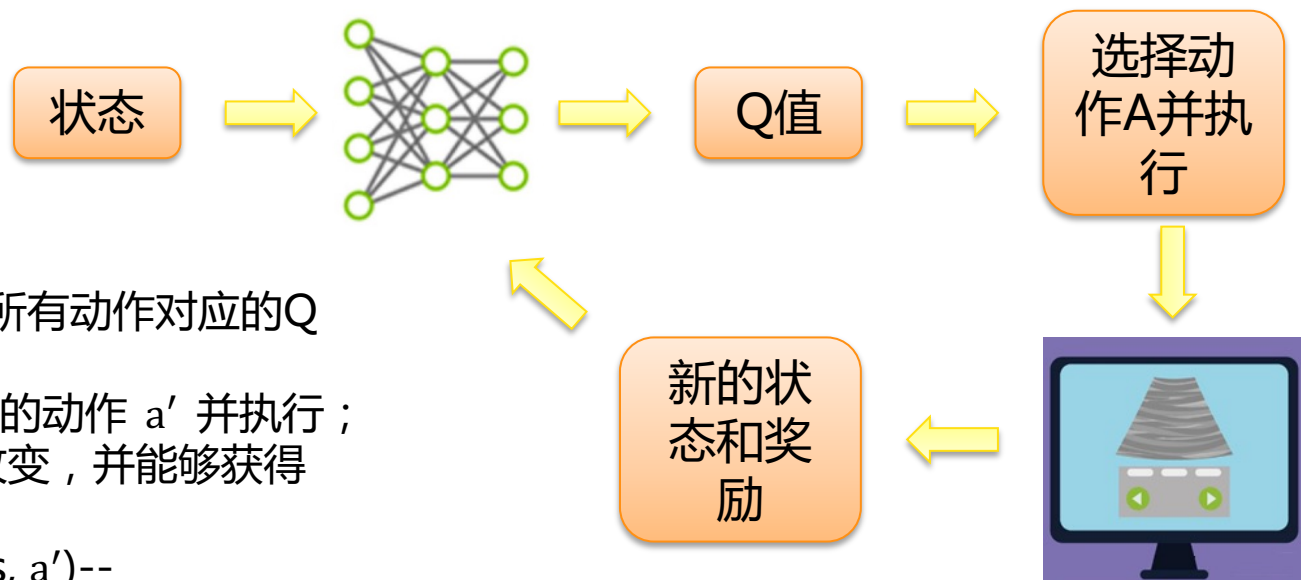


Deep Q Network (DQN)

- Deep Q Network (DQN) : 是将神经网络(neural network) 和Q-learning结合, 利用神经网络近似模拟函数 $Q(s,a)$, 输入是问题的状态 (e.g., 图形), 输出是每个动作a对应的Q值, 然后依据Q值大小选择对应状态执行的动作, 以完成控制。
- 神经网络的参数: 应用监督学习完成

DQN学习过程



学习流程：

1. 状态 s 输入，获得所有动作对应的Q值 $Q(s,a)$ ；
2. 选择对应Q值最大的动作 a' 并执行；
3. 执行后环境发生改变，并能够获得环境的奖励 r ；
4. 利用奖励 r 更新 $Q(s, a')$ --
利用新的 $Q(s, a')$ 更新网络参数

DQN算法流程

初始化D：用于存放采集的
(S_t, a_t, r_t, S_{t+1}) 状态
转移过程，用于网络参数的
训练

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{D} to capacity N

Initialize action-value function Q with random weights

for episode = 1, M **do**

 Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 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 $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

进行多次采样

一次完整的采样

DQN算法流程

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{D} to capacity N

Initialize action-value function Q with random weights

随机初始化神经网络的参数

for episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 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 $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

进行多次采样

一次完整的采样

DQN算法流程

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{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 $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 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 $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

获取环境的初始状态（ x 是采集的图像，使用图像作为agent的状态；预处理过程是说，使用4张图像代表当前状态，这里可以先忽略掉）

进行多次采样

一次完整的采样

DQN算法流程

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{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 sequence $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

ϵ 贪心策略：使用 ϵ 概率随机选取动作或 $1 - \epsilon$ 的概率根据神经网络的输出选择动作

进行多次采样

一次完整的采样

DQN算法流程

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{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 $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 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 $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

进行多次采样

一次完整的采样

在模拟器中执行选定的动作，获得奖励 r_t 和一个观察 x_{t+1}

DQN算法流程

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{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 $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 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 $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

设置 S_{t+1} ，并将状态转移过程 (S_t, a_t, r_t, S_{t+1}) 存放在 \mathcal{D} 中

进行多次采样

一次完整的采样

DQN算法流程

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{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 $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 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 $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

进行多次采样

一次完整的采样

从D中进行随机采样，
获得一部分状态转移过程历史信息

DQN算法流程

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{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 $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 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 $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to eq

end for

end for

进行多次采样

一次完整的采样

使用Q-learning方法更新状态值函数的值 (终止与非终止状态的更新不同)

DQN算法流程

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{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 $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 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 $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equ

end for

end for

进行多次采样

一次完整的采样

使用监督学习方法更新网络的参数

实例：自主学习Flappy Bird游戏

ML---



礼欣



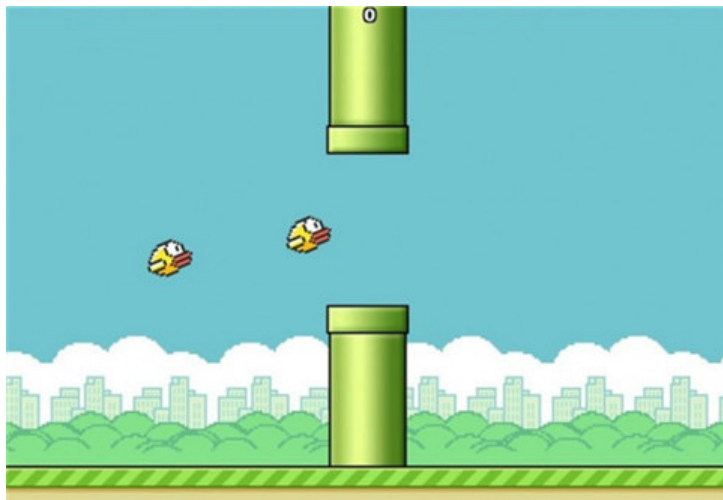
“自主学习Flappy Bird游戏” 实例介绍

深度强化学习

2013年，Deep Mind团队在NIPS上发表《Playing Atari with Deep Reinforcement Learning》一文，在该文中首次提出Deep Reinforcement Learning一词，并且提出DQN（Deep Q-Network）算法，实现了从纯图像输入完全通过学习来玩Atari游戏。

Flappy Bird游戏

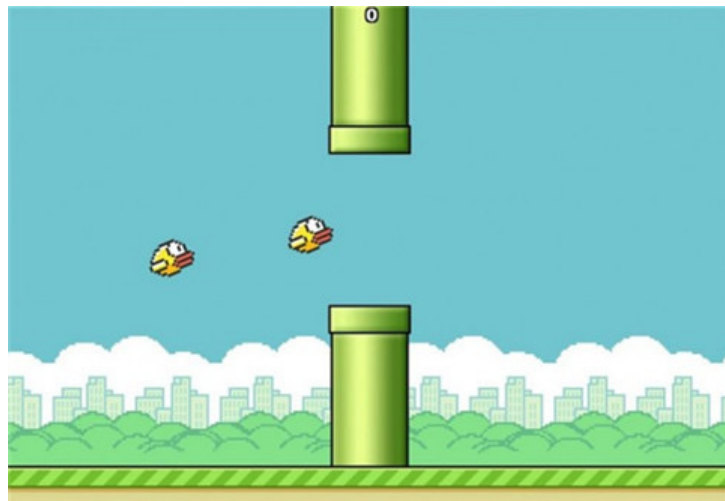
Flappy Bird：是由来自越南的独立游戏开发者开发的一款游戏。在游戏中，玩家需要点击屏幕控制小鸟跳跃，跨越由各种不同长度水管组成的障碍。



Flappy Bird

Flappy Bird游戏

Flappy Bird游戏和Atari游戏的操作方法很相似，同样可以使用DQN进行学习。



Flappy Bird

自主学习Flappy Bird游戏

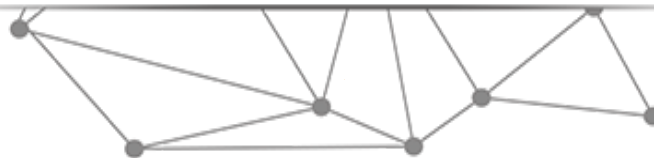
目标：使用深度强化学习方法自主学习Flappy Bird游戏策略，达到甚至超过人类玩家的水平。

技术路线：Deep Q-Network

使用工具：tensorflow + pygame + cv2

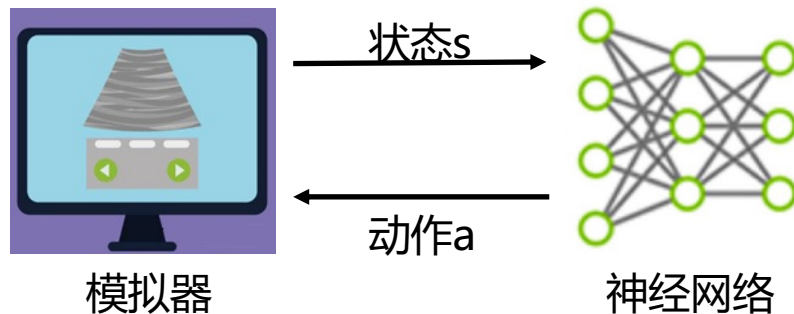


程序基本框架



程序与模拟器交互

训练过程也就是神经网络（agent）不断与游戏模拟器（Environment）进行交互，通过模拟器获得状态，给出动作，改变模拟器中的状态，获得反馈，依据反馈更新策略的过程。

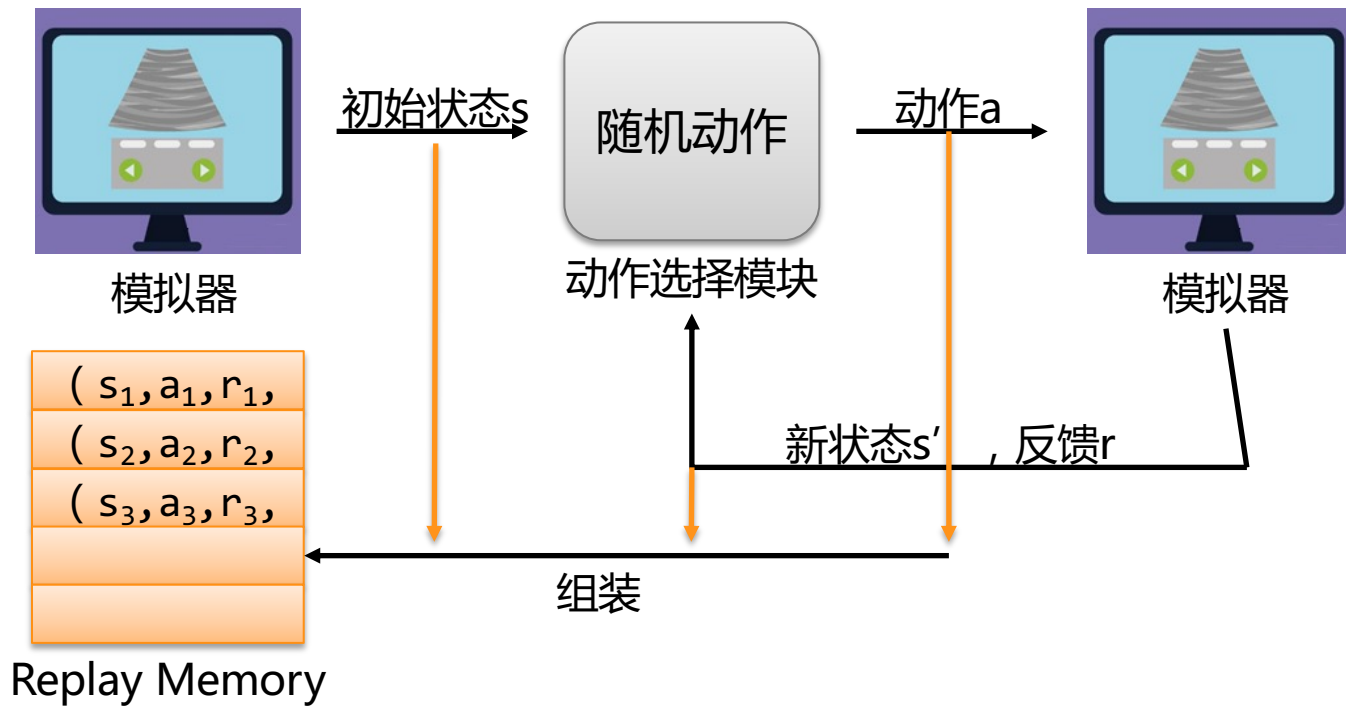


训练过程

训练过程过程主要分为以下三个阶段：

1. 观察期（OBSERVE）：程序与模拟器进行交互，随机给出动作，获取模拟器中的状态，将状态转移过程存放在D（Replay Memory）中；
2. 探索期（EXPLORE）：程序与模拟器交互的过程中，依据Replay Memory中存储的历史信息更新网络参数，并随训练过程降低随机探索率 ϵ ；
3. 训练期（TRAIN）： ϵ 已经很小，不再发生改变，网络参数随着训练过程不断趋于稳定。

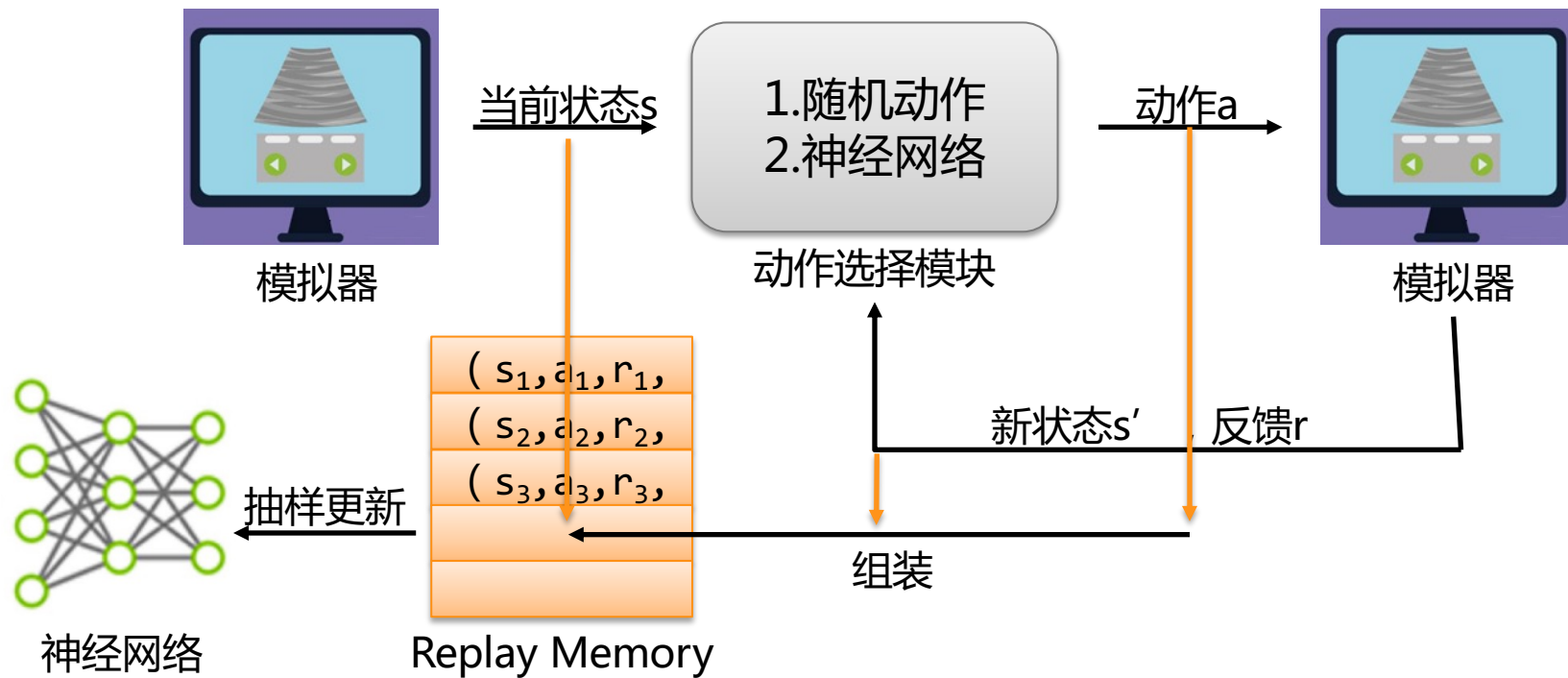
整体框架—观察期



整体框架—观察期

1. 打开游戏模拟器，不执行跳跃动作，获取游戏的初始状态
2. 根据 ϵ 贪心策略获得一个动作（由于神经网络参数也是随机初始化的，在本阶段参数也不会进行更新，所以统称为随机动作），并根据迭代次数减小 ϵ 的大小
3. 由模拟器执行选择的动作，能够返回新的状态和反馈奖励
4. 将上一状态 s ，动作 a ，新状态 s' ，反馈 r 组装成 (s, a, s', r) 放进Replay Memory中用作以后的参数更新
5. 根据新的状态 s' ，根据 ϵ 贪心策略选择下一步执行的动作，周而复始，直至迭代次数到达探索期

整体框架—探索期



整体框架—探索期

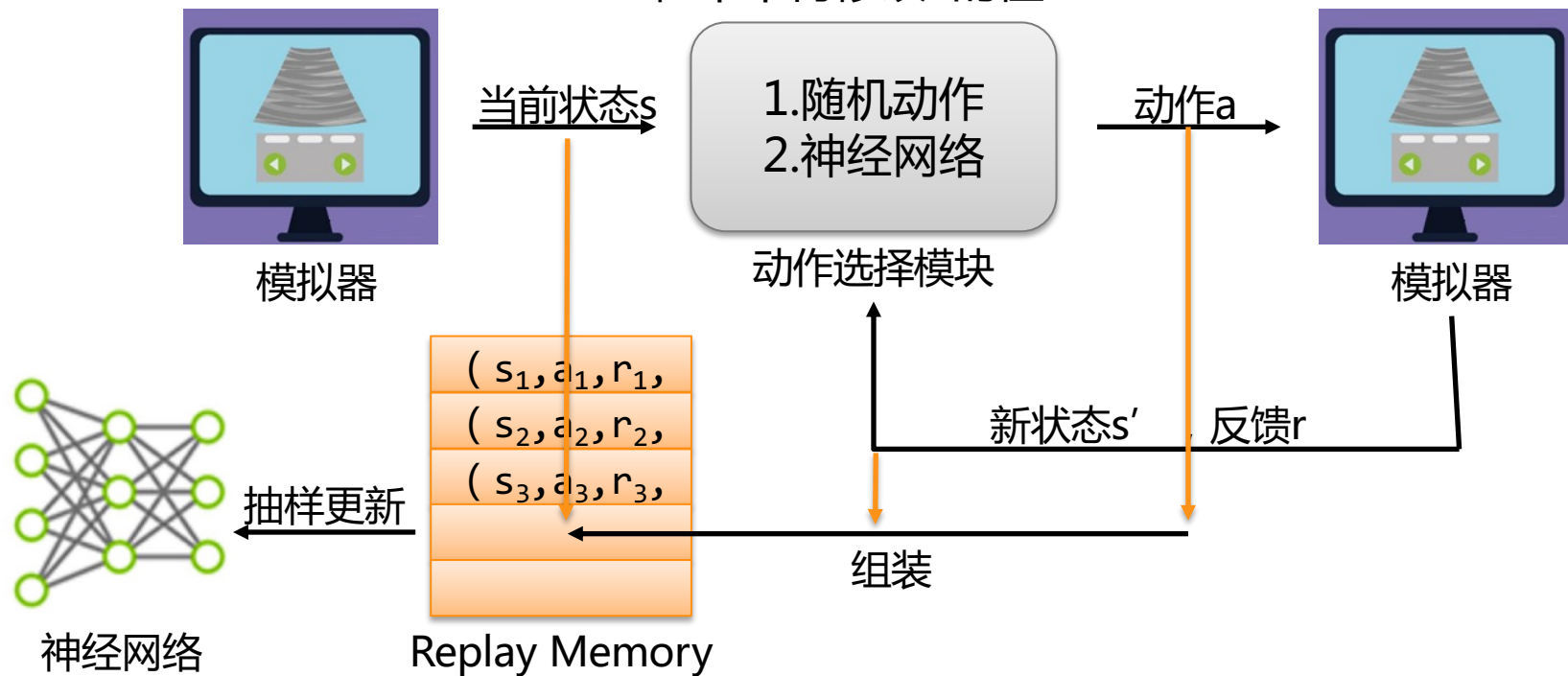
探索期与观察期的唯一区别在于会根据抽样对网络参数进行更新。

1. 迭代次数达到一定数目，进入探索期，根据当前状态 s ，使用 ϵ 贪心策略选择一个动作（可以是随机动作或者由神经网络选择动作），并根据迭代次数减小 ϵ 的值
2. 由模拟器执行选择的动作，能够返回新的状态和反馈奖励
3. 将上一状态 s ，动作 a ，新状态 s' ，反馈 r 组装成 (s, a, s', r) 放进Replay Memory中用作参数更新
4. 从Replay Memory中抽取一定量的样本，对神经网络的参数进行更新
5. 根据新的状态 s' ，根据 ϵ 贪心策略选择下一步执行的动作，周而复始，直至迭代次数到达训练期

整体框架—训练期

迭代次数达到一定数目，进入训练期，本阶段跟探索期的过程相同，只是在迭代过

程中不再修改 ϵ 的值



模拟器

游戏模拟器：使用Python的Pygame模块完成的Flappy Bird游戏程序，为了配合训练过程，在原有的游戏程序基础上进行了修改。参考以下网址查看游戏源码：

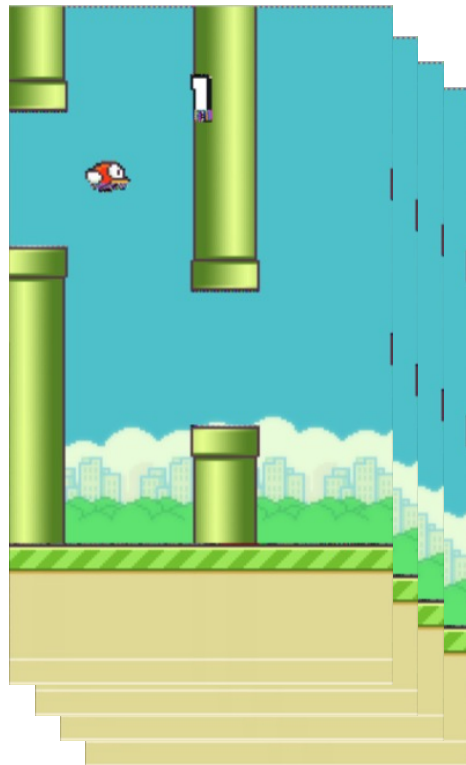
链接：<https://github.com/sourabhv/FlapPyBird>



模拟器

模拟器

- 图示通过模拟器获取游戏的画面。
- 训练过程中使用连续4帧图像作为一个状态 s ，用于神经网络的输入。



动作选择模块

动作选择模块：为 ϵ 贪心策略的简单应用，以概率 ϵ 随机从动作空间 A 中选择动作，以 $1 - \epsilon$ 概率依靠神经网络的输出选择动作：

$$\pi(s, a) = \begin{cases} \operatorname{argmax}_a Q(s, a) & \text{以概率 } 1 - \epsilon \\ \text{随机从 } A \text{ 中选取动作} & \text{以概率 } \epsilon \end{cases}$$

深度神经网络-CNN

- DQN：用卷积神经网络对游戏画面进行特征提取，这个步骤可以理解
为对状态的提取。
- 卷积神经网络(CNN)：右侧展示卷积操作。

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

CNN-卷积核

卷积核：这里的卷积核指的就是
移动中3*3大小的矩阵。

1	0	1
0	1	0
1	0	1

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

CNN-卷积操作

卷积操作：使用卷积核与数据进行对应位置的乘积并加和，不断移动卷积核生成卷积后的特征。

1	0	1
0	1	0
1	0	1

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

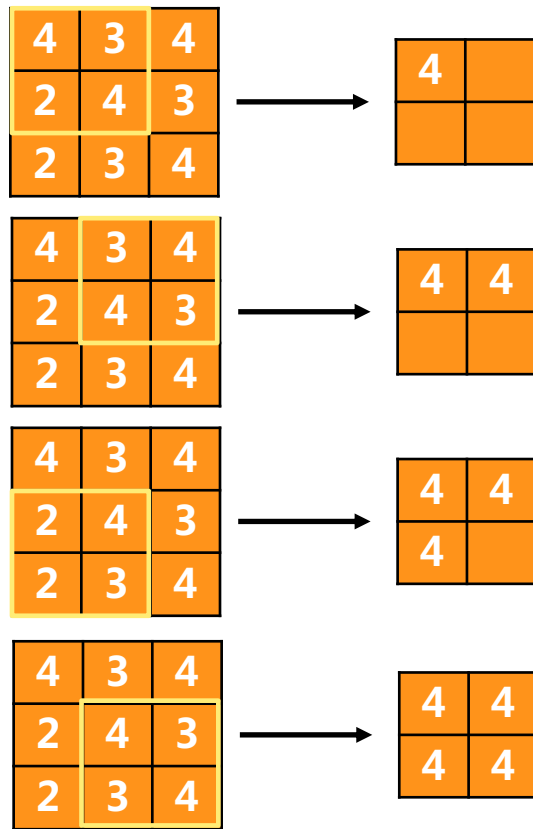
Image

4		

Convolved
Feature

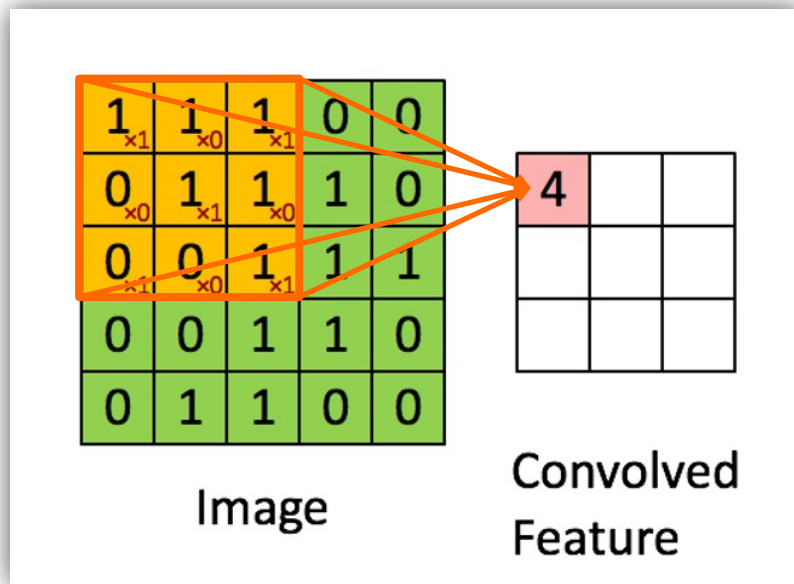
CNN-池化操作

池化操作：对卷积的结果进行操作。最常用的是最大池化操作，即从卷积结果中挑出最大值，如选择一个 $2*2$ 大小的池化窗口（操作如图示）：

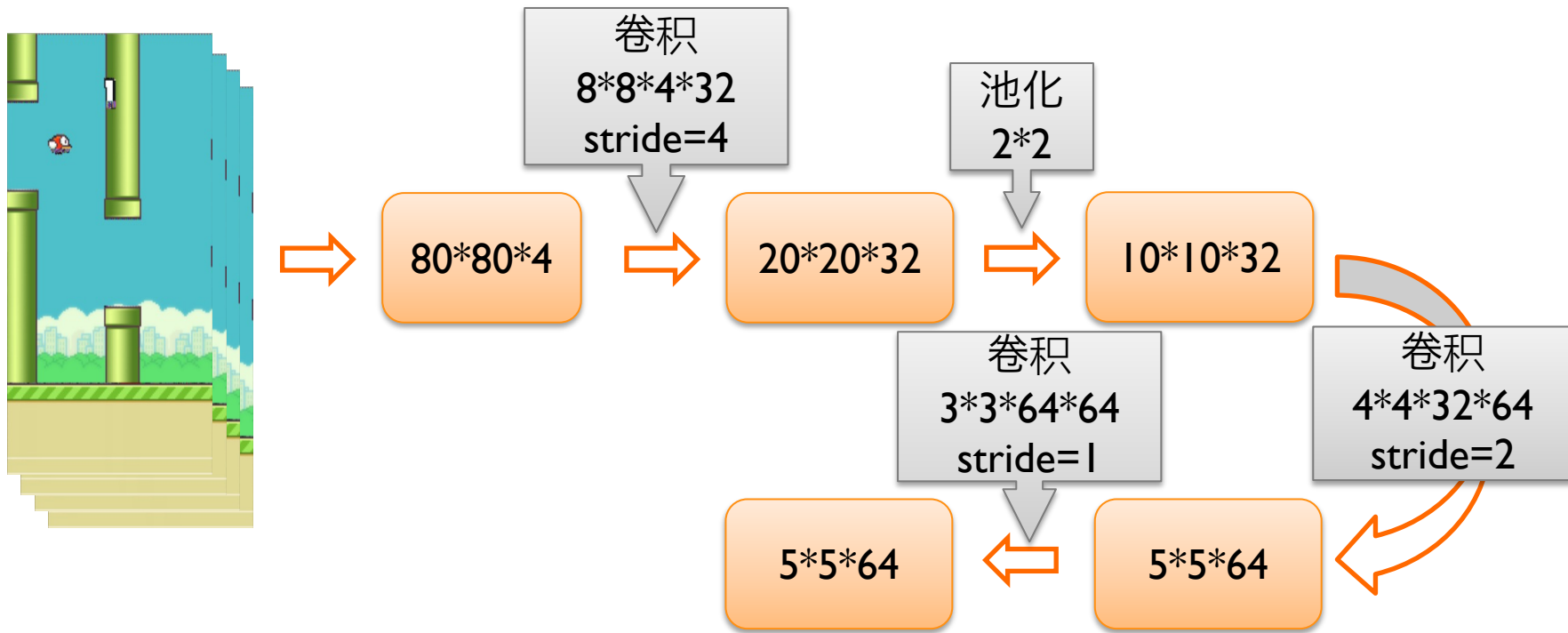


卷积神经网络

卷积神经网络：把Image矩阵中的每个元素当做一个神经元，那么卷积核就相当于输入神经元和输出神经元之间的链接权重，由此构建而成的网络被称作卷积神经网络。



Flappy Bird-深度神经网络

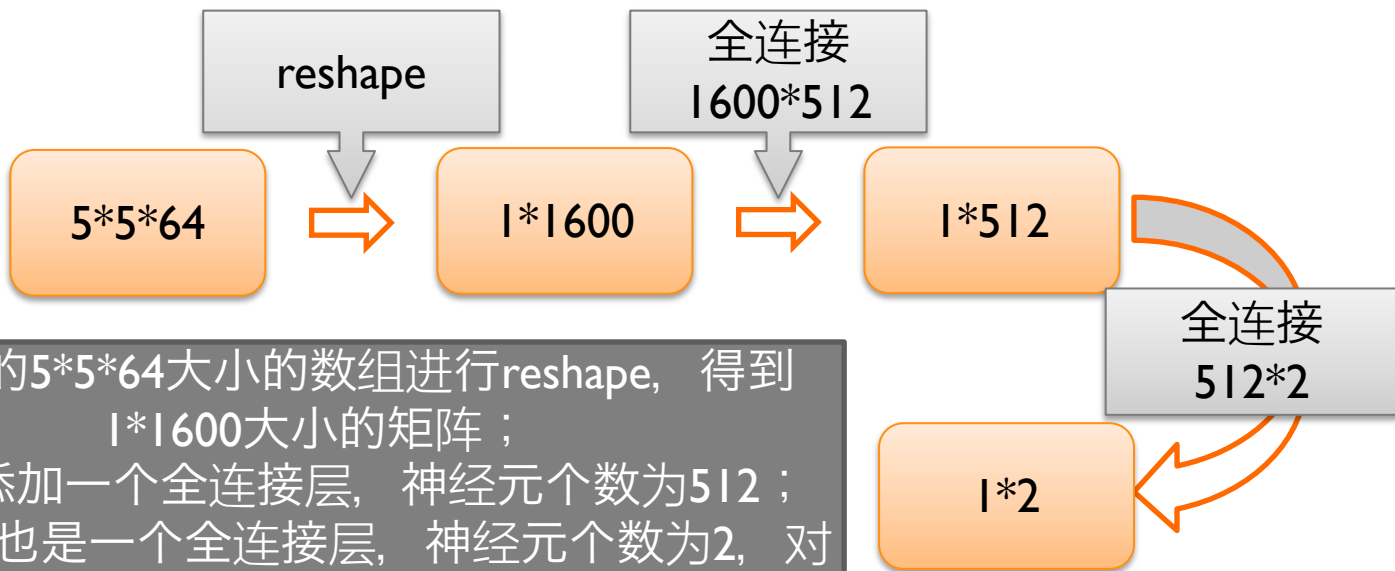


本实验中使用的深度神经网络结构就是多个卷积操作和池化操作的累加。

Flappy Bird-深度神经网络

- 对采集的4张原始图像进行预处理, 得到 $80*80*4$ 大小的矩阵;
- 使用32个 $8*8*4$ 大小步长4的卷积核对以上矩阵进行卷积, 得到 $20*20*32$ 大小的矩阵; 注: 在tensorflow中使用4维向量表示卷积核[输入通道数, 高度, 宽度, 输出通道数], 对应于上面的 $[4,8,8,32]$, 可以理解为32个 $8*8*4$ 大小的卷积核;
- 对以上矩阵进行不重叠的池化操作, 池化窗口为 $2*2$ 大小, 步长为2, 得到 $10*10*32$ 大小的矩阵;
- 使用64个 $4*4*32$ 大小步长为2的卷积核对以上矩阵进行卷积, 得到 $5*5*64$ 的矩阵;
- 使用64个 $3*3*64$ 大小步长为1的卷积核对以上矩阵进行卷积, 得到 $5*5*64$ 的矩阵;

Flappy Bird-深度神经网络



- 将输出的 $5 \times 5 \times 64$ 大小的数组进行reshape, 得到 1×1600 大小的矩阵；
- 在之后添加一个全连接层, 神经元个数为512；
- 最后一层也是一个全连接层, 神经元个数为2, 对应的是就是两个动作的动作值函数；

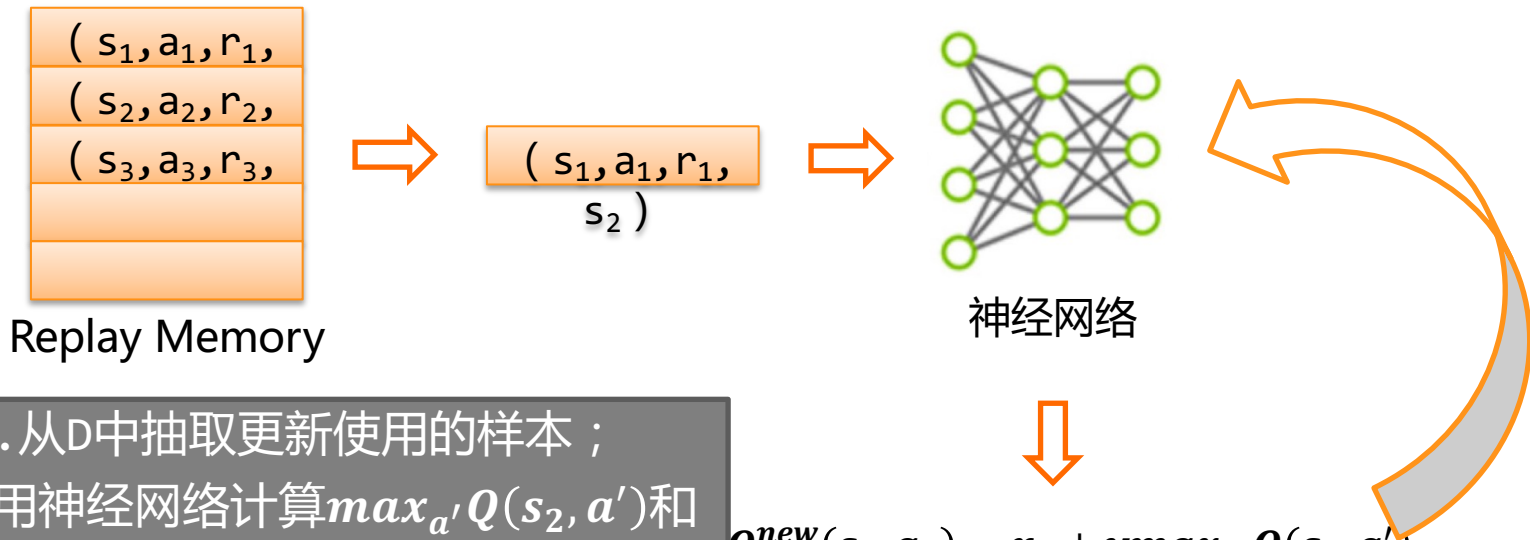
输出分别对应于两个动作, 即不做操作和跳跃的状态动作值函数。

Flappy Bird-深度神经网络

通过获得输入 s ，神经网络就能够：

- 输出 $Q(s,a1)$ 和 $Q(s,a2)$ 比较两个值的大小，就能够评判采用动作 $a1$ 和 $a2$ 的优劣，从而选择要采取的动作
- 在选择并执行完采用的动作后，模拟器会更新状态并返回回报值，然后将这个状态转移过程存储进 D ，进行采样更新网络参数。

网络参数更新

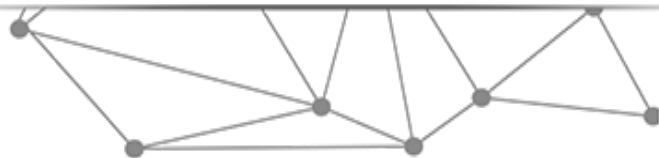


1. 从D中抽取更新使用的样本；
2. 利用神经网络计算 $\max_{a'} Q(s_2, a')$ 和 $Q^{old}(s_1, a_1)$ ；
3. 计算 $Q^{new}(s_1, a_1)$ ，并通过 $Q^{new}(s_1, a_1)$ 和 $Q^{old}(s_1, a_1)$ 差值更新网络参数。

$$Q^{new}(s_1, a_1) = r_1 + \gamma \max_{a'} Q(s_2, a') - Q^{old}(s_1, a_1)$$



相关库的简介



tensorflow库

- TensorFlow是谷歌2015年开源的一个人工智能学习系统。主要目的是方便研究人员开展机器学习和深度神经网络方面的研究，目前这个系统更具有通用性，也可广泛用于其他计算领域。
- Tensorflow 支持多种前端语言，包括Python（Python也是tensorflow支持最好的前端语言），因此一般大家利用python实现对tensorflow的调用。

OpenCV库

- OpenCV是一个开源的跨平台的计算机视觉库，实现了大量的图像处理和计算机视觉方面的通用算法。
 - 本实验采用opencv对采集的游戏画面进行预处理。

PyGame库

- Pygame是一个跨平台的模块，专为电子游戏设计。
- Pygame相当于是一款游戏引擎，用户无需编写大量的基础模块，而只需完成游戏逻辑本身就可以了。
- 本实验游戏模拟器采用Pygame实现。



相关库安装

tensorflow库安装

在确保网络通畅的情况下，打开windows的DOS命令行窗口，使用pip命令安装：

```
pip install tensorflow
```

OpenCV库安装

在下载地址中找到opencv的
相关下载链接，依据Python的
具体版本下载对应的文件。

OpenCV, a real time computer vision library.

[opencv_python-2.4.13.2-cp27-cp27m-win32.whl](#)

[opencv_python-2.4.13.2-cp27-cp27m-win_amd64.whl](#)

[opencv_python-3.1.0-cp27-cp27m-win32.whl](#)

[opencv_python-3.1.0-cp27-cp27m-win_amd64.whl](#)

[opencv_python-3.1.0-cp34-cp34m-win32.whl](#)

[opencv_python-3.1.0-cp34-cp34m-win_amd64.whl](#)

[opencv_python-3.2.0+contrib-cp35-cp35m-win32.whl](#)

[opencv_python-3.2.0+contrib-cp35-cp35m-win_amd64.whl](#)

[opencv_python-3.2.0+contrib-cp36-cp36m-win32.whl](#)

[opencv_python-3.2.0+contrib-cp36-cp36m-win_amd64.whl](#)

[opencv_python-3.2.0-cp35-cp35m-win32.whl](#)

[opencv_python-3.2.0-cp35-cp35m-win_amd64.whl](#)

[opencv_python-3.2.0-cp36-cp36m-win32.whl](#)

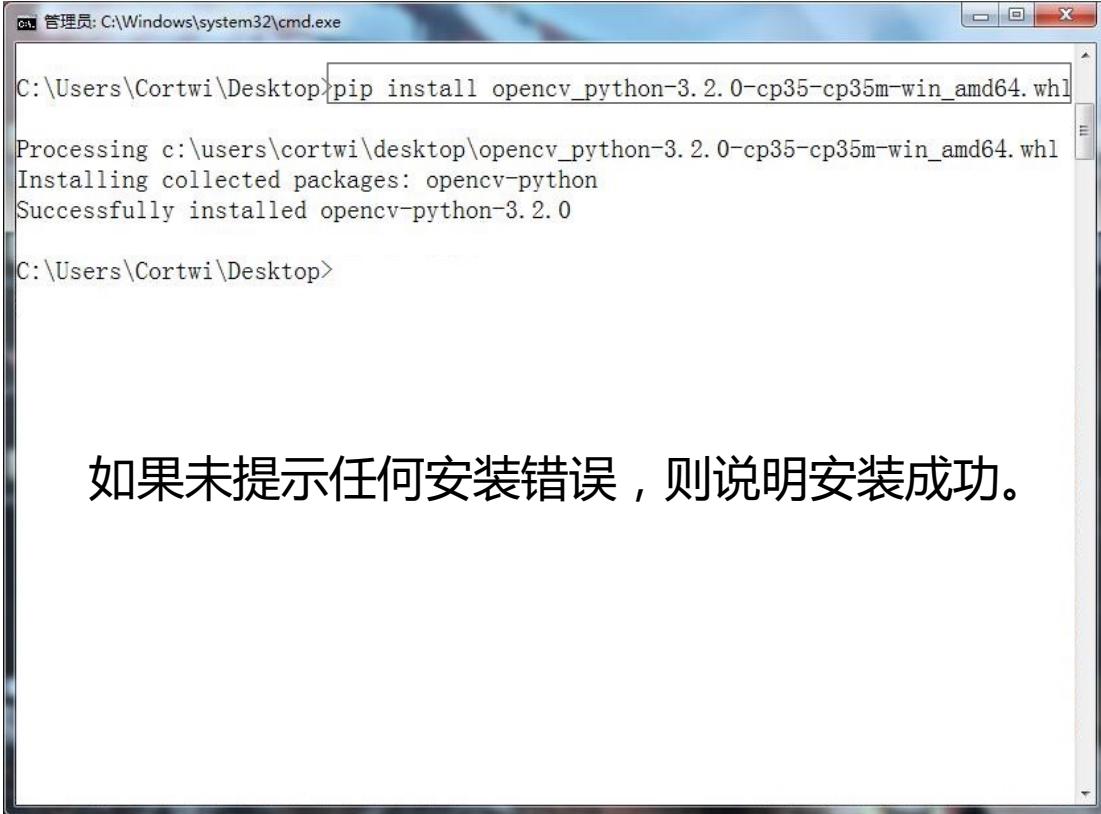
[opencv_python-3.2.0-cp36-cp36m-win_amd64.whl](#)

OpenCV库的安装

找到下载的文件的路径，打开windows的DOS命令行窗口，使用如下命令：

```
pip install opencv_python-3.2.0-cp35-cp35m-win_amd64.whl
```

OpenCV库安装



```
管理员: C:\Windows\system32\cmd.exe

C:\Users\Cortwi\Desktop>pip install opencv_python-3.2.0-cp35-cp35m-win_amd64.whl

Processing c:\users\cortwi\desktop\opencv_python-3.2.0-cp35-cp35m-win_amd64.whl
Installing collected packages: opencv-python
Successfully installed opencv-python-3.2.0

C:\Users\Cortwi\Desktop>
```

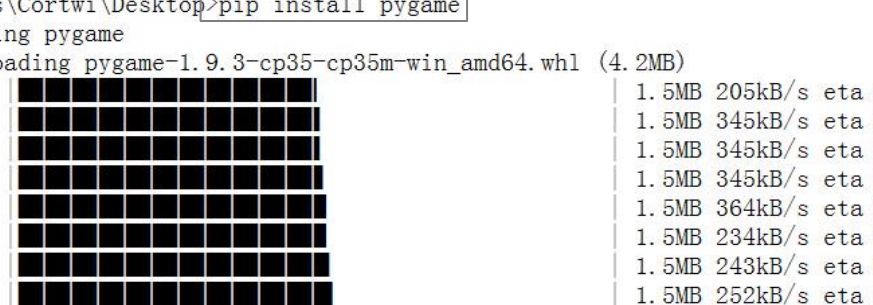
如果未提示任何安装错误，则说明安装成功。

Pygame库安装

在确保网络通畅的情况下，打开windows的DOS命令行窗口，使用如下命令：

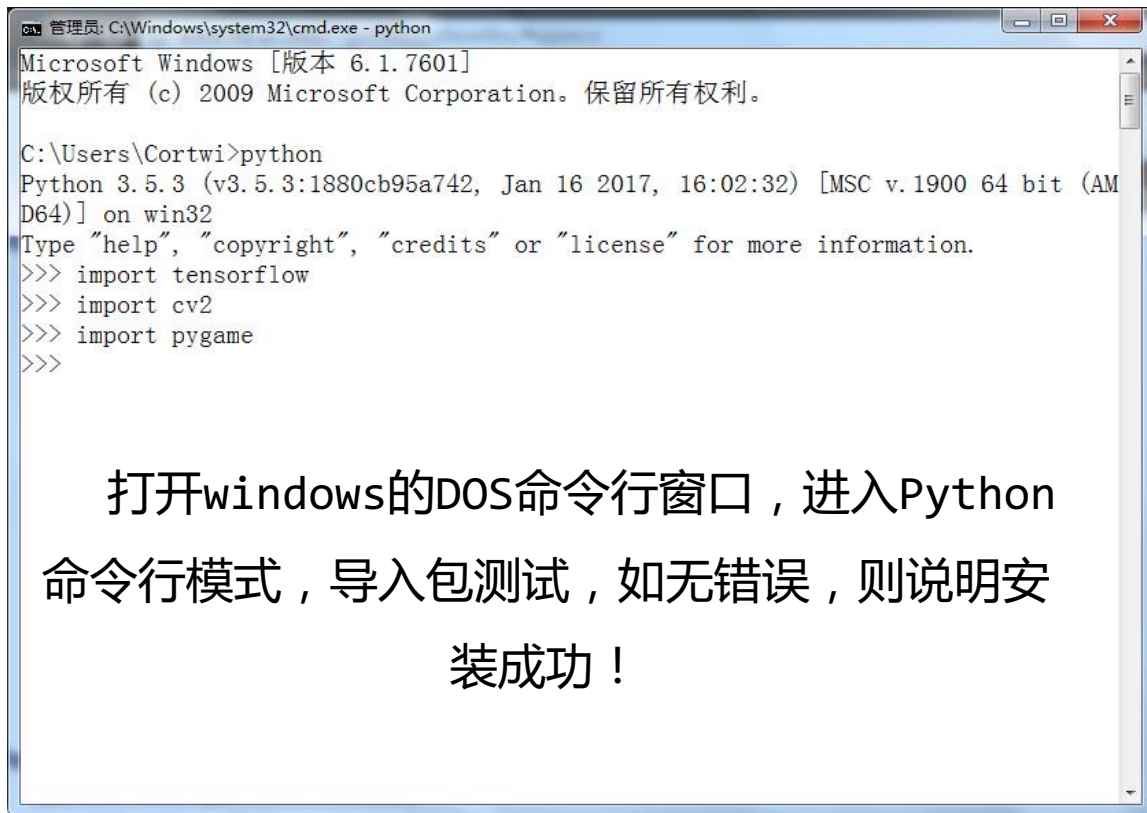
```
pip install pygame
```


Pygame库安装



```
管理员: C:\Windows\system32\cmd.exe
C:\Users\Cortwi\Desktop>pip install pygame
Collecting pygame
  Downloading pygame-1.9.3-cp35-cp35m-win_amd64.whl (4.2MB)
    34% [#####] 1.5MB 205kB/s eta 0:00:14
    35% [#####] 1.5MB 345kB/s eta 0:00:08
    35% [#####] 1.5MB 345kB/s eta 0:00:08
    35% [#####] 1.5MB 345kB/s eta 0:00:08
    35% [#####] 1.5MB 364kB/s eta 0:00:08
    36% [#####] 1.5MB 234kB/s eta 0:00:12
    36% [#####] 1.5MB 243kB/s eta 0:00:12
    36% [#####] 1.5MB 252kB/s eta 0:00:11
    36% [#####] 1.6MB 218kB/s eta 0:00:13
    37% [#####] 1.6MB 298kB/s eta 0:00:09
    37% [#####] 1.6MB 328kB/s eta 0:00:09
    37% [#####] 1.6MB 328kB/s eta 0:00:09
    37% [#####] 1.6MB 364kB/s eta 0:00:08
    38% [#####] 1.6MB 226kB/s eta 0:00:1
    38% [#####] 1.6MB 226kB/s eta 0:00:1
    38% [#####] 1.6MB 345kB/s eta 0:00:0
    38% [#####] 1.6MB 298kB/s eta 0:00:0
    38% [#####] 1.6MB 298kB/s eta 0:00:0
    39% [#####] 1.7MB 312kB/s eta 0:00:0
    39% [#####] 1.7MB 312kB/s eta 0:00:0
    39% [#####] 1.7MB 364kB/s eta 0:00:0
```

测试



```
管理员: C:\Windows\system32\cmd.exe - python
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Cortwi>python
Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017, 16:02:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow
>>> import cv2
>>> import pygame
>>>
```

打开windows的DOS命令行窗口，进入Python
命令行模式，导入包测试，如无错误，则说明安
装成功！



项目实战

tensorflow基本使用

理解TensorFlow:

- 使用图(graph)来表示计算任务；
- 在被称之为会话(Session)的上下文(context)中执行图；
- 使用tensor (张量) 表示数据；
- 通过变量(Variable)维护状态；
- 使用feed和fetch可以为任意的操作(arbitrary operation)赋值或者从其中获取数据。

tensorflow基本使用

- TensorFlow是一个编程系统，使用图来表示计算任务。图中的节点被称作Op（Operation），op可以获得0个或多个tensor，产生0个或多个tensor。每个tensor是一个类型化的多维数组。例如：可以将一组图像集表示成一个四维的浮点数组，四个维度分别是[batch, height, weight, channels]。
- 图（graph）描述了计算的过程。为了进行计算，图必须在会话中启动，会话负责将图中的op分发到CPU或GPU上进行计算，然后将产生的tensor返回。在Python中，tensor就是numpy.ndarray对象。

tensorflow基本使用

- TensorFlow程序通常被组织成两个阶段：构建阶段和执行阶段。
 - 构建阶段:op的执行顺序被描述成一个图；
 - 执行阶段:使用会话执行图中的op。
 - 例如：通常在构建阶段创建一个图来表示神经网络，在执行阶段反复执行图中的op训练神经网络。

tensorflow基本使用

实例1：

```
>>> import tensorflow as tf    #导入tensorflow库
>>> mat1 = tf.constant([[3., 3.]]) #创建一个1*2的矩阵
>>> mat2 = tf.constant([[2.],[2.]]) #创建一个2*1的矩阵
>>> product = tf.matmul(mat1, mat2) #创建op执行两个矩阵的乘法
>>> sess = tf.Session()         #启动默认图
>>> res = sess.run(product)     #在默认图中执行op操作
>>> print(res)                  #输出乘积结果
[[ 12.]]
>>> sess.close()                #关闭session
```

tensorflow基本使用

交互式会话 (InteractiveSession) :

为了方便使用Ipython之类的Python交互环境，可以使用交互式会话 (InteractiveSession) 来代替Session，使用类似Tensor.run()和Operation.eval()来代替Session.run()，避免使用一个变量来持有会话。

tensorflow基本使用

实例2：

```
>>> import tensorflow as tf    #导入tensorflow库
>>> sess = tf.InteractiveSession() #创建交互式会话
>>> a = tf.Variable([1.0, 2.0]) #创建变量数组
>>> b = tf.constant([3.0, 4.0]) #创建常量数组
>>> sess.run(tf.global_variables_initializer()) #变量初始化
>>> res = tf.add(a, b)          #创建加法操作
>>> print(res.eval())          #执行操作并输出结果
[4. 5.]
```

tensorflow基本使用

Feed操作：

前面的例子中，数据均以变量或常量的形式进行存储。Tensorflow还提供了Feed机制，该机制可以临时替代图中任意操作中的tensor。最常见的用例是使用`tf.placeholder()`创建占位符，相当于是作为图中的输入，然后使用Feed机制向图中占位符提供数据进行计算，具体使用方法见接下来的样例。

tensorflow基本使用

实例3：

```
>>> import tensorflow as tf    #导入tensorflow库
>>> sess = tf.InteractiveSession() #创建交互式会话
>>> input1 = tf.placeholder(tf.float32) #创建占位符
>>> input2 = tf.placeholder(tf.float32) #创建占位符
>>> res = tf.mul(input1, input2)      #创建乘法操作
>>> res.eval(feed_dict={input1:[7.], input2:[2.]}) #求值
array([ 14.], dtype=float32)
```



自主学习flappy bird实例程序编写

实例程序编写

1. 建立工程，导入相关工具包：

```
>>> import tensorflow as tf    #导入tensorflow库
>>> import cv2                 #导入opencv库
>>> import sys                  #导入sys模块
>>> sys.path.append("game/")    #添加game目录到系统环境变量
>>> import wrapped_flappy_bird as game #加载游戏
>>> import random               #加载随机模块
>>> import numpy as np          #加载numpy模块
>>> from collections import deque #导入双端队列
```

实例程序编写

2. 设置超参数：

```
>>> GAME = 'bird'           #设置游戏名称
>>> ACTIONS = 2              #设置游戏动作数目（点击不点击屏幕）
>>> GAMMA = 0.99             #设置增强学习更新公式中的累计折扣因子
>>> OBSERVE = 10000.         #观察期 1万次迭代（随机指定动作获得D）
>>> EXPLORE = 2000000.       #探索期
>>> FINAL_EPSILON = 0.0001   #设置 $\epsilon$ 的最终最小值
>>> INITIAL_EPSILON = 0.1    #设置 $\epsilon$ 贪心策略中的 $\epsilon$ 初始值
>>> REPLAY_MEMORY = 50000    #设置Replay Memory的容量
>>> BATCH = 32               #设置每次网络参数更新时用的样本数目
>>> FRAME_PER_ACTION = 1     #设置几帧图像进行一次动作
```

实例程序编写

3. 创建深度神经网络：

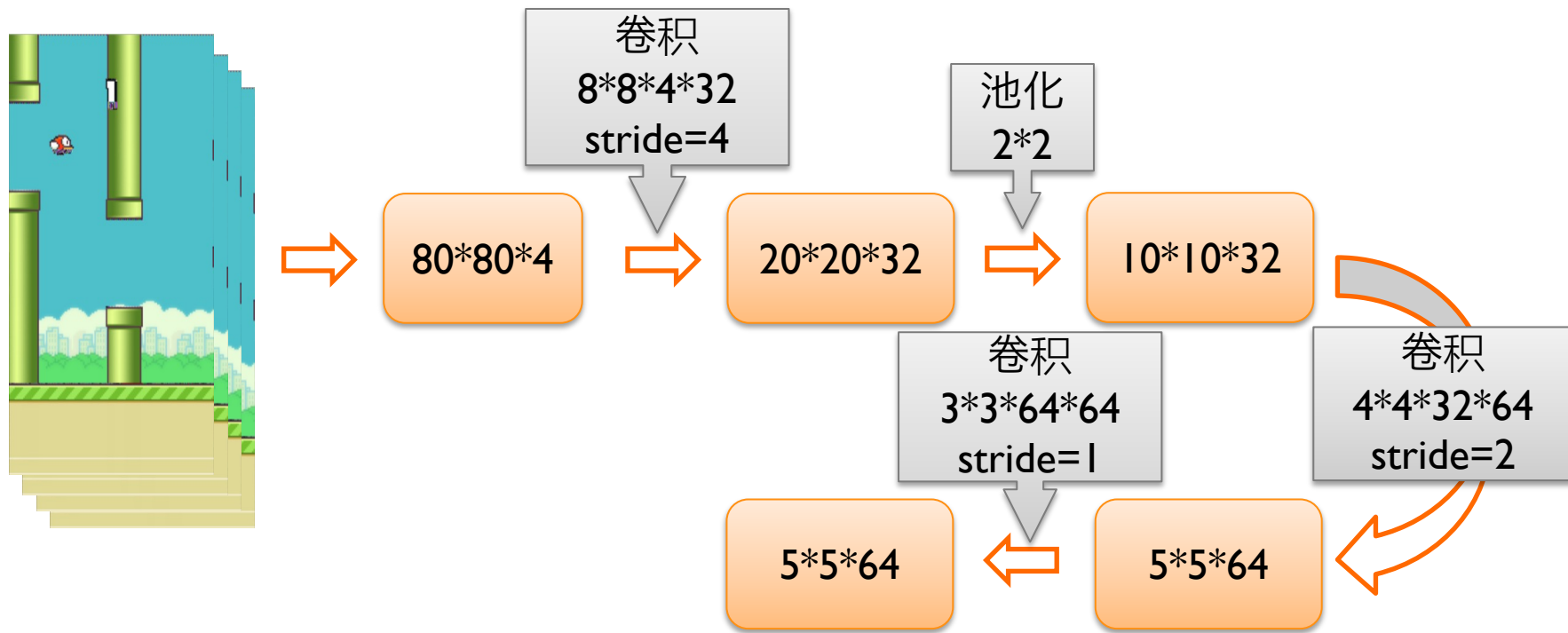
```
>>> def weight_variable(shape):  
    initial = tf.truncated_normal(shape, stddev = 0.01)  
    return tf.Variable(initial)  
#首先定义一个函数，该函数用于生成形状为shape的张量（高维数组）  
#张量中的初始化数值服从正太分布，且方差为0.01  
>>> def bias_variable(shape):  
    initial = tf.constant(0.01, shape = shape)  
    return tf.Variable(initial)  
#定义另外一个函数，用于生成偏置项，初始值为0.01
```

实例程序编写

3. 创建深度神经网络：

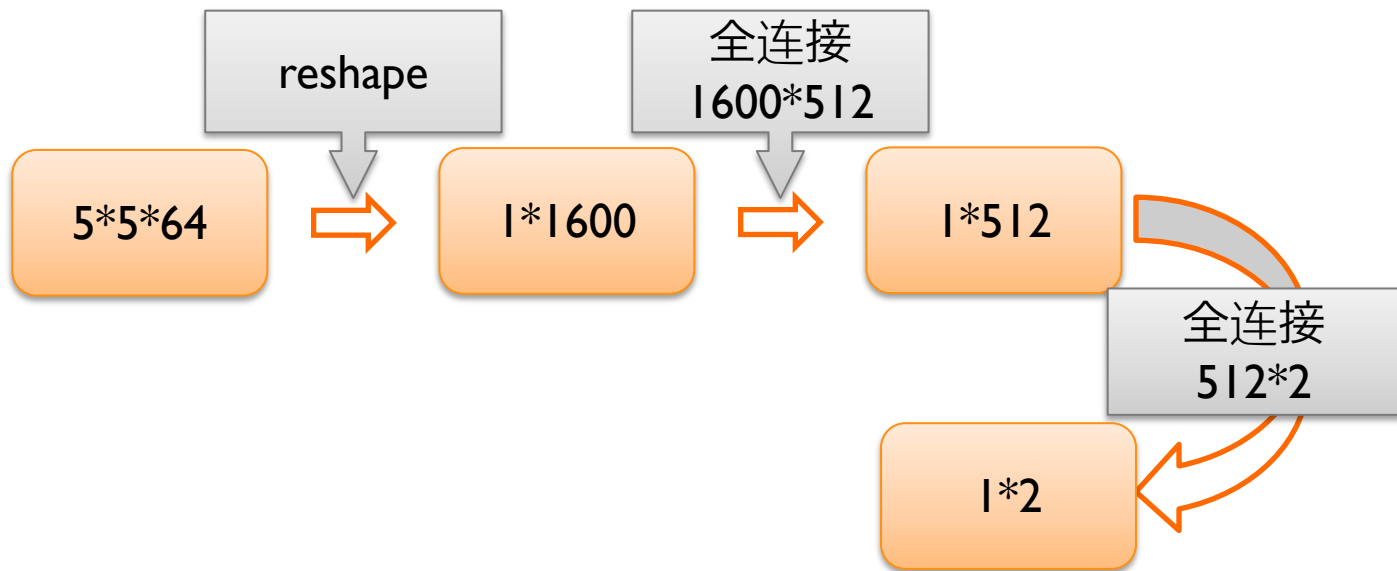
```
>>> def conv2d(x, W, stride):  
    return tf.nn.conv2d(x, W, strides=[1,stride,stride,1],  
padding = "SAME")  
#定义卷积操作，实现卷积核W在数据x上卷积操作  
#strides为卷积核的移动步长，padding为卷积的一种模式，参数为same  
表示滑动范围超过边界时，超过的部分进行补零  
  
>>> def max_pool_2x2(x):  
    return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,  
2,1], padding = "SAME")  
#定义池化函数，此程序中通过调用max_pool执行最大池化操作，大小为  
2*2，stride步长为2.
```


深度神经网络-框架回顾



本实验中使用的深度神经网络结构就是多个卷积操作和池化操作的累加。

深度神经网络-框架回顾



输出分别对应于两个动作，即不做操作和跳跃的状态动作值函数。

实例程序编写

3. 创建深度神经网络，定义网络结构：

```
def createNetwork():  
    #定义深度神经网络的参数和偏置  
    W_conv1 = weight_variable([8, 8, 4, 32])  
    b_conv1 = bias_variable([32])  
  
    W_conv2 = weight_variable([4, 4, 32, 64])  
    b_conv2 = bias_variable([64])  
  
    W_conv3 = weight_variable([3, 3, 64, 64])  
    b_conv3 = bias_variable([64])  
  
    W_fc1 = weight_variable([1600, 512])  
    b_fc1 = bias_variable([512])  
  
    W_fc2 = weight_variable([512, ACTIONS])  
    b_fc2 = bias_variable([ACTIONS])
```

第一个卷积层

第二个卷积层

第三个卷积层

第一个全连接层

第二个全连接层

实例程序编写

创建深度神经网络：

```
# 输入层
s = tf.placeholder("float", [None, 80, 80, 4])

# 隐藏层
h_conv1 = tf.nn.relu(conv2d(s, W_conv1, 4) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2, 2) + b_conv2)
h_conv3 = tf.nn.relu(conv2d(h_conv2, W_conv3, 1) + b_conv3)
h_conv3_flat = tf.reshape(h_conv3, [-1, 1600])

h_fc1 = tf.nn.relu(tf.matmul(h_conv3_flat, W_fc1) + b_fc1)

# 输出层
readout = tf.matmul(h_fc1, W_fc2) + b_fc2

return s, readout, h_fc1
```

输入层，placeholder用于占位，
可用作网络的输入

对各层进行连接

实例程序编写

4. 训练深度神经网络-1：

```
def trainNetwork(s, readout, h_fc1, sess):  
    # 定义损失函数  
    a = tf.placeholder("float", [None, ACTIONS])  
    y = tf.placeholder("float", [None])  
    readout_action = tf.reduce_sum(tf.multiply(readout, a),  
                                   reduction_indices=1)  
    cost = tf.reduce_mean(tf.square(y - readout_action))  
    train_step = tf.train.AdamOptimizer(1e-6).minimize(cost)  
  
    # 开启游戏模拟器，会打开一个模拟器的窗口，实时显示游戏的信息  
    game_state = game.GameState()  
  
    # 创建双端队列用于存放replay memory  
    D = deque()
```

定义神经网络训练函数；
定义损失函数。

4. 训练深度神经网络-2：

```
# 获取游戏的初始状态，设置动作为不执行跳跃，并将初始状态修改成80*80*4大小
do_nothing = np.zeros(ACTIONS)
do_nothing[0] = 1
x_t, r_0, terminal = game_state.frame_step(do_nothing)
x_t = cv2.cvtColor(cv2.resize(x_t, (80, 80)), cv2.COLOR_BGR2GRAY)
ret, x_t = cv2.threshold(x_t, 1, 255, cv2.THRESH_BINARY)
s_t = np.stack((x_t, x_t, x_t, x_t), axis=2)

# 用于加载或保存网络参数
saver = tf.train.Saver()
sess.run(tf.initialize_all_variables())
checkpoint = tf.train.get_checkpoint_state("saved_networks")
if checkpoint and checkpoint.model_checkpoint_path:
    saver.restore(sess, checkpoint.model_checkpoint_path)
    print("Successfully loaded:", checkpoint.model_checkpoint_path)
else:
    print("Could not find old network weights")
```

将像素值大于等于1的
像素点处理成255，也
就是黑白二值图

这里需要使用
Opencv对图
像进行预处理

实例程序编写

4. 训练深度神经网络-2：

```
# 开始训练
epsilon = INITIAL_EPSILON
t = 0
while "flappy bird" != "angry bird":
    # 使用epsilon贪心策略选择一个动作
    readout_t = readout.eval(feed_dict={s : [s_t]})[0]
    a_t = np.zeros([ACTIONS])
    action_index = 0
    if t % FRAME_PER_ACTION == 0:
        # 执行一个随机动作
        if random.random() <= epsilon:
            print("-----Random Action-----")
            action_index = random.randrange(ACTIONS)
            a_t[random.randrange(ACTIONS)] = 1
        # 由神经网络计算的Q(s,a)值选择对应的动作
    else:
        action_index = np.argmax(readout_t)
        a_t[action_index] = 1
    else:
        a_t[0] = 1 # 不执行跳跃动作
```


实例程序编写

4. 训练深度神经网络-2：

```
# 随游戏的进行，不断降低epsilon，减少随机动作
if epsilon > FINAL_EPSILON and t > OBSERVE:
    epsilon -= (INITIAL_EPSILON - FINAL_EPSILON) / EXPLORE

# 执行选择的动作，并获得下一状态及回报
x_t1_colored, r_t, terminal = game_state.frame_step(a_t)
x_t1 = cv2.cvtColor(cv2.resize(x_t1_colored, (80, 80)),
                    cv2.COLOR_BGR2GRAY)
ret, x_t1 = cv2.threshold(x_t1, 1, 255, cv2.THRESH_BINARY)
x_t1 = np.reshape(x_t1, (80, 80, 1))
s_t1 = np.append(x_t1, s_t[:, :, :3], axis=2)

# 将状态转移过程存储到D中，用于更新参数时采样
D.append((s_t, a_t, r_t, s_t1, terminal))
if len(D) > REPLAY_MEMORY:
    D.popleft()
```


4. 训练深度神经网络-2 :

```
# 过了观察期，才会进行网络参数的更新
if t > OBSERVE:
    # 从D中随机采样，用于参数更新
    minibatch = random.sample(D, BATCH)

    # 分别将当前状态、采取的动作、获得的回报、下一状态分组存放
    s_j_batch = [d[0] for d in minibatch]
    a_batch = [d[1] for d in minibatch]
    r_batch = [d[2] for d in minibatch]
    s_j1_batch = [d[3] for d in minibatch]

    #计算Q(s,a)的新值
    y_batch = []
    readout_j1_batch = readout.eval(feed_dict = {s : s_j1_batch})
    for i in range(0, len(minibatch)):
        terminal = minibatch[i][4]
        # 如果游戏结束，则只有反馈值
        if terminal:
            y_batch.append(r_batch[i])
        else:
            y_batch.append(r_batch[i] +
                           GAMMA * np.max(readout_j1_batch[i]))

    # 使用梯度下降更新网络参数
    train_step.run(feed_dict = {
        y : y_batch,
        a : a_batch,
        s : s_j_batch
    })
```

实例程序编写

4. 训练深度神经网络-2：

```
# 状态发生改变，用于下次循环
s_t = s_t1
t += 1

# 每进行10000次迭代，保留一下网络参数
if t % 10000 == 0:
    saver.save(sess, 'saved_networks/' + GAME + '-dqn', global_step=t)

# 打印游戏信息
state = ""
if t <= OBSERVE:
    state = "observe"
elif t > OBSERVE and t <= OBSERVE + EXPLORE:
    state = "explore"
else:
    state = "train"

print("TIMESTEP", t, "/ STATE", state, \
      "/ EPSILON", epsilon, "/ ACTION", action_index, "/ REWARD", r_t, \
      "/ Q_MAX %e" % np.max(readout_t))
```

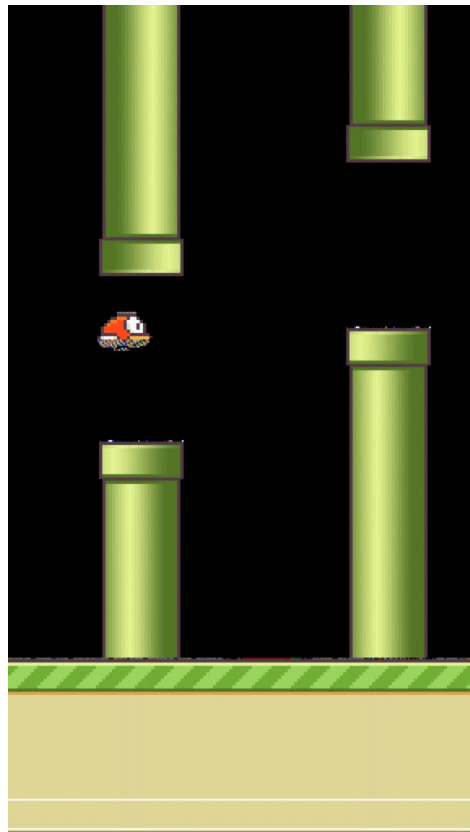
实例程序编写

5. 开启整个训练过程：

```
def playGame():  
    sess = tf.InteractiveSession()  
    s, readout, h_fc1 = createNetwork()  
    trainNetwork(s, readout, h_fc1, sess)  
  
if __name__ == "__main__":  
    playGame()
```

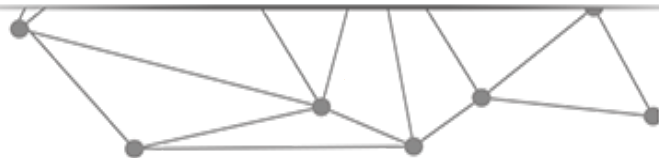
前面已经定义深度神经网络创建函数和网络训练函数，开启训练过程

结果展示





训练结果分析



训练参数加载

- 参数训练完成之后，修改程序中的超参数INITIAL_EPSILON=0，即不使用随机动作，直接由神经网络输出动作。
- saved_networks文件夹下，保存了最近几次检查点保留的网络参数，只需使用tf.train.Saver()加载参数就可以使用了。

bird-dqn-2530000.data-00000-of-00...	2017/4/19 17:52	DATA-00000-OF...	10,532 KB
bird-dqn-2530000.index	2017/4/19 17:52	INDEX 文件	2 KB
bird-dqn-2530000.meta	2017/4/19 17:52	META 文件	78 KB
bird-dqn-2540000.data-00000-of-00...	2017/4/19 18:03	DATA-00000-OF...	10,532 KB
bird-dqn-2540000.index	2017/4/19 18:03	INDEX 文件	2 KB
bird-dqn-2540000.meta	2017/4/19 18:03	META 文件	78 KB
bird-dqn-2550000.data-00000-of-00...	2017/4/19 18:15	DATA-00000-OF...	10,532 KB
bird-dqn-2550000.index	2017/4/19 18:15	INDEX 文件	2 KB
bird-dqn-2550000.meta	2017/4/19 18:15	META 文件	78 KB
bird-dqn-2560000.data-00000-of-00...	2017/4/19 18:27	DATA-00000-OF...	10,532 KB
bird-dqn-2560000.index	2017/4/19 18:27	INDEX 文件	2 KB
bird-dqn-2560000.meta	2017/4/19 18:27	META 文件	78 KB
bird-dqn-2570000.data-00000-of-00...	2017/4/19 18:39	DATA-00000-OF...	10,532 KB
bird-dqn-2570000.index	2017/4/19 18:39	INDEX 文件	2 KB
bird-dqn-2570000.meta	2017/4/19 18:39	META 文件	78 KB
checkpoint	2017/4/20 8:53	文件	1 KB

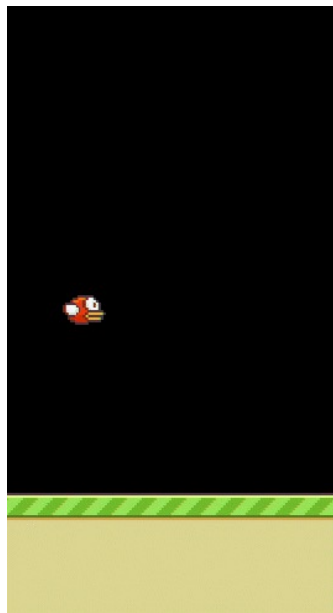
训练参数加载

修改checkpoint配置文件, 加载保存的网络参数, 默认加载最后一次保存的参数。修改完成, 运行程序即可。

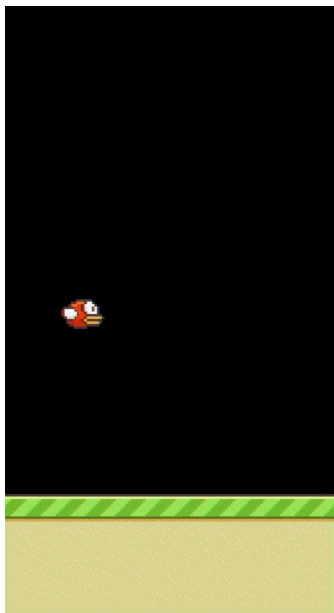
bird-dqn-2530000.data-00000-of-00...	2017/4/19 17:52	DATA-00000-OF...	10,532 KB
bird-dqn-2530000.index	2017/4/19 17:52	INDEX 文件	2 KB
bird-dqn-2530000.meta	2017/4/19 17:52	META 文件	78 KB
bird-dqn-2540000.data-00000-of-00...	2017/4/19 18:03	DATA-00000-OF...	10,532 KB
bird-dqn-2540000.index	2017/4/19 18:03	INDEX 文件	2 KB
bird-dqn-2540000.meta	2017/4/19 18:03	META 文件	78 KB
bird-dqn-2550000.data-00000-of-00...	2017/4/19 18:15	DATA-00000-OF...	10,532 KB
bird-dqn-2550000.index	2017/4/19 18:15	INDEX 文件	2 KB
bird-dqn-2550000.meta	2017/4/19 18:15	META 文件	78 KB
bird-dqn-2560000.data-00000-of-00...	2017/4/19 18:27	DATA-00000-OF...	10,532 KB
bird-dqn-2560000.index	2017/4/19 18:27	INDEX 文件	2 KB
bird-dqn-2560000.meta	2017/4/19 18:27	META 文件	78 KB
bird-dqn-2570000.data-00000-of-00...	2017/4/19 18:39	DATA-00000-OF...	10,532 KB
bird-dqn-2570000.index	2017/4/19 18:39	INDEX 文件	2 KB
bird-dqn-2570000.meta	2017/4/19 18:39	META 文件	78 KB
checkpoint	2017/4/20 8:53	文件	1 KB

model_checkpoint_path: "bird-dqn-2570000"
all model checkpoint paths: "bird-dqn-2570000"

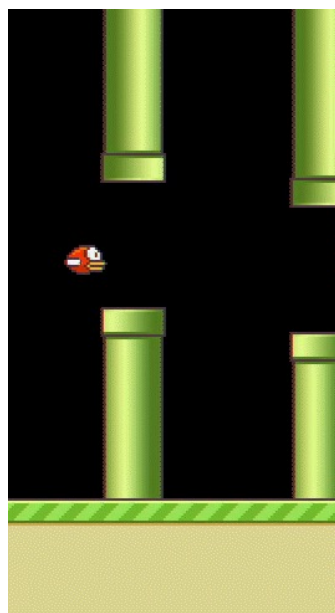
结果展示



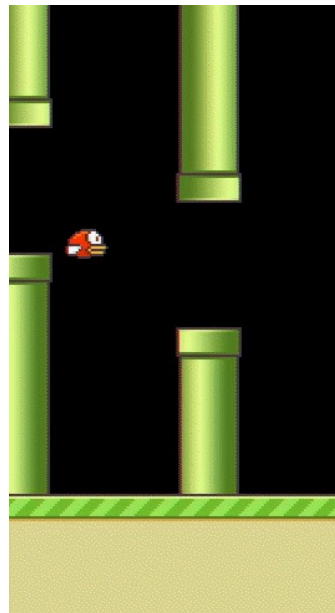
t=25万 2-3次



t=100万 24次



t=210万 死不掉



t=280万 死不掉

以上为不同迭代次数下，训练的神经网络能达到的游戏水平