**人工智能自主实验报告**

鲍锋雄 网工1701 201601390702

1. **选题介绍**

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。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。

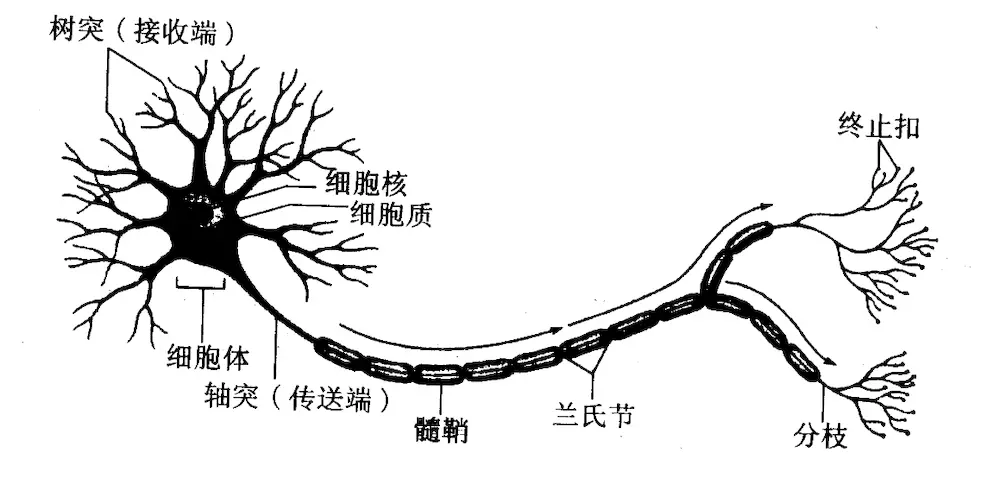
Flappy Bird是个极其简单又困难的游戏，风靡一时。它的游戏逻辑十分简单，小鸟只有两个动作，拍拍翅膀小鸟上升，不动翅膀，小鸟下落，玩家需要控制小鸟的动作来避开障碍物。

我们要做的是利用DQN来实现通过屏幕学习玩Flappy Bird，最终能实现小鸟不死的效果。

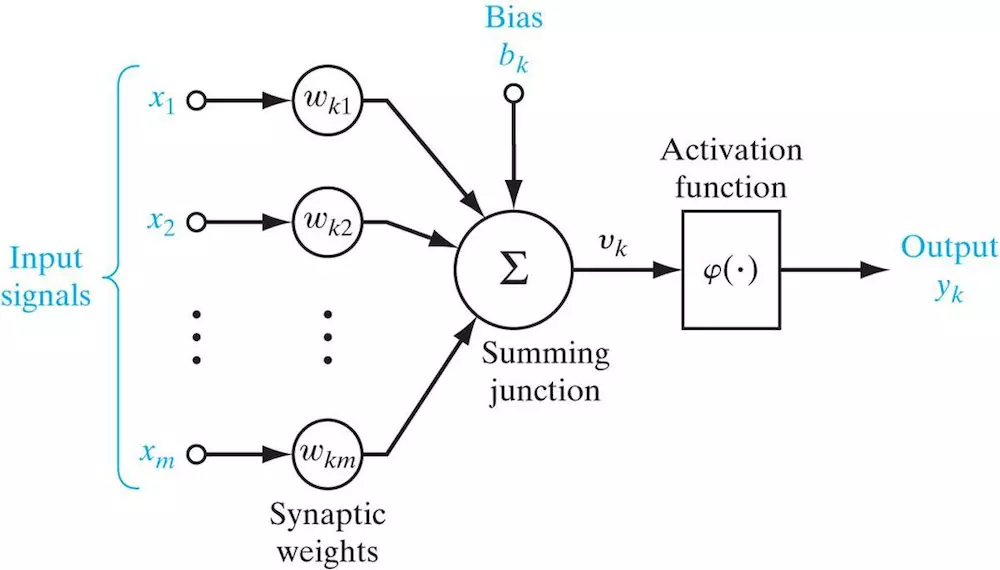
1. **所用到的人工智能知识**

本次课程实验我们组使用了卷积神经网络搭建模型来进行训练，并使用增强学习来作为训练网络的方法。

神经网络算法是由众多的神经元可调的连接权值连接而成，具有大规模并行处理、分布式信息存储、良好的自组织自学习能力等特点。人工神经元与生物神经元结构类似，其结构对比如下图所示



生物神经元



人工神经元

人工神经元的输入（x1,x2...xm）类似于生物神经元的树突，输入经过不同的权值（wk1, wk2, ....wkn），加上偏置，经过激活函数得到输出，最后将输出传输到下一层神经元进行处理。

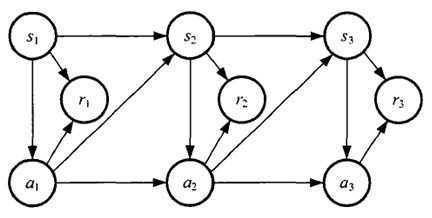
卷积神经网络（Convolutional Neural Network），简称CNN起源于动物的视觉系统，主要是用于图像识别、图像分类、对象检测、人脸识别等。

卷积神经网络与人类的视觉类似，采用局部感知，低层的神经元只负责感知局部的信息，在向后传输的过程中，高层的神经元将局部信息综合起来得到全局信息。每个卷积都是一种特征提取方式，那么对于整幅图像来讲，单个卷积核提取的特征肯定是不够的，那么对同一幅图像使用多种卷积核进行特征提取，就能得到多幅特征图（feature map）。得到特征图之后，可以使用提取到的特征去训练分类器，但依然会面临特征维度过多，难以计算，并且可能过拟合的问题。从图像识别的角度来讲，图像可能存在偏移、旋转等，但图像的主体却相同的情况。也就是不同的特征向量可能对应着相同的结果，那么池化就是解决这个问题的。

DQN属于机器学习中的强化学习的范畴。

强化学习过程有两个组成部分：

* **智能代理（学习系统）**
* **环境**

如图所示，在每步迭代过程中，首先智能代理（学习系统）接收环境的状态st，然后产生动作at作用于环境，环境接收动作at，并且对其进行评价，反馈给智能代理rt。不断的循环这个过程，就会产生一个状态/动作/反馈的序列：*（s1, a1, r1, s2, a2, r2.....,sn, an, rn）*

DQN的基础是Qlearning算法

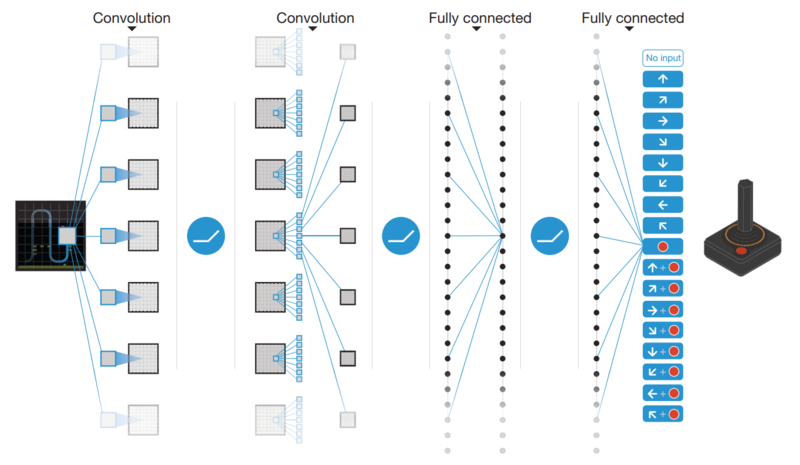
Q-Learning是根据值迭代的思路来进行学习的。该算法中，Q值更新的方法如下：

Q值更新方法

虽然根据值迭代计算出目标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，朝目标迈近一小步，取决于α，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

而DQN是Q-Learning的演化版本。

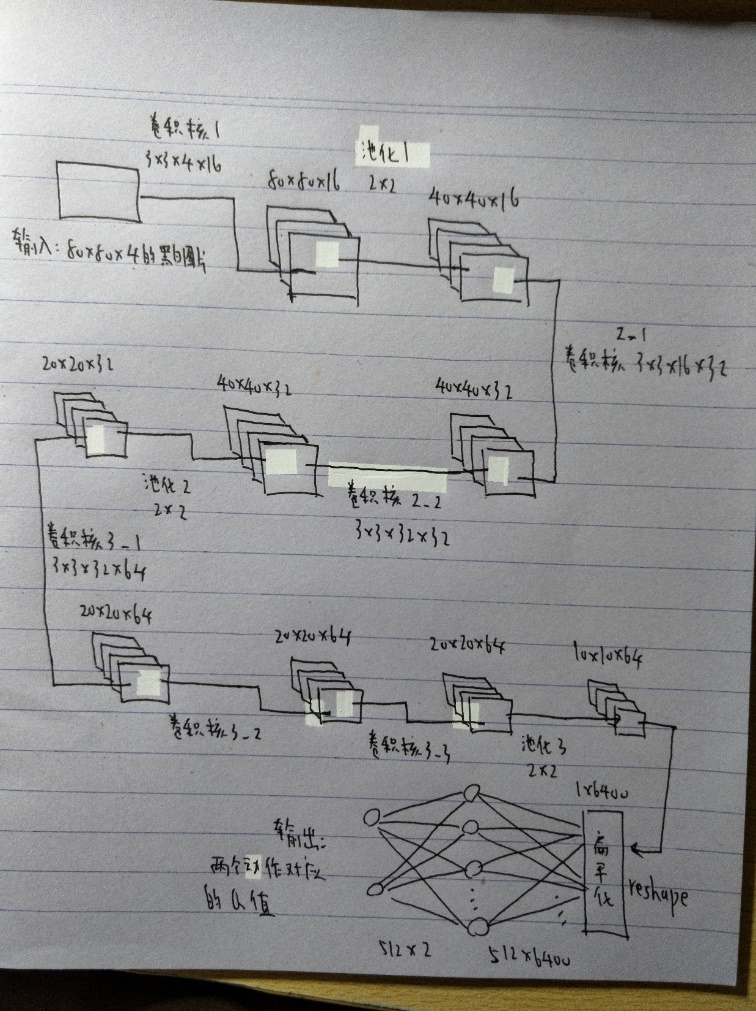
DQN采用值函数近似的方法，用函数的形式来代替庞大的Q表，其中的w1、w2、b参数与神经网络中的权值和偏置类似，因此， Q-learning可以与神经网络进行结合。用神经网络代替庞大的Q表。神经网络训练的过程其实就是一个最优化方程求解的过程，定义系统的损失函数，然后让损失函数最小化的过程。



1. **我负责的部分**

在本次课程实验中，我主要是负责卷积神经网络模型的搭建。搭建的模型具有一个输入层，三个连续卷积加池化层，一个全连接层和一个输出层。

网络结构具体如下图所示：



神经网络模型

可以看到神经网络分为4个层次，第一个层次是输入层，卷积层和池化层，第二个层次是连续的两个卷积层和池化层，第三个层次是三个连续的卷积层和池化层，最后一层就是全连接层和输出层

图片输入是80\*80大小，经过一层卷积，然后经过一层池化后变成40\*40大小，再经过两层卷积，然后经过一层池化后变成20\*20大小，再经过两层卷积，然后经过一层池化后变成10\*10大小，最后经过全连接层输出得到两个动作对应的Q值。

在这个CNN网络中，采用了3\*3卷积核 连续卷积堆叠的做法，为什么不采取大卷积核的做法而采用小卷积核堆叠的做法？这里涉及到感受野的大小，两个3\*3卷积核堆叠的感受野的大小等于一个5\*5的卷积核，3个3\*3卷积核堆叠的感受野的大小等于一个7\*7卷积核的大小，多个卷积核堆叠比一个大卷积有更好的非线性，使网络有更好的拟合能力，多个3\*3的卷积核，比一个大卷积核有更少的参数，训练起来也会更快。

搭建网络的代码：

1. # 搭建神经网络
2. **def** createNetwork():
3. # 输入层
4. # 定义80\*80\*4的输入层s（处理过的连续4帧的游戏图像）
5. s = tf.placeholder('float', [None, 80, 80, 4])
7. # 隐藏层
9. # 在达到相同感受野的情况下，卷积核越小，所需要的参数和计算量越小
10. # 卷积 池化过程
11. w\_conv1\_1 = get\_weight([3, 3, 4, 16])  # 第一个卷积核 3\*3 4通道 16核
12. b\_conv1\_1 = get\_bias([16])  # 偏置b
13. # 卷积
14. h\_conv1\_1 = tf.nn.relu(conv2d(s, w\_conv1\_1) + b\_conv1\_1)
15. # 池化
16. h\_pool1 = max\_pool\_2x2(h\_conv1\_1)
18. # 连续两个卷积+一个池化过程
19. w\_conv2\_1 = get\_weight([3, 3, 16, 32])
20. b\_conv2\_1 = get\_bias([32])
21. # 卷积2\_1
22. h\_conv2\_1 = tf.nn.relu(conv2d(h\_pool1, w\_conv2\_1) + b\_conv2\_1)
23. w\_conv2\_2 = get\_weight([3, 3, 32, 32])
24. b\_conv2\_2 = get\_bias([32])
25. h\_conv2\_2 = tf.nn.relu(conv2d(h\_conv2\_1, w\_conv2\_2) + b\_conv2\_2)
26. # 池化
27. h\_pool2 = max\_pool\_2x2(h\_conv2\_2)
29. # 连续三个卷积+一个池化过程
30. w\_conv3\_1 = get\_weight([3, 3, 32, 64])
31. b\_conv3\_1 = get\_bias([64])
32. # 卷积3\_1
33. h\_conv3\_1 = tf.nn.relu(conv2d(h\_pool2, w\_conv3\_1) + b\_conv3\_1)
34. w\_conv3\_2 = get\_weight([3, 3, 64, 64])
35. b\_conv3\_2 = get\_bias([64])
36. h\_conv3\_2 = tf.nn.relu(conv2d(h\_conv3\_1, w\_conv3\_2) + b\_conv3\_2)
37. w\_conv3\_3 = get\_weight([3, 3, 64, 64])
38. b\_conv3\_3 = get\_bias([64])
39. h\_conv3\_3 = tf.nn.relu(conv2d(h\_conv3\_2, w\_conv3\_3) + b\_conv3\_3)
40. # 池化
41. h\_pool3 = max\_pool\_2x2(h\_conv3\_3)  # 10\*10\*64
43. # 扁平化
44. # tf.reshape 函数原型为
45. # def reshape(tensor, shape, name=None)
46. # 第1个参数为被调整维度的张量 第2个参数为要调整为的形状
47. # 返回一个shape形状的新tensor
48. # 注意shape里最多有一个维度的值可以填写为 - 1，表示自动计算此维度
49. h\_conv3\_flat = tf.reshape(h\_pool3, [-1, 6400])  # 1\*6400
51. # 全连接层 得到小鸟每个动作对应的Q值out
52. # 第一层
53. w\_fc1 = get\_weight([6400, 512])
54. b\_fc1 = get\_bias([512])
55. h\_fc1 = tf.nn.relu(tf.matmul(h\_conv3\_flat, w\_fc1) + b\_fc1)
57. # 输出层
58. W\_fc2 = get\_weight([512, ACTIONS])
59. b\_gc2 = get\_bias([ACTIONS])
60. out = tf.matmul(h\_fc1, W\_fc2) + b\_gc2  # 第二层输出：结果out 前向传播
62. **return** s, out  # 返回输入和网络输出

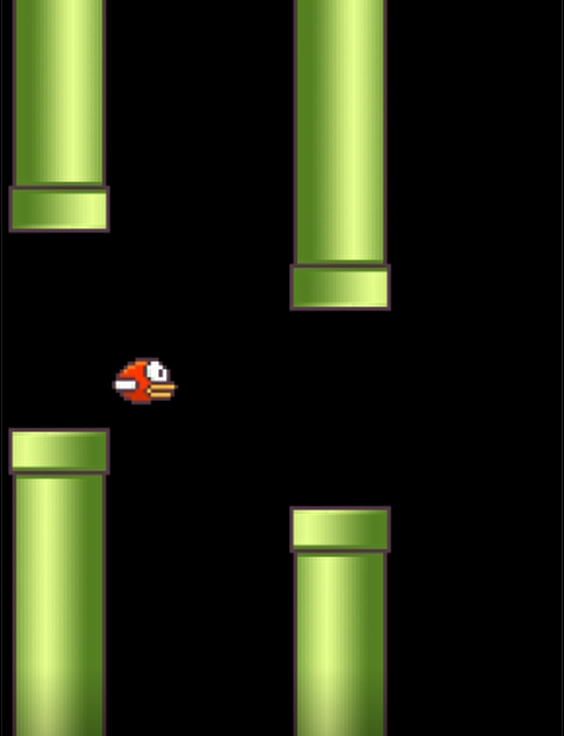
反向传播部分：

1. # 定义损失函数
2. x = tf.placeholder(float, [None, ACTIONS])
3. y = tf.placeholder(float, [None])
4. out\_action = tf.reduce\_sum(tf.multiply(out, x), reduction\_indices=1)  # Q估计
5. # 计算实际和预测结果的均方误差
6. loss = tf.reduce\_mean(tf.square(y - out\_action))  # Q现实-Q估计
7. # 定义反向传播方法
8. # 学习率：决定参数每次更新的幅度 1e-6
9. train\_step = tf.train.AdamOptimizer(1e-6).minimize(loss)  # Adam优化器
10. 实验结果

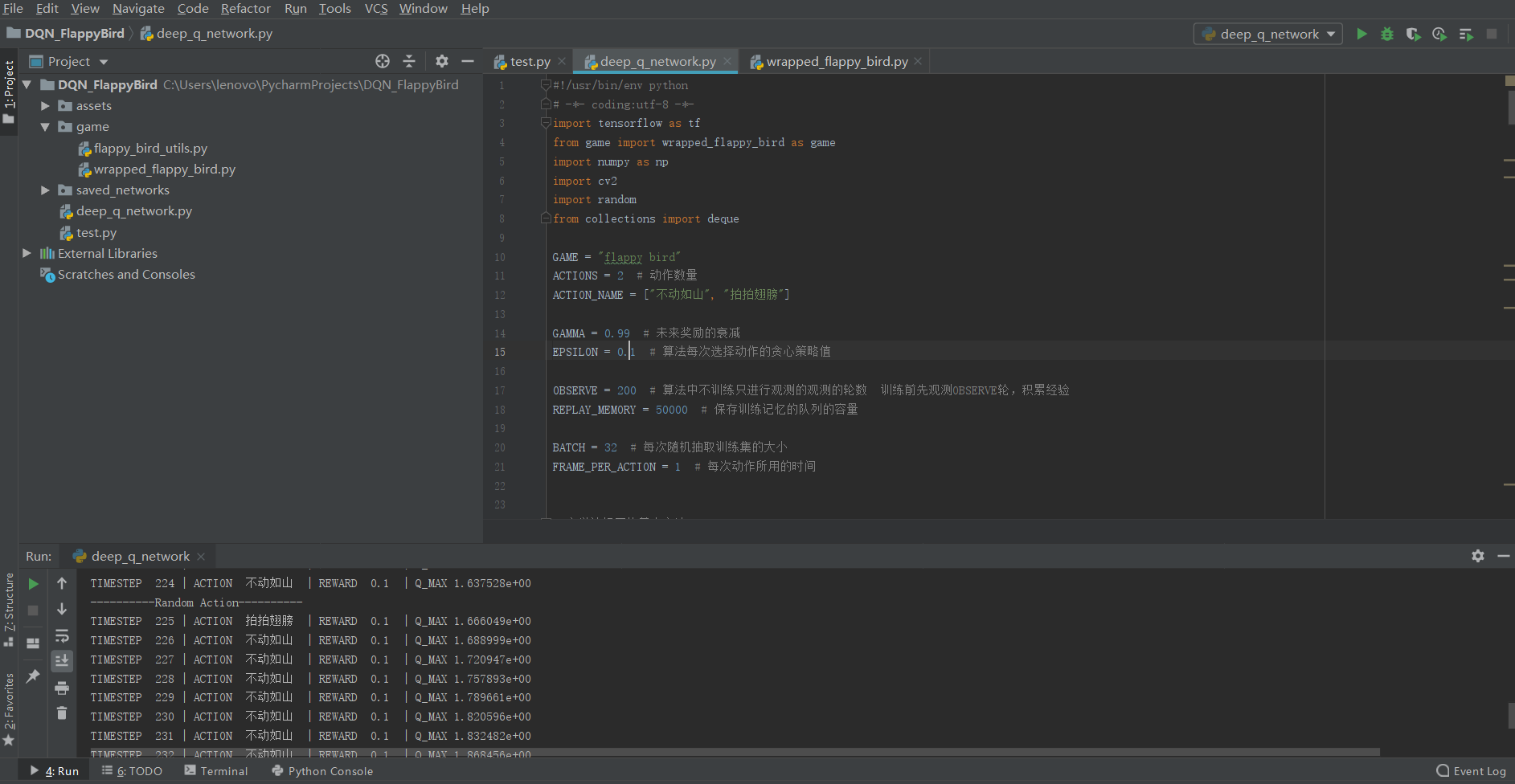
预期达到的效果为小鸟能在没有人干预的情况下，自主的通过所有障碍物，实现不死的效果。

但在我们实际训练过程中，由于网络搭的比较大，实际所需计算量十分巨大，计算机训练一轮大概需要耗费将近一秒的时间，而据估计训练完得到一个完善的网络大概需要训练50万轮以上，显然以我们组的设备水平无法完成这么庞大的训练。

所以我们训练了大概3000轮，来观察所搭网络的效果



小鸟已经可以通过自主学习通过第一个障碍物



最大Q值不断上升，说明小鸟在不断学习。

1. **实验体会**

通过这次课程设计，我更加深入的了解了卷积神经网络的搭建和传播过程，通过自己动手搭建实现网络，将课本上学到的理论知识与代码实践紧密结合在一起，提高了自己的能力，并且让我对人工智能在现实生活中的应用有了更好的了解。