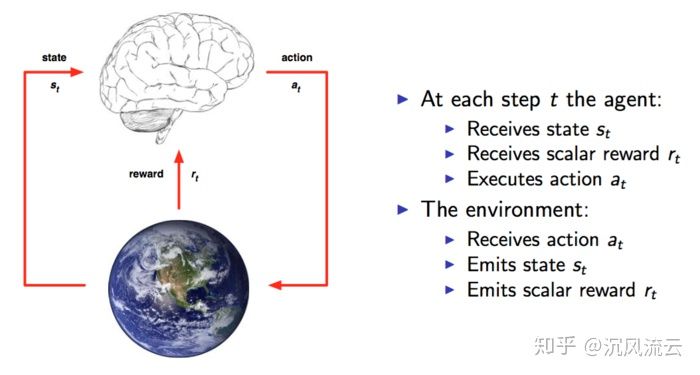
技术概述：

强化学习：

强化学习（英语：Reinforcement learning，简称RL）是机器学习中的一个领域，强调如何基于环境而行动，以取得最大化的预期利益。其灵感来源于心理学中的行为主义理论，即有机体如何在环境给予的奖励或惩罚的刺激下，逐步形成对刺激的预期，产生能获得最大利益的习惯性行为。这个方法具有普适性，因此在其他许多领域都有研究，例如博弈论、控制论、运筹学、信息论、仿真优化、多主体系统学习、群体智能、统计学以及遗传算法。在运筹学和控制理论研究的语境下，强化学习被称作“近似动态规划”（approximate dynamic programming，ADP）。在最优控制理论中也有研究这个问题，虽然大部分的研究是关于最优解的存在和特性，并非是学习或者近似方面。在经济学和博弈论中，强化学习被用来解释在有限理性的条件下如何出现平衡。



机器接收到当前的状态S,和外部的奖赏r，做出反应动作a；外部环境接收这次的动作a，环境根据这次动作生成状态s,并做出这次的奖赏r。机器不断的根据外部环境的反馈做出更好的判断，生成更合理的行动，产生更聪明的决策。

Q-learning：

Q-learning算法的主要思想是通过建立一个Q-Table的结构，也就是记录每个状态和行为的Q值（可以理解为一种评分），通过不断的训练，可以得到在任何一个状态下，如果做出每个行为的Q值是多少，这样就可以快速的选择出好的行为。

其中Q值的更新，可以看做是加入学习率α的更新方式，具体公式如下：

preview

其中α为学习率，γ为奖励性衰变系数，采用时间差分法的方法进行更新。

Q-learning存在的问题：

（1）Q-learning需要一个Q table，在状态很多的情况下，Q table会很大，查找和存储都需要消耗大量的时间和空间。因此如何减少状态空间是使用Q-learning的重点。

（2）Q-learning存在过高估计的问题。因为Q-learning在更新Q函数的时候使用的是下一时刻最优值对应的action，这样就会导致“过高”的估计采样过的action，而对于没有采样到的action，便不会被选择为最优的action。

具体实现：

研究方法：本代码采用强化学习、Q-learning的方法实现机器自动学习flappybird游戏策略。与DQN（Deep Q-Learning）方法不同，没有使用深度神经网络进行处理，而是自己定义状态空间。

技术路线：

在该实现中，将小鸟与管子（pipe）之间的横纵距离，以及小鸟此时的垂直速度作为一个状态。为了减少状态空间，将小鸟与管子较近的状态离散为10x10的网格，较远的状态离散成70x60的网格（例：定义状态为dis\_x : dis\_y : velocity，离散后12:21:3与16:24:3会被归结为同一状态10:20:3上）。这样做可以极大地缩小状态空间，加快收敛速度。

该实现有三部分：

Bot.py —— Q-learning部分

Learning.py —— 调用无可视化部分游戏及强化学习部分，进行训练

Showflappybird.py —— 运行可视化游戏程序展示效果，使用训练后的Q-table采取动作

Outputdata.py —— 提取训练时产生的迭代和分数的数据，进行分析，绘制散点图

Bot.py：

创建Bot类，定义了5个方法：

1. load\_qvalues(self):从data/qvalue.json中加载Q-table以便继续上次训练，若不存在Q-table，则初始化。
2. act(self, xdif, ydif, vel):输入小鸟与管子横纵距离以及此时小鸟的垂直速度作为当前状态，使用贪心算法，执行能够获得最大Q值的行动，之后将该状态和动作记录在history中，并返回。
3. update\_scores(self, dump\_qvalue = True):在每次迭代后从后向前遍历history，获得reward值（死亡前的倒数两个状态reward为-1000，如果是因为向上飞撞到上方管子而死亡，则最近一个采取向上飞动作的状态reward也被设为-1000，其他状态reward为1），以公式https://pic1.zhimg.com/80/v2-42d461d49b906c6cc5cbe60dca438564_hd.jpg更新Q-table，并将history清空。
4. map\_state(self, xdif, ydif, vel): 输入小鸟与管子横纵距离以及此时小鸟的垂直速度，返回被离散后的状态（将小鸟与管子较近的状态离散为10x10的网格，较远的状态离散成70x60的网格）。
5. dump\_qvalues(self, force=False):将更新后的Q-table导出到json文件。

Learning.py:

1. 实例化bot。
2. main()函数增加两个命令行参数：--iter:控制迭代次数，默认为1000；--analysis:将迭代次数和对应的分数输出至analysis.csv，以便于接下来进行数据分析（画散点图）。
3. main()在while True中调用showWelcomeAnimation()、mainGame(movementInfo)、showGameOverScreen(crashInfo)函数（均是游戏原代码）进行游戏。
4. 删除showWelcomeAnimation()函数中的显示部分以加快训练。
5. 在mainGame()函数中调用bot.act获取一个动作，在游戏结束时（一次迭代结束）调用bot.update\_scores更新Q-table。
6. 在showGameOverScreen()函数中调用bot.dump\_qvalues(),当达到迭代次数时输出至json文件。

Showflappybird.py:

在原游戏代码的基础上将从键盘获取动作改为从bot.act()（即Q-table）中获取动作,用来展示训练效果。

Outputdata.py:

根据导出的数据（.csv文件），使用matplotlib库绘制散点图，进行数据分析。

实验数据：由机器自动运行flappybird游戏产生，每死亡一次作为一次迭代。

开发环境：Ubuntu 18.04.3 LTS、python 3.6.9

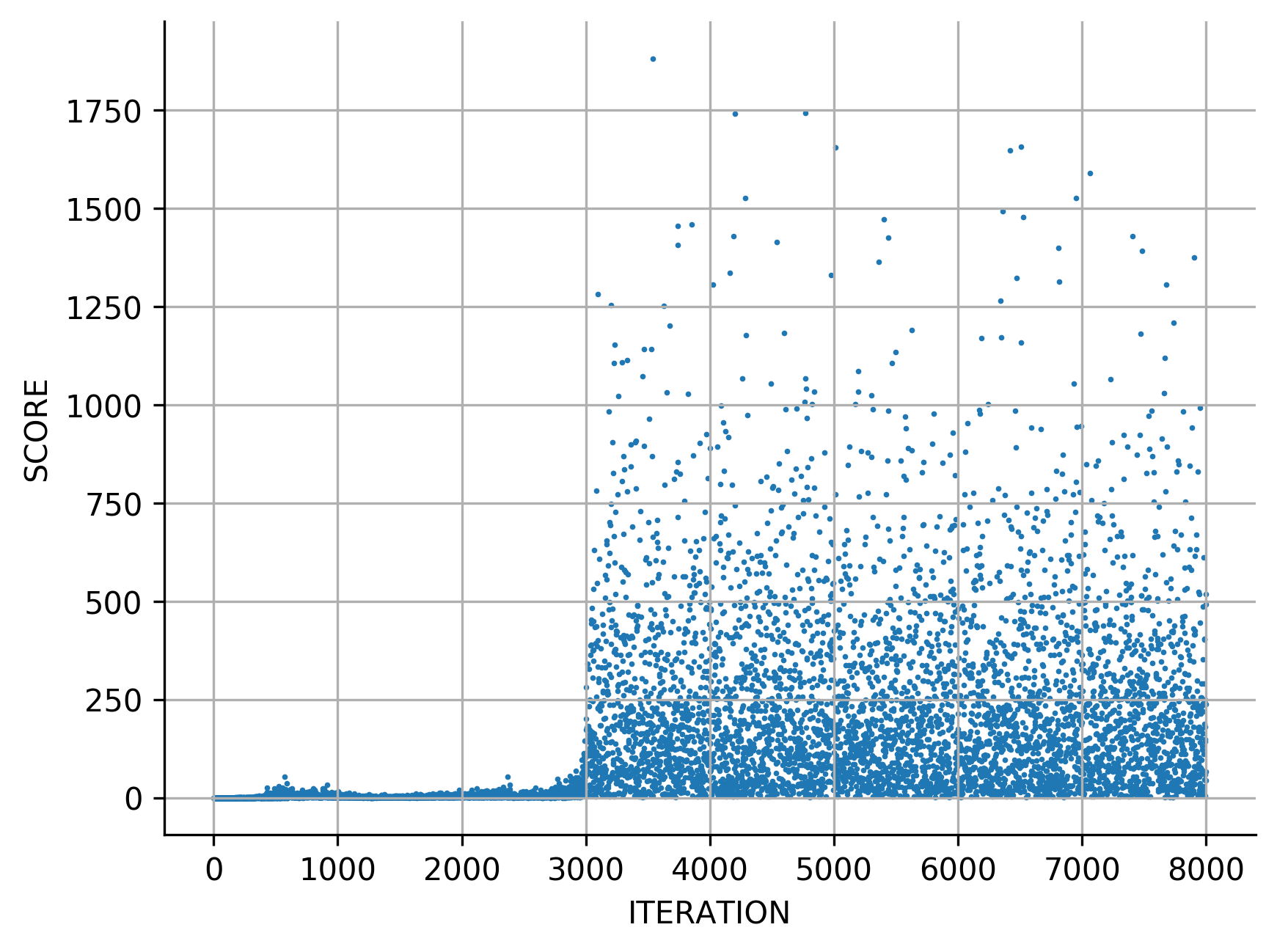
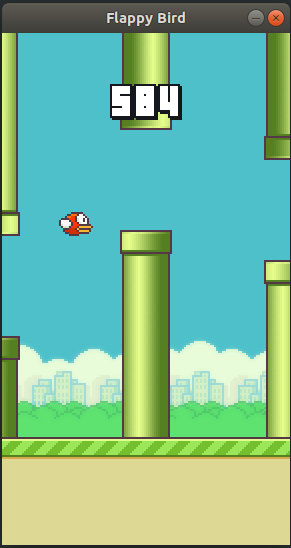
运行结果：

删除/data中的.json文件后，运行以下命令行指令：

Python3 learning.py –iter 8000 –analysis

Python3 outputdata.py

Python3 showflappybird.py



实验结论：

如散点图所示，大约在3000次迭代之后收敛，分数大约在250分左右。相比于连续地定义状态空间，离散定义状态空间后的效率提升非常明显。

（关于连续地定义状态空间，可参考<http://sarvagyavaish.github.io/FlappyBirdRL/>，

该作者大约花了6-7小时才收敛）

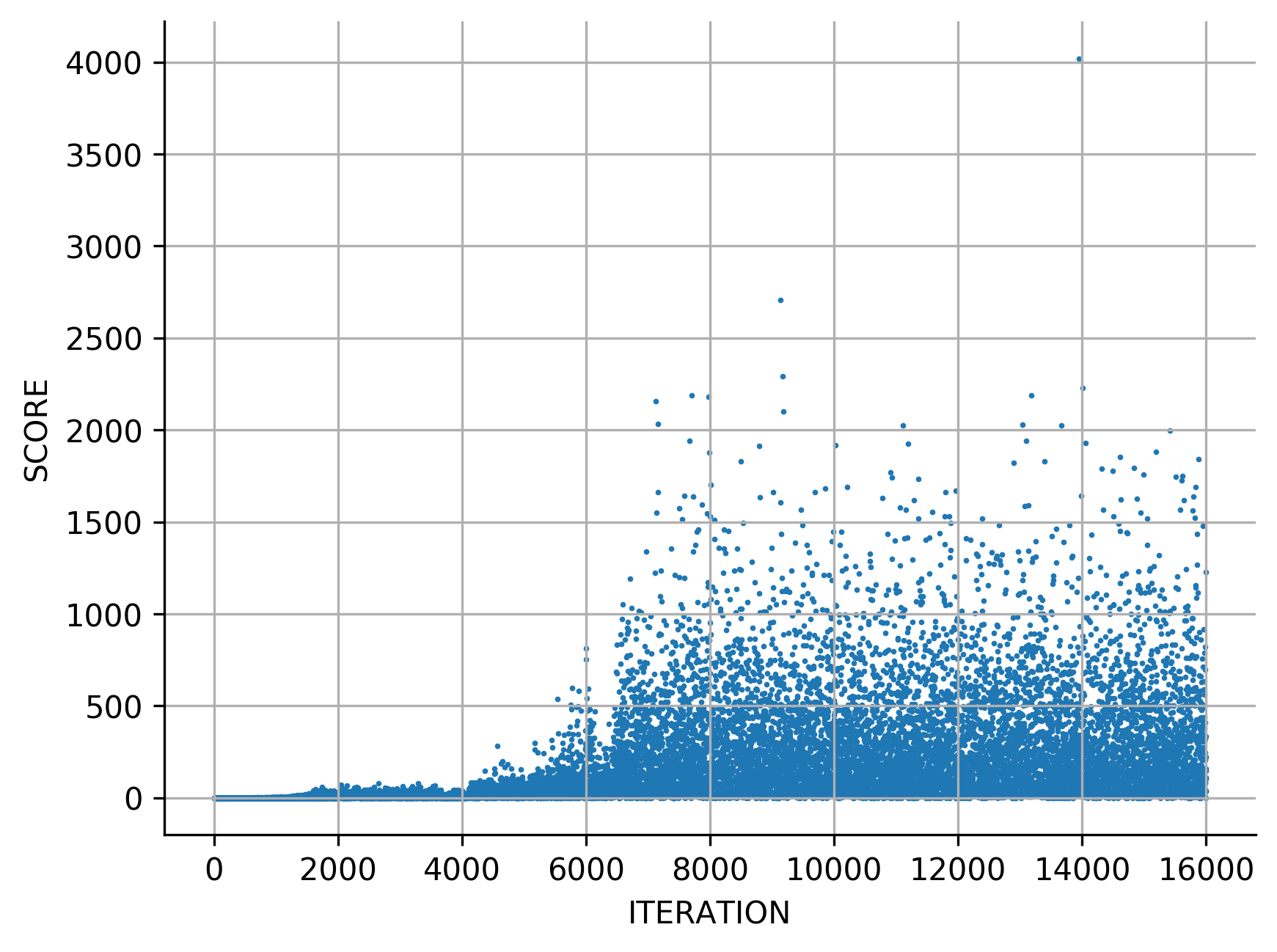
改进：

离散定义状态空间仍有一些不足。在这种定义下，很多相似的情况会被合并。有些情况虽然相似，但采取相同的动作a可能会导致不同的结果，使得最后收敛的效果并不好。猜想可以通过缩小离散的网格以改善收敛效果，尝试将网格缩小为5x5，迭代次数提升至16000次，结果如下：

（删除/data中的.json文件并修改原代码后，运行以下命令行指令：

Python3 learning.py –iter 16000 –analysis

Python3 outputdata.py）



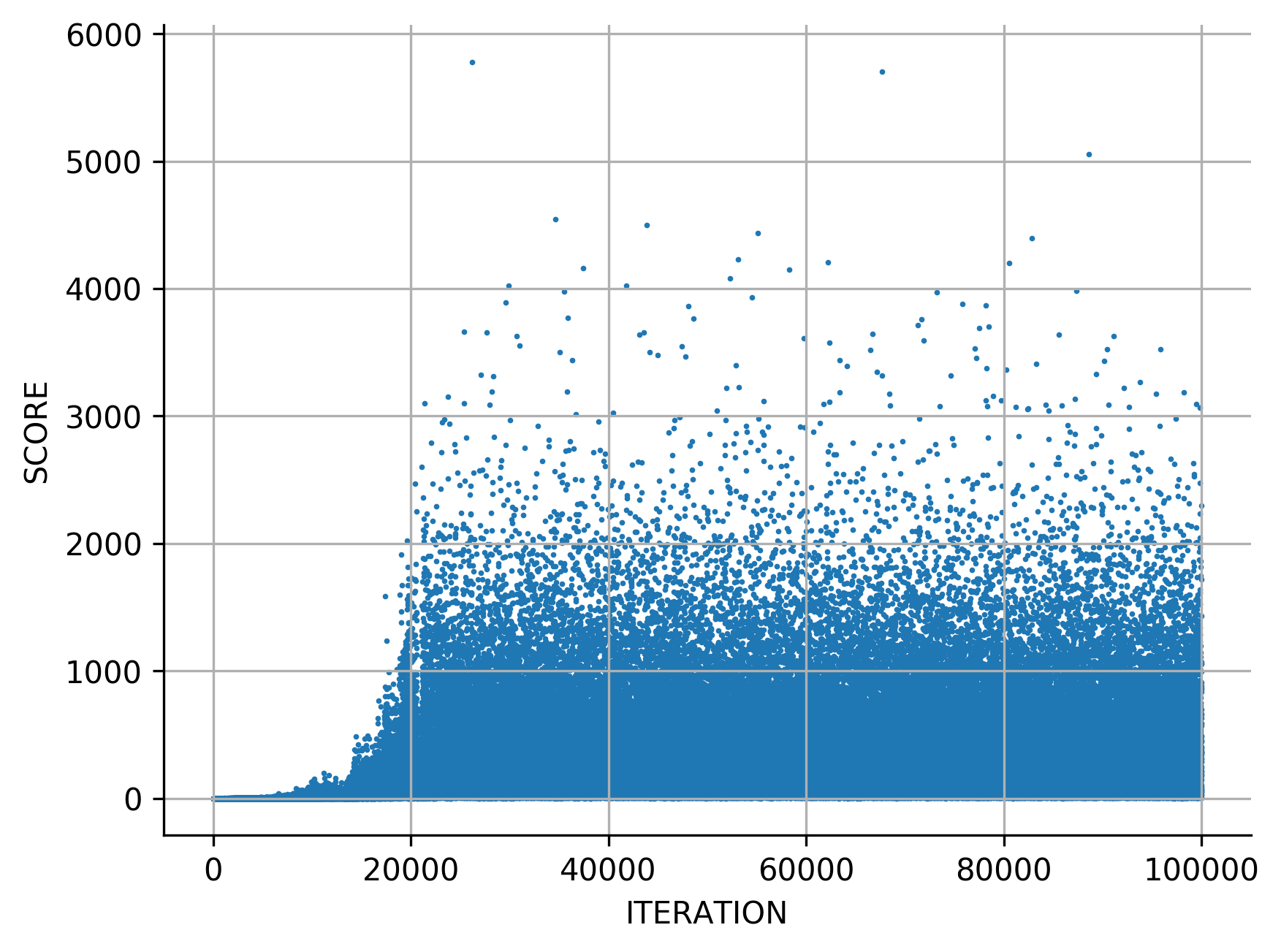
如图，结果大约在6000至7000次迭代后收敛，分数大约在500分左右，甚至多次出现2000以上的高分，效果提升明显，但收敛所需的迭代次数也明显增多。

进一步将网格缩小为2x2，迭代次数增加至100000次，结果如下：

（删除/data中的.json文件后，运行以下命令行指令：

Python3 learning.py –iter 100000 –analysis

Python3 outputdata.py

Python3 showflappybird.py）

如图，结果大约在20000次迭代后收敛，分数大约在1000分以上，甚至多次出现3000以上的高分，效果提升非常明显，但收敛所需的迭代次数增加非常多，耗时很长，因此效率较低。