利用神经网络启发的A*算法解决15数码问题

摘要

为了解决十五数码问题, 常用的方法是A*算法, A*算法的好坏完全决定于启发函数h的选择. 在15数码问题中, 常用的启发函数有错位数和曼哈顿距离, 但这样的启发函数距离真实的代价差距还是很大的. 同时我们可以注意到, 启发函数可以看作当前状态到预测代价的映射, 因此利用神经网络训练启发函数是完全合理的. 实验证明, 在搜索深度比较大的情况下, 神经启发函数比曼哈顿距离找到最优解的速度可能会快上甚至上千倍

代码说明

`main.py` 程序的入口

`board.py` 负责一些基本操作: 比如是否为解, 判断可行的移动方向, 移动棋子

`utils.py` 数据结构类: Node用于存储最优路径, PriorityQueue负责探索f(state)最小的状态

`intelligence.py` 基本的A*算法, 曼哈顿距离和判断是否有解的算法

`neural_network.py` 用于训练和产生神经启发函数

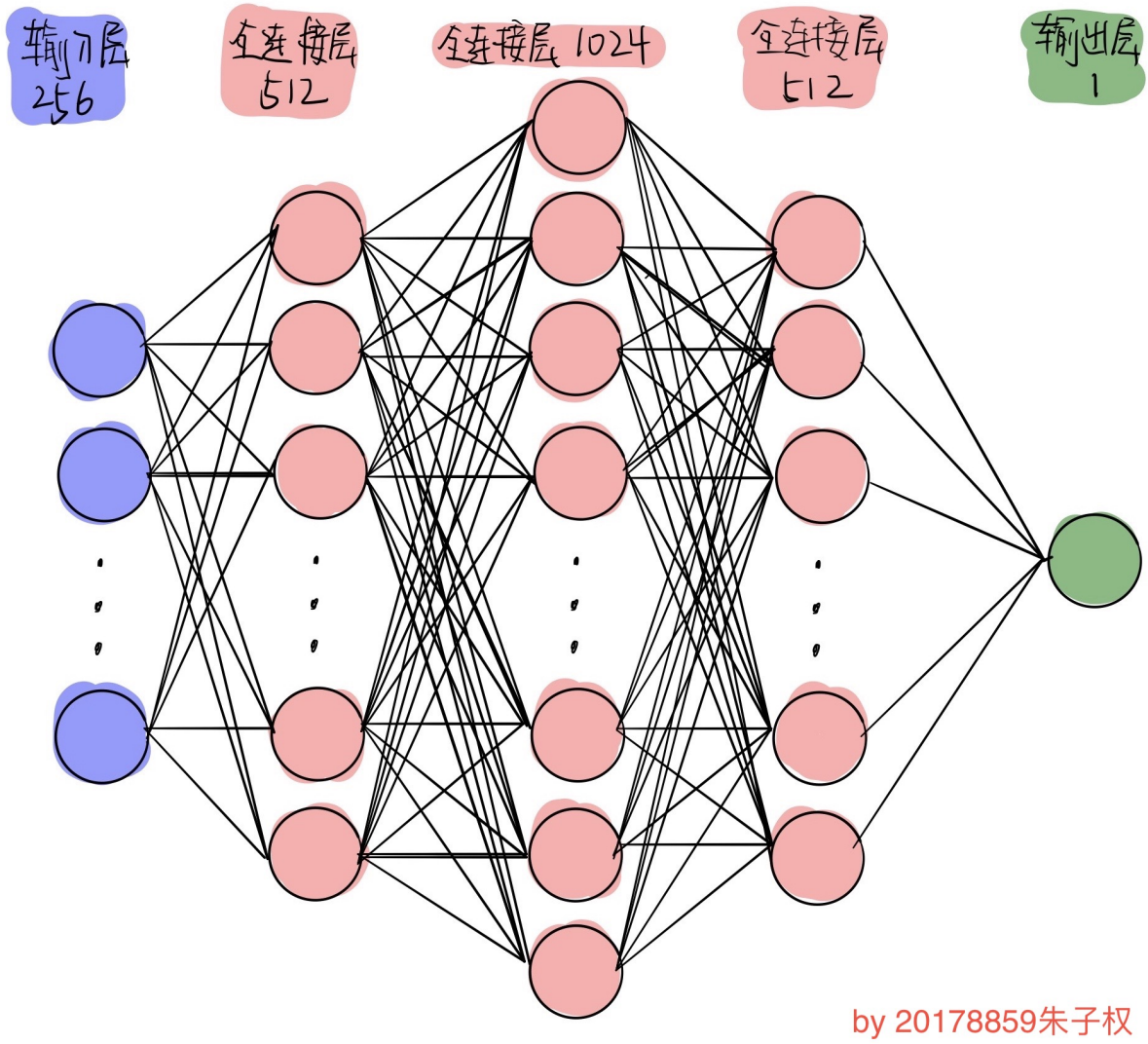
`ans/neu_ans[num].txt或man_ans[num].txt` 存储使用neu和man的解(主要是判断neu产生的解是最优的), 还记录了所花费的总时间和解的深度

`models/` 存储了训练好了的的模型

算法说明

判断十五数码是否有解, 我们只需要计算N=逆序数对之和，e=空白所在的行数。若N+e为偶数，则有解，反之无解(参考文献3)

本模型提供了两种启发函数的A*算法: 一种是基于Manhattan距离的A*算法(MHA*)(参考文献1), 另一种是基于Neural Netword Heuristic的A*算法(NNHA*)(参考文献2). 其中NNHA*的网络结构如下:



如何使用该程序

训练神经网络你可以直接运行`neural_network.py`, 前提是你安装了keras并且去除了代码中的注释.

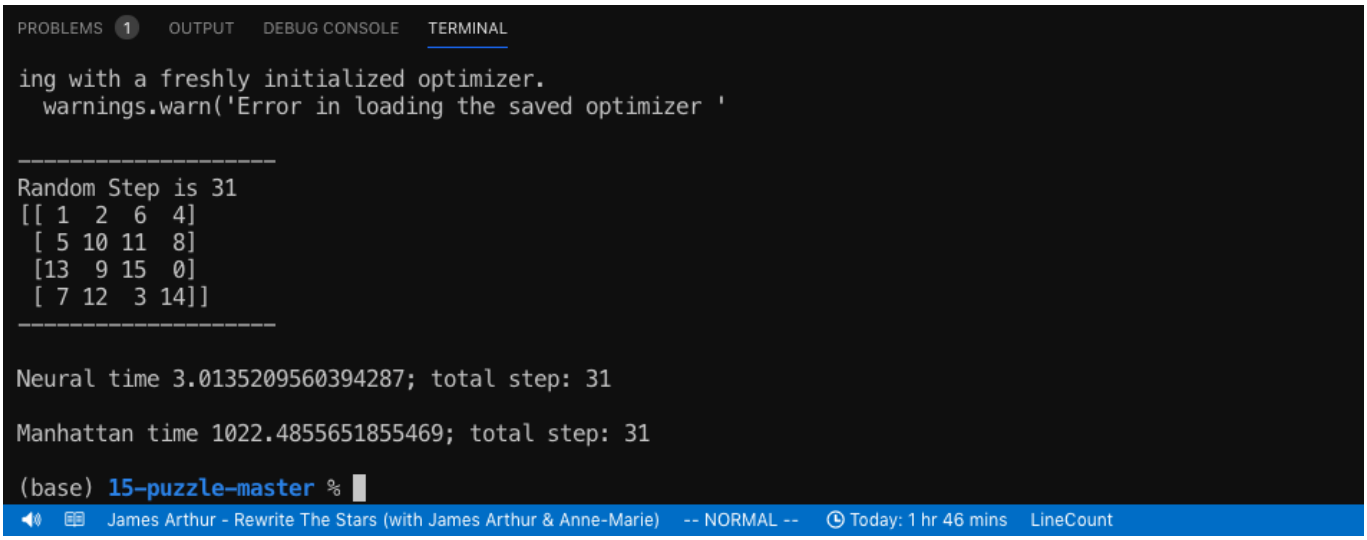
因为训练的时间比较长, 所以在程序运行过程中目标状态是固定的. 但如果你想要使用不同的目标状态, 只需要修改`neural_network.py`中`traning_data`函数中`this_board`的数组值为你所需要的的目标状态, 修改曼哈顿函数, 然后重新训练即可.

进行搜索我们直接可以运行`main.py`, 程序会进行搜索, 并且会在中断打印出当前搜索的游戏, 已经NNHA*和MHA*的时间以及结果的步数, 并且保存为txt文件存储于当前目录(`main.py`利用Manhattan的部分已经被我注释掉, 如果相比较效率区别请取消注释)

实验

在深度较浅的情况下, MHA*比NNHA*往往效率要好, 但不过两者差距不大, 都在可接受范围内. 在大多数深度较深的情况下, NNHA*的速度比MHA*有时快了上千倍.

下面是运行过程中的一个NNHA*效率远远高于MHA*的例子:



参考

1. The Fifteen Puzzle - The Algorithm
2. Using Neural Networks for Evaluation in Heuristic Search Algorithm
3. Solvability of the Tiles Game