《强化学习理论及应用》课程大作业

> 小组成员:
>
> 松山鐘迪 2101213070 信息科学技术学院
>
> 齐藤文美吉 2101213051 信息科学技术学院
>
> 樊雨萱 2101111483 信息科学技术学院

一、实验目的

本次课程大作业需要使用强化学习方法解决《2048》小游戏。《2048》小游戏是4*4网格上进行的游戏，每一回合会在随机的位置生成一个数字2或者数字4的块，玩家可以向上、下、左、右整体移动盘面上的所有块，每次移动，在该移动方向上相邻且数字大小相同的2个块都会被合并。合成的数字越大，合成次数越多，游戏的得分越高。玩家的目标是在盘面没有被占满的情况下，合成数字2048或者更大的数字。

图1 《2048》小游戏示意图

二、实验原理

经过调研，我们总结出三种在强化学习任务上表现较好的方法：DQN, MCTS，NTN，并利用这些方法解决《2048》小游戏。本节将介绍这三种方法的基本原理。

1. DQN

DQN(Deep Q-Network)是深度学习与强化学习算法Q-Learning的结合，使用深度神经网络来拟合Q-Learning中的Q-Table，从而能够更好地处理高维连续的状态和动作空间。

DQN算法从初始状态出发，对每个状态都通过预处理函数得到该状态的序列。在每一步，通过epsilon法进行探索或利用，执行所选择的动作并得到下一轮的状态和当前轮的回报。(当前状态序列、下一轮状态序列、当前动作、当前回报)的四元组依次被存在缓冲区中并不断被抽样训练。DQN的算法流程如图2。在这个过程中，DQN使用到了两个重要的结构：Replay Buffer和Target Network。

- **Replay Buffer:**

经验回放(Replay Buffer)用来处理深度学习与强化学习间存在的两个问题：（1）深度学习需要大量标注的训练样本，而强化学习从reward学习，reward本身是稀疏且延迟的。（2）深度学习假设样本是独立同分布的，而强化学习中采样到的数据是强相关的。Replay Buffer机制按照时间先后顺序将采样到的数据依次存储到一个缓冲区中，并每隔一段时间就从缓冲区中均匀随机地采样一批样本进行学习，这

样就能够减少连续样本的前后相关性，并提高数据的利用率。

- **Target Network:**

此外，DQN还引入Target Network来增强模型的稳定性。原本的模型被称为Behavior Network，开始时，Target Network和Behavior Network使用完全相同的参数。在训练过程中， Behavior Network负责与环境交互，得到交互样本。在学习过程中，Target Network的参数相对固定，用来计算Q-Target值算得到; Behavior Network的参数不断更新，用来获取Q-估计值。Q-Target值和Q-估计值的差就是模型的损失函数，通过最小化损失函数实现模型的训练。每当训练完成一定轮数的迭代，Behavior Network模型的参数就会同步给Target Network。引入Target Network可以减轻模型的波动性，使模型尽快收敛。

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

图2 DQN算法流程伪代码

2. MCTS

蒙特卡洛树搜索(Monte Carlo Tree Search, MCTS)是一种寻找最优决策的方法，该方法通过蒙特卡洛模拟来估计价值函数，可以有效解决《2048》游戏中搜索空间巨大的问题。在MCTS方法中，树的节点表示不同的游戏状态，节点的父子关系代表在当前游戏状态下采取行动可到达的新的游戏状态。对每一个游戏状态，MCTS算法要输出下一步采取什么行动更可能获胜，也就是在当前节点的子节点中寻找获胜概率最高的节点。

MCTS算法按顺序重复执行以下4个步骤：选择，扩展，模拟，反向传播，如图3所示。

- **选择(Selection)**：MCTS算法从树的根节点出发，根据一定的策略，选择下一个节点进行访问。若被选择的节点未被访问过，则执行扩展操作；若已经被访问过，则访问该节点并递归进行选择操作。具体选择策略由upper confidence bounds(UCB)公式确定。
- **扩展(Expansion)**：MCTS 算法在搜索的过程中是有选择地访问节点，并把所有访问过的节点构建成一个树。扩展就是把选择步骤中遇到的未访问节点添加到树中的过程。
- **模拟(Simulation)**：扩展出子节点后，可以对该子节点进行模拟。具体来说，从被扩展的节点开始，随机选择后续动作，直至游戏结束，记录本轮的reward值。
- **反向传播ropagation)**：也称回溯更新，每次模拟结束后，从被扩展的节点向上到根节点都会用模拟中得到的reward更新节点价值函数。

MCTS算法的终止条件一般为最大根节点搜索次数或者最大搜索时间，超出预定义的上限后自动结束搜索过程，并将选择访问次数最多的节点作为当前状态下该采取的动作。

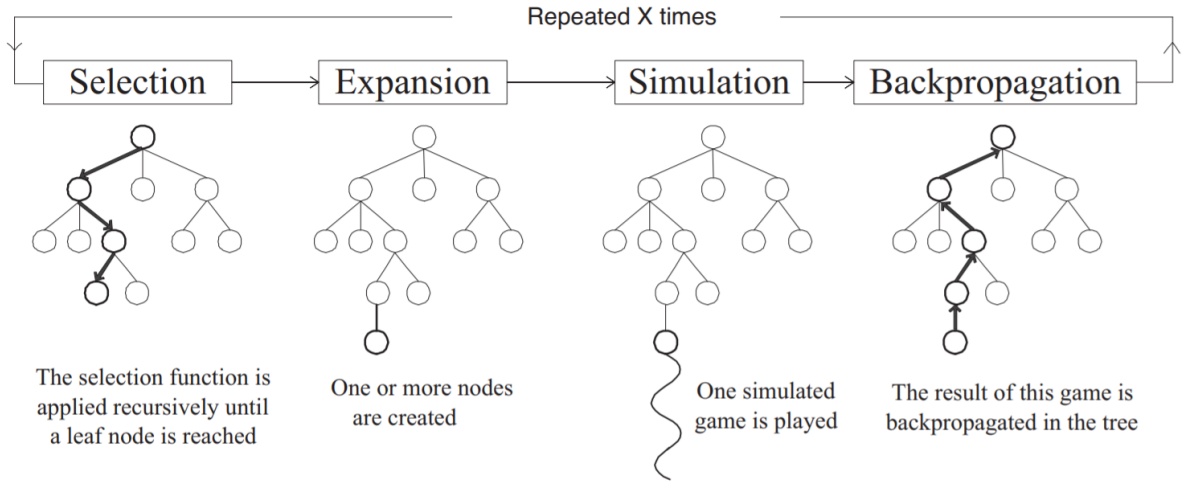

图3 MCTS算法流程

3. NTN

NTN是指N-Tuple Network，是模拟值函数的一种网络结构。如果在小状态空间的问题中，可以用一个查询表来值函数V，其中每个值单独存储。在《2048》游戏中，设单个$2^{17} = 131072$，在此基础上《2048》游戏约有$(4\times4)^{18} \approx 4.7\times1021$个状态。因此，使用显式Q-Table在计算上是不可行的。必须使用一个函数来近似，它采用一类参数化函数来替换Q-Table。因此我们利用N-Tuple Network来作为近似函数，一个N-Tuple Network由m个n_i元组组成，n_i是元组的大小。对于给定的游戏状态，它计算各个n元组返回值的和。第i个n_i元组(i = 1...M)，由一个预定的位置序列 $(loc_{ij})i = l..i$，和一个查找表LUT_i构成。查询表LUT_{i}$包含在位置序列上观察到的每种模式的权重。

其中N-Tuple Network的实现函数f为:

$$
\begin{aligned}
f(s) = \sum_{i=1}^{m} f_i(s) &= \sum_{i=1}^{m} \mathrm{LUT}_i \left[\mathrm{index}\left(s_{loc_{i1}}, \ldots, s_{loc_{in_i}}\right)\right], \\
\mathrm{index}\left(\mathbf{v}\right) &= \sum_{j=1}^{|\mathbf{v}|} v_j c^{j-1},
\end{aligned}
$$

其中$s_{loc_{ij}}$是位loc_{ij}置上数字的编码值，v是一个按照某种规则确定的编码值序列，使得$0 < v_i < c$。这里的c是一个常数，表示所有可能的编码值数目,《2048》游戏中认为c的值理论上为18，实验中方便限制网络中权重数量，假设c为15。计算过程中空的正方形被编码为0，包含值v的正方形被编码为$log_2(v)$，例如，128被编码为7。

在《2048》游戏中编码的序列元组包含多个权重，一种编码是一个权重。为了获得具有可管理数量的权值的网络，将元组的长度限制为4或6，即4或6个数字统一编码成一个元组。实验中使用了3种类型的元组表格，由4个水平的4元组和4个垂直的4元组以及6个6元组组成。一个网络总共包含8个4元组和6个6元组。

三、实验内容

在这一部分，我们将描述我们对DQN的基础实验及DQN模型调优，并尝试使用MCTS和NTN解决2048问题。此外，我们还将融合在DQN、MCTS和NTN的探索，对这一问题给出更完善的解决方案。

1. DQN及其调优实验

我们首先复现了DQN baseline的代码。然而，Baseline版本的DQN算法学习能力提升较慢，并且loss没有收敛，我们推断这可能是模型没有找到足够高分的数据，或者高分数据没有被训练到导致的。

针对高分数据不容易被采样的问题，我们进行了两点优化：

（1）首先调整e-greedy方法中epsilon的值，在10000 episode以内设置epsilon呈指数衰减，使模型尽可能地进行探索操作；在10000 episode以后让epsilon快速收敛到0，使模型更多地利用探索得到的经验。

（2）进行reward shaping，调整奖励机制，将模型合成的最大数值(max tile)的增量和合并的单元数也作为奖励函数的一部分，这样能激励模型更快地进行单元合并，并不断合成更大的数值。

针对被采样到的高分数据可能没有被训练到的问题，我们进行了三点优化：

（1）首先是将Replay Buffer机制优化为Priority Replay Buffer，抛弃之前按照时间先后顺序替换缓冲区数据的做法，每次缓冲区满时，优先替换其中reward最低的数据。这样能尽可能地将高分数据保留在缓冲区中，从而提高训练数据的整体reward值。

（2）接着，调整memory size至6000，这样能够避免过估计问题，并且提高高分数据在全部训练数据中的可能比例。

（3）最后，利用tanh函数来自动调整学习率，使得学习率和数据的reward相匹配，reward越大学习率越大， reward越小学习率越小。

2. MCTS+DQN

由于DQN调优之后的效果不佳，我们引入了MCTS并与DQN进行结合。具体来说就是，在执行每一步动作之前，先对当前状态开始的环境，进行100次模拟蒙特卡洛树搜索。

对于MCTS中的**选择阶段**，我们使用UCT方法对已经生成的MC树进行分支选择，其中UCT中V值，我们使用到当前节点为止，目前所有的reward求和，进行表示，而探索部分依然根据MCTS中的对不同分支探索次数的比例进行计算，最终UCT的值就是这两个部分的求和，具体UCT公式如是：$score = \sum_{t=1}^{T} \gamma^{t-1} r_t + c\sqrt{\frac{lnN_i}{n_i}}$，其中，Ni是所有模拟次数，c 是探索常数，理论值为√2，可根据经验调整，c 越大就越偏向于广度搜索，c 越小就越偏向于深度搜索。最后我们选择分数最高的动作节点。

对于MCTS中的**扩展阶段**，为了方便计算不同节点的信息，我们直接创建与当前状态可以执行动作数量相同的节点数，即为每个动作后导致的状态建立一个新的MC树的子节点。这些节点信息都为空，是在模拟阶段才计算信息进行填充的。

对于MCTS中的**模拟阶段**，我们遵循的策略就是每次选择当前状态下每个动作Q值最大的那个动作，即使用DQN进行策略的游走。这样，当DQN估值越来越精确的时候，MCTS也会选择越来越优良的策略动作，形成良性循环。使得DQN也能更容易获得高分数据，同时优化DQN和MCTS的策略。

对于MCTS中的**反向传播阶段**，我们保存到每个节点的reward之和，这样方便选择阶段的UCT的计算。我们同时也保存到该节点的次数，方便计算出reward之和的期望。

3. MCTS+NTN

在MCTS+NTN结合的尝试中，我们采取与MCTS+DQN类似的思路，在每个状态进行100次搜索后再使用UCT执行当前最优的动作。唯一不同的是在扩展后的模拟阶段，我们不再执行当前DQN中Q值最大的动作，而是使用NTN来替换DQN，选择使得下一个状态的价值V最大的动作进行模拟。这里的状态价值函数就是通过NTN计算的。

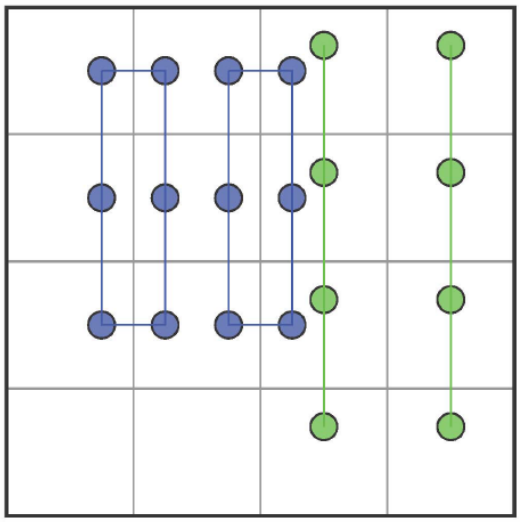

Figure 10: A network consisting of all possible 2×3 rectangle tuples (blue) and all possible straight 4-tuples (green). This network makes use of board symmetry (symmetric sampling), thus only two n-tuples of each kind are used.

图4 NTN的元组示例

对于NTN模型，我们将状态价值拆分为3种类型的子状态价值，分别是横四行的价值，纵四列的价值与如上图所示连续6格的价值。而对于状态价值的描述就是计算完这三种子价值后求和。

这三种价值表格的状态都使用同一种编码方式。如下所示，即每一个格子的数字编码为自身的以2为底的对数。

$$S_{ij}^{new} = floor(log_2(S_{ij} + 1))$$

则对于第一种横四行编码方式即为，一列的四个放在各自的位子上，公式如下，其中c=15，因为可以认为格子数字不会超过$32768 = 2^{15}$。最终的子价值为每一列的编码在表格中的价值求和。这对于第二种、第三种编码方式也同样适用。其中第三种编码方式不一样的是它同时编码6个相连的格子，如上图所示。

$$\text{index}\,(\mathbf{v}) \quad = \quad \sum_{j=1}^{|\mathbf{v}|} v_j c^{j-1},$$

每次实际执行一个动作后，我们获得reward和新的状态，我们就会结合下面两个公式对NTN表格进行更新，如下图所示。

$$V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s)). \qquad (1)$$

$$f(s) = \sum_{i=1}^{m} f_i(s) = \sum_{i=1}^{m} \mathrm{LUT}_i \left[\mathrm{index} \left(s_{loc_{i1}}, \ldots, s_{loc_{in_i}} \right) \right],$$

$$\mathrm{index}\left(\mathbf{v}\right) = \sum_{j=1}^{|\mathbf{v}|} v_j c^{j-1},$$

四、实验结果

在这一部分，我们将描述我们的实验环境设置，并展示实验结果。

我们的实验环境是：CPU: Intel Core i7-11800H，GPU：GeForce GTX 1080 Ti，python版本：Python 3.8.10，cuda版本：Cuda 11.5，torch版本：torch 1.10.0。我们使用了标准游戏环境的代码来评价模型的表现，并汇报模型的平均得分、最高得分、块值分布、最大块值、训练时间等指标。不同模型的实验结果对比见表1。

Method	Max Reward	Avg Reward	Max Tile	Training Episodes
DQN baseline	3000	2100	256	20000
DQN refine	6000	3000	512	40000
MCTS+DQN	31498	7000	2048	20000
MCTS+NTN	32878	15000	2048	20000

表1 不同模型的实验结果

1. DQN baseline结果

DQN baseline使用到的各个参数如下所示：

- learning rate=0.01
- reward decay=0.9
- eplison=0.96
- memory size=500
- batch size=32

该实验进行了20000 episodes,均分在2100，最大分在3000，能得到的最大数字为256。

2. DQN调优结果

在DQN baseline的基础上引入reward shaping、priority replay buffer、调整memory size、调整e-greedy策略、自动调整学习率等优化方法之后，我们的DQN在40000 episodes迭代之后可以达到3000的均分和6000的最高分，能得到的最大数字是512。这部分实验的块数字的分布情况见表2，reward值变化见图5。

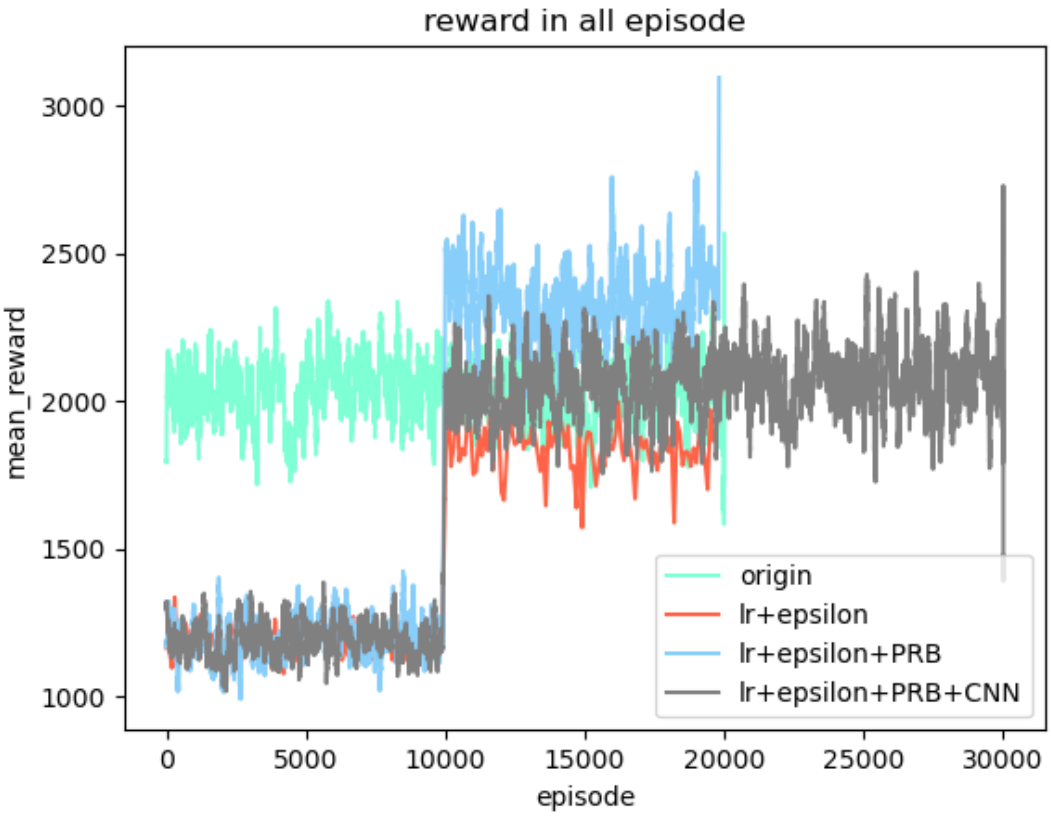

图5 DQN调优实验reward值

tile number	percentage
512	10%
256	70%
128	15%
64	3%
32	2%

表2 DQN调优实验的块数值分布情况

3. MCTS+DQN结果

这部分实验的reward值变化见图6，块数字的分布情况见表3。

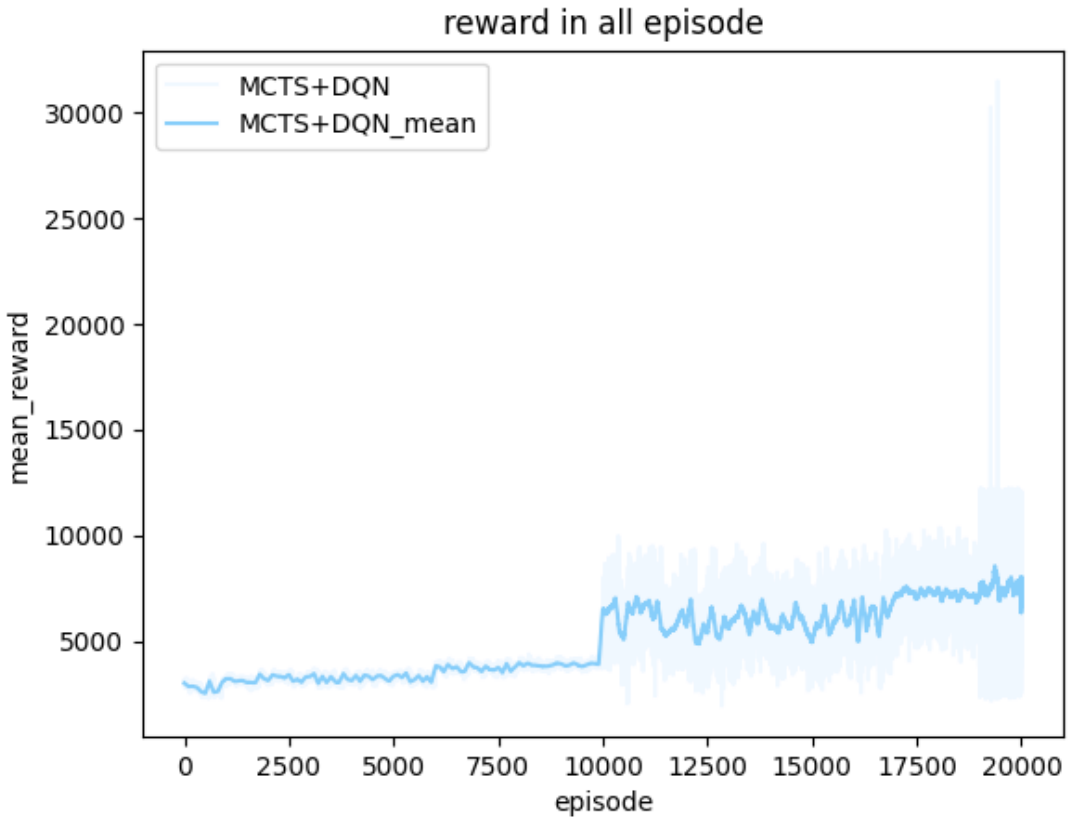

图6 MCTS+DQN实验reward值

tile number	percentage
2048	0.2%
1024	22%
512	47%
256	20%
128	10.8%
64	0%
32	0%

表3 MCTS+DQN实验的块数值分布情况

4. MCTS+NTN结果

这部分实验的reward值变化见图7，块数字的分布情况见表4。

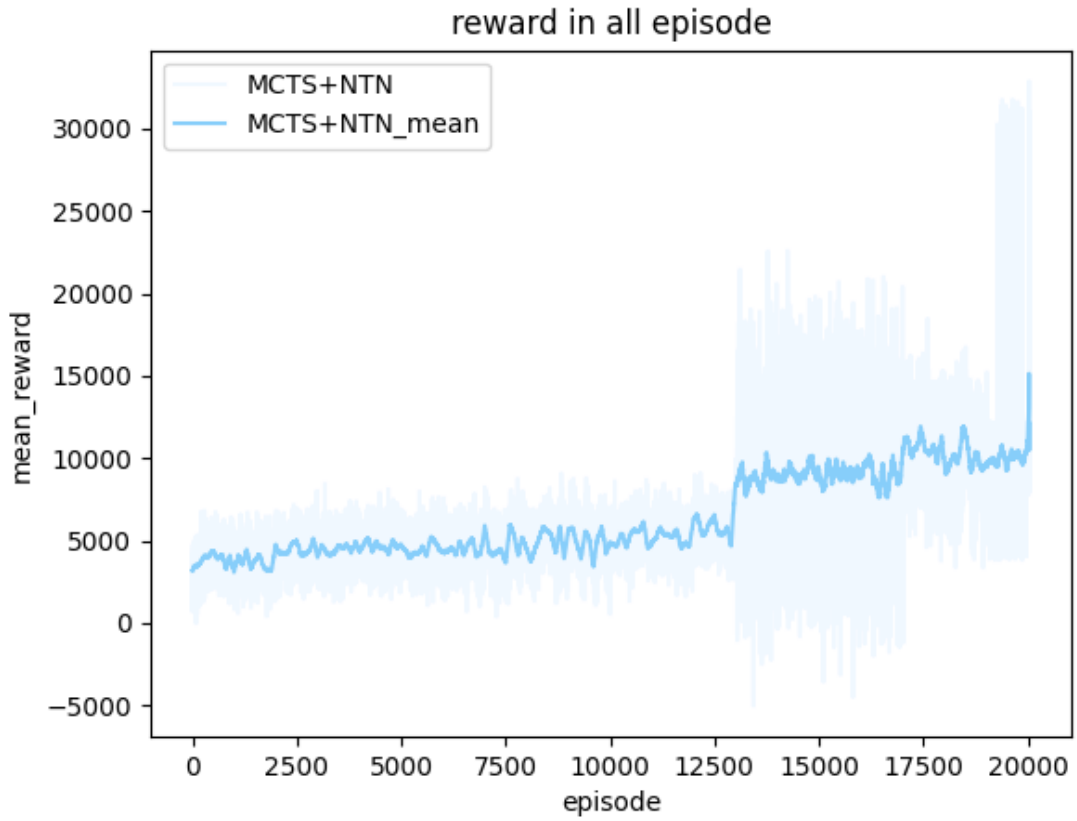

图7 MCTS+NTN实验reward值

tile number	percentage
2048	1.5%
1024	45%
512	50%
256	3.5%
128	0%
64	0%
32	0%

表4 MCTS+NTN实验的块数值分布情况

五、实验结果分析

1. DQN及其调优结果分析

从DQN的实验结果可以看出两个问题。一个是DQN学习能力提升较慢，另一个是DQN的loss不收敛。经过分析，我们发现问题的原因就在于1. DQN在探索过程中没有找到足够高分的数据。2. 即使找到了高分数据，这些高分数据没有被训练到，被其他不太好的数据淹没了。

根据前面两点原因，我们分别进行了优化，但是提升效果并不明显。我们确定了是因为游戏状态空间比较大，要想纯DQN训练出非常好的性能，需要大量reward较高的数据。因此是DQN本身学习能力限制了策略性能。于是我们选择了DQN结合mcts构建模型进行学习。

2. MCTS+DQN结果分析

从MCTS+DQN的实验结果来看，我们会发现MCTS很好地解决了DQN的两个问题。使得DQN在很大程度上对高分数据有了较为准确的估计。使得DQN在20000个episodes时已经可以有30000分的高分，同时达到2048。但是还是不稳定，我们相信继续训练，可以使得算法稳步优化，分数也会稳步增加。希望能加快对策略的优化，我们尝试了另一种对状态估值的方法NTN。

3. MCTS+NTN结果分析

从MCTS+NTN的实验结果来看，我们会发现我们根据参考文献[2]设计的3种类型的表格，很好地解决了DQN对于状态估计偏差比较大的问题，即NTN对于状态的价值估计更精确。再结合MCTS强大的探索能力，使得算法在20000个episodes时已经可以有平均分10000分的高分，让1024迅速稳定下来，同时可以达到2048。尽管2048不稳定，但我们发现该算法已经有比MCTS+DQN更快更稳定的性能。

六、实验感想及总结

本次项目我们尝试了3种RL相关的算法。其中，DQN方法尝试各种不同的调优方法，训练了很长的时间，也没有达到很好的效果，最高分仅仅达到了512。说明了DQN对于稍微复杂一些的环境来说，不是一种精确的值估计方法。剩下两种方法分别是MCTS结合DQN和MCTS结合NTN，最终都得到不错的效果。实验说明了MCTS本身强大的搜索能力，能有效提升策略性能，同时NTN作为棋盘类游戏的半人工估值方法，是一种较为精确的值估计方法。

在实现本次项目中，我们体会到了强化学习理论与实际的差别。需要对强化学习算法有很强的实现能力与信心，才有可能发挥出强化学习这种与环境交互学习的作用。

参考文献

[1] Oka, K., & Matsuzaki, K. (2016, June). Systematic selection of N-tuple networks for 2048. In *International Conference on Computers and Games* (pp. 81-92). Springer, Cham.

[2] Szubert, M., & Jaśkowski, W. (2014, August). Temporal difference learning of n-tuple networks for the game 2048. In *2014 IEEE Conference on Computational Intelligence and Games* (pp. 1-8). IEEE.

[3] Levine, Z. Learning 2048 with Deep Reinforcement Learning.

[4] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., ... & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), 1-43.