Theory of Computer Games 2018 - Project 2

In the series of projects, you are required to develop AI programs that play *Threes!*, the origin of other 2048-like games.

Overview: **Write a player to play *Threes!* with high win rates**.

1. Implement the function approximator with n-tuple network.
2. Implement TD(0) after-state training method.
3. Build an AI based on n-tuple network and TD learning (no extra search is required).

Specification:

1. The environment and the rules are the same as Project 1's.
2. Train the agent by temporal difference learning.
3. The **testing speed** should be at least ~~**50,000**~~ **actions per second** (time limit).
   (approximate value, see Scoring Criteria for details)
4. Statistic is required, and the requirement is the same as Project 1's requirement.
   a. Average score.
   b. Maximum score.
   c. Speed (action per second).
   d. Win rate of each tiles.
5. Implementation details:
   a. Your program should be able to compile and run under the workstation of NCTU CS.
      i. Write a makefile (or CMake) for the project.
      ii. C++ is highly recommended for TCG.
      You may choose other programming language to implement your project, ~~however, the scoring criteria (time limit) will keep unchanged~~. (TBD)
   b. Your implementation needs to follow the statistic output format.
      (see the Methodology for details)
6. Your program should be able to serialize and deserialize the weight tables of N-tuple network.

Methodology:

1. As a player, your program should calculate all the after-states (at most 4). **Determine the value of available after-states by the N-tuple network**. Finally, select a proper action based on the values.
    a. Design and **implement the N-tuple network**.
    b. States and actions should be saved during an episode.
    c. After the episode done, **train the N-tuple network by TD(0) method**.
2. Some useful advices for designing the N-tuple network of *Threes!*:
    a. Try the simplest 8 × 4-tuple network first.
    b. The maximum tile is 6144-tile, so the size is $15^4$ for a 4-tuple network.
    c. Isomorphic patterns are able to speed up the training process.
    d. The initial value of weight tables should be set as 0.
    e. Larger networks need more time to converge.
3. Some useful advices for the reward system:
    a. Threes! **does NOT define the reward** of actions.
    b. You can define the reward system by yourself, such as 2048-like merged score, difference of score, or even a const for every valid action.
4. Some useful advices for training:
    a. Do not forget to train **the terminal state**. Its TD target should be 0.
    b. Do not forget to take **the immediately reward** into account.
    c. You should train the network based on "**after-state**", not on "before-state".
    d. The learning rate is based on the number of features. For example, the learning rate could be $0.1 \div 32 = 0.003125$ if you have four N-tuples with 8 isomorphic patterns (32 features).
    e. Reduce the learning rate if the network is converged.
5. **Sample program is provided**, which is a non-implement AI that plays *2048*. You are allowed to modify everything (remember to follow the specification).
    a. 2048-game is treated as two-player game in the sample program.
        i. The evil (a.k.a. the environment) puts new tiles.
        ii. The player slides the board and merge the tiles.
    b. The process of 2048-game is designed as:
        i. A game begins with an empty board, the evil puts two tiles first.
        ii. Then, the player and the evil take turns to take action.
        iii. If the player is unable to find any action, the game terminated.

Submission:

1. Your solution **should be archived in zip/rar/7z file**, and **named as `XXXXXXX.zip`**, where XXXXXXX is the student ID (e.g. `0356168.zip`).
   a. Pack your **source files**, **makefiles**, and other relative files in the archive.
   b. Do **NOT** upload the statistic output or the network weights.
   c. Provide the version control repository of your project (URL), while do **NOT** upload the hidden folder (e.g. **.git** folder).
2. Your project should be able to run under the workstations of NCTU CS (Arch Linux).
   a. **Test your project on workstations**. Use the NCTU CSCC account to login: **TBA**.
   b. Only run your project on workstations reserved for TCG (tcglinux). Do not occupied the normal workstations (linux1 ~ linux6), otherwise you will get banned.

Scoring Criteria:

1. Demo: **TBD**.
2. Framework: Pass the statistic file test.
   a. A **judger** which is similar to HW1's will be released later, you can test the statistic file by yourself before project due.
   b. **Your score is not counted if the statistic file cannot pass the judge**.
3. Win rate of 384-tile (100 points): Calculated by $\lceil \mathrm{WinRate}_{384} \rceil$.
   a. $\mathrm{WinRate}_{384}$ is the win rate of 384-tile calculated in 1000 games.
4. Maximum tile (5 points): Calculated by $\max(k - 9,\ 0)$.
   a. $k$-index is the max tile calculated in 1000 episodes.
5. Penalty:
   a. Time limit exceeded (–30%): ~~50,000~~ is an approximate speed, your program should run faster than the **sample program**.
   b. Late work (–30%): Note that late work including but not limited to **uncompilable sources** or **any modification** after due.
   c. No version control (–30%).

References:


[1] Szubert, Marcin, and Wojciech Jaśkowski. "Temporal difference learning of N-tuple networks for the game 2048." 2014 IEEE Conference on Computational Intelligence and Games. IEEE, 2014.

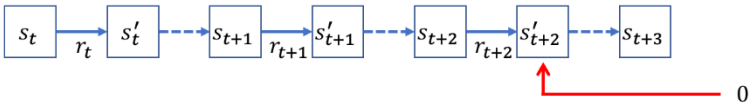
[2] Kun-Hao Yeh, I-Chen Wu, Chu-Hsuan Hsueh, Chia-Chuan Chang, Chao-Chin Liang, and Han Chiang, Multi-Stage Temporal Difference Learning for 2048-like Games, accepted by IEEE Transactions on Computational Intelligence and AI in Games (SCI), doi: 10.1109/TCIAIG.2016.2593710, 2016.

[3] Oka, Kazuto, and Kiminori Matsuzaki. "Systematic selection of n-tuple networks for 2048." International Conference on Computers and Games. Springer International Publishing, 2016.

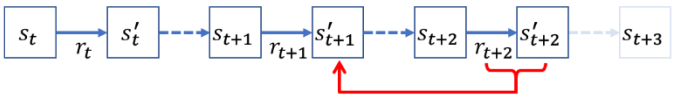

Appendix:

1. Backward training:

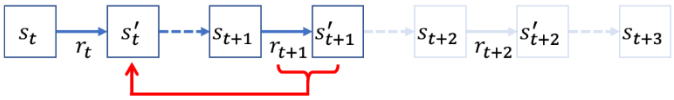
   Step 1: after game over ($s_{t+3}$), update the last state ($s'_{t+2}$)




   Step 2: update the previous *afterstate* ($s'_{t+1}$)

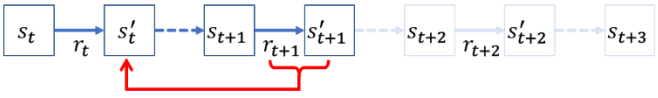



   Step 3: update the previous *afterstate* ($s'_t$)

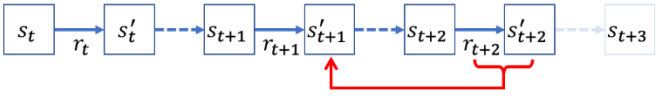



2. Forward training:

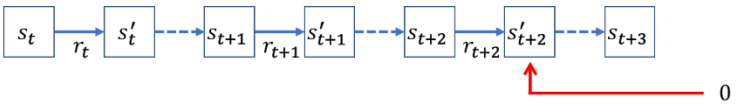
   Step 1: apply an action ($s'_{t+1}$), update previous state ($s'_t$).



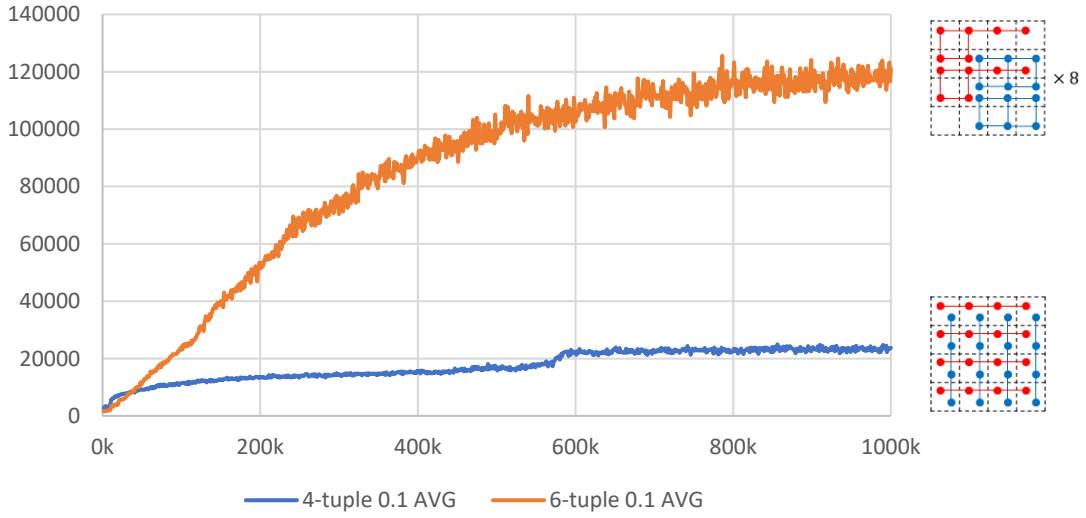

   Step 2: apply an action ($s'_{t+2}$), update previous state ($s'_{t+1}$).




   Step 3: after game over ($s_{t+3}$), update the last state ($s'_{t+2}$)



3. Average score during the training of a million episodes (without peeking the next tile):




```
4-tuple (8 x 4-tuple):
0123 4567 89ab cdef 048c 159d 26ae 37bf
6-tuple (8 x 4 x 6-tuple):
012345 37bf26 fedcba c840d9 321076 048c15 cdef89 fb73ea 456789 26ae15
ba9876 d951ea 7654ba 159d26 89ab45 ea62d9 012456 37b26a fedba9 c84d95
321765 048159 cde89a fb7ea6 45689a 26a159 ba9765 d95ea6 765ba9 15926a
89a456 ea6d95
```

Hints:

You are NOT required to use the framework of project 2 since the it only provides the weight table and some hints (you can keep working on project 1's framework, however, remember to create some branches).

Version control (GitHub, Bitbucket, Git, SVN, etc.) is recommended but not required.

Having some problems? Feel free to ask on the Discussion of e3 platform.

You may use Github Student Developer Pack or Bitbucket for the version control.

Remember to share the sources on sharing platform, for example, GitHub Gist.