Theory of Computer Games 2021 – Project 2

Overview: **Train a player to play *2584 Fibonacci* with high win rates**.

1. Implement the n-tuple network.
2. Implement the TD(0) afterstate learning framework.
3. Train a player based on TD learning and n-tuple network.

Specification:

1. The environment and the rules are the same as those in Project 1.
2. The player should be trained by **the TD(0) afterstate learning framework** as in [1].
   a. An episode is defined as $s_0 \cdots \dashrightarrow s_t \xrightarrow[r_t]{a_t} s'_t \dashrightarrow s_{t+1} \xrightarrow[r_{t+1}]{a_{t+1}} s'_{t+1} \dashrightarrow \cdots s_T$
   b. The TD(0) afterstate learning framework adjusts the afterstate values, $V(s'_t)$, by $\boldsymbol{V(s'_t) \leftarrow V(s'_t) + \alpha\left(r_{t+1} + V(s'_{t+1}) - V(s'_t)\right)}$. See Methodology for more details.
      i. You can train the player with more sophisticated TD variants if needed.
      ii. The total number of episodes for training is not limited.
3. The n-tuple network for storing afterstate values **can only use 2GB memory at most**.
   a. The structure of the n-tuple network (e.g., 8×4, 4×5, 4×6, etc) is not limited.
   b. At most 536870912 entries are allowed in the lookup table when using 32-bit floating-point numbers. See Methodology for more details.
4. The player **takes actions based on the rewards and the afterstate values, $\boldsymbol{r_t + V(s'_t)}$**:
   a. Not required to perform extra searching, i.e., just expand 4 afterstates.
   b. The program speed should be **at least 50000 actions per second**. (an approximate value, see Scoring Criteria for details)
5. The statistic is required, and the requirements are the same as those in Project 1.
6. Other implementation requirements:
   a. The program should be able to execute in the Linux environment.
      i. C/C++ is highly recommended for TCG projects since the methods involved are sensitive to CPU speed. If you need to use other programming languages (e.g., Python) for the projects, contact TAs for more details.
      ii. The makefile (or CMake) for the program should be provided.
   b. The recorded episodes should be serializable in the same format as in Project 1.
   c. The n-tuple network weight tables should be serializable.

Methodology:

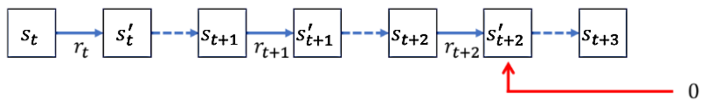
1. The player should calculate all available afterstates, **determine the afterstate values by the n-tuple network**, then select a proper action according to the immediate rewards and the afterstate values. To achieve this, you need to
   a. Design and implement an n-tuple network.
   b. Implement the TD(0) afterstate learning algorithm.
   c. Train an n-tuple network with TD(0) afterstate learning algorithm.
   d. Use immediate rewards and afterstate values to take action.

2. Precautions for implementing an n-tuple network of *2584 Fibonacci*:
   a. **Try the simplest 8×4-tuple network first** (4 vertical lines and 4 horizontal lines). It has 8 weight tables $\Theta_1, \ldots, \Theta_8$, corresponding to 8 tuples, $\phi_1, \ldots \phi_8$:
      i. $\phi_1(s'_t)$ maps the first row to an index for accessing feature weights in Θ_1, e.g.,

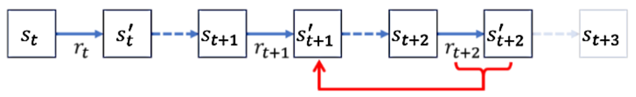
      maps to 0x6241. See Appendix for more details.
      ii. **Be careful to implement feature extraction and index encoding**.
      iii. Once the simplest network works, you may refer to [2] for a more powerful n-tuple network design or even design it yourself.
   b. Note that 2584 Fibonacci **has more possible tile types**, which may significantly increase the size of the network, especially when using a large network.
      i. In 2048, tiles larger than 32768-tile are too rare to be seen. It is safe to consider only 16 possible types in practice. The size of a 6-tuple is 16^6 (64MB).
      ii. In 2584 Fibonacci, the possible tile types are much more than that in 2048. If we set 75025-tile (the 24th) as the maximum, the size of a 6-tuple will be 25^6 (931MB). Note that 75025-tile is not even the largest possible tile in practice.
      iii. Since the limit of the network size is 2GB, to use a 4×6-tuple network in 2584 Fibonacci, you need to allocate a smaller lookup table, e.g., 4×22^6 (1.7GB), and filter out indexes larger than 21 when accessing the lookup table, e.g., `min(index, 21)`. This may introduce some bias, but if there are not too many larger tiles, the problem should be not serious and you can still successfully train a good player.
      iv. You may use a network with different n, e.g., two 6-tuples and two 5-tuples.
   c. Isomorphic patterns with shared weights can speed up the training process.
      i. Be careful with the order of accessing indexes. You should use the same order when accessing isomorphic patterns.
      ii. Note that the simplest 8×4-tuple network does not consider weight sharing. It has 8 lines, each corresponding to a unique weight table. When weight sharing is applied, Only 2 weight tables are used to store 2 sets of tuples: the outer lines, and the inner lines.
      iii. When you are using a larger network, it is highly recommended to use isomorphic patterns. In addition to speed up the training, it also significantly reduces the amount of required memory.
3. Precautions for implementing the TD(0) algorithm:
   An episode is defined as $s_0 \cdots \dashrightarrow s_t \xrightarrow[r_t]{a_t} s'_t \dashrightarrow s_{t+1} \xrightarrow[r_{t+1}]{a_{t+1}} s'_{t+1} \dashrightarrow \cdots s_T$
   a. Take the simplest 8×4-tuple network as an example, there are 8 weight tables $\Theta_1, \ldots, \Theta_8$, corresponding to 8 tuples, $\phi_1, \ldots \phi_8$:
      i. The value function $\boldsymbol{V(s'_t)}$ **is calculated as** $\boldsymbol{\Theta_1[\phi_1(s'_t)] + \cdots + \Theta_8[\phi_8(s'_t)]}$.
      ii. For a value function $V(s'_t)$, only its corresponding 8 feature weights are adjusted: $\Theta_1[\phi_1(s'_t)], \ldots, \Theta_8[\phi_8(s'_t)]$. These 8 feature weights **are adjusted with the same TD error:** $\boldsymbol{\Theta[\phi(s'_t)] \leftarrow \Theta[\phi(s'_t)] + \alpha\big(r_{t+1} + V(s'_{t+1}) - V(s'_t)\big)}$.

b. Do not **forget to train the last afterstate** s'_{T-1}. Its TD target should be 0.
c. Do not **confuse the immediate rewards**. The TD target for $V(s'_t)$ is $r'_{t+1} + V(s'_{t+1})$.
d. Do not **confuse the feature weight and the value function** when calculating TD errors. The TD error for $\Theta[\phi(s'_t)]$ is NOT $(r_{t+1} + \Theta[\phi(s'_{t+1})] - \Theta[\phi(s'_t)])$.
e. Do not **confuse the states and the afterstates**.
f. The backward method and the forward method are both common implementations:
   i. The backward method updates the afterstates from the end to the beginning.

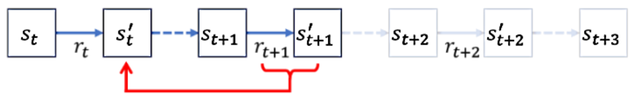
   Step 1: after game over (s_{t+3}), update the last state (s'_{t+2})



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

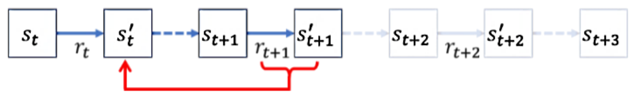


   Step 3: update the previous *afterstate* (s'_t)

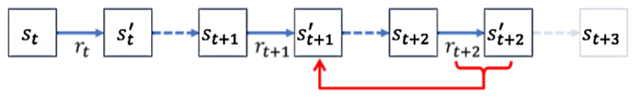


   ii. The forward method updates the afterstates from the beginning to the end.

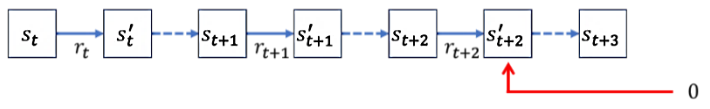
   Step 1: apply an action (s'_{t+1}), update previous state (s'_t).



   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})



4. Precautions for training the n-tuple with TD(0) algorithm learning algorithm:
   a. The initial learning rate α should be calculated based on the number of features. For example, **α can be $0.1 \div 8 = 0.0125$ for the simplest 8×4-tuple network**.
      i. For an isomorphic 4×6-tuple network, $\alpha = 0.1 \div 32$ since it has 32 features.
      ii. Reduce the learning rate α only if the network is converged. In addition, do not reduce the learning rate too fast.
   b. When using the provided framework for training, **always set `--total`, `--block`, and `--limit`**, e.g, `--total=100000 --block=1000 --limit=1000`.

c. Always keep the statistic files and log files. They are useful for further analysis.
   i. To record the training log, use the `tee` command.
   ii. It is recommended to use `tmux` or `screen` command to launch your program.

d. Remember to regularly save network snapshots.
   i. Even if all the code is bug-free, the network may sometimes become worse during the training, especially when the learning rate α is low. Snapshots help you recover from such a situation.
   ii. You can refer to the README.md in the sample code for an example of taking snapshots during the long training.

Scoring Criteria:

1. **Win rate (95 points)**: Calculated by $\lceil \mathrm{WinRate}_{2584} \times 95\% \rceil$.
   a. $\mathrm{WinRate}_{2584}$ is the percentage of the reaching rate of 2584-tile in 1000 games.
   b. A **sample judge program** is provided, you should check the correctness of your work by yourself before the project is due.
2. **Maximum tile (8 points)**: Calculated by $\min(\max(\mathrm{k} - 17, 0), 8)$.
   a. k is the index of the maximum tile calculated in 1000 games. Note that tiles more than 2584-tiles (the 17th tile) are necessary to get the points.
3. **Report (5 points, optional)**: Graded according to the completeness of the report.
   a. Summarize the network design, the used method, the training process, and so on.
4. Penalties:
   a. **Time limit exceeded (–30%)**: 50000 is an approximate speed measured on our standard machine. You should check your work with the judge program.
   b. **Memory limit exceeded (–30%)**: 2GB is the limitation of the n-tuple network, the overall memory usage of the program can be slightly higher. Note that the judge program will NOT detect this issue, so please remember to check it yourself.
   c. **Late work (–30%)**: Late work refers to any modification after the due date.
   d. **No version control (–30%)**: If it is found that there is no version control during the spot check, points will be deducted.
5. The final grade is the sum of the indicators minus the penalties, and the maximum is 100 points. However, if the program does not pass the judge, the final grade is 0.
   a. Note that the report is optional. **You can choose NOT to submit a report**, especially if you have already scored more than 100 points from the other two indicators.

Submission:

1. The submitted files **should be archived as a ZIP file** and **named `ID.zip`**, where **`ID`** is your student ID, e.g. `0356168.zip`.
   a. Pack your **report**, **source files**, **makefiles**, and other relative files.
   b. Submit the archive through the E3 platform.
   c. Do not upload the version control hidden folder, e.g., the `.git` folder.

   d. **Do not upload the trained weight tables**. We will announce another location for placing these weight files.
   e. **Do not upload the statistic file**, e.g., stat.txt. We will regenerate it for grading.
2. The program **should be able to run under the provided Linux workstations**.
   a. Available hosts: tcglinux1.cs.nctu.edu.tw, tcglinux2.cs.nctu.edu.tw, …, tcglinux10.cs.nctu.edu.tw (more information will be announced on the E3 platform)
      i. Use the NYCU CSIT account to log in via SSH.
   b. The projects will be graded on the provided workstations.
      i. You may use your machine for development.
      ii. If the program cannot be compiled or executed on our workstations without additional modifications, the project may be regarded as late work.
   c. Do not occupy the workstations. Contact TAs if the workstations are crowded.
      i. Each workstation has 4 threads and 7.8GB of memory, which can be shared by 3–4 people. **Pay attention to the resources used by your program**.
3. Version control (e.g., Github or Bitbucket) is required during the development.

References:

[1] M. Szubert and W. Jaśkowski, "**Temporal difference learning of N-tuple networks for the game 2048**," DOI: 10.1109/CIG.2014.6932907. [Online]. Available: http://www.cs.put.poznan.pl/mszubert/pub/szubert2014cig.pdf

[2] K. Matsuzaki, "**Systematic selection of N-tuple networks with consideration of interinfluence for game 2048**," DOI: 10.1109/TAAI.2016.7880154.

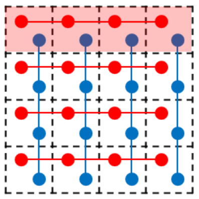
[3] K.-H. Yeh, I-C. Wu, C.-H. Hsueh, C.-C. Chang, C.-C. Liang, and H. Chiang, "Multistage temporal difference learning for 2048-like games," DOI: 10.1109/TCIAIG.2016.2593710.

[4] W. Jaśkowski, "Mastering 2048 with delayed temporal coherence learning, multistage weight promotion, redundant encoding and carousel shaping," DOI: 10.1109/TCIAIG.2017.2651887.

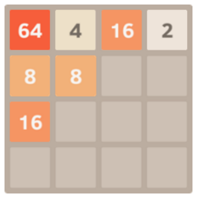
[5] H. Guei, L.-P. Chen, and I-C. Wu, "Optimistic Temporal Difference Learning for 2048," DOI: 10.1109/TG.2021.3109887.

Appendix:

1. Using the simplest 8×4-tuple network in 2048:
   This network has 8 different 4-tuples, $\phi_1, \dots, \phi_8$, including 4 horizontal lines, and 4 vertical lines, corresponding to different cell locations. For example, the horizontal 4-tuple highlighted in red, says ϕ_1, corresponds to cell locations 0, 1, 2, 3.



   When applying ϕ_1 to a given afterstate s'_t, it extracts a feature $\phi_1(s'_t)$ by extracting tile values at the predefined cell locations 0, 1, 2, 3. Take the right board as an example, $\phi_1(s'_t) = [64, 4, 16, 2]$. By ignoring tiles larger than 32768-tiles, we can simply encode this feature to 0x6241.



Each 4-tuple is with a unique weight table, i.e, there are 8 weight tables in total.
A weight table is a large array to store weights of all possible features, i.e., for a 4-tuple, the size of a weight table is 16^4.

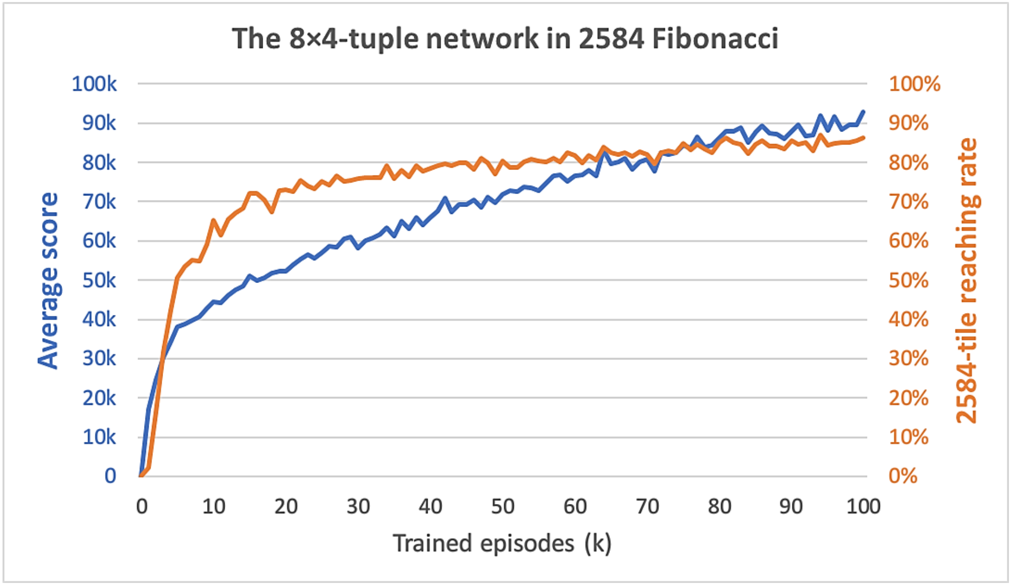
Suppose that the weight table corresponds to ϕ_1 is Θ_1. The feature weight is then accessed by $\Theta_1[0x6241]$. The value estimation can finally be calculated by all corresponding feature weights: $\boldsymbol{V(s_t') = \Theta_1[\phi_1(s_t')] + \cdots + \Theta_8[\phi_8(s_t')]}$.

With the afterstate value function $V(s_t')$, the best move of a given state s_t can be determined by $\boldsymbol{\pi(s_t) = \mathbf{argmax}_{a_t}\left(r_t + V(s_t')\right)}$. Suppose that the backward TD(0) method is used, the weight tables are adjusted after the end of an episode. Let an episode be defined as: $s_0 \cdots \dashrightarrow s_t \xrightarrow[r_t]{a_t} s_t' \dashrightarrow s_{t+1} \xrightarrow[r_{t+1}]{a_{t+1}} s_{t+1}' \dashrightarrow \cdots s_T$, the adjustments of each afterstate values $V(s_t')$ are described as follows.

First, adjust the value of the last afterstate s_{T-1}'. The adjustment is $\alpha(0 - V(s_{T-1}'))$. Note that α is the learning rate, assuming $\alpha = 0.1 \div 8$, since there are 8 feature weights; and $V(s_{T-1}') = \Theta_1[\phi_1(s_{T-1}')] + \cdots + \Theta_8[\phi_8(s_{T-1}')]$. The adjustment should be applied to the 8 feature weights related to s_{T-1}', say $\Theta_i[\phi_i(s_{T-1}')]$ for i from 1 to 8. Namely, $\Theta_i[\phi_i(s_{T-1}')]$ is adjusted as $\boldsymbol{\Theta_i[\phi_i(s_{T-1}')] \leftarrow \Theta_i[\phi_i(s_{T-1}')] + \alpha(0 - V(s_{T-1}'))}$.

Then, for the other afterstates s_t' from $t = T - 2$ to $t = 0$, the adjustment is $\alpha(r_{t+1} + V(s_{t+1}') - V(s_t'))$; and the related feature weights of each s_t' are adjusted by $\boldsymbol{\Theta_i[\phi_i(s_t')] \leftarrow \Theta_i[\phi_i(s_t')] + \alpha(r_{t+1} + V(s_{t+1}') - V(s_t'))}$.

2. The progress of training a player with the simplest 8×4-tuple network in 2584 Fibonacci:


The 8×4-tuple network in 2584 Fibonacci
Average score
100k
90k
80k
70k
60k
50k
40k
30k
20k
10k
2584-tile reaching rate
Trained episodes (k)