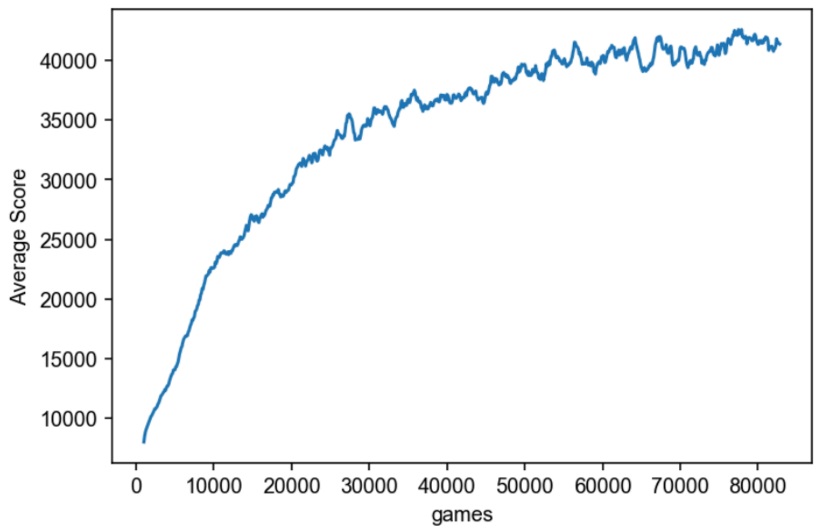
RL Lab1 Report

Name : 林伯偉　　Student ID : 109612019

• Report



• Bonus

1. Describe the implementation and the usage of n-tuple network.

- Implementation

  n-tuple network is a type of neural network architecture used in reinforcement learning, particularly in the context of developing agents for playing games. It is particularly for problems with continuous state spaces by approximating the Q-function and enable agents to learn and make optimal decisions in complex environments.

- Usage

  there is some usage of n-tuple leaning such as:

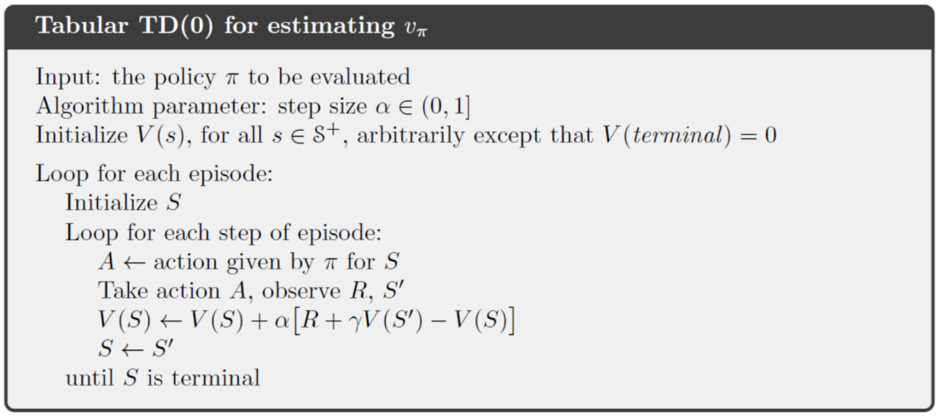
  - Game Connect-4 (https://link.springer.com/chapter/10.1007/978-3-642-32937-1_19)
  - Game Othello (https://repository.essex.ac.uk/3820/)
  - Game 2048

2. Explain the mechanism of TD(0).

Temporal-Difference TD(0) learning updates the estimated value of a state V for policy based on the reward the agent received and the value of the state it transitioned to. Specifically, if our agent is in a current state St takes the action At and receives the reward Rt then we update our estimate of V following:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

The value of [Rt+1+γV(St+1) – V(St)] is called the TD error, which is the difference between the current estimate for Vt, the discounted value estimate of Vt+1 and the actual reward gained from transitioning between St and St+1. Hence correcting the error in Vt slowly over many passes through. α is a constant step-size parameter that impacts how quickly the TD algorithm learns. For the algorithms following, we generally require α to be suitably small to guarantee convergence, however the smaller the value of alpha the smaller the changes made for each update, and therefore the slower the convergence. A simple mechanism of TD(0) can be seen below.



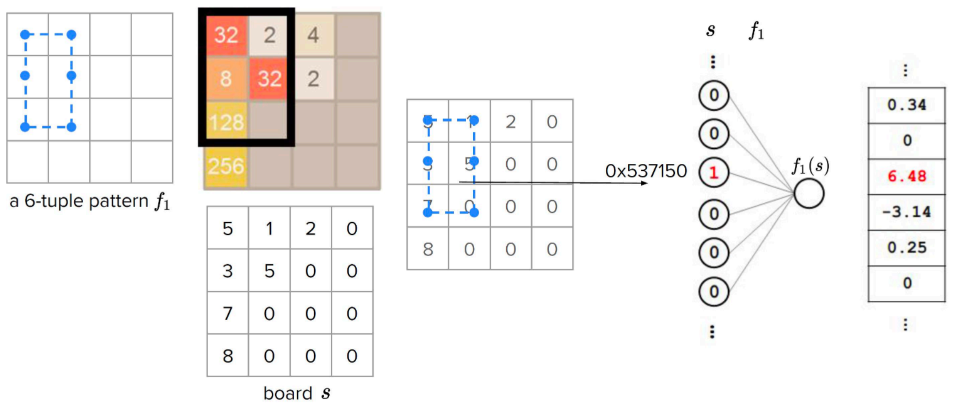
Tabular TD(0) for estimating v_π

```
Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$
Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    until $S$ is terminal
```

3. Describe your implementation in detail including action selection and TD-backup diagram

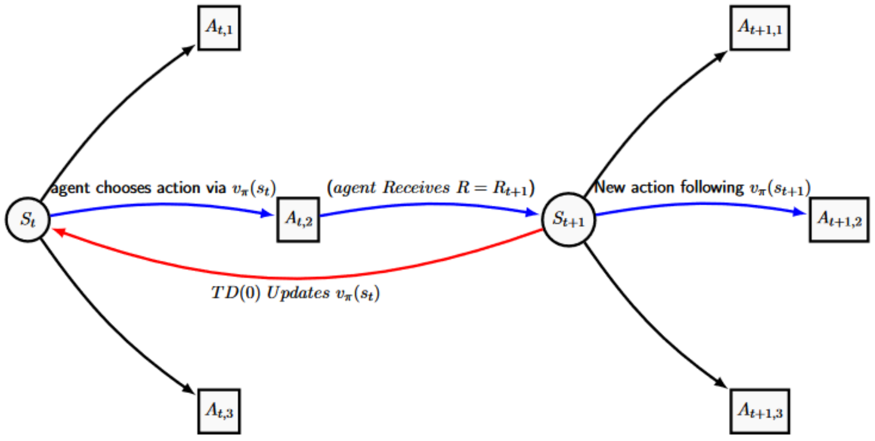
- **Representation**

The game 2048 has too many states to fit in the memory. An n-tuple network is introduced to break the board into smaller patterns called "tuples", so each tuple has a feasible number of states to fit in memory.The network stores the maximum expected total score to be obtained by playing a board state.



- **Action selection**

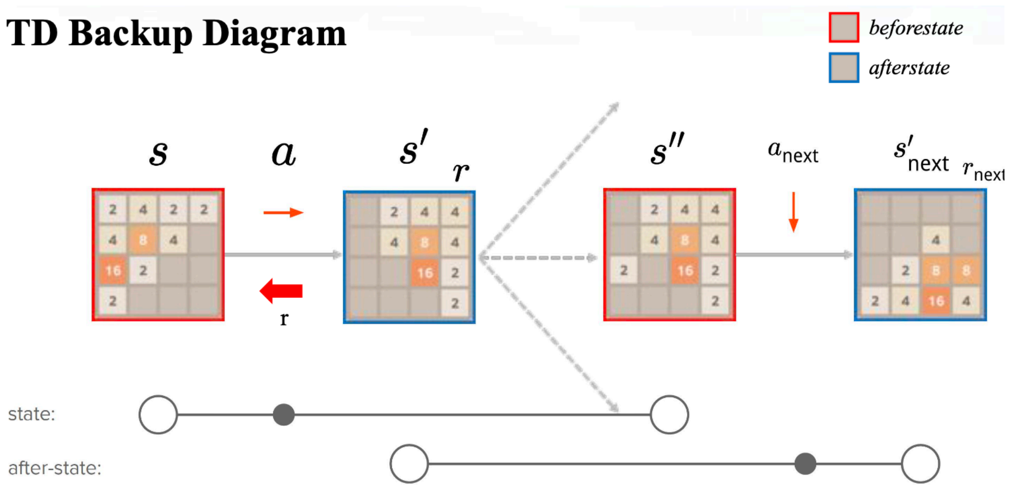
  To decide the best action given a board, the network works like an ensemble system by asking each tuple what they think of doing each possible action on the board and taking the average of their opinion as the final output. The action with the highest output is chosen.



$A_{t,1}$
$A_{t,2}$
$A_{t,3}$
S_t
agent chooses action via $v_\pi(s_t)$
$(agent\ Receives\ R = R_{t+1})$
S_{t+1}
New action following $v_\pi(s_{t+1})$
$A_{t+1,1}$
$A_{t+1,2}$
$A_{t+1,3}$
$TD(0)\ Updates\ v_\pi(s_t)$

- **Learning**

  The network learns by improving its expected score for a state using the reward from the best action and the expected score of the next state. Concretely, the target score is the action reward plus the expected score on the resulting next state (i.e. *TD(0)*). Since a state learns from the next state, each gameplay is learned backward to enhance the back propagation experiences, which the TD backup diagram and pseudo code for training is shown below.



A pseudocode of the game engine and training. (modified backward training method)

```
function PLAY GAME
  $score \leftarrow 0$
  $s \leftarrow$ INITIALIZE GAME STATE
  while IS NOT TERMINAL STATE($s$) do
    $a \leftarrow \underset{a' \in A(s)}{\text{argmax}}$ EVALUATE($s, a'$)
    $r, s', s'' \leftarrow$ MAKE MOVE($s, a$)
    SAVE RECORD($s, a, r, s', s''$)
    $score \leftarrow score + r$
    $s \leftarrow s''$
  for ($s, a, r, s', s''$) FROM TERMINAL DOWNTO INITIAL do
    LEARN EVALUATION($s, a, r, s', s''$)
  return $score$

function MAKE MOVE($s, a$)
  $s', r \leftarrow$ COMPUTE AFTERSTATE($s, a$)
  $s'' \leftarrow$ ADD RANDOM TILE($s'$)
  return ($r, s', s''$)
```

TD-state

```
function EVALUATE($s, a$)
  $s', r \leftarrow$ COMPUTE AFTERSTATE($s, a$)
  $S'' \leftarrow$ ALL POSSIBLE NEXT STATES($s'$)
  return $r + \Sigma_{s'' \in S''} P(s, a, s'') V(s'')$

function LEARN EVALUATION($s, a, r, s', s''$)
  $V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$
```