# Temporal Difference Learning of N-Tuple Networks for the Game 2048

Marcin Szubert    Wojciech Jaśkowski

Institute of Computing Science

POZNAN UNIVERSITY OF TECHNOLOGY


August 27, 2014


## Rules

- single-player, nondeterministic
- $4 \times 4$ board
- actions: left, right, up, down
- merging: score the sum
- every move: 2 or 4 in random position
- goal: construct tile 2048

http://gabrielecirulli.github.io/2048


2048
digiplex.in
2048 Number puzz
Estoty Entertainment La
2048 Plus
Sun.Rain
2048
2048 Game
2048
Presselite
2048
Uberspot
Circular 2048
TTY Games
2048 by Gabriele C
Gabriele Cirulli
Super 2048
Bo Long
2048 Extended
Alexander Kulikovski
2048 PvP Arena
Estoty Entertainment La
Hexic 2048
SmartPlayland
2048
2048 Bird World
Doge 2048
mcarrano
2048
S2Apps.com
2048
MS Developers
2048
Androbros
2048 (Ads Free)
tpcstld
2048
PhoneMaster
2048
Puzzle Game Lab
Advanced 2048
Bo Long
2048 Mania
Smoote Mobile
2048
Icomania Logo Quiz 204
2048 Puzzle
SubMad Group
2048
Android Ardent
2048 Collection
Prabh Arora
2048 (using Kivy)
Mathieu Virbel
2048 game
QUEEN GAME
2048 Total
Free Games Developers
2048
Shinnx Droid
2048
Endemion
2048 Fibonacci
Time Killing Machine
2048
easygames
2048 game
Time Killing Machine
LoL 2048
miniboss
2048 Numbers
CanadaDroid
2048!
Tigao Mobile
2048 Monsters
BitMango
2048
Shreyas Sood
2048
Candy Crush
2048 Fibonacci
MS Developers
2048
delight.im
2048
Pavel Golubev
2048 Super (2048
KK App Team
2048 Backwards
Mobile Bats
2048
2048: TwoZeroFourEigh
Words 2048
Silver Labs
2048 Color Match
Parudi Ltd

## Motivation

- Popularity:
  - 4mln visitors in the first weekend
  - 3000 man-years in 3 weeks
- Easy to learn but not trivial to master → ideal test bed for CI/AI
- No previous studies

## Goal

- **Learning without expert knowledge and without search**

## 2048 as MDP

- $S$ — **states**: board positions,
- $A$ — **actions**: legal moves,
- $R(s, a)$ — **reward**: score obtained by action $a$ in state $s$
- $P(s, a, s'')$ — stochastic **transition function**, probability of transition to state $s''$ in result of taking action $a$ in state $s$. Defined implicitely by the game rules.

## Value function

$$V : S \to \mathbb{R}$$

$V(s)$ — expected number of points the agent will get from state $s$ till the end of the game.

## Making moves with $V$

$$\pi(s) = \operatorname*{argmax}_{a \in A(s)} \left[ R(s,a) + \sum_{s'' \in S} P(s,a,s'') V(s'') \right].$$

## Learning of $V$

- After a move the agents gets a new experience $\langle s, a, r, s'' \rangle$
- Modify $V$ in response to the experience by Sutton's TD(0) update rule:

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

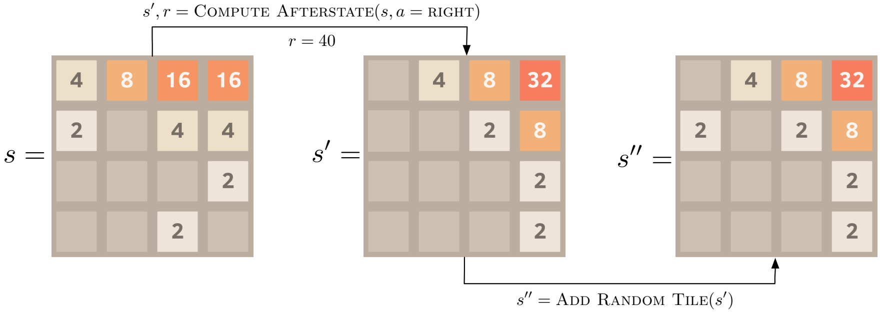
$\alpha$ — learning rate

## General Idea

- Reconcile neighboring states $V(s)$ and $V(s'')$, so that (ideally, in the long run) Bellman equaltion holds:

$$V(s) = \max_{a \in A(s)} \left( R(s,a) + \sum_{s'' \in S} P(s, a, s'') V(s'') \right)$$

# Afterstates



# State Value Function

Move selection:

$$\pi(s) \leftarrow \underset{a \in A(s)}{\operatorname{argmax}} \left( R(s,a) + \sum_{s'' \in S} P(s,a,s'') V(s'') \right)$$

Learning:

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

## State Value Function

Move selection:

$$\pi(s) \leftarrow \operatorname*{argmax}_{a \in A(s)} \left( R(s,a) + \sum_{s'' \in S} P(s,a,s'') V(s'') \right)$$

Learning:

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

## Afterstate Value Function

Move selection:

$$\pi(s) \leftarrow \operatorname*{argmax}_{a \in A(s)} \left( R(s,a) + V(s') \right)$$

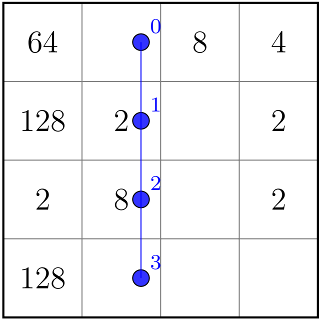
Learning:

$$V(s') \leftarrow V(s') + \alpha(r_{next} + V(s_{next}) - V(s'))$$

$r_{next}$, $s_{next}$ are obtained by taking an action from $s''$ according to the current policy.

# Value Function Approximation with N-tuple Networks

2048 has ca. $10^{21}$ states $\rightarrow$ function approximator




| 0123 | weight |
|---|---|
| 0000 | 3.04 |
| 0001 | −3.90 |
| 0002 | −2.14 |
| ⋮ | ⋮ |
| 0010 | 5.89 |
| ⋮ | ⋮ |
| **0130** | **-2.01** |
| ⋮ | ⋮ |

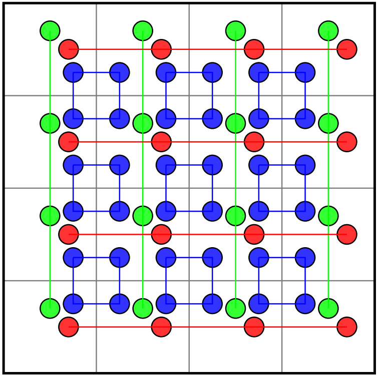
Network response:

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

Buro, Michael, "From Simple Features to Sophisticated Evaluation Functions", 1999
Lucas, Simon M., "Learning to Play Othello with N-tuple Systems", 2007

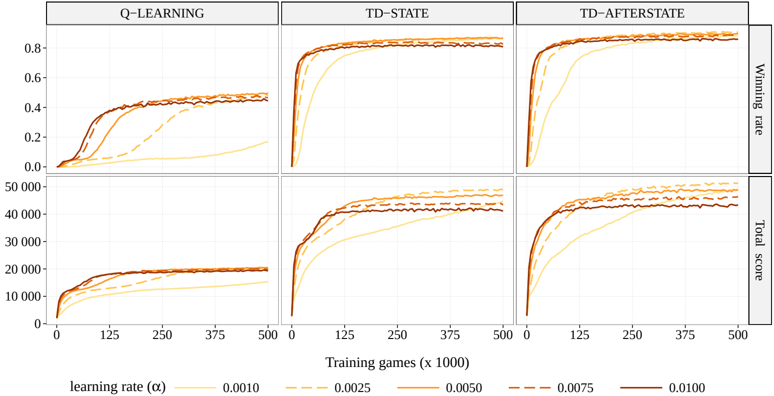
## Settings

- Systematic N-Tuple Network with 17 tuples of size 4 $\longrightarrow$ 860 625 weights.
- TD-state, TD-afterstate, Q-learning
- 0.5 mln training games



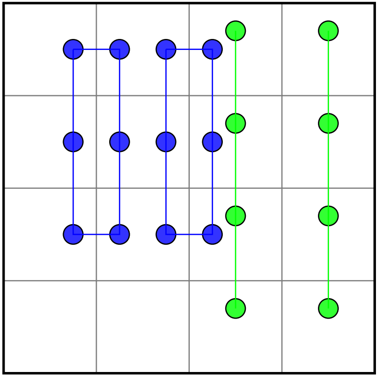
# Comparison of Learning Methods

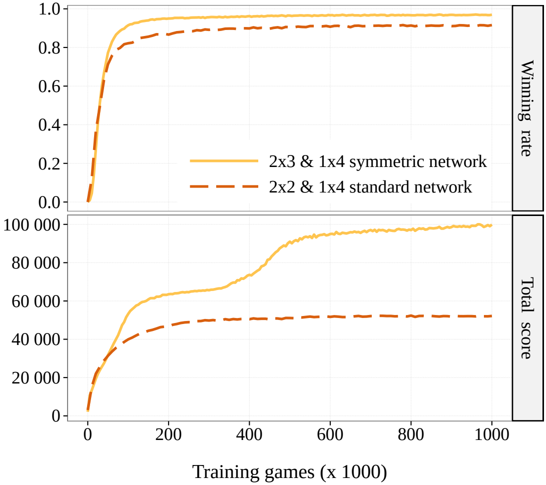
| Algorithm | Best winning rate | Best total score | CPU time [s] |
|---|---|---|---|
| Q-LEARNING | $0.4980 \pm 0.0078$ | $20504.6 \pm 163.5$ | $3136.8 \pm 61.7$ |
| TD-STATE | $0.8672 \pm 0.0122$ | $48929.6 \pm 702.5$ | $24334.7 \pm 405.7$ |
| TD-AFTERSTATE | $0.9062 \pm 0.0051$ | $51320.9 \pm 358.4$ | $7967.5 \pm 165.3$ |



## Settings

- TD-afterstate with $\alpha = 0.0025$
- two tuples of size 4 and two of size 6 (22 882 500 weights)
- exploiting the board symmetry
- 1 mln training games






## Winning rate

- "Small": ≈ 91%
- "Large": ≈ 98%

## Summary

- 2048: new interesting **challenge for AI/CI** with simple rules and highly popular, quick to play (20ms for one game)
- Learned a very quick agent, win ratio nearly 98% at 1-ply:
  - **afterstate value function** — for environments where the agent can simulate the immediate effects of its moves, but it is difficult to obtain the entire state transition.
  - **n-tuple network** — evidence of scalability (22 mln weights)

## Summary

- 2048: new interesting **challenge for AI/CI** with simple rules and highly popular, quick to play (20ms for one game)
- Learned a very quick agent, win ratio nearly 98% at 1-ply:
  - **afterstate value function** — for environments where the agent can simulate the immediate effects of its moves, but it is difficult to obtain the entire state transition.
  - **n-tuple network** — evidence of scalability (22 mln weights)

## Open questions

- What is the expected score of the optimal policy? Currently $[99\,916, 3\,932\,100)$
- Highest possible winning rate for 2048, 4096, 8192, 16384, 32768,. . . ?

# Results: Optimizing the Learning Rate

| Learning rate | Winning rate | | |
|---|---|---|---|
| | Q-LEARNING | TD-STATE | TD-AFTERSTATE |
| 0.0010 | $0.1672 \pm 0.0262$ | $0.8622 \pm 0.0059$ | $0.8821 \pm 0.0068$ |
| 0.0025 | $0.4796 \pm 0.0058$ | $\mathbf{0.8672 \pm 0.0122}$ | $\mathbf{0.9062 \pm 0.0051}$ |
| 0.0050 | $\mathbf{0.4980 \pm 0.0078}$ | $0.8660 \pm 0.0120$ | $0.8952 \pm 0.0089$ |
| 0.0075 | $0.4658 \pm 0.0090$ | $0.8253 \pm 0.0131$ | $0.8867 \pm 0.0077$ |
| 0.0100 | $0.4438 \pm 0.0103$ | $0.8083 \pm 0.0170$ | $0.8601 \pm 0.0090$ |

Winning rate of learning agents after 0.5 mln training games with 95% confidence interval.

# Game engine

```
function PLAY GAME
    score ← 0
    s ← INITIALIZE GAME STATE
    while ¬IS TERMINAL STATE(s) do
        a ← argmax_{a′∈A(s)} EVALUATE(s, a′)
        r, s′, s″ ← MAKE MOVE(s, a)
        if LEARNING ENABLED then
            LEARN EVALUATION(s, a, r, s′, s″)
        score ← score + r
        s ← s″
    return score
function MAKE MOVE(s, a)
    s′, r ← COMPUTE AFTERSTATE(s, a)
    s″ ← ADD RANDOM TILE(s′)
    return (r, s′, s″)
```

# Q-Learning

**function** $\textsc{Evaluate}(s, a)$

    **return** $V_a(s)$

**function** $\textsc{Learn Evaluation}(s, a, r, s', s'')$

    $v_{next} \leftarrow \max_{a' \in A(s'')} V_{a'}(s'')$

    $V_a(s) \leftarrow V_a(s) + \alpha(r + v_{next} - V_a(s))$

Figure: The *action evaluation function* and Q-learning.

# State Value Function TD-learning

**function** $\textsc{Evaluate}(s, a)$
    $s', r \leftarrow \textsc{Compute Afterstate}(s, a)$
    $S'' \leftarrow \textsc{All Possible Next States}(s')$
    **return** $r + \sum_{s'' \in S''} P(s, a, s'') V(s'')$

**function** $\textsc{Learn Evaluation}(s, a, r, s', s'')$
    $V(s) \leftarrow V(s) + \alpha(r + V(s'') - V(s))$

Figure: The *state evaluation function* and TD(0).

# Afterstate Value Function TD-learning

**function** EVALUATE($s, a$)
    $s', r \leftarrow$ COMPUTE AFTERSTATE($s, a$)
    **return** $r + V(s')$

**function** LEARN EVALUATION($s, a, r, s', s''$)
    $a_{next} \leftarrow \mathrm{argmax}_{a' \in A(s'')}$ EVALUATE($s'', a'$)
    $s'_{next}, r_{next} \leftarrow$ COMPUTE AFTERSTATE($s'', a_{next}$)
    $V(s') \leftarrow V(s') + \alpha(r_{next} + V(s'_{next}) - V(s'))$

Figure: The *afterstate evaluation function* and a dedicated variant of the TD(0) algorithm.