Temporal Difference Learning

- Sutton, R.S. and Barto, A.G., Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.
  - http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html
  - Bible in this area.

Acknowledgement:

- Some of the slides in this chapter are partially modified from those by Hsin-Ti Tsai (蔡心迪) and Kun-Hao Yeh (葉騉豪).

Outline

- Reinforcement Learning
- Temporal Difference Learning
- Case Studies
  - 2048
  - Connect6

Reinforcement Learning

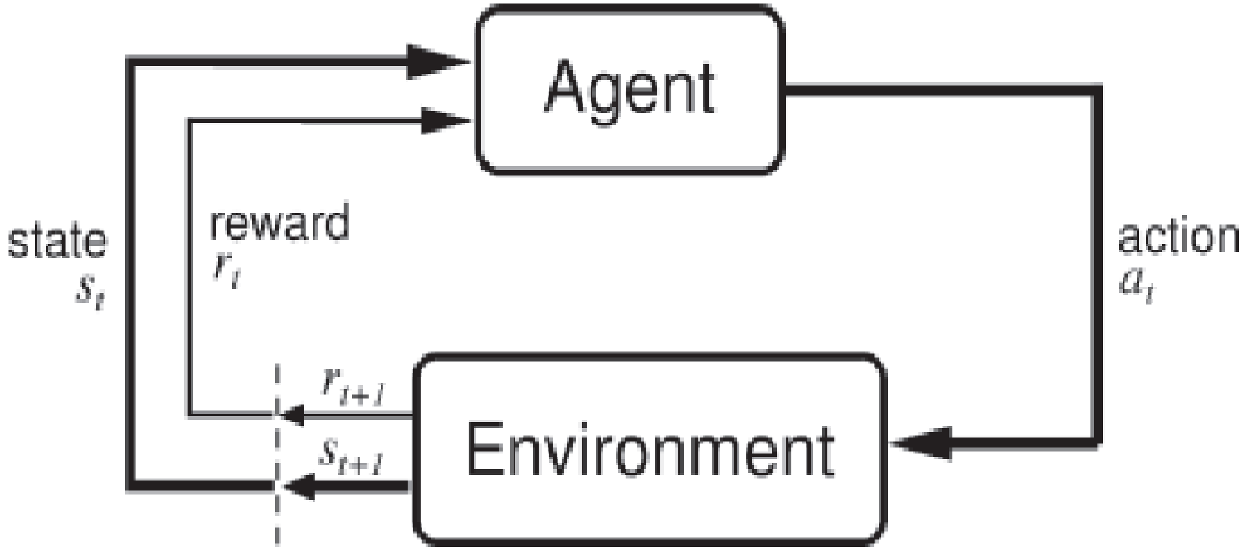
- A **computational approach** to learning from **interaction**
  - Explore designs for machines that are effective in
    - solving learning problems of scientific or economic interest,
    - evaluating the designs through mathematical analysis or computational experiments.
  - Focus on **goal-directed learning** from interaction, when compared with other approaches to machine learning.
  - The learner must discover which actions yield the most reward by trying them.
    - Two characteristics: most important distinguishing features of reinforcement learning.
      - **trial-and-error search**
      - **delayed reward**
  - different from **supervised learning**, like statistical pattern recognition, and artificial neural networks.

Successful Examples

- In AI, it has been used to defeat human champions at games of skill (Tesauro, 1994);
  - For Connect6/2048/Threes!, it has been used to reach the top levels.
  - For Go, it has been used in Monte-Carlo Tree Search.
- In robotics, to fly stunt maneuvers in robot-controlled helicopters (Abbeel et al., 2007).
- In neuroscience it is used to model the human brain (Schultz et al., 1997);
- In psychology to predict animal behavior (Sutton and Barto, 1990).
- In economics, it is used to understand the decisions of human investors (Choi et al., 2007), and to build automated trading systems (Nevmyvaka et al., 2006)
- In engineering, it has been used to allocate bandwidth to mobile phones and to manage complex power systems (Ernst et al., 2005).

Agent-Environment Interaction Framework

- Agent: The learner and decision-maker.
- Environment: The thing it interacts with, comprising everything outside the agent.
- State: whatever information is available to the agent.
- Reward: single numbers.



Agent-Environment Interaction

- The agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, \ldots$, or $t = 0, 1, 2, \ldots T$ (T: the terminated time, if any.)
  - $S = \{s_0, s_1, s_2, \ldots, s_t, \ldots\}$.
    - s_t: some representation of the environment's state at time step t.
  - a_t: action at time step t,
  - r_t: rewards at time step t,
  - π_t: the agent's policy,
    - a mapping from states to probabilities of selecting each possible action
    - $\pi_t(s, a)$: the probability that $a_t = a$ if $s_t = s$.

Examples

- 2048-like games:
  - Make moves with rewards. Then, tiles are popped up randomly.
- Bioreactor:
  - Suppose reinforcement learning is being applied to determine moment-by-moment temperatures and stirring rates for a bioreactor
- Pick-and-Place Robot:
  - Consider using reinforcement learning to control the motion of a robot arm in a repetitive pick-and-place task.
- Recycling Robot
  - A mobile robot has the job of collecting empty soda cans in an office environment.
  - This agent has to decide whether the robot should
    - (1) actively search for a can for a certain period of time,
    - (2) remain stationary and wait for someone to bring it a can, or
    - (3) head back to its home base to recharge its battery.

Example: Recycling Robot

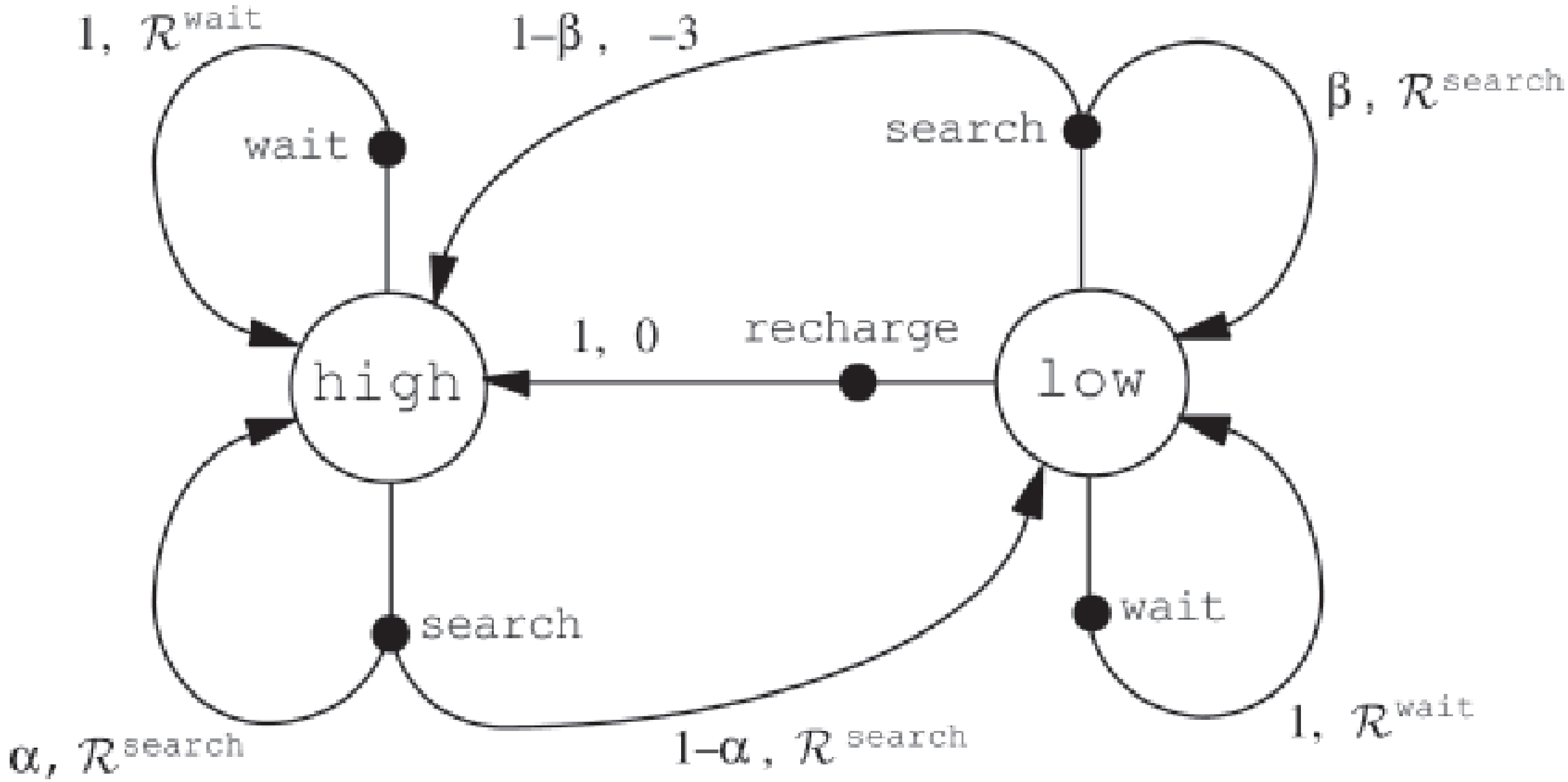
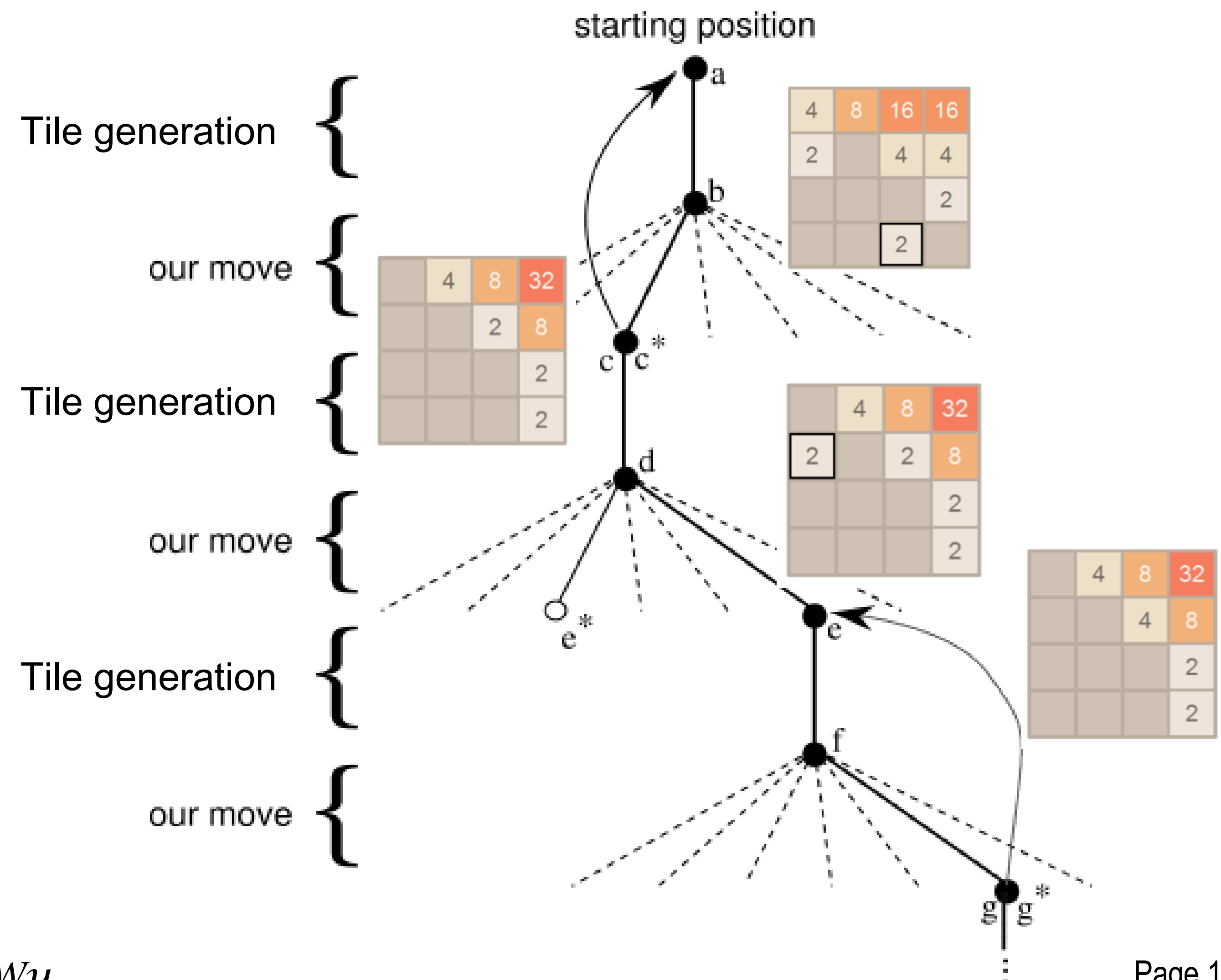


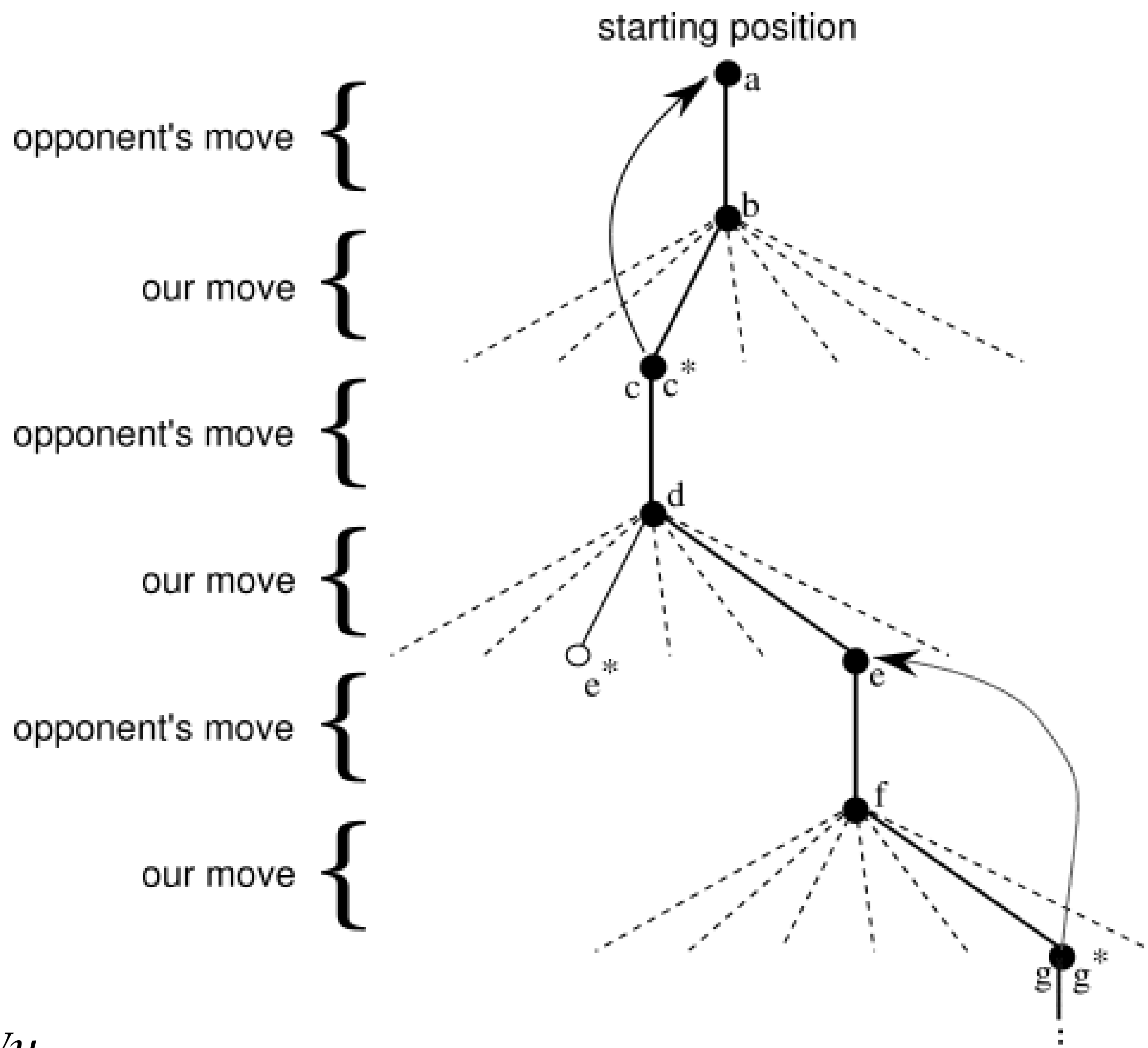
Figure 3.3: Transition graph for the recycling robot example.

$s = s_t$	$s' = s_{t+1}$	$a = a_t$	$\mathcal{P}^a_{ss'}$	$\mathcal{R}^a_{ss'}$
high	high	search	α	$\mathcal{R}^{\texttt{search}}$
high	low	search	$1 - \alpha$	$\mathcal{R}^{\texttt{search}}$
low	high	search	$1 - \beta$	-3
low	low	search	β	$\mathcal{R}^{\texttt{search}}$
high	high	wait	1	$\mathcal{R}^{\texttt{wait}}$
high	low	wait	0	$\mathcal{R}^{\texttt{wait}}$
low	high	wait	0	$\mathcal{R}^{\texttt{wait}}$
low	low	wait	1	$\mathcal{R}^{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	$0.$

2048


starting position
Tile generation
our move
Tile generation
our move
Tile generation
our move
a
b
c
c*
d
e
e*
f
g
g*
4 8 16 16
2 4 4
2
2
4 8 32
2 8
2
2
4 8 32
2 2 8
2
2
4 8 32
4 8
2
2

Tic-Tac-Toe


starting position
opponent's move
our move
opponent's move
our move
opponent's move
our move
a
b
c
c*
d
e*
e
f
g
g*

Rewards

- Rewards: A way of formulating goal:
- Example: 2048
  - Straightforwardly set the earned scores to rewards.
- Example: recycling robot.
  - 0 for most of the time,
  - +1 for each can collected,
  - -3 in case of running out of electricity.
- Example: learning to play checkers/chess/Go,
  - +1 for winning,
  - -1 for losing, and
  - 0 for drawing and for all nonterminal positions.

Comments

- Critical:
  - **the rewards we set up truly indicate what we want accomplished.**
- Reward signals:
  - **A way of "what"** you want to achieve, **not "how"** you want to it achieved. (no prior knowledge about "how")
  - Not the place to impart to the agent prior knowledge about how to achieve what we want it to do.
  - Example: for chess-playing, rewarded only for real winning, not for sub-goals, like taking pieces.

Goals

- Goal: Maximize the *expected return*.
- Returns: Total rewards of the episode
  - $R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots$
- Episodic tasks: (with a terminal state)
  - Example: 2048, chess, Go, etc.
  - $R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$
    - T: a final time step. (s_T is a terminal state.)
    - Episode: $S^+ = \{s_{t+1}, s_{t+2}, s_{t+3}, \dots, s_T\}$.
      Note that: $S = \{s_{t+1}, s_{t+2}, s_{t+3}, \dots, s_{T-1}\}$.
- Continuing tasks: T=∞.
  - Example: Recycling Robot.
  - Problem: R_t could be infinite.
  - Solution: Add the concept of "discounting".
  - Change it to "discounted return":
  - $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$
    - γ: discount rate. $0 \le \gamma \le 1$

Goal

- Rewards: A way of formulating goal:
  - Example: 2048
    - Straightforwardly set the earned scores to rewards.
- Goal: Maximize the *expected return*.
  - Returns: Total rewards of the episode
    - $R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$
      - T: a final time step. (s_T is a terminal state.)
    - Example: 2048, chess, Go, etc.
- Episodic tasks: (with a terminal state)
  - Example: 2048, chess, Go, etc.
  - $R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$
    - T: a final time step. (s_T is a terminal state.)
    - Episode: $S^+ = \{s_{t+1}, s_{t+2}, s_{t+3}, \ldots, s_T\}$.
      Note that: $S = \{s_{t+1}, s_{t+2}, s_{t+3}, \ldots, s_{T-1}\}$.

Temporal Difference (TD) Learning

- Temporal Difference Learning
  - Is a kind of reinforcement learning
  - Is to adjust weights automatically
  - Was successfully applied to many games such as
    - Backgammon
    - Checkers
    - Chess
    - Shogi
    - Go
    - Chinese Chess
    - Connect6
    - 2048

Value Function

- Expected return or
  Estimate how good it is for the agent to be in a given state
  - $V(s)$: the estimated value of state s.
    - the expected return when starting in s thereafter.
    - also called the state-value function.
  - $Q(s, a)$: the value of taking action a in state s under a policy π.
    - the expected return starting from s, taking the action a:
    - called the action-value function.
  - Omit policy π. (See Reinforcement Learning)

TD Prediction

- Prediction:
  - $V(s_t)$ is approximate to actual return R_t.
  - Error: $\delta_t = R_t - V(s_t)$.
  - Adjust: $V(s_t) = V(s_t) + \alpha\delta_t = V(s_t) + \alpha\big(R_t - V(s_t)\big)$
    - α: a step-size parameter to control the learning rate.
- Problem:
  - To get R_t, we must wait until the episode ends.
  - Can we do that earlier?

TD(0)

- Change error:
  - From: $\delta_t = R_t - V(s_t)$
  - To: $\delta_t = \left(r_{t+1} + V(s_{t+1})\right) - V(s_t)$

    $= r_{t+1} + V(s_{t+1}) - V(s_t)$
    - For simplicity, $\gamma = 1$.
- Thus, change value function
  - From: $V(s_t) = V(s_t) + \alpha\left(R_t - V(s_t)\right)$
  - To: $V(s_t) = V(s_t) + \alpha\left(r_{t+1} + V(s_{t+1}) - V(s_t)\right)$
- This is called TD(0).

TD(λ)

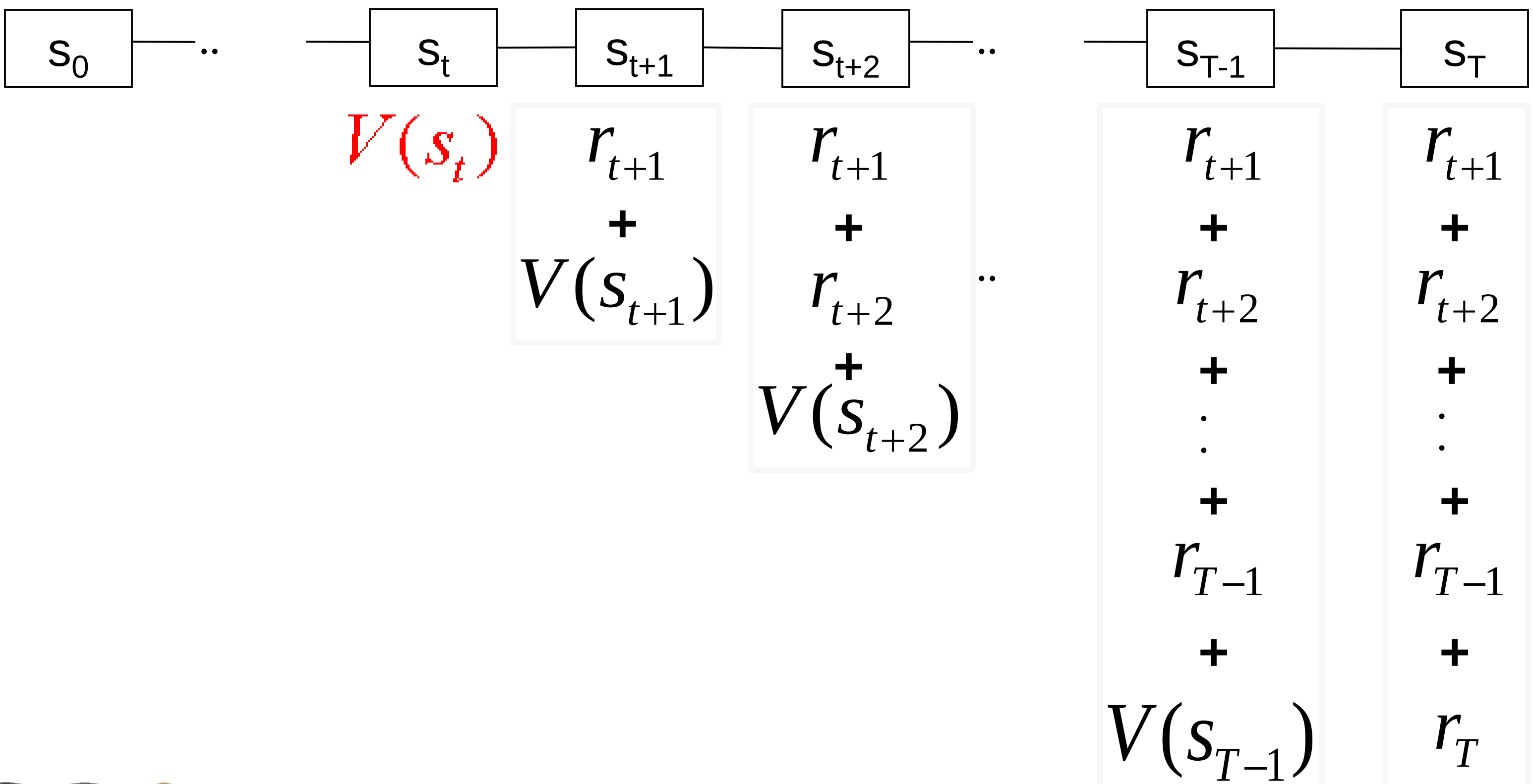
- Change error:
  - From: $\delta_t = R_t - V(s_t)$
  - To:

$$\delta_t = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} V(s_{t+n}) + \lambda^{T-t-1} V(s_T) - V(s_t)$$

- This is called TD(λ).

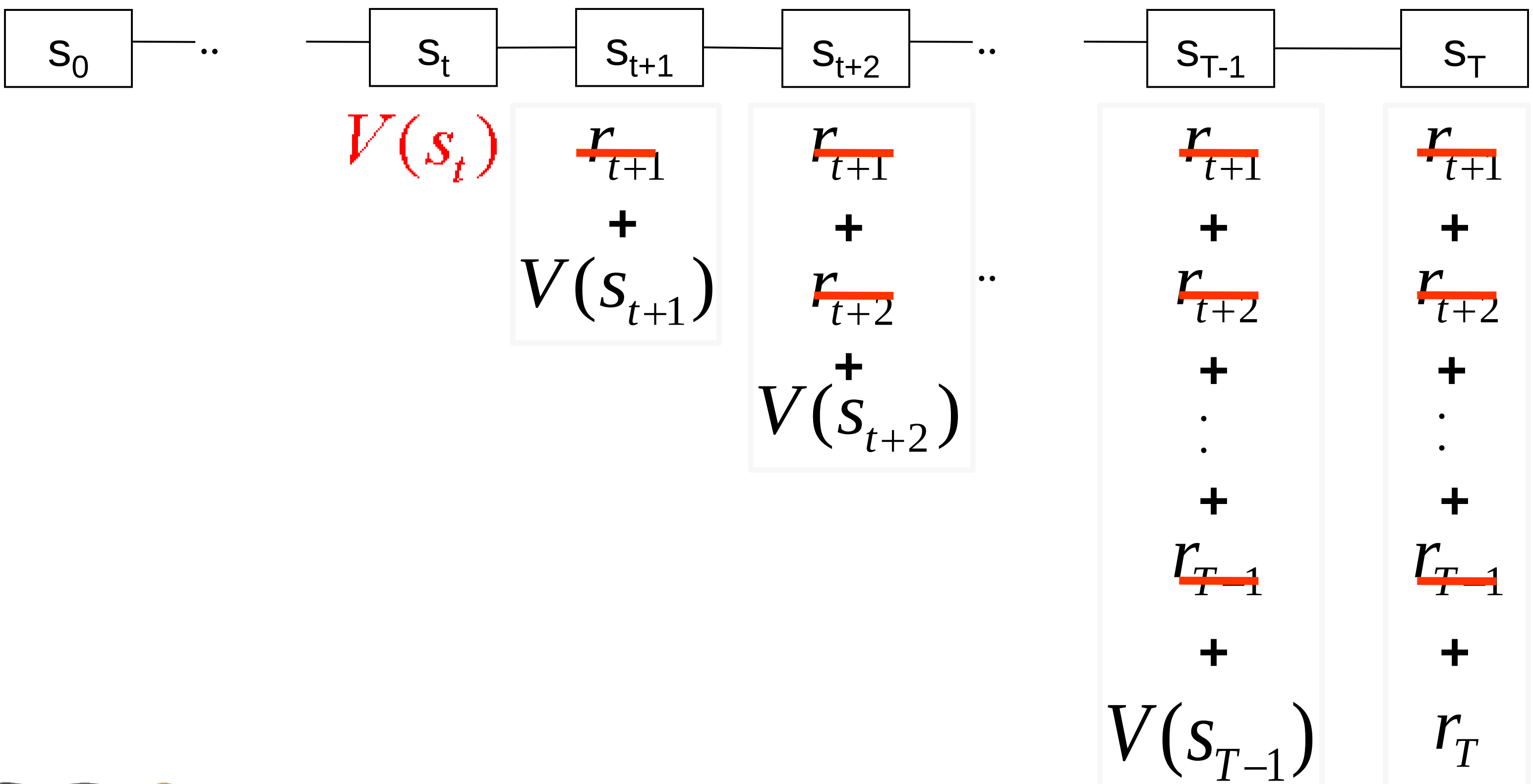
TD Learning

- TD(λ)



TD Learning

- TD(λ)



TD Learning

- TD(λ)

s_0 — .. — s_t — s_{t+1} — s_{t+2} — .. — s_{T-1} — s_T

$V(s_t)$

$\cancel{r_{t+1}} + V(s_{t+1})$

$\cancel{r_{t+1}} + \cancel{r_{t+2}} + V(s_{t+2})$

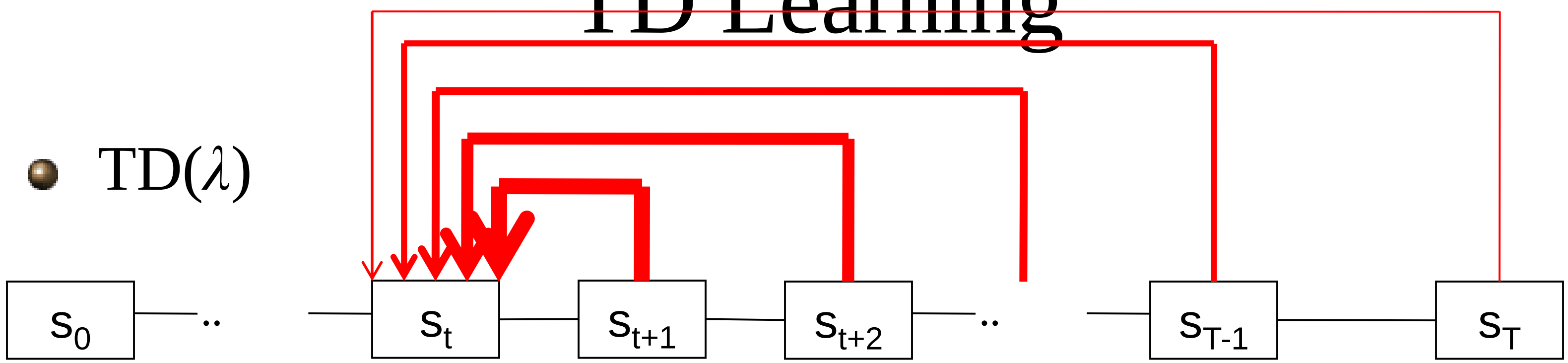
..

$\cancel{r_{t+1}} + \cancel{r_{t+2}} + \cdots + \cancel{r_{T-1}} + V(s_{T-1})$

$\cancel{r_{t+1}} + \cancel{r_{t+2}} + \cdots + \cancel{r_{T-1}} + \cancel{r_T}$ $V(s_T)$

TD Learning

- TD(λ)



$V(s_t)$ $\quad V(s_{t+1})$ $\quad V(s_{t+2})$ .. $\quad V(s_{T-1})$ $\quad V(s_T)$

$\quad\quad\quad *\quad\quad\quad *\quad\quad\quad *\quad\quad\quad *$

Proportion : $1-\lambda$ $\quad (1-\lambda)\lambda$ .. $\quad (1-\lambda)\lambda^{T-t-2}$ $\quad \lambda^{T-t-1}$

$$\underbrace{\quad\quad\quad\quad\quad\quad\quad\quad}_{\sum = 1}$$

Given Weights

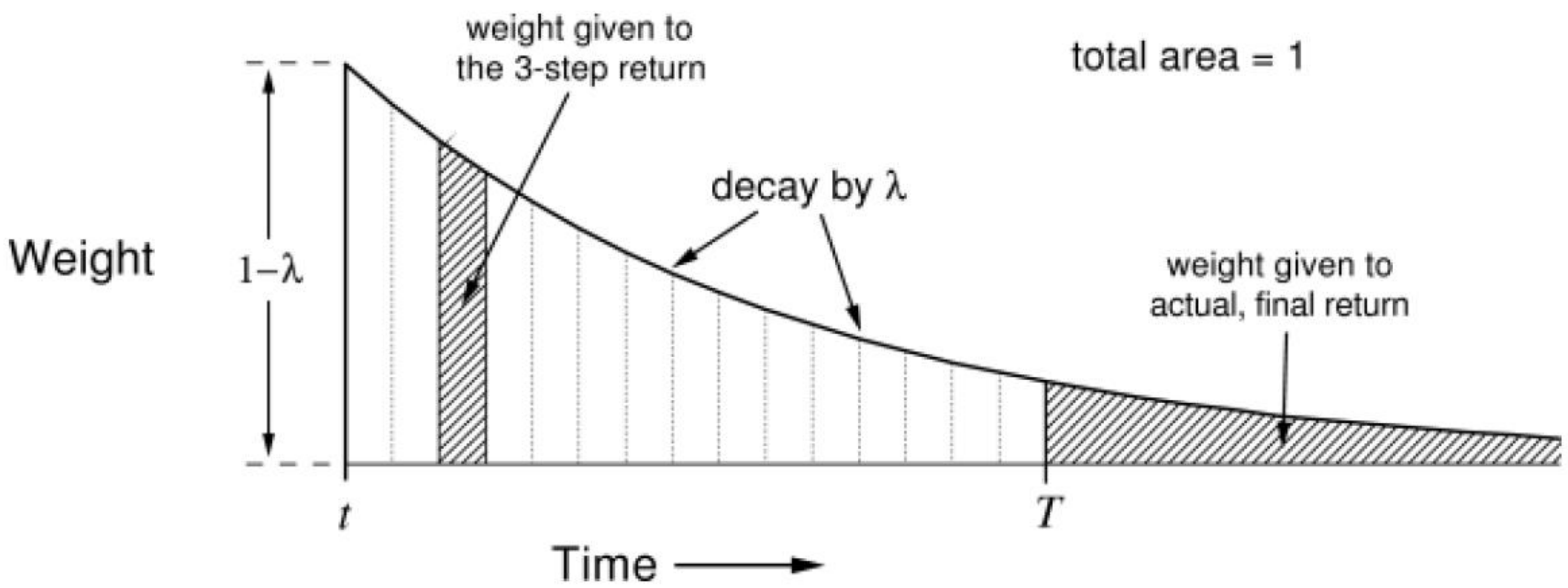
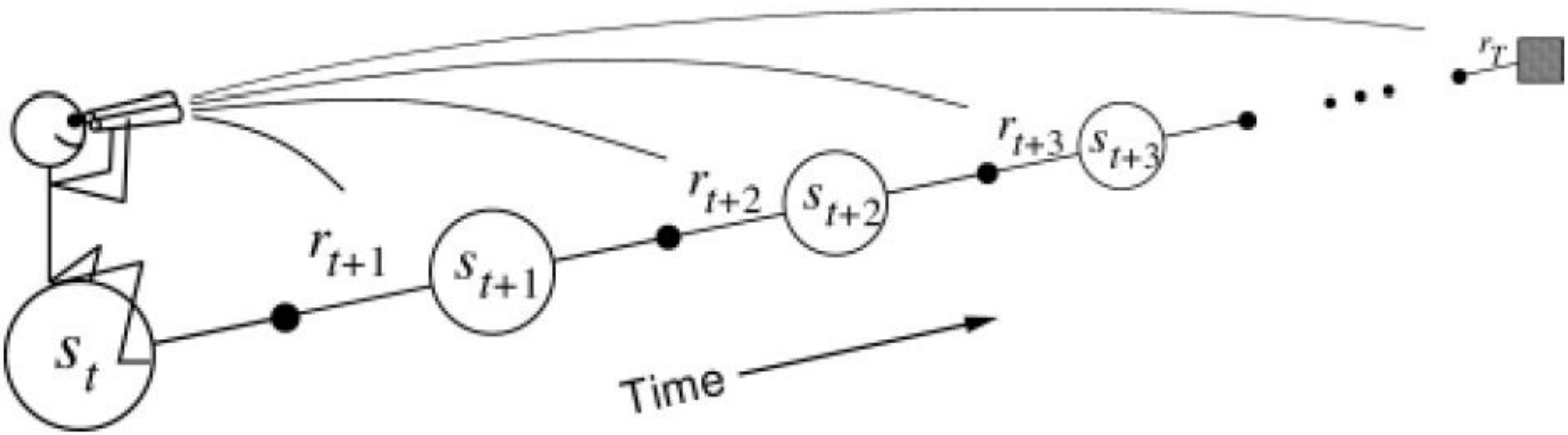


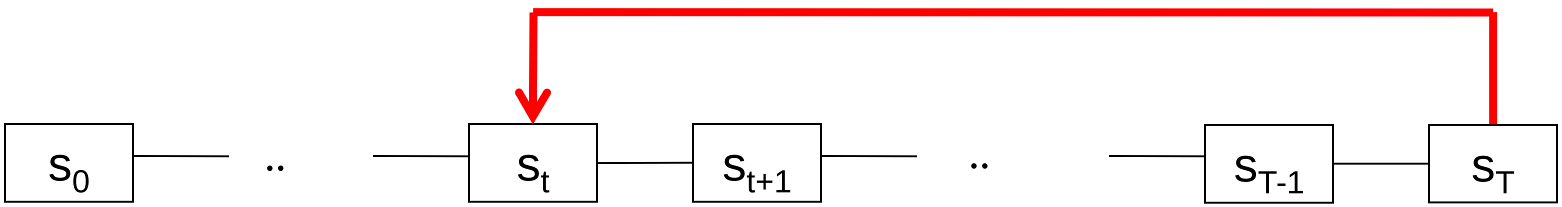
Figure 7.4: Weighting given in the λ-return to each of the n-step returns.

The Forward View



TD(1)

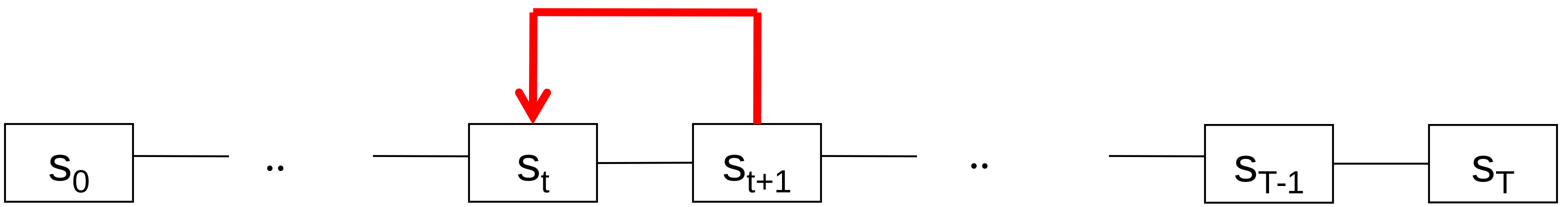
- Monte-Carlo tree search.



$$\Delta V(s_t) = \alpha[V(s_T) - V(s_t)]$$

$$V(s_t) = V(s_t) + \Delta V(s_t)$$

TD(0)

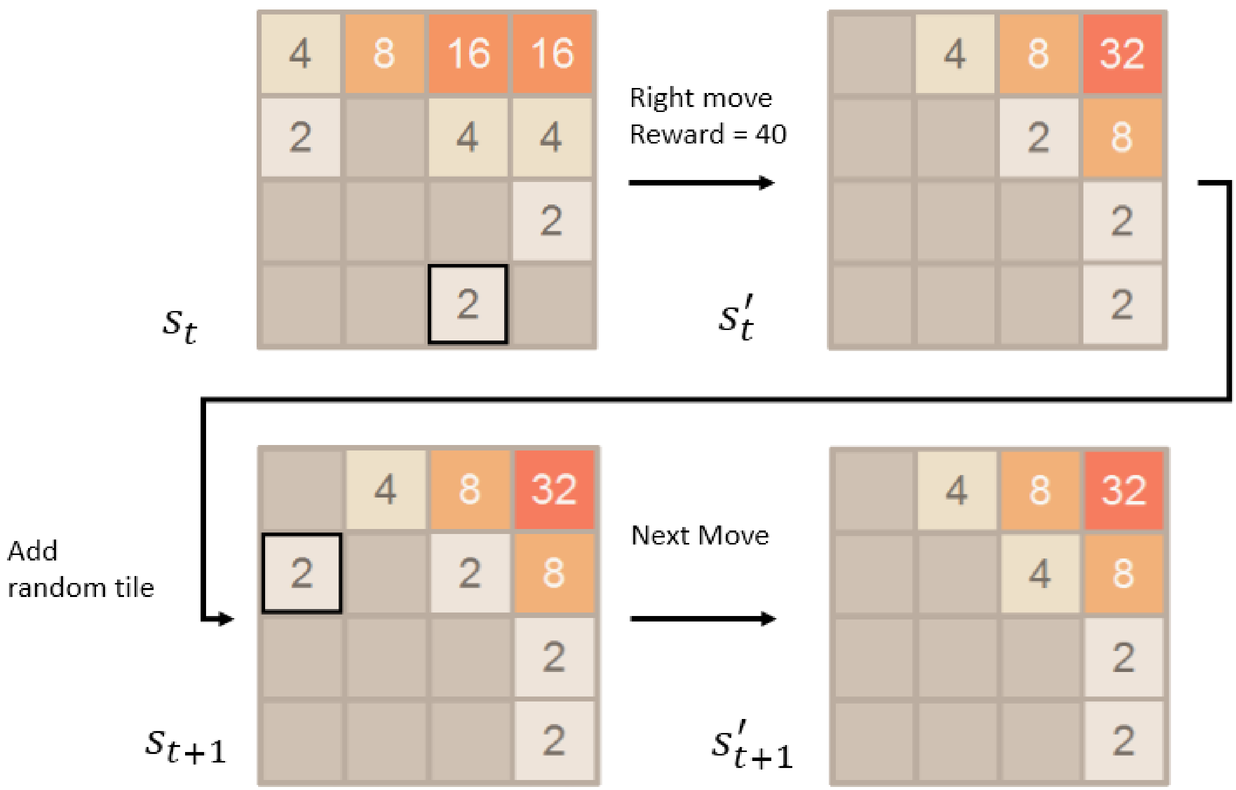


$$\Delta V(s_t) = \alpha[V(s_{t+1}) - V(s_t)]$$

$$V(s_t) = V(s_t) + \Delta V(s_t)$$

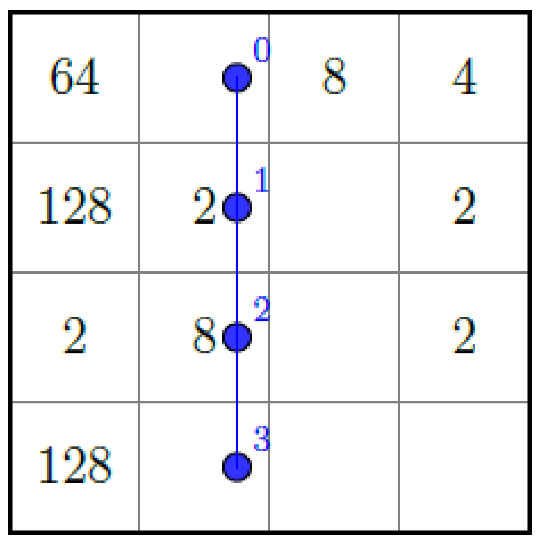
Case Study: 2048

- [Szubert and Jaskowaski 2014]



N-Tuple Network

- Example: 8 4-tuple networks as shown.
  - Each cell has 16 different tiles
  - 16^4 features for this network.
    - But only one is on, others are 0.
    - So, we can use table lookup to find the feature weight.



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

Evaluation on Feature Weights

- General evaluation function:
  - $\boldsymbol{V(s) = F(\varphi(s))}$
    - $\varphi(s)$: a vector of feature occurrences in s
- Linear evaluation function:
  - $\boldsymbol{V(s) = \varphi(s) \cdot \theta}$
    - θ: a vector of feature weights

Evaluate a Position

- The value of a position is evaluated based on
  - linear combination of features, i.e.,
  - $\boldsymbol{V(s) = \varphi(s) \cdot \theta}$
    - $\varphi(s)$: a vector of feature occurrences in s
    - θ: a vector of feature weights
- Features:
  - $\varphi(s)$: 8 x 16^4 features, [$\underline{0, 1, 0, \ldots}$, $\underline{0, 0, 1, \ldots}$, …, $\underline{1, 0, 0, \ldots}$]
    - All 0s, except for 8 ones.
      - One 1 every 16^4 features.
      - Let their indices be $\boldsymbol{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8}$.
- So, $\boldsymbol{V(s) = \varphi(s) \cdot \theta}$

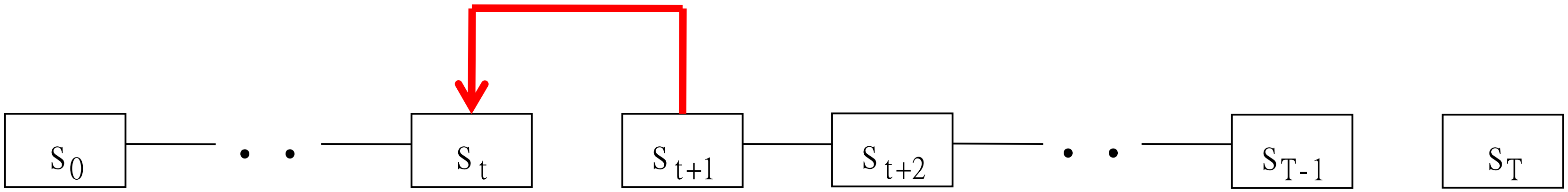
$$\boldsymbol{= \sum_i^{8\times 16^4} \varphi_i(s) \cdot \theta_i}$$

$$\boldsymbol{= \sum_i^{8} \varphi_{f_i}(s) \cdot \theta_{f_i}}$$

$$\boldsymbol{= \sum_i^{8} \theta_{f_i}}$$ (simply lookup table for f_i)

TD(0)

- To minimize the error.



- Error:

$$\delta_t = r_{t+1} + V(s_{t+1}) - V(s_t)$$

  - So, $\Delta V(s_t) = \alpha\delta_t = \alpha\big(r_{t+1} + V(s_{t+1}) - V(s_t)\big)$

- Adjustment

$$\Delta\theta = \Delta V(s_t)\frac{\varphi(s_t)}{\|\varphi(s_t)\|} = \alpha\delta_t\frac{\varphi(s_t)}{\|\varphi(s_t)\|}$$

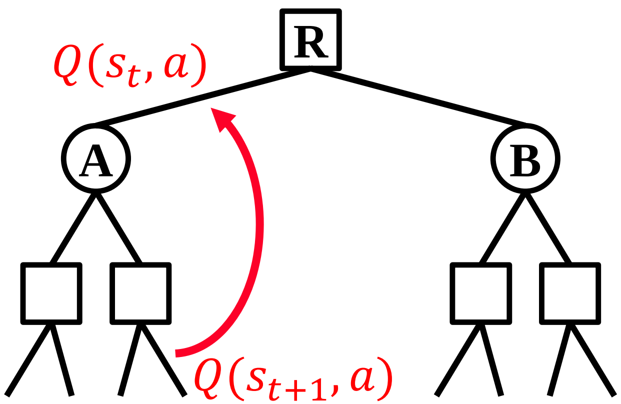
- Since $\|\varphi(s_t)\|$ is constant in 2048, no need for normalization.

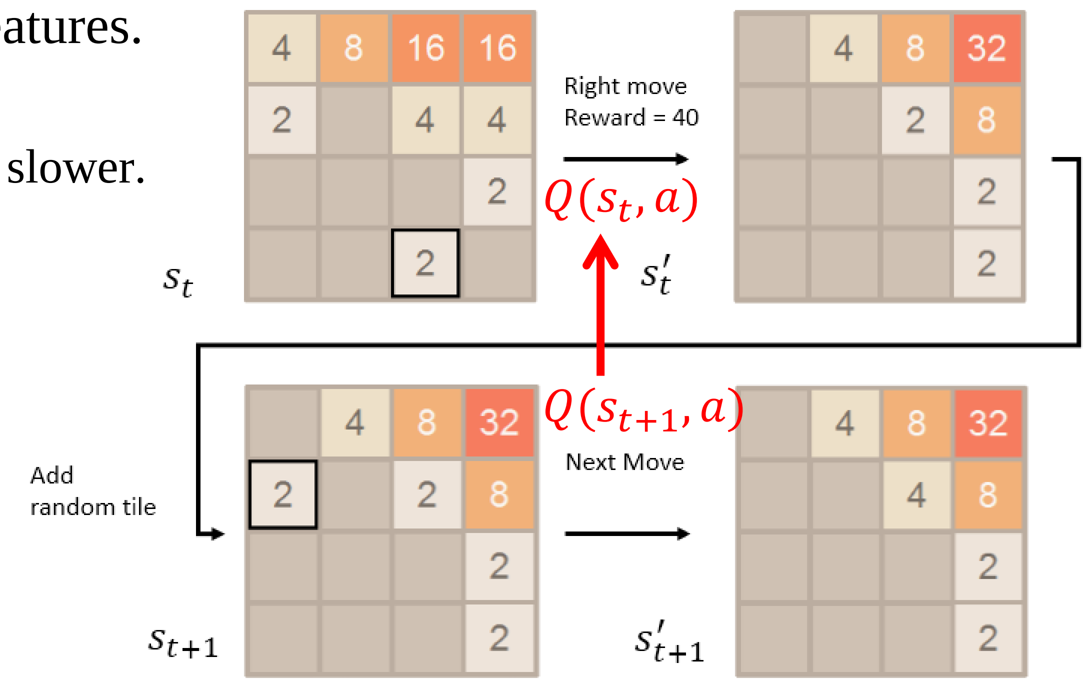
$$\Rightarrow \Delta\theta = \alpha'\delta_t\varphi(s_t),$$

  - where $\alpha' = \frac{\alpha}{\|\varphi(s_t)\|}$

Three Methods of Evaluating Values

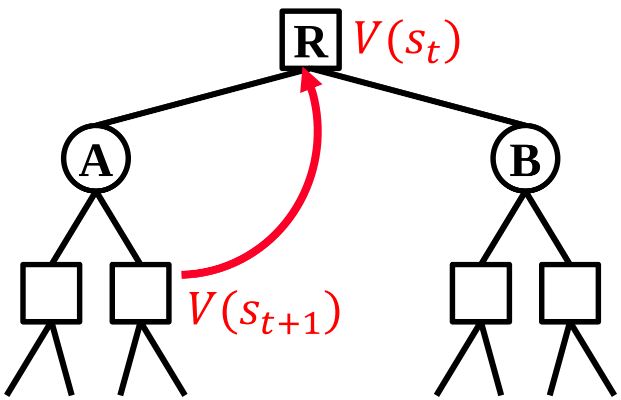
1. Evaluate actions: $Q(s,a)$.
   Select $arg_a \max Q(s,a)$
   - Also called Q-learning
   - Problem: Too many features.
     - 4 times more!
     - This makes learning slower.

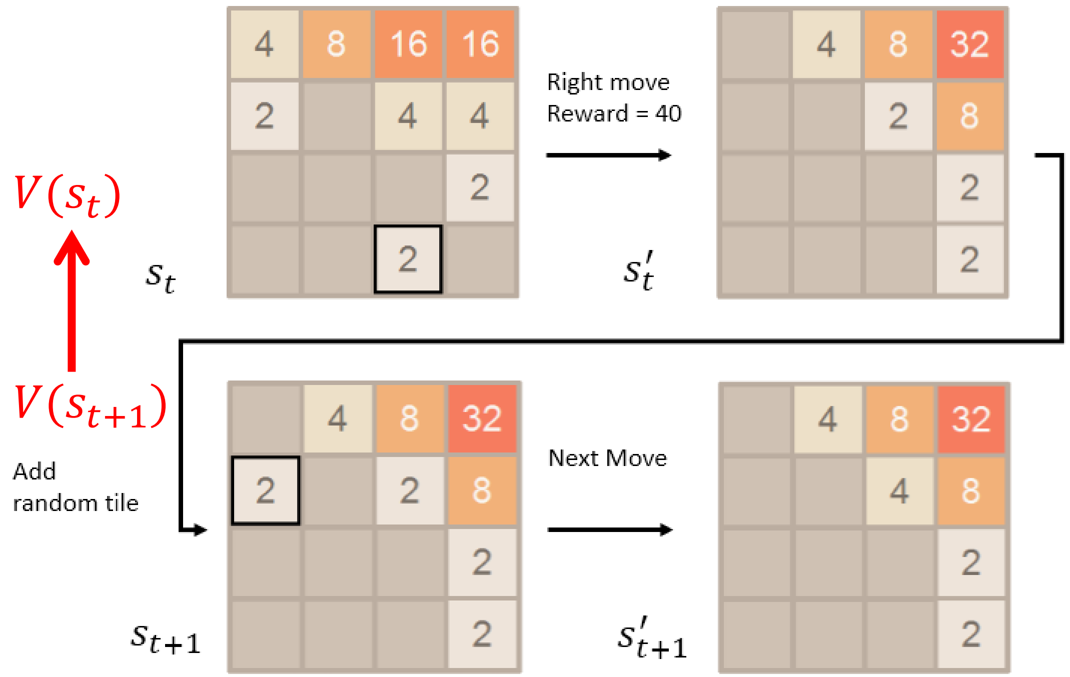




Three Methods of Evaluating Values

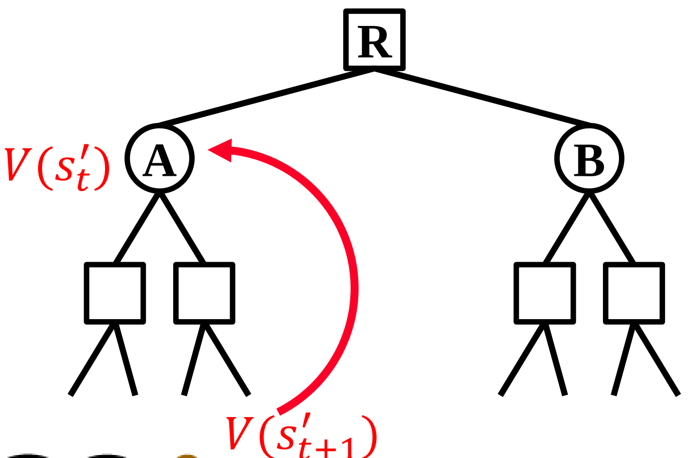
2. Evaluate states to play: $V(s_t)$.
   Select $arg_a \max\left(R(s_t, a) \sum_{s_{t+1}} P(s_t, a, s_{t+1}) V(s_{t+1})\right)$
   – Problem: Higher time complexity.

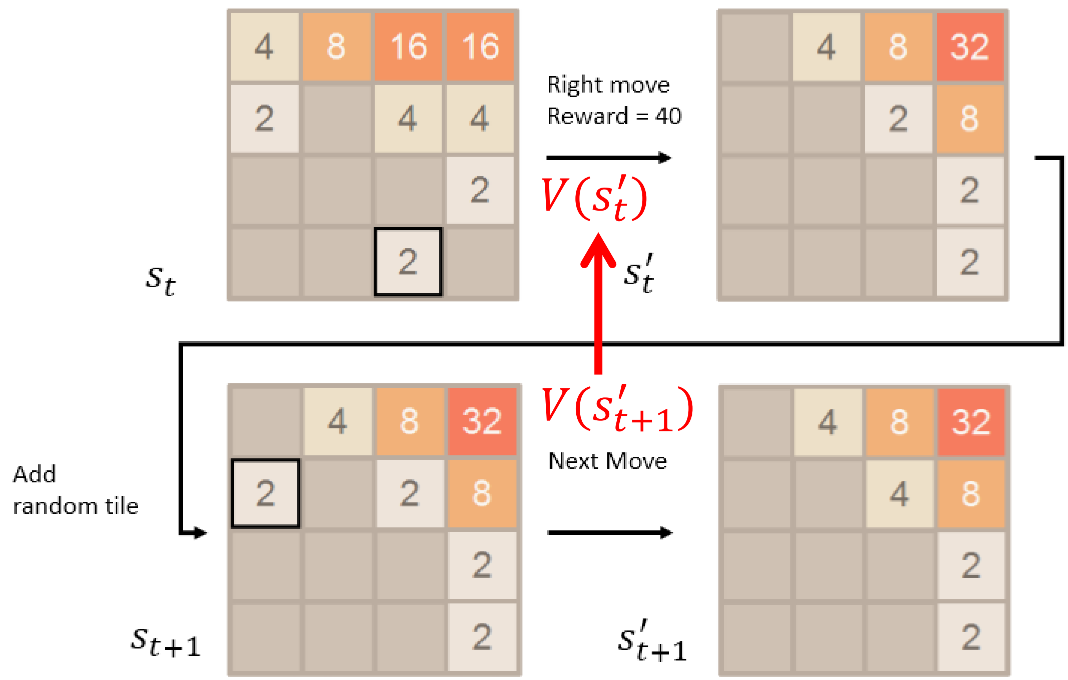




Three Methods of Evaluating Values

3. Evaluate states after an action. $V(s_t')$.
   Select $arg_a max[R(s_t, a) + V(s_t')]$
   - These states are also called **afterstates**.
   - **The best solution!!**





Afterstate Evaluation Function

function EVALUATE(s, a)

  $s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)

  return $r + V(s')$

function LEARN EVALUATION(s, a, r, s', s'')

  $a_{next} \leftarrow \arg\max_{a' \in A(s'')}$ 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 6: The *afterstate evaluation function* and a dedicated variant of the TD(0) algorithm.

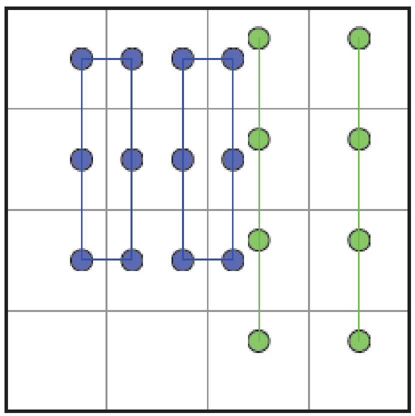
function PLAY GAME
  $score \leftarrow 0$
  $s \leftarrow$ INITIALIZE GAME STATE
  while $\neg$IS TERMINAL STATE(s) **do**
    $a \leftarrow \arg\max_{a' \in A(s)}$ EVALUATE(s, a')
    $r, s', s'' \leftarrow$ MAKE MOVE(s, a)
    if LEARNING ENABLED **then**
      LEARN EVALUATION(s, a, r, s', s'')
    $score \leftarrow score + r$
    $s \leftarrow s''$
  return $score$

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

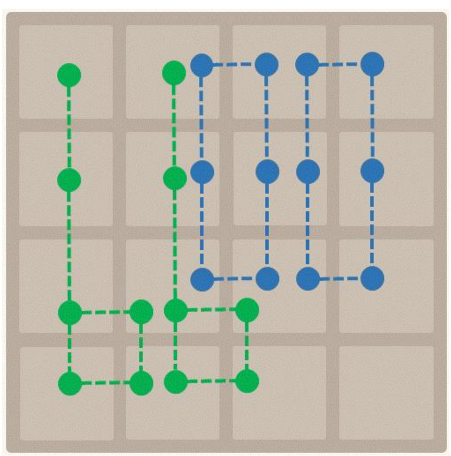
Figure 3: A pseudocode of a game engine with moves selected according to the evaluation function. If learning is enabled, the evaluation function is adjusted after each move.

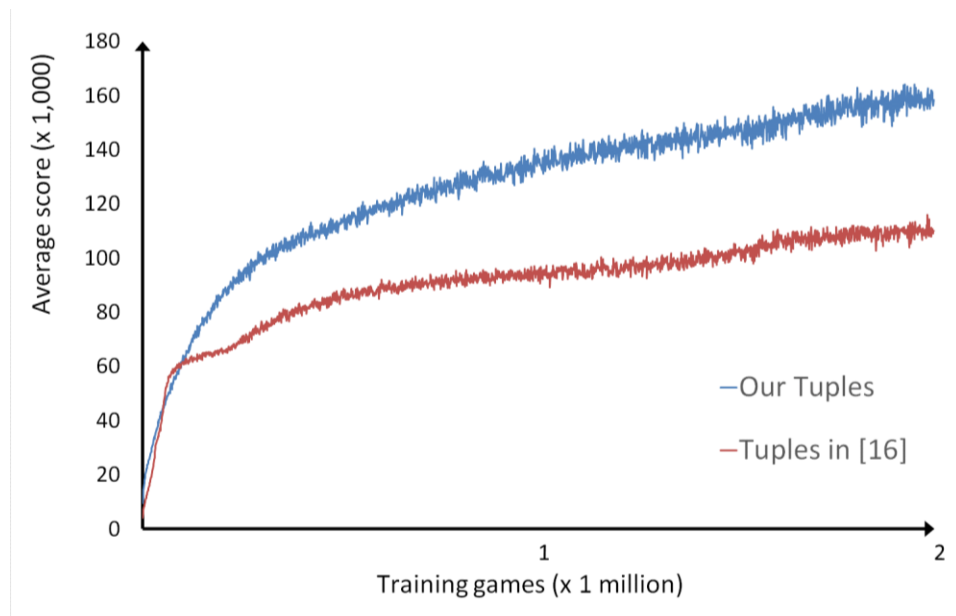
The N-Tuple Networks Used

- Use the following [Szubert and Jaskowaski 2014]



- Ours:



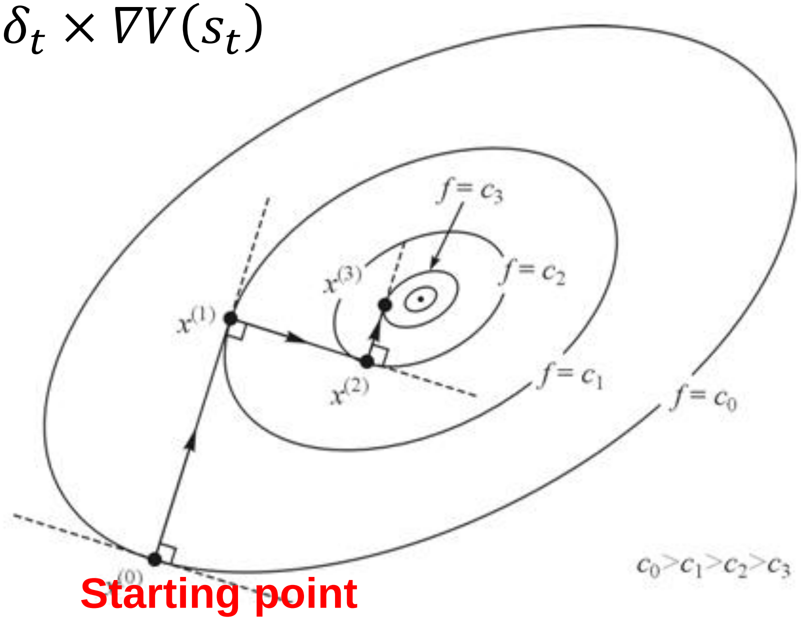


Issues

- Features
  - Multi-stages
  - Monotonicity
  - Number of distinct tiles
  - Number of empty squares
  - Big tiles
  - Rotation/Mirroring
  - Sizes
- Step-size parameter: α.
  - Our experience: 0.0025,
    - A better version: 0.00025 after 1,000,000 learning games.
- Learning backwards.
- Bitboard
- Expectimax search

What If $V(s)$ is non-linear?

- Use gradient: $\nabla V(s_t)$
  - The Normal (法向量)
    - (等高線圖的梯度最大者.)
- Adjustment:
  - $\Delta\theta = \Delta V(s_t) \times \nabla V(s_t) = \alpha\delta_t \times \nabla V(s_t)$
- Example:
  - For linear: $\nabla V(s_t) = \dfrac{\varphi(s_t)}{\|\varphi(s_t)\|}$

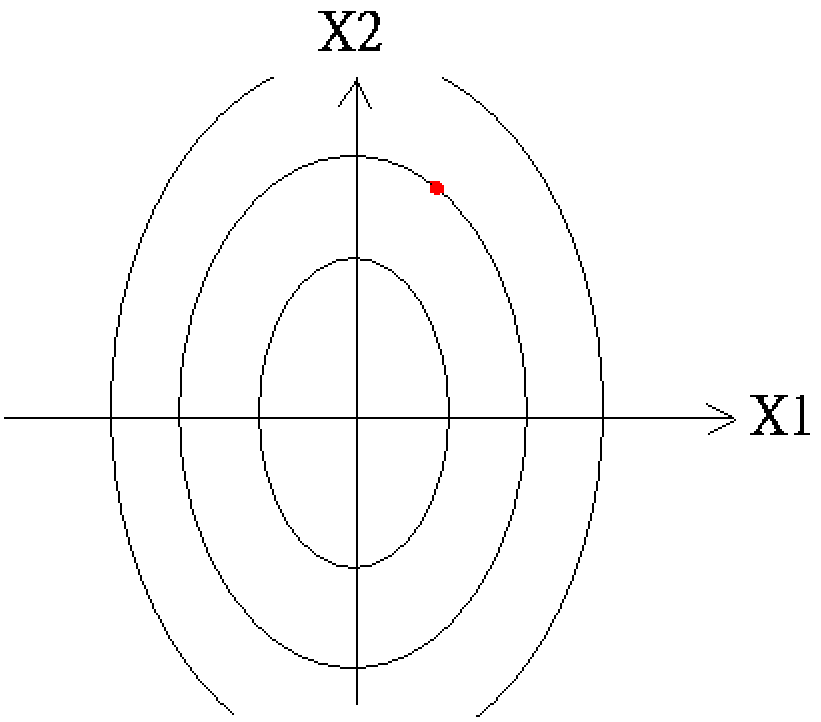


Example

- x = [x1,x2]

$$f(x) = -4X_1^{\ 2} - X_2^{\ 2}$$

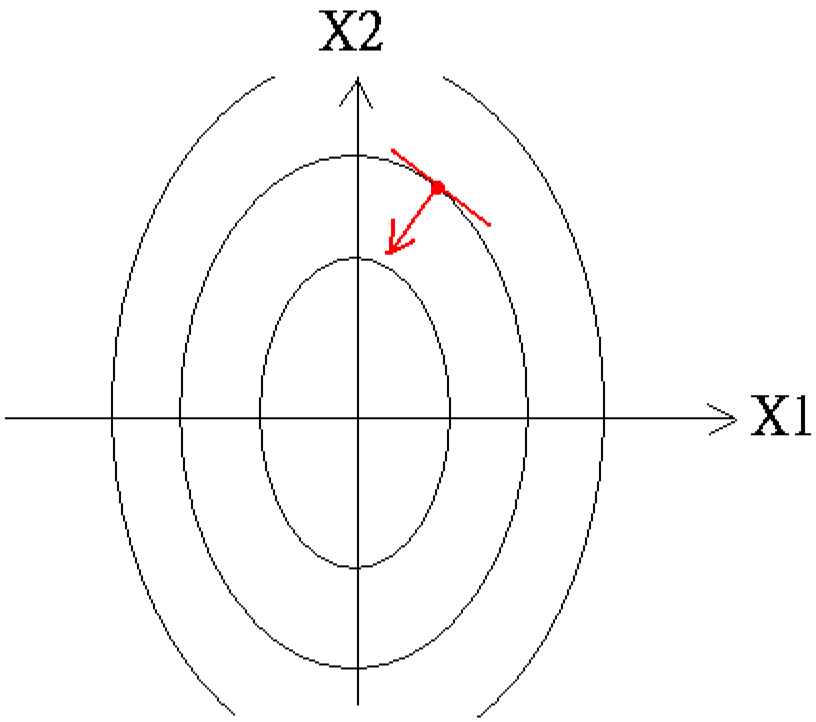
- Find the max.
- Starting at (1,3)



Example

$$f'(x) = [-8X_1, -2X_2]$$

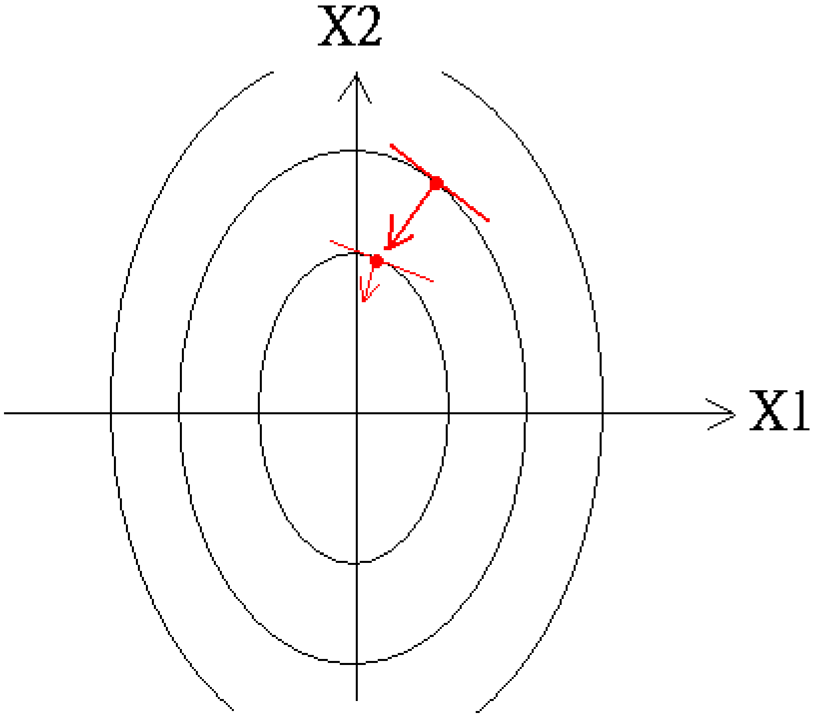
- From (1,3).
- Gradient: [-8 , -6]
- Adjustment:
  - $\alpha = 0.1$
  - $(1,3) + \alpha(-8,-6)$
    $= (0.2 , 2.4)$



Example

$$f'(x) = [-8X_1, -2X_2]$$

- From (0.2 , 2.4)
- Gradient: [-1.6 , -4.8]
- Adjustment:
  - $\alpha = 0.1$
  - $(0.2 , 2.4) + \alpha(-1.6,-4.8) = (0.04 , 1.92)$

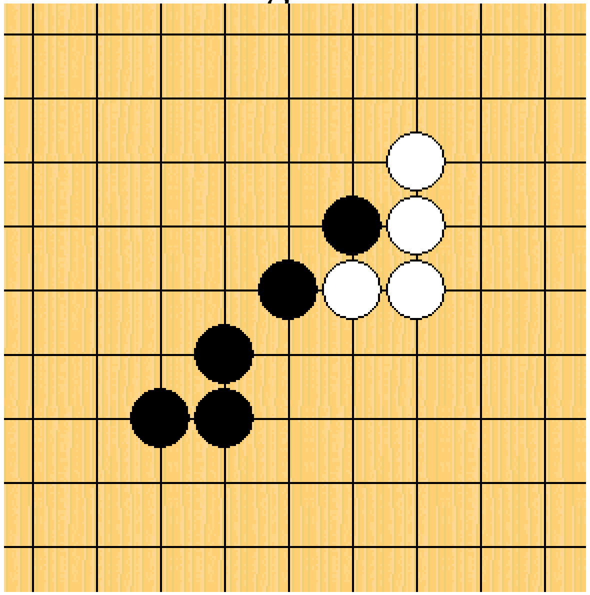


Case Study: Connect6

- Connect6 is a kind of six-in-a-row game.
  - Two players, named Black and White.
    - Place two black and white stones, respectively.
      - Black plays first and places one stone initially.
    - A player wins
      - If the player gets six or more consecutive stones horizontally, vertically or diagonally.

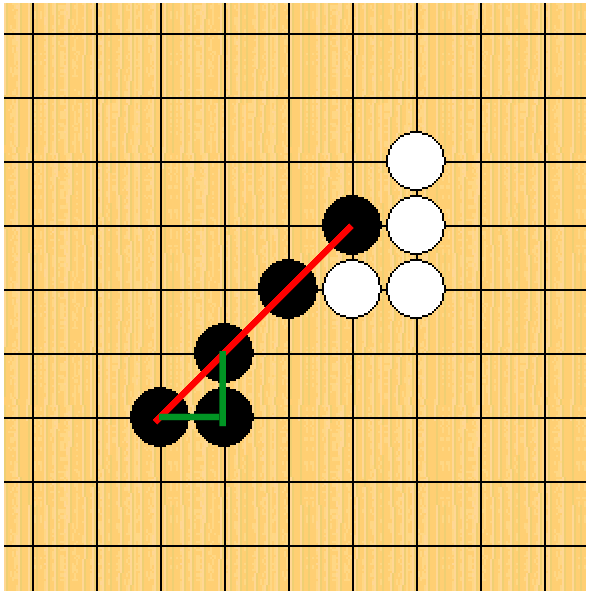
Evaluate a Position

- Basically, we evaluate the value of a position based on the features of Connect6.
  - Usually use **linear combination $V(s) = \varphi(s) \cdot \theta$,**
    - $\varphi(s)$: a vector of feature occurrences in s,
    - θ: a vector of feature weights



Evaluate a Position (An Illustration)

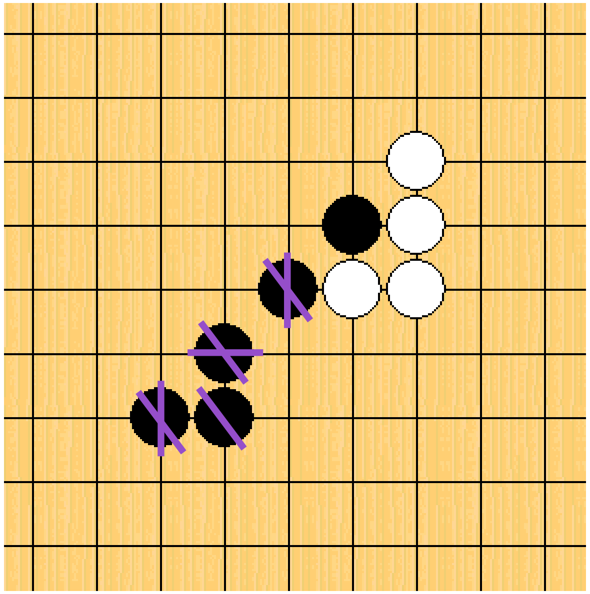
- For simplicity of discussion, use linear combination to evaluate the value of a position. E.g.,
  - Black: 1600 + 200*2 +



T1 : 1600
L3 : 800
D3 : 400
L2 : 200
D2 : 100
L1 : 50

Evaluate a Position (An Illustration)

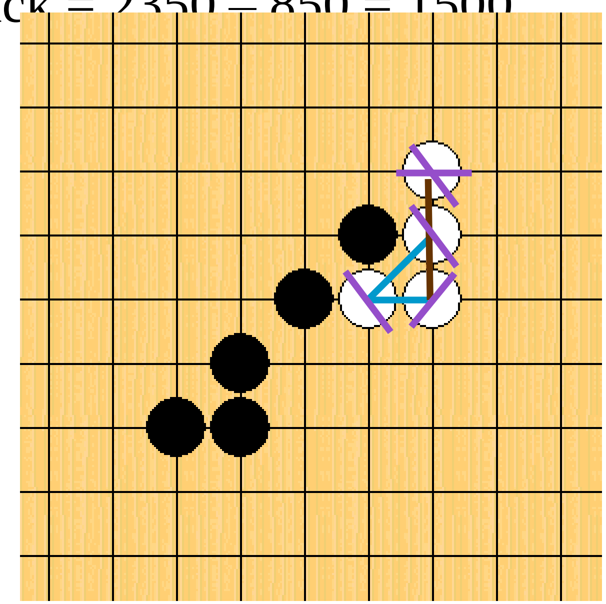
- For simplicity of discussion, use linear combination to evaluate the value of a position. E.g.,
  - Black: 1600 + 200*2 + 50*7 = 2350



T1 : 1600
L3 : 800
D3 : 400
L2 : 200
D2 : 100
L1 : 50

Evaluate a Position (An Illustration)

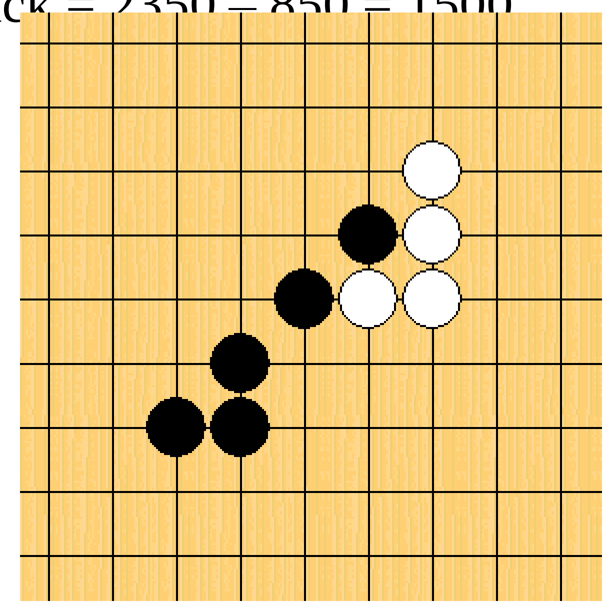
- For simplicity of discussion, use linear combination to evaluate the value of a position. E.g.,
  - Black: 1600 + 200*2 + 50*7 = 2350
  - White: 400 + 100*2 + 50*5 = 850
  - Value for Black = 2350 – 850 = 1500



T1 : 1600
L3 : 800
D3 : 400
L2 : 200
D2 : 100
L1 : 50

Goal

- For simplicity of discussion, use linear combination to evaluate the value of a position. E.g.,
  - Black: 1600 + 200*2 + 50*7 = 2350
  - White: 400 + 100*2 + 50*5 = 850
  - Value for Black = 2350 – 850 = 1500

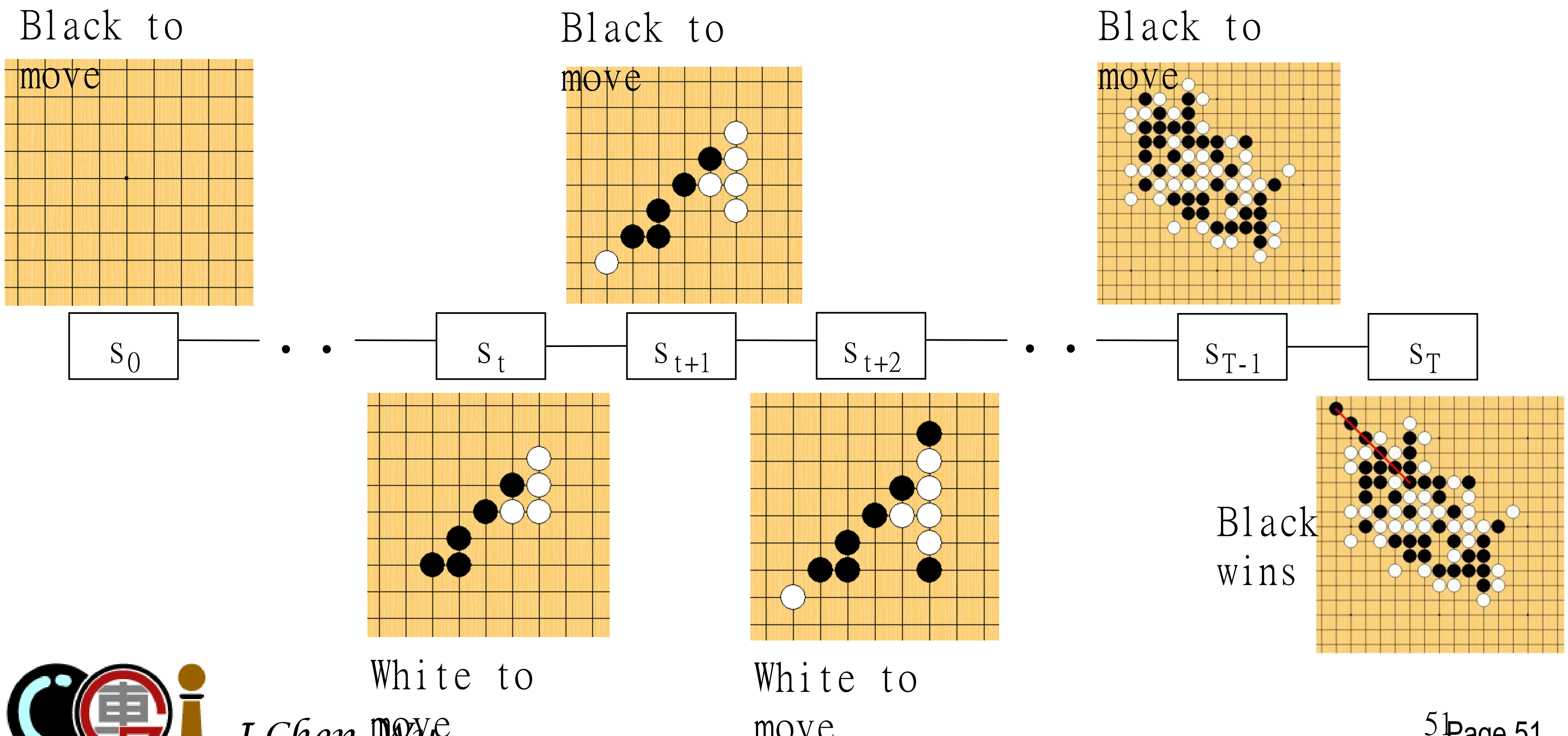


T1 : 1600
L3 : 800
D3 : 400
L2 : 200
D2 : 100
L1 : 50

We want to adjust these weights for accuracy.

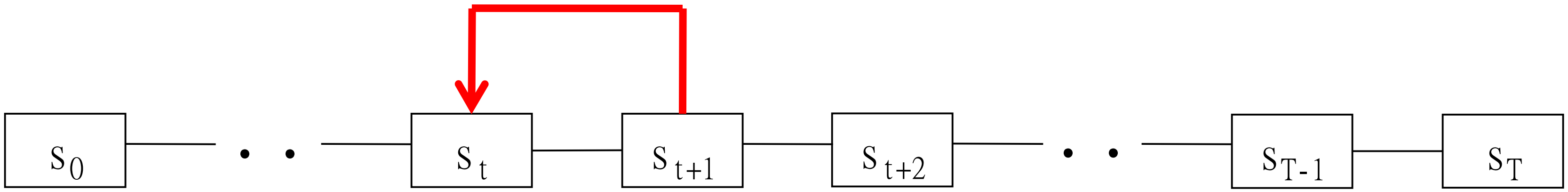
TD Learning

- A sequence of positions.



TD(0)

- To minimize the error.



- Error:

$$\delta_t = V(s_{t+1}) - V(s_t)$$
$$\Delta V(s_t) = \alpha\delta_t = \alpha\big(V(s_{t+1}) - V(s_t)\big)$$

  - Note: reward is at the final state s_T

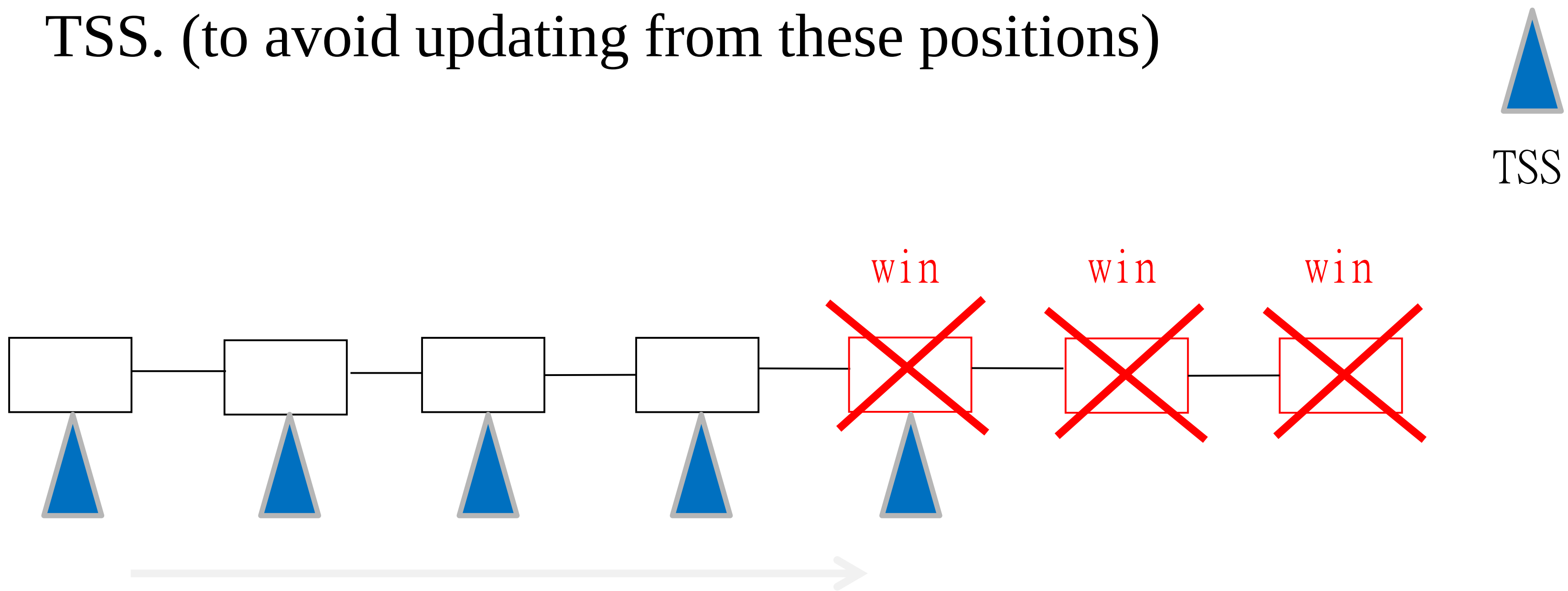
- Adjustment:

$$\Delta\theta = \Delta V(s_t)\frac{\varphi(s_t)}{\|\varphi(s_t)\|} = \alpha\delta_t\frac{\varphi(s_t)}{\|\varphi(s_t)\|}$$

  - $\|\varphi(s_t)\|$ is for normalization.

Threat Space Search (TSS)

- We use TSS to remove the winning positions found by TSS. (to avoid updating from these positions)



Trained Weights

- The weights of some features after training.

Feature Weights	With TSS	Without TSS
W_{T2}	0.52982 Not high & close	1.73220 Too high.
W_{T1}	0.51070	0.83796
W_{L3}	0.49358	0.73046
W_{D3}	0.27506	0.25531
W_{L2}	0.20028	0.07715

  – T2: double threats (or live 4)
  – T1: single threat (or dead 4)
- "without TSS" overweighs threats.

Discussion

- We successfully use TD(0) to improve the strength of NCTU6, a Connect6 champion program.
  - We got 58% win rate against the original NCTU6.
- We raise an important issue.
  - It is very important to remove the winning/losing positions found by TSS (or RZOP) in TD Learning.