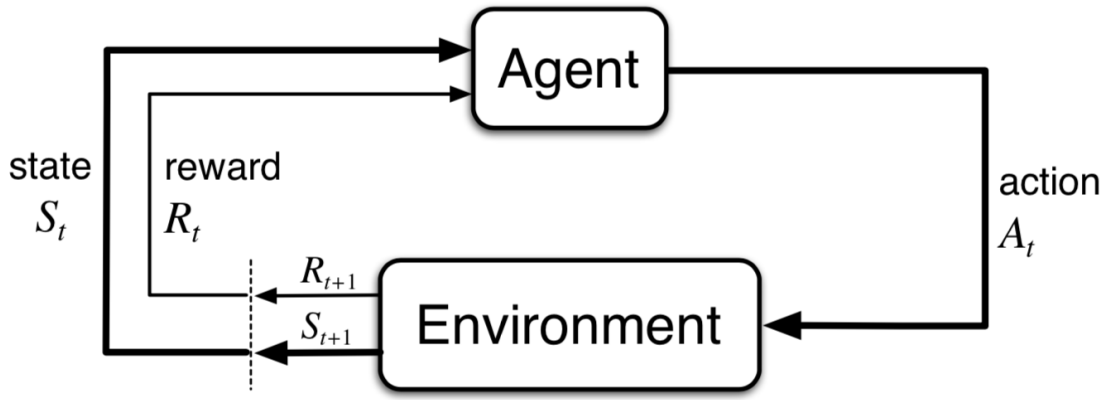
Outline

Reminder: states, actions and policy



$\mathcal{S}$, $\mathcal{A}$ = state/action spaces.

A (determinisitic) policy is a function
$\pi : \mathcal{S} \to \mathcal{A}$

.

Gain and value function

The gain is:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= R_{t+1} + \gamma G_{t+1}, \end{aligned}$$

where $\gamma \in (0, 1)$ is the discount factor.

Gain and value function

The gain is:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= R_{t+1} + \gamma G_{t+1}, \end{aligned}$$

where $\gamma \in (0, 1)$ is the discount factor.

The value function V and action-value function Q are:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}\left[G_{t+1} \mid S_t = s, \pi\right] \\ Q_\pi(s, a) &= \mathbb{E}\left[G_{t+1} \mid S_t = s, A_t = a, \pi\right] \end{aligned}$$

Two problems

- Policy evaluation

For a given policy π, find
$V^\pi(x)$ and $Q^\pi(x,a)$.

Two problems

- Policy evaluation

For a given policy π, find
$V^\pi(x)$ and $Q^\pi(x,a)$.

- Control problem / optimization

Find / use π^* such that
$V^{\pi^*} = \max_\pi V^\pi(x)$.

Bellman's equation

$$V^*(s) =$$

$$Q^*(s,a) =$$

Bellman's equation

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a)$$

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} V^*(s') p(s' \mid s,a)$$

Two problems:

- Requires the knowledge of systems dynamics and rewards.
- $|\mathcal{S}|$ can be large

Bellman's equation

$$r(s,a) = \sum_{s'} s' \mathbb{P}(s' \mid s, a)$$

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a)$$

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} V^*(s')) p(s' \mid s, a)$$

~~Two problems:~~

- ~~Requires the knowledge of systems dynamics and rewards.~~
  - We assume to have access to a simulator.
- ~~$|\mathcal{S}|$ can be large~~
  - We assume $|\mathcal{S}|$ to be small for now.

Table of contents

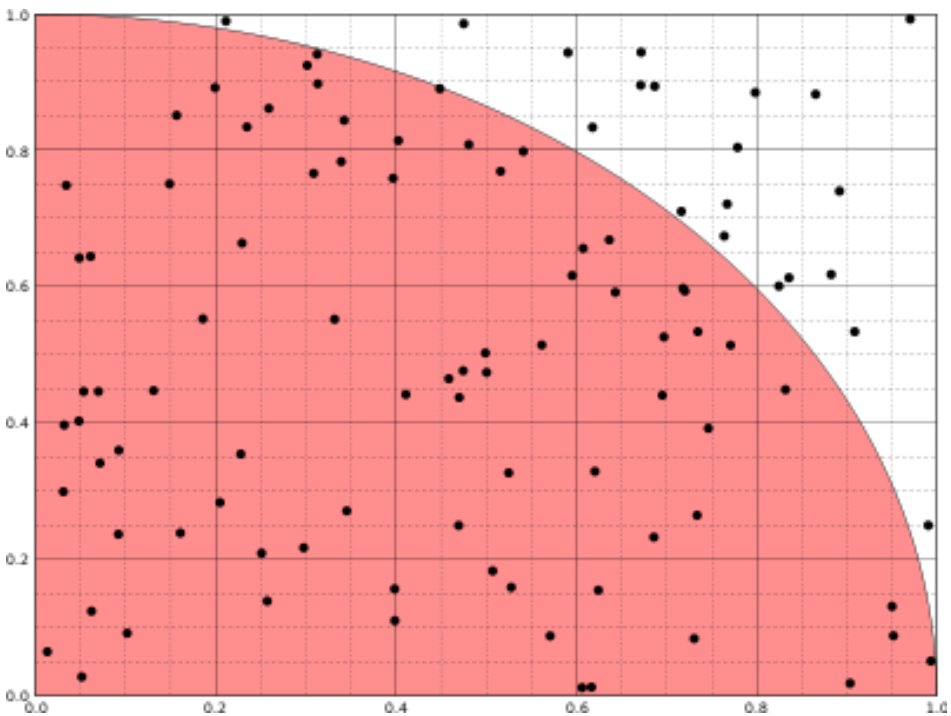
Monte Carlo methods

Class of algorithms where we replace a deterministic computation by an estimation of $\mathbb{E}[X]$. We then sample many values of X and compute the average (law of large numbers: $\frac{1}{n}\sum_{i=1}^{n} X_i \approx \mathbb{E}[X]$).

Monte Carlo methods

Class of algorithms where we replace a deterministic computation by an estimation of $\mathbb{E}[X]$. We then sample many values of X and compute the average (law of large numbers: $\frac{1}{n}\sum_{i=1}^{n} X_i \approx \mathbb{E}[X]$).

Example:



Source: wikipedia

- Area is $\pi/4$. A point (x, y) is in the red zone if $x^2 + y^2 \leq 1$.

Monte Carlo for policy Evaluation

$$V^{\pi}(S_t) = \mathbb{E}\left[G_t \mid S_t = s, \pi\right].$$

Monte-Carlo = sample G_t by using rollout.

Monte Carlo for policy Evaluation

$$V^\pi(S_t) = \mathbb{E}\left[G_t \mid S_t = s, \pi\right].$$

Monte-Carlo = sample G_t by using rollout.

Recipe:

- Play many episodes with π
- Record the return from the first visit to each state
- Return the average as an approximation of $V^\pi(s)$.

Note: every-visit also works but the samples are not independent.

Monte Carlo learning algorithm

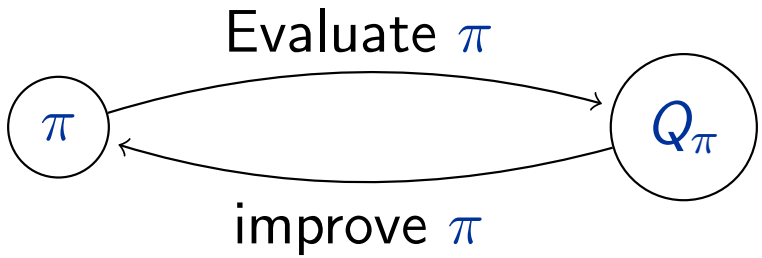
First-visit Monte-Carlo

For all s: R(s):={}
while True **do**
  Simulate an episode from 0 to T using π
  Set $G_T := 0$
  for $t = T$ to 0 (backward) **do**
    $G_t = R_{t+1} + \gamma G_{t+1}$.
    If S_t does not appear in $S_0 \dots S_{t-1}$, $R(S_t).append(G_t)$.
  end for
end while
$V(s) = mean(R(s))$.

If a state has been seen n times, the error is $O(1/\sqrt{n})$.

Monte-Carlo optimization

Monte-Carlo can be used to evaluate the state-action function $Q(s, a)$.

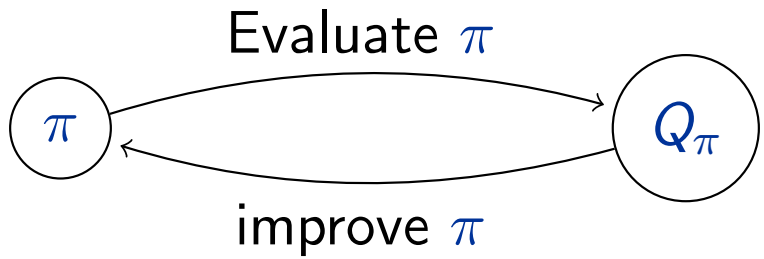


Recall: improve can be done by using greedy:

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a).$$

Monte-Carlo optimization

Monte-Carlo can be used to evaluate the state-action function $Q(s, a)$.



Recall: improve can be done by using greedy:

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a).$$

Possible problems:

- One may need many samples for all actions.
- Some action-pair might not be visited.

Solutions: exploration/exploitation tradeoff (course 4), importance sampling.

Table of contents

The temporal difference (TD) error

Bellman's equation states:

$$\begin{aligned} V(S_t) &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \dots\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma V(S_{t+1})\right]. \end{aligned}$$

$$V_\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} V_\pi(s') P(s' | s, \pi(s))$$

The temporal difference (TD) error

Bellman's equation states:

$$\begin{aligned} V(S_t) &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \dots\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma V(S_{t+1})\right]. \end{aligned}$$

α_t "small"

This is equivalent to

$$0 = \mathbb{E}\left[\underbrace{R_{t+1} + \gamma V(S_{t+1}) - V(S_t)}_{\text{TD error}}\right]$$

The TD learning algorithm uses the updates:

$$V(S_t) := V(S_t) + \alpha_t(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))),$$

where α is a learning rate.

Goal: Estimate $\mathbb{E}[X]$

$$S_n = \frac{1}{n} \sum_{i=1}^{n} X_i \qquad \lim_{n \to \infty} S_n = \mathbb{E}[X] \quad \text{(a.s.)}$$

$$S_{n+1} = S_n + \frac{1}{n+1}\left(X_{n+1} - S_n\right)$$

$$\sum_{i=1}^{n+1} X_i = (n+1) S_{n+1} = n S_n + X_{n+1} = (n+1) S_n + X_{n+1} - S_n$$

TD learning algorithm

TD(0) for evaluating V^π

1: Initialize $V(s)$ arbitrarily.
2: **while** True **do**
3: Initialize S
4: **for** While S' is not a terminal state **do**
5: Sample $A \sim \pi(S)$ and simulate a transition $S', R \sim p(\cdot \mid S, A)$.
6: $V(S) := V(S) + \alpha_t(R + \gamma V(S') - V(S))$.
7: $S := S'$
8: **end for**
9: **end while**

TD-learning: proof of convergence

TD-update:

$$V(S_t) := V(S_t) + \alpha_t(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))).$$

Theorem

Fix a policy π that visits all states and let $\gamma < 1$.
Assume that we use the TD-update with α_t be decreasing and such that:

- $\sum_t \alpha_t = +\infty$ *and* $\sum_t \alpha_t^2 < +\infty$.

Then the TD-learning converges to V^π almost surely.

Proof

Let $\beta_t(s)$ be such that

$$\beta_t(s) = \begin{cases} 0 & \text{if } s = S_t \\ \alpha_t & \text{otherwise} \end{cases}$$

Let V_t be the V-table at time t. The definition of β_t implies that for all s:

$$V_{t+1}(s) := V_t(s) + \beta_t(s) \left(\underbrace{R_{t+1} + \gamma V_t(S_{t+1})}_{= T^\pi V_t + \text{noise}} - V_t(s) \right).$$

with $\sum_t \beta_t(s) = \infty$ and $\sum_t \beta_t^2(s) < \infty$.

Proof

Let $\beta_t(s)$ be such that

$$\beta_t(s) = \begin{cases} 0 & \text{if } s = S_t \\ \alpha_t & \text{otherwise} \end{cases}$$

Let V_t be the V-table at time t. The definition of β_t implies that for all s:

$$V_{t+1}(s) := V_t(s) + \beta_t(s) \left(\underbrace{R_{t+1} + \gamma V_t(S_{t+1})}_{= T^\pi V_t + \text{noise}} - V_t(s) \right).$$

with $\sum_t \beta_t(s) = \infty$ and $\sum_t \beta_t^2(s) < \infty$.

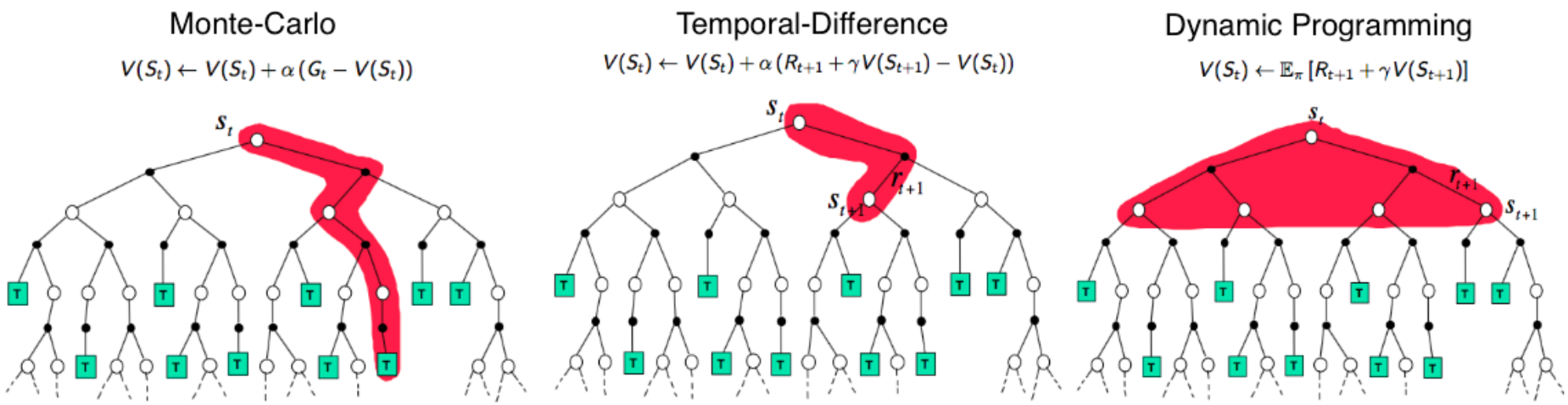
As T^π is contracting, Theorem 1 of On the convergence of stochastic iterative dynamic programming algorithms., Jaakkola, Jordan, Singh, NeurIPS 93 shows that this implies $\lim_{t\to\infty} V_t = V^\pi$ almost surely.

Relation between MC, TD and DP

$$V(S_t) = \mathbb{E}[G_t] \qquad MC$$

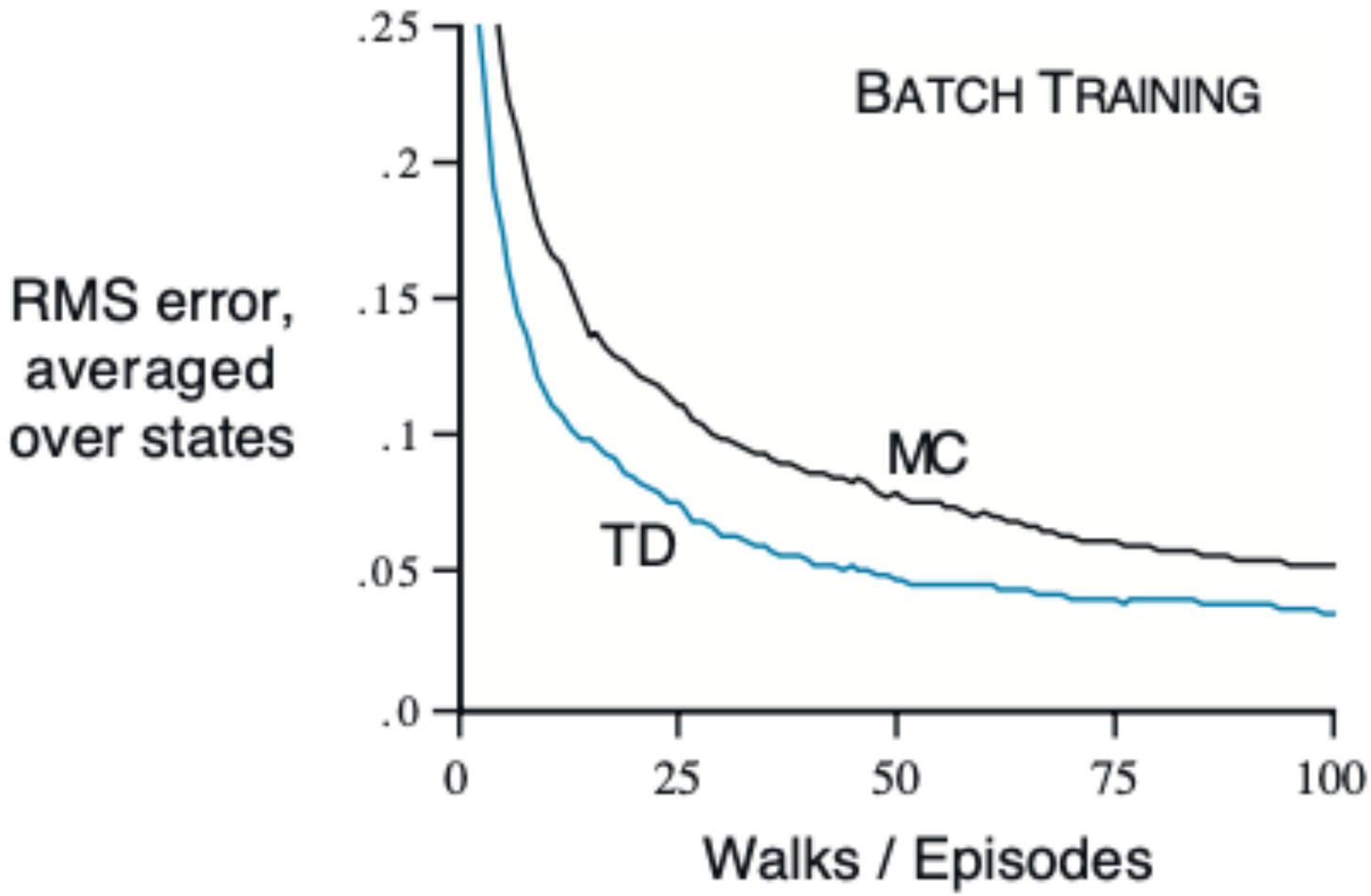
$$V(S_t) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})] \qquad TD$$

$$V(S_t) = \mathbb{E}[R_{t+1}] + \gamma \sum_{s'} V(S_{t+1}) \mathbb{P}(S_{t+1} = s') \qquad DP$$



- MC simulates a full trajectory
- TD samples one-step and uses a previous estimation of V.
- DP needs all possible values of $V(s')$.

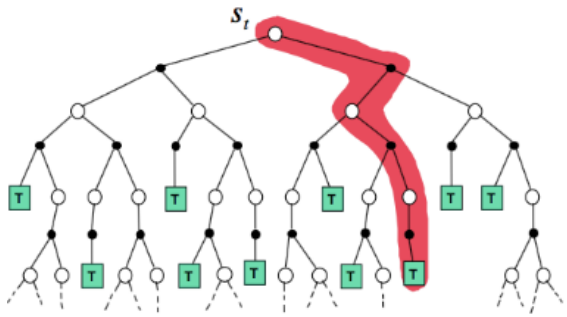
TD vs MC comparison: general case



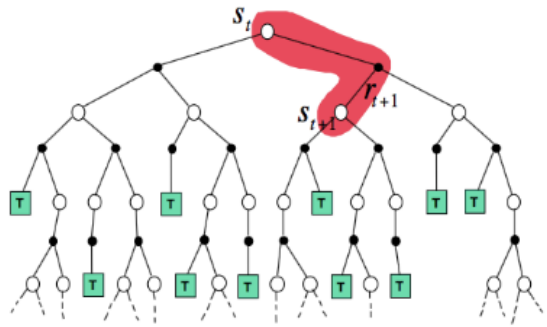
source: Sutton, Barto 2018. For a random-walk example.

Warning: this might very well depend on the choice of learning parameter α_t!

TD v.s. MC and tradeoffs

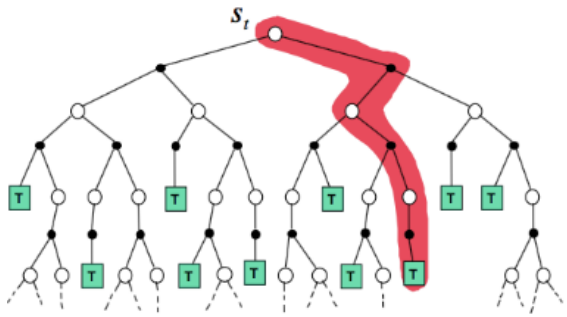


One full trajectory for update

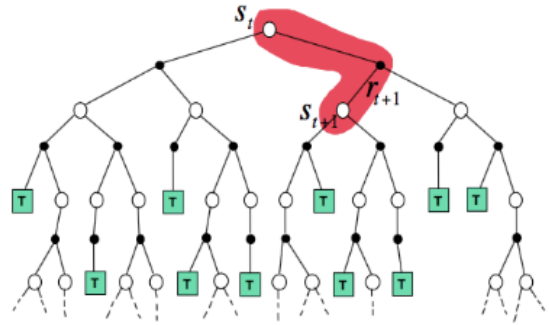


Updates take time to propagate

TD v.s. MC and tradeoffs



One full trajectory for update



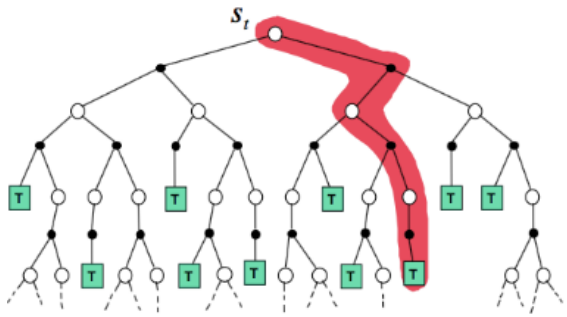
Updates take time to propagate

Tradeoff:

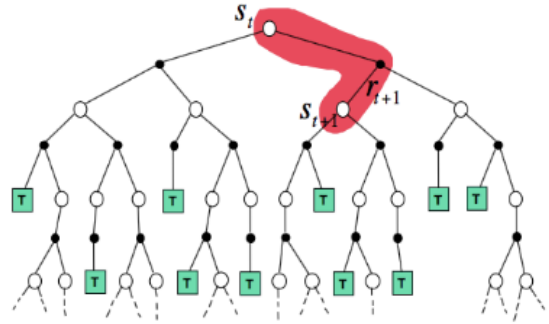
- Use n-step returns (see Sutton-Barto, chapter 7).

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{t+n} V(S_{t+n}).$$

TD v.s. MC and tradeoffs



One full trajectory for update



Updates take time to propagate

Tradeoff:

- Use n-step returns (see Sutton-Barto, chapter 7).

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{t+n} V(S_{t+n}).$$

- $TD(\lambda)$ (see Sutton-Barto, chapter 12 or Szepesvári, Section 2.1.3).

$$G_t(\lambda) = (1-\lambda) \sum_{n=1}^{T} \lambda^{n-1} G_{t:t+n} + \lambda^T G_t.$$

Table of contents

TD learning = policy evaluation. What about optimization?

Bellman's equations are:

$$V^\pi(S_t) = \mathbb{E}^\pi \left[R_{t+1} + \gamma V^\pi(S_{t+1}) \right]$$ to evaluate π

$$R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

TD learning = policy evaluation. What about optimization?

Bellman's equations are:

$$V^\pi(S_t) = \mathbb{E}^\pi \left[R_{t+1} + \gamma V^\pi(S_{t+1}) \right] \qquad \text{to evaluate } \pi$$

$$Q^*(S_t, A_t) = \mathbb{E}\left[R_{t+1} + \gamma \underbrace{\max_a Q^*(S_{t+1}, a)}_{V^*(S_{t+1})} \right] \qquad \text{to find the best policy}$$

TD difference for Q:

$$R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$$

$S_t \to A_t \to R_{t+1}, S_{t+1} \to A_{t+1} \to \cdots$

TD learning = policy evaluation. What about optimization?

Bellman's equations are:

$$V^\pi(S_t) = \mathbb{E}^\pi \left[R_{t+1} + \gamma V^\pi(S_{t+1}) \right] \qquad \text{to evaluate } \pi$$

$$Q^*(S_t, A_t) = \mathbb{E}\left[R_{t+1} + \gamma \max_a Q^*(S_{t+1}, a) \right] \qquad \text{to find the best policy}$$

This leads to two variant of:

- Q-learning = off-policy learning.
  - Choose $A_t \sim \pi$.
  - Apply TD-learning replacing $V(s)$ by $\max_a Q(s,a)$.
- SARSA = on-policy learning:
  - Choose $A_{t+1} \sim \arg\max_{a \in \mathcal{A}} Q(S_{t+1}, a)$.
  - Apply TD-learning replacing $V(s)$ by $Q(s, A_{t+1})$.

Q-learning and convergence guarantee

$$A_t \sim \pi$$

$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right).$$

Q-learning and convergence guarantee

$$A_t \sim \pi$$

$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right).$$

Theorem

Assume that $\gamma < 1$ and that:

- *Any station-action pair (a, s) is visited infinitely often.*
- $\sum_t \alpha_t = \infty$ *and* $\sum_t \alpha_t^2 < \infty$.

Then: Q converges almost surely to the optimal Q^*-table as t goes to infinity.

Proof: Identical to the proof of TD-learning.

ex: $\alpha_t = \frac{1}{t}$

$$\sum_{i=1}^{t} \alpha_i \simeq \log t$$

$$\sum_{i=1}^{t} (\alpha_i)^2 = \sum_i \frac{1}{i^2} < \frac{\pi^2}{6}$$

$$\alpha_t = t^{-\beta}, \quad \beta \in \left(\frac{1}{2}; 1\right)$$

$$\alpha_t = \frac{1000}{t}$$

Q-Learning and SARSA

Q-learning, (one of the most popular RL algorithm):

$$A_t \sim \pi$$

$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right).$$

Q-Learning and SARSA

Q-learning, (one of the most popular RL algorithm):

$$A_t \sim \pi$$
$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right).$$

SARSA (name comes from $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$)

$$A_{t+1} \sim \arg\max Q(S_t, A_t) \text{ (or } \varepsilon\text{-greedy)}$$
$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right).$$

Q-learning pseudo-code

The Q learning algorithm

1: Initialize $Q(s, a)$ arbitrarily.
2: **while** True **do**
3: Initialize S
4: **while** S' is not a terminal state **do**
5: $\pi =$ policy derived from Q (e.g. ε-greedy).
6: Sample $A \sim \pi(S)$ and simulate a transition $S', R \sim p(\cdot \mid S, A)$.
7: $Q(S, A) := Q(S, A) + \alpha_t(R + \gamma \max_a Q(S', a) - Q(S, A))$.
8: $S := S'$
9: **end while**
10: **end while**

(in orange, the difference with TD-learning).

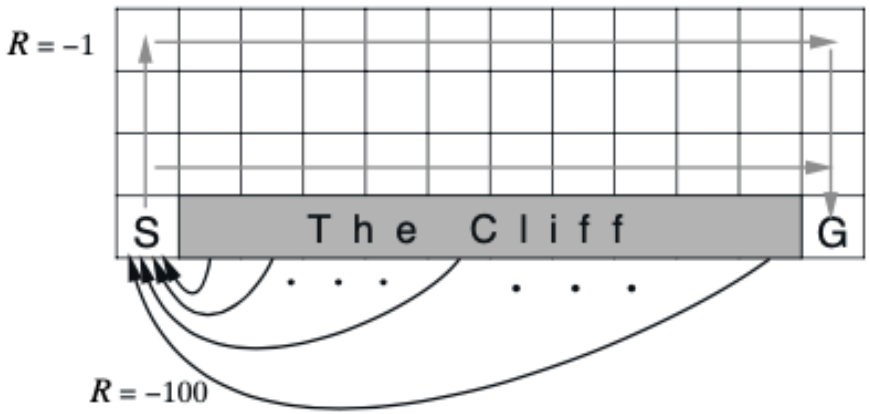
SARSA

SARSA algorithm

Initialize $Q(s, a)$ arbitrarily.
while True **do**
  Initialize S and A
  while S' is not a terminal state **do**
    π = policy derived from Q (e.g. ε-greedy).
    Simulate $S', R \sim p(\cdot \mid S, A)$ and $A' := \pi(S')$.
    $Q(S, A) := Q(S, A) + \alpha_t(R + \gamma Q(S', A') - Q(S, a))$.
    $S := S', A := A'$
  end while
end while

(in orange, the difference with Q-learning).

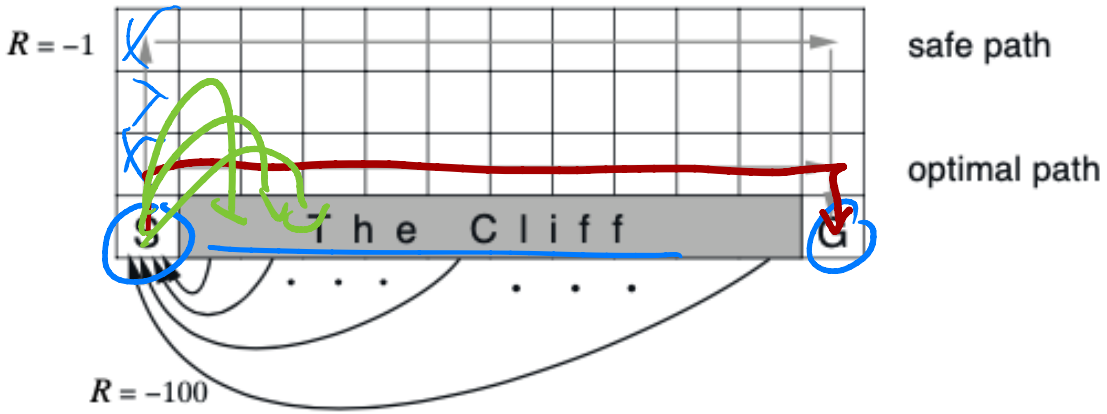
SARSA vs Q-learning



- Model is deterministic.
- Exploration policy (π) is ε-greedy.

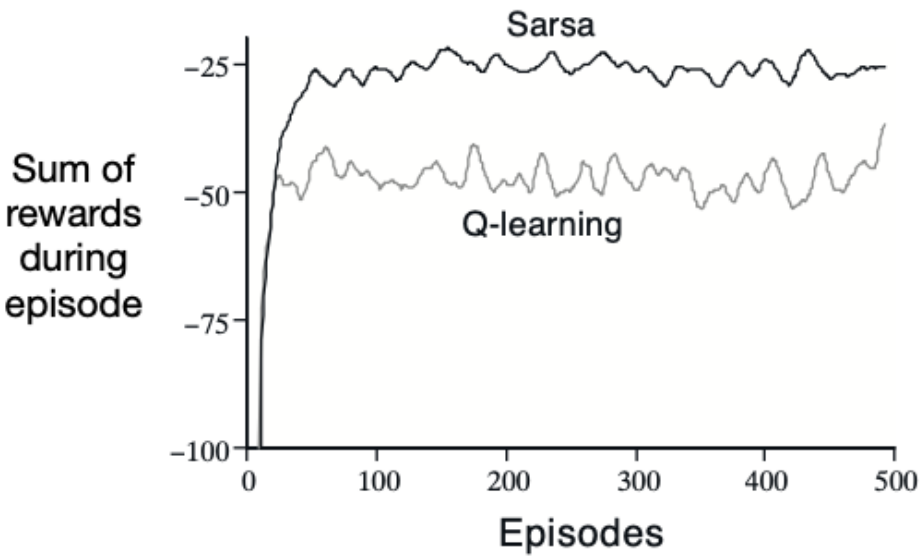
SARSA or Q-learning: what will be the difference?

SARSA vs Q-learning



- Model is deterministic.
- Exploration policy (π) is ε-greedy.

SARSA or Q-learning: what will be the difference?



- For large ε, SARSA will avoid the optimal shortest path.
- Q-learning will learn the shortest path but will often fall.

How to choose the learning rate and guarantee exploration?

Recall: for Q learning, you are given an exploration policy π and apply:

$$A_{t+1} \sim \pi$$

$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right).$$

Questions:

- How to choose π?
- How to choose α_t?

Solution: exploration/exploitation tradeoff (course 3), and Q-learning with UCB Exploration is Sample Efficient for Infinite-Horizon MDP by Dong et al 2019.

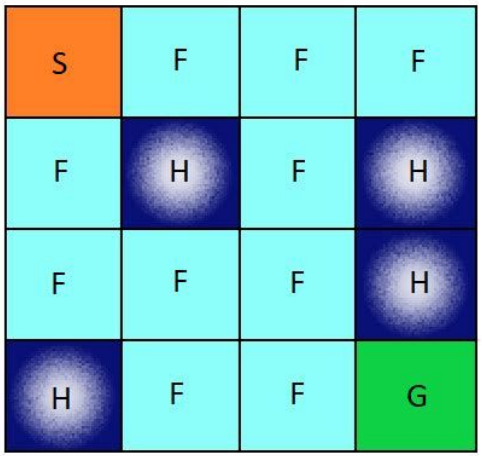
Table of contents

Important notions

(your job here)

TD and Q-learning are tabular method

They can be proven to converge.

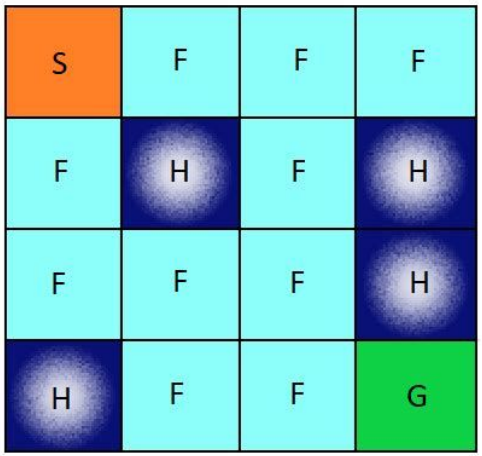


S	$V(S)$
(0,0)	
(0,1)	
(0,2)	
(0,3)	
(1,0)	
(1,1)	
(1,2)	
(1,3)	
⋮	

S \ A	N	S	E	W
(0,0)				
(0,1)				
(0,2)				
(0,3)				
(1,0)				
(1,1)				
(1,2)				
⋮				

TD and Q-learning are tabular method

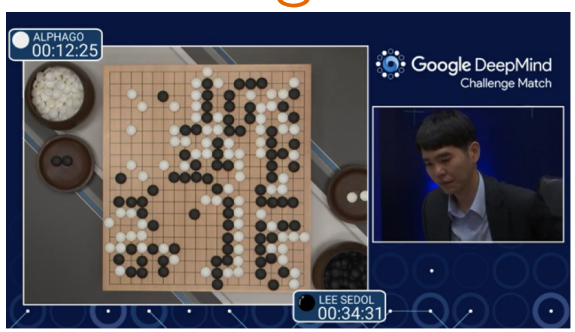
They can be proven to converge.



S	$V(S)$
(0,0)	
(0,1)	
(0,2)	
(0,3)	
(1,0)	
(1,1)	
(1,2)	
(1,3)	
⋮	

S \ A	N	S	E	W
(0,0)				
(0,1)				
(0,2)				
(0,3)				
(1,0)				
(1,1)				
(1,2)				
⋮				

What about large state spaces?



Outline

Reminder: Tabular MDP

We want to find $Q(s,a) \approx Q^*(s,a)$.

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a).$$

Two types of methods:

- MC methods:

$$Q^\pi(s,a) = \frac{1}{K} \sum_{k=1}^{K} G^{(k)}$$

- TD methods (SARSA / Q-learning)

Reminder: Tabular MDP

We want to find $Q(s,a) \approx Q^*(s,a)$.

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s,a).$$

Does it scale?

The complexity is $\Omega(|\mathcal{S}||\mathcal{A}|)$.

$Q(s,a)$	a_1	a_2	a_3	...
s_1				
s_2				
s_3				
s_4				
$\vdots$				

Two types of methods:

- MC methods:

$$Q^\pi(s,a) = \frac{1}{K} \sum_{k=1}^{K} G^{(k)}$$

- TD methods (SARSA / Q-learning)

What are typical state space sizes?

The curse of dimensionality

Managing a portfolio of 10 types of product, with 100 product each max.

- $|\mathcal{S}| = 100^{10} = 10^{20}$.
- $\mathcal{A}$ = possible orders (=10×100?)

$\vec{s} = (\#\text{prod } 1, \#\text{prod } 2 \ldots \#\text{prod } 10)$

$\vec{s} = (s^1, s^2 \ldots s^{10})$

What are typical state space sizes?

The curse of dimensionality

Managing a portfolio of 10 types of product, with 100 product each max.

- $|\mathcal{S}| = 100^{10} = 10^{20}$.
- $\mathcal{A}$ = possible orders ($=10 \times 100$?)

Game of go

- $|\mathcal{S}| = 3^{19 \times 19}$ (19×19 board game).
- $|\mathcal{A}| = 19 \times 19$.

There are $\approx 10^{170}$ Q-values.

What are typical state space sizes?

The curse of dimensionality

Breakout (1976) ▸ Atari games

- $|\mathcal{S}| = 8^{84 \times 84}$ (84×84 screen, 8 colors).
- $|\mathcal{A}| = 2$ (left, right).

There are $\approx 10^{2000}$ Q-values.

What are typical state space sizes?

The curse of dimensionality

Breakout (1976) ▸ Atari games

- $|\mathcal{S}| = 8^{84\times 84}$ (84×84 screen, 8 colors).
- $|\mathcal{A}| = 2$ (left, right).

There are $\approx 10^{2000}$ Q-values.

Starcraft ▸ alphastar

- $|\mathcal{S}| \gg |\mathcal{A}| \approx +\infty$??

We need approximations.

Table of contents

TD-learning and function approximation

The tabular TD-learning or Q-learning algorithm is:

$$V(S_t) := V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$

$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right).$$

This does not scale if $|\mathcal{S}|$ (or $|\mathcal{A}|$) are large.

Function approximation

We replace the exact Q-table (or value function V) by an approximation:

$$Q(S, A) \approx q_{\mathsf{w}}(S, A),$$

where w is a vector parameter to be found.

Function approximation

We replace the exact Q-table (or value function V) by an approximation:

$$Q(S, A) \approx q_{\mathsf{w}}(S, A),$$

where w is a vector parameter to be found.

- (classic): Use a linear approximation. For instance:

  $$Q(S, A) = \mathsf{w}^T \phi(s, a),$$

  where $\phi(s, a)$ is a feature vector.

Function approximation

We replace the exact Q-table (or value function V) by an approximation:

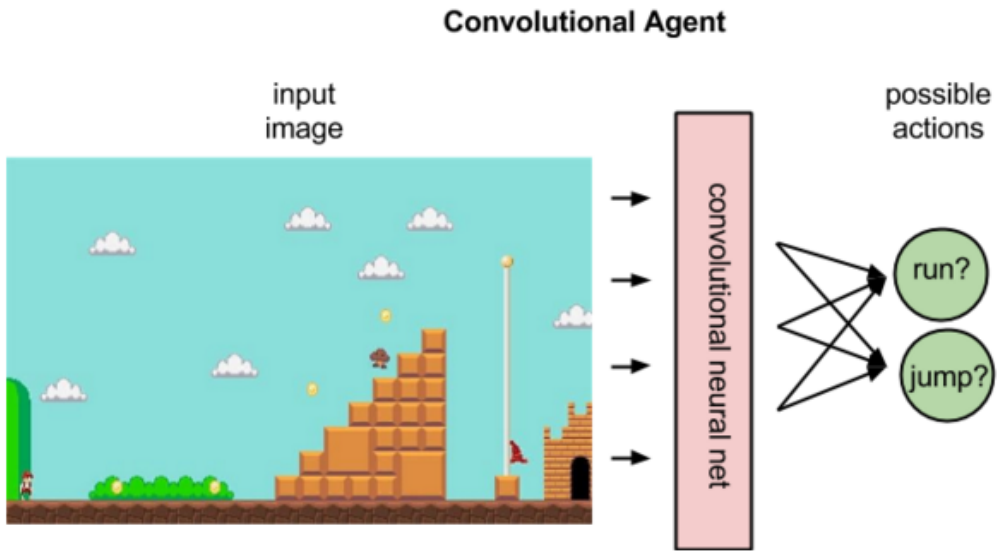
$$Q(S, A) \approx q_{\mathsf{w}}(S, A),$$

where w is a vector parameter to be found.

- (classic): Use a linear approximation. For instance:

  $$Q(S, A) = \mathsf{w}^T \phi(s, a),$$

  where $\phi(s, a)$ is a feature vector.
- ("modern"): q_{w} is a deep neural network.



From Q-learning to deep Q-learning

The original Q-learning uses that:

$$Q(S_t, A_t) = \mathbb{E}\left[R_{t+1} + \max_{a \in \mathcal{A}} Q(S_{t+1}, a)\right].$$

We want to find w such that $\underbrace{q_{\mathsf{w}}(S_t, A_t)}_{\text{predictor}} \approx \underbrace{\mathbb{E}\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}} q_{\mathsf{w}}(S_{t+1}, a)\right]}_{\text{target}}$.

From Q-learning to deep Q-learning

The original Q-learning uses that:

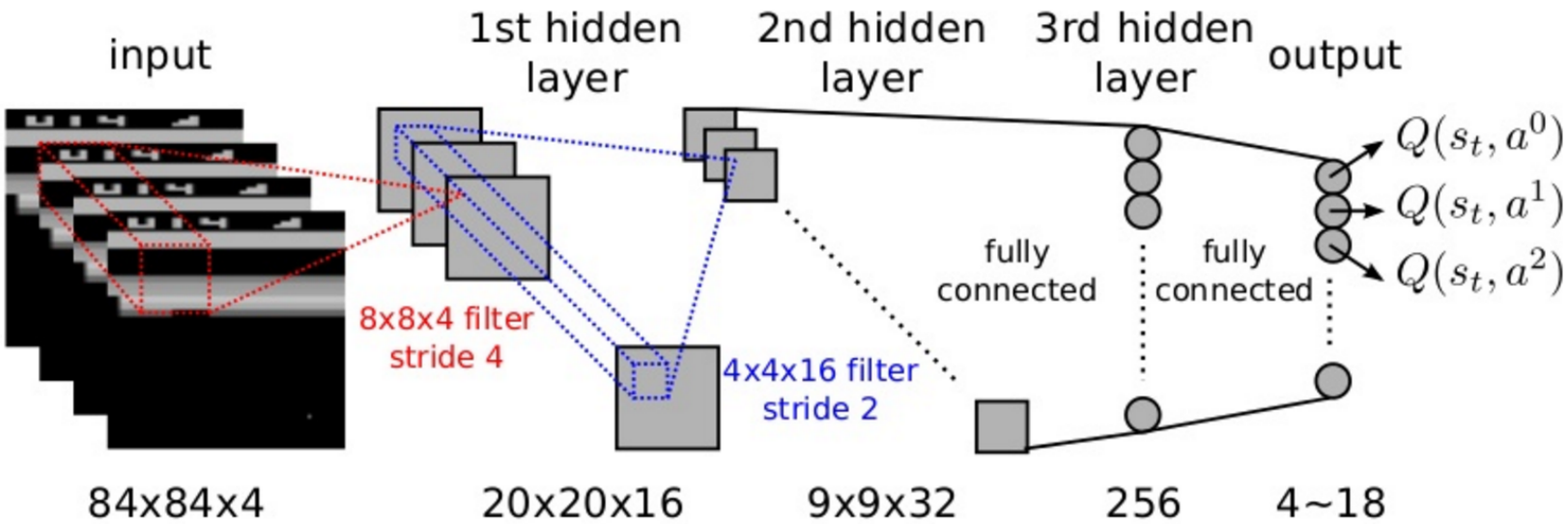
$$Q(S_t, A_t) = \mathbb{E}\left[R_{t+1} + \max_{a \in \mathcal{A}} Q(S_{t+1}, a)\right].$$

We want to find w such that $\underbrace{q_{\mathsf{w}}(S_t, A_t)}_{\text{predictor}} \approx \underbrace{\mathbb{E}\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}} q_{\mathsf{w}}(S_{t+1}, a)\right]}_{\text{target}}$.

Deep Q-learning minimizes the L_2 norm and use gradient descent:

$$\mathsf{w} := \mathsf{w} + \alpha \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} q_{\mathsf{w}}(S_t, a) - q_{\mathsf{w}}(S_t, A_t)\right) \nabla_{\mathsf{w}}(q_{\mathsf{w}}(S_t, A_t)).$$

Example of breakout



Why is vanilla unstable?

We want to find w such that $\underbrace{q_{\mathsf{w}}(S_t, A_t)}_{\text{predictor}} \approx \underbrace{\mathbb{E}\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}} q_{\mathsf{w}}(S_{t+1}, a)\right]}_{\text{target}}$.

For that, we do:

$$\mathsf{w} := \mathsf{w} + \alpha \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} q_{\mathsf{w}}(S_t, a) - q_{\mathsf{w}}(S_t, A_t)\right) \nabla_{\mathsf{w}}(q_{\mathsf{w}}(S_t, A_t)).$$

Problems:

Why is vanilla unstable?

We want to find w such that $\underbrace{q_{\mathsf{w}}(S_t, A_t)}_{\text{predictor}} \approx \underbrace{\mathbb{E}\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}} q_{\mathsf{w}}(S_{t+1}, a)\right]}_{\text{target}}$.

For that, we do:

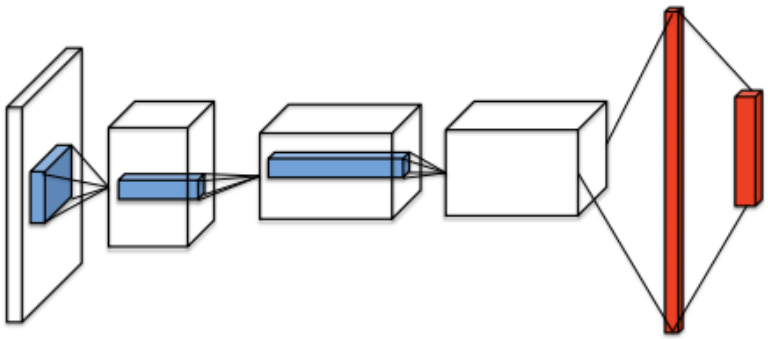
$$\mathsf{w} := \mathsf{w} + \alpha \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} q_{\mathsf{w}}(S_t, a) - q_{\mathsf{w}}(S_t, A_t) \right) \nabla_{\mathsf{w}} (q_{\mathsf{w}}(S_t, A_t)).$$

Problems:

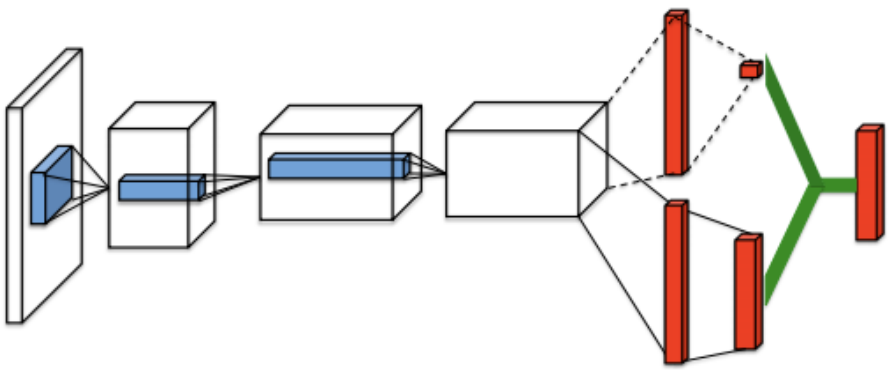
- Target and sources are highly correlated
- Target changes as we learn.
- Exploration is not guaranteed.

Learning algorithm can be unstable.

Possible solution: replay buffer or separate target network



Vanilla Q-learning uses a single network



DDQN uses a slow learning target network and a fast learning q-network.

Applications of Deep RL

- Resource management (energy)
- Computer vision and robotics
- Finance
- ...

Fundamental idea is simple but making the system stable and fast is an issue. Also, delayed actions or sparse rewards is difficult.

Table of contents

Policy search

We are given a family of policies π_w parametrized by $w \in \mathcal{W}$. Typically:

$$\pi_w(a \mid s) \propto \exp(w^T \phi(s, a)),$$

where $\phi(s, a)$ is a feature vector.

Policy search

We are given a family of policies π_w parametrized by $w \in \mathcal{W}$. Typically:

$$\pi_w(a \mid s) \propto \exp(w^T \phi(s, a)),$$

where $\phi(s, a)$ is a feature vector.

Let $J(w) := V^{\pi_w}(s_0)$ be its performance. We want to find w that maximizes $J(w)$.

Policy search

We are given a family of policies π_w parametrized by $w \in \mathcal{W}$. Typically:

$$\pi_w(a \mid s) \propto \exp(w^T \phi(s, a)),$$

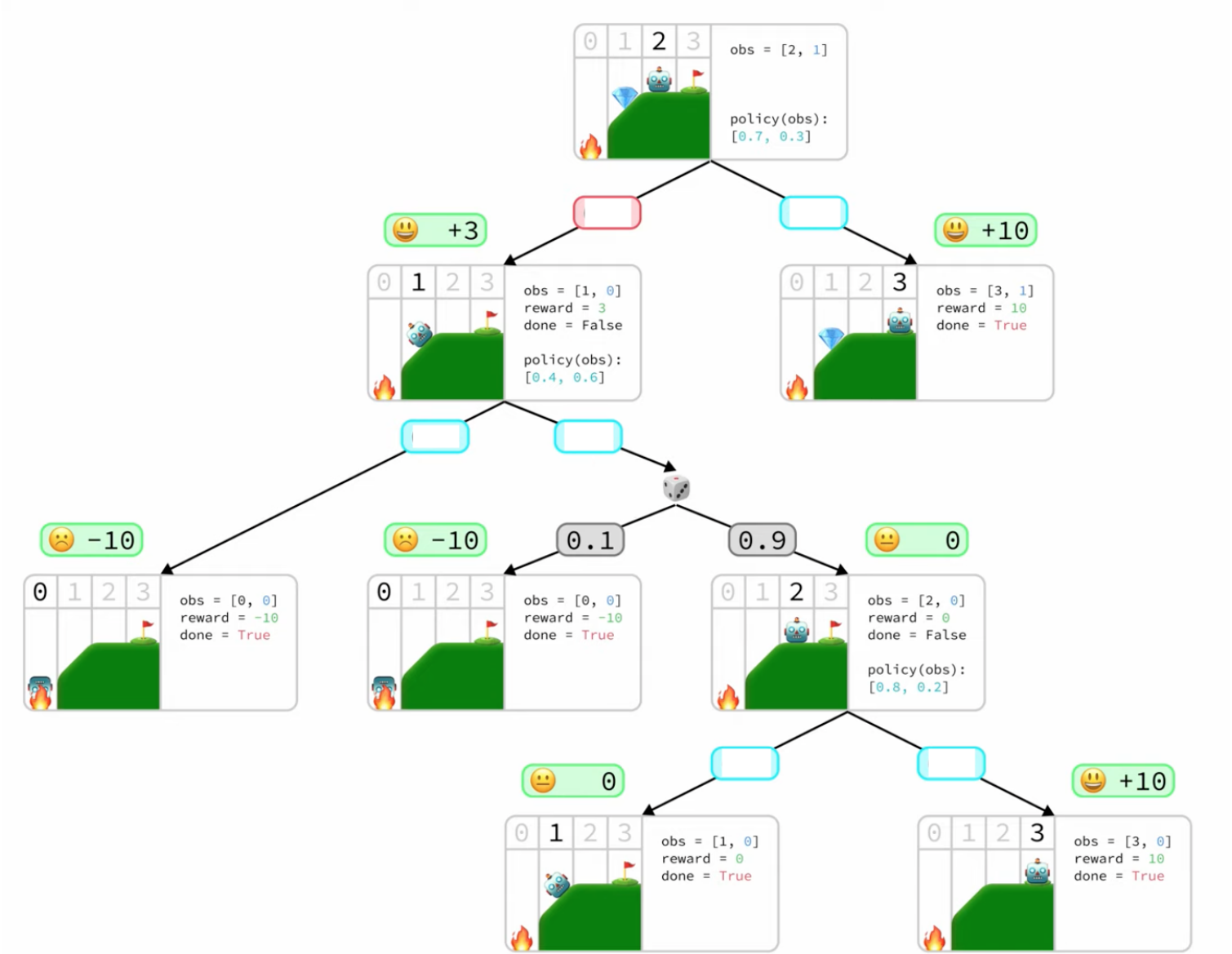
where $\phi(s, a)$ is a feature vector.

Let $J(w) := V^{\pi_w}(s_0)$ be its performance. We want to find w that maximizes $J(w)$.

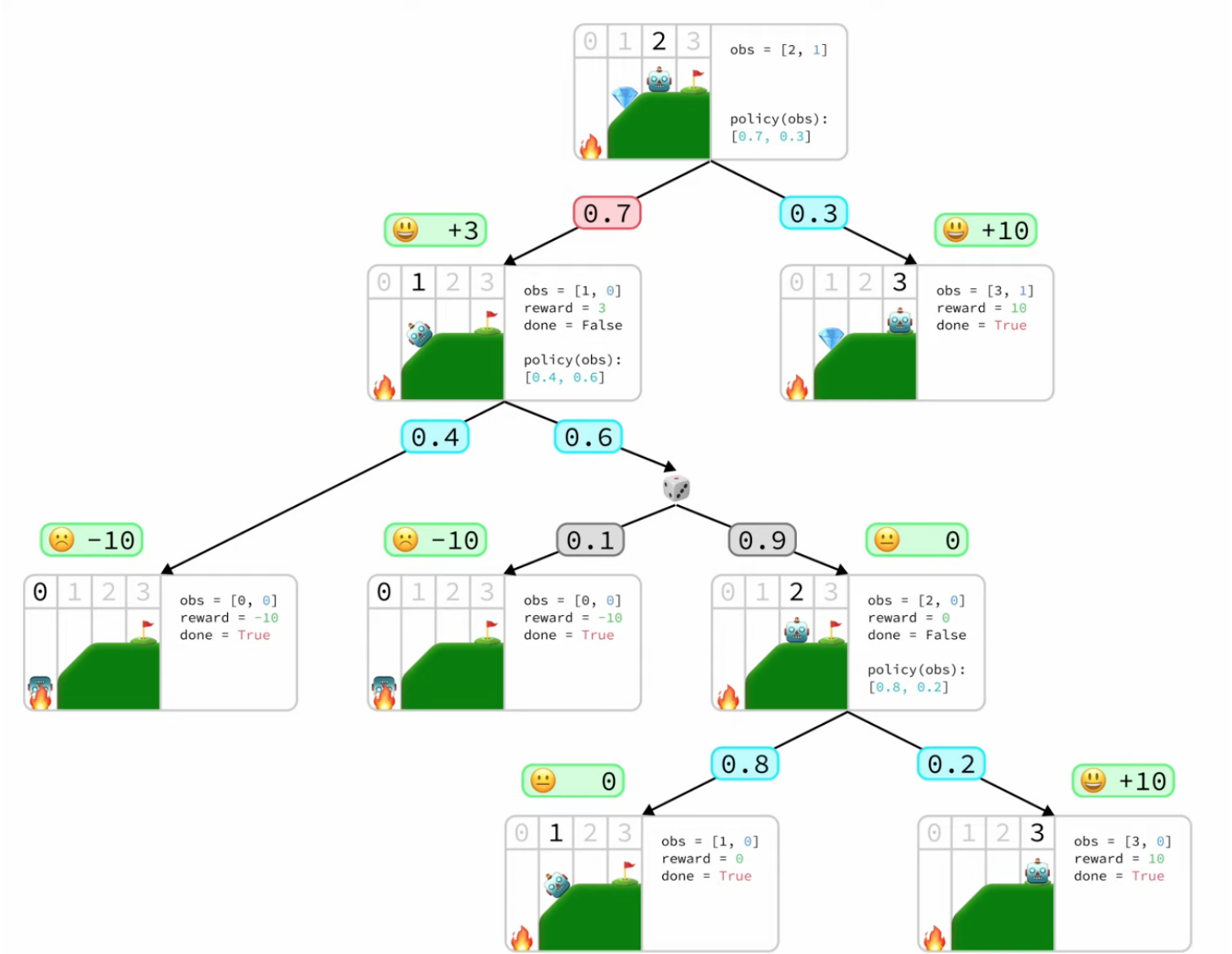
- Sometimes, this works well with direct methods (brute-force)
- We can also use policy gradients:

$$w := w + \alpha \nabla_w J(w).$$

On an example https://www.youtube.com/watch?v=cQfOQcpYRzE

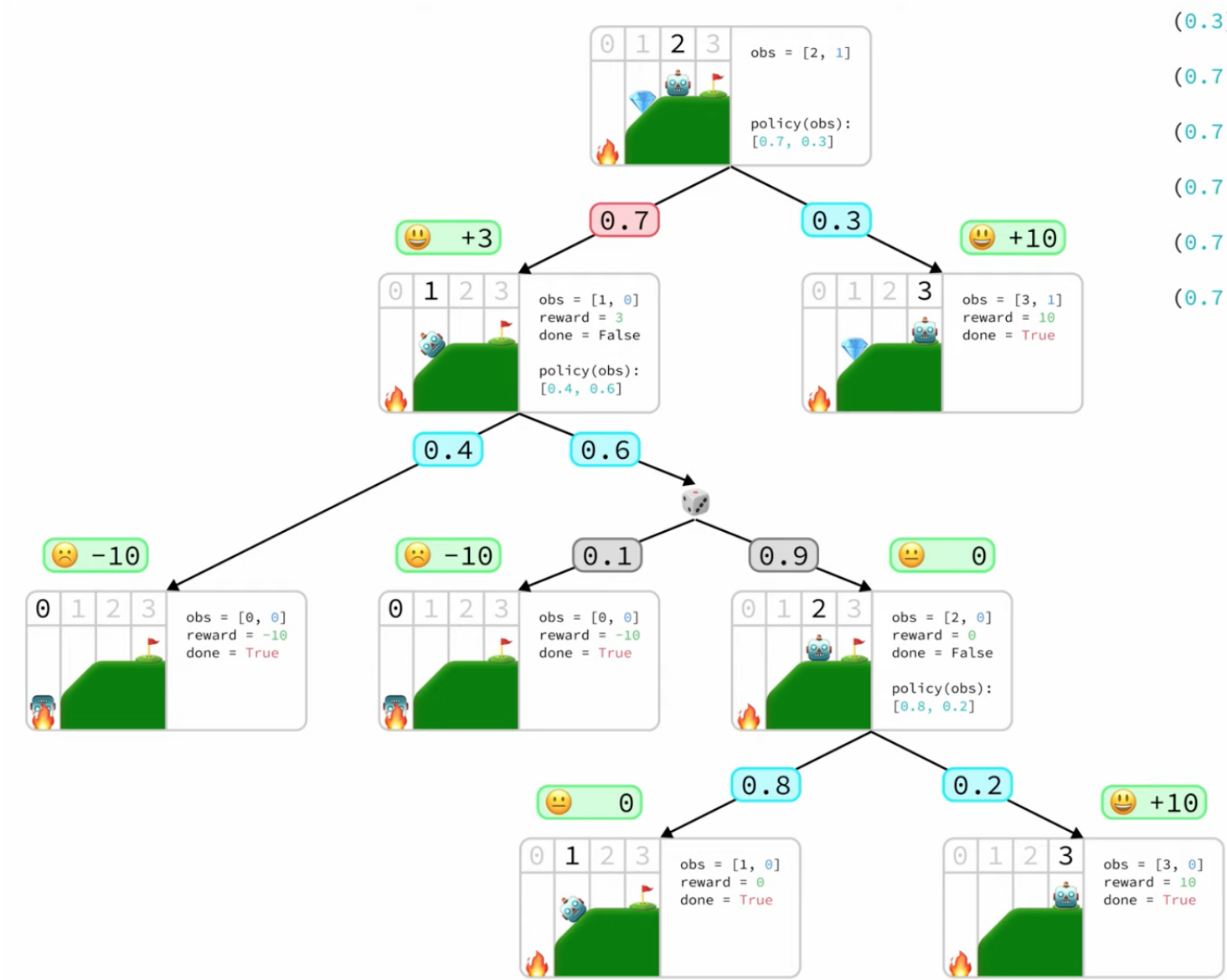


On an example https://www.youtube.com/watch?v=cQfOQcpYRzE



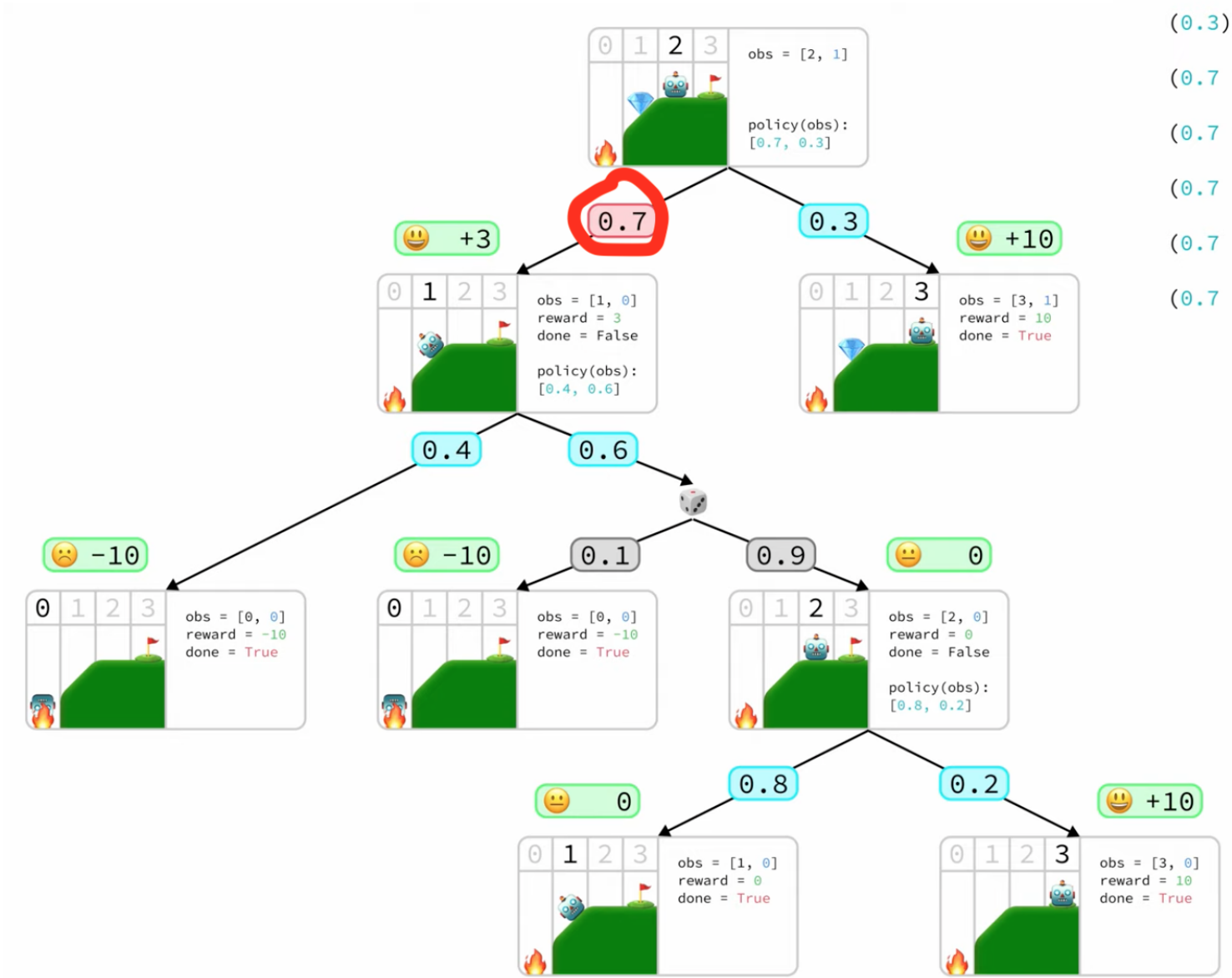
On an example https://www.youtube.com/watch?v=cQfOQcpYRzE

```
Expected Return (G) =

    (0.7) * (3) +

    (0.3) * (10) +

    (0.7 * 0.4) * (-10) +

    (0.7 * 0.6 * 0.1) * (-10) +

    (0.7 * 0.6 * 0.9) * (0) +

    (0.7 * 0.6 * 0.9 * 0.8) * (0) +

    (0.7 * 0.6 * 0.9 * 0.2) * (10)
```



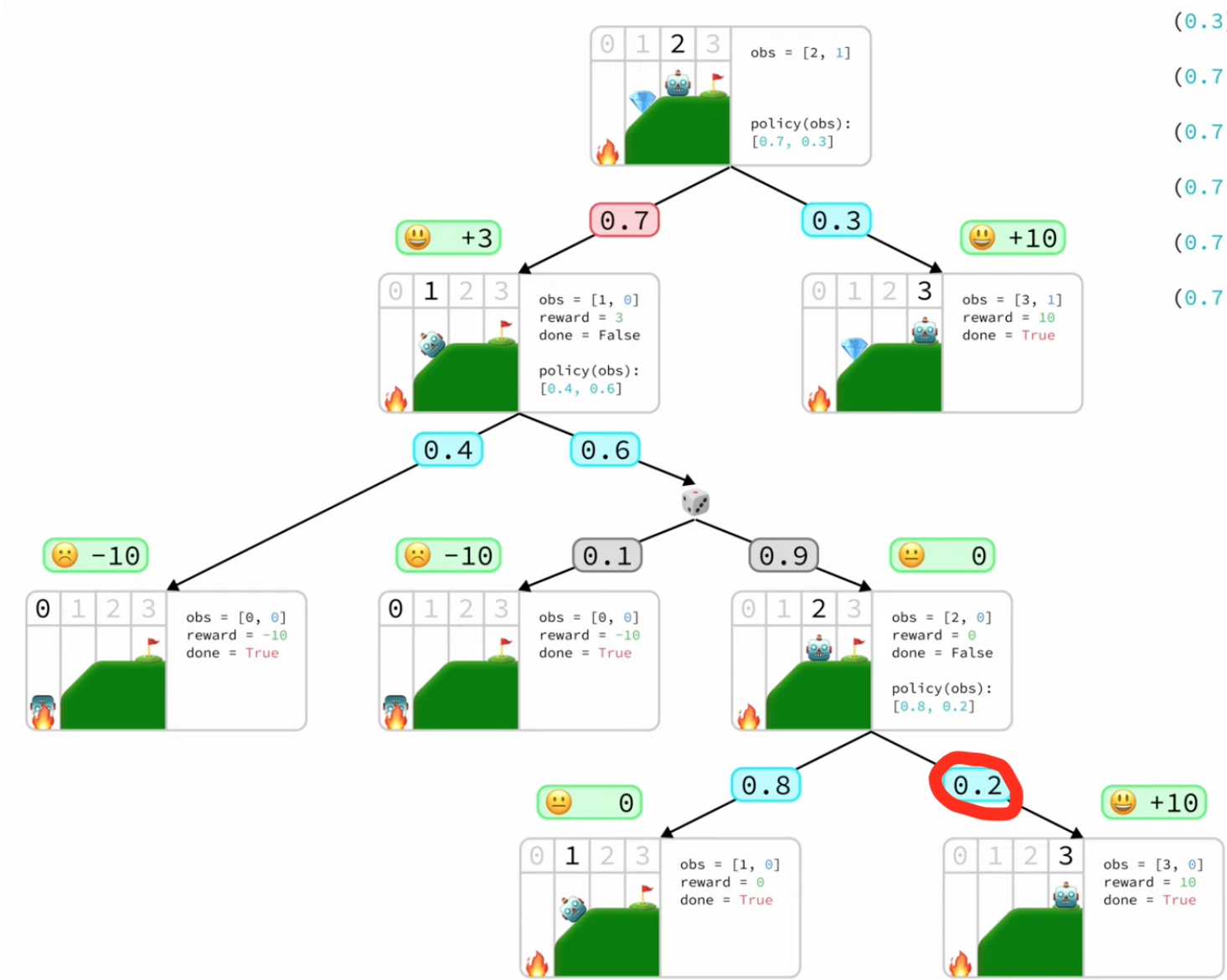
On an example https://www.youtube.com/watch?v=cQfOQcpYRzE

```
Expected Return (G) =

    (0.7) * (3) +

    (0.3) * (10) +

    (0.7 * 0.4) * (-10) +

    (0.7 * 0.6 * 0.1) * (-10) +

    (0.7 * 0.6 * 0.9) * (0) +

    (0.7 * 0.6 * 0.9 * 0.8) * (0) +

    (0.7 * 0.6 * 0.9 * 0.2) * (10)
```



On an example https://www.youtube.com/watch?v=cQfOQcpYRzE

```
Expected Return (G) =

    (0.7) * (3) +

    (0.3) * (10) +

    (0.7 * 0.4) * (-10) +

    (0.7 * 0.6 * 0.1) * (-10) +

    (0.7 * 0.6 * 0.9) * (0) +

    (0.7 * 0.6 * 0.9 * 0.8) * (0) +

    (0.7 * 0.6 * 0.9 * 0.2) * (10)
```



How to estimate the gradient with trajectories?

Assume for simplicity that each state is visited only once.
The probability of choosing a in state s is $\pi(a|s)$.

$$\begin{aligned}\nabla_{\pi(a|s)}\mathbb{E}\left[G_0\right] &= \mathbb{P}(\text{attaining } s)Q(s,a)\\ &= \frac{1}{\pi(a|s)}\mathbb{P}(\text{observing } (s,a))Q(s,a)\end{aligned}$$

How to estimate the gradient with trajectories?

Assume for simplicity that each state is visited only once.
The probability of choosing a in state s is $\pi(a|s)$.

$$\begin{aligned}\nabla_{\pi(a|s)}\mathbb{E}\left[G_0\right] &= \mathbb{P}(\text{attaining } s)Q(s,a)\\ &= \frac{1}{\pi(a|s)}\mathbb{P}(\text{observing } (s,a))Q(s,a)\end{aligned}$$

Algorithm: We want to compute $\text{gradient}(S,A) = \nabla_{\pi(a|s)}\mathbb{E}\left[G_0\right]$.

- Run a trajectory and observe S_t, A_t.
- For each t:

$$\widehat{gradient}(S_t,A_t) = \frac{1}{\pi(A_t|S_t)}G_t.$$

Theorem. For all s,a: $\mathbb{E}\left[\widehat{gradient}(s,a)\right] = \nabla_{\pi(a|s)}\mathbb{E}\left[G\right]$.

The policy gradient theorem

Assume that $\pi(a|s) = f_w(s, a)$. We have:

$$\nabla_w \mathbb{E}[G_0] = \sum_{s,a} \nabla_w \pi(a|s) \nabla_{\pi(a|s)} \mathbb{E}[G_0]$$

The policy gradient theorem

Assume that $\pi(a|s) = f_w(s,a)$. We have:

$$\nabla_w \mathbb{E}[G_0] = \sum_{s,a} \nabla_w \pi(a|s) \nabla_{\pi(a|s)} \mathbb{E}[G_0]$$

Hence, an unbiased estimate of the gradient $\nabla_w \mathbb{E}[G_0]$ is

$$\sum_t \frac{(\nabla_w \pi(A_t|S_t))}{\pi(A_t|S_t)} G_t.$$

By using that $\nabla log(y) = \nabla(y)/y$, we get:

An unbiased estimate of the gradient is:

$$\nabla_w \mathbb{E}[G_0] = \mathbb{E}\left[\sum_t (\nabla_w \log \pi(A_t|S_t)) G_t\right].$$

Why is $\nabla \log \pi(a|s)$ easy to compute?

Reminder: if $p_i = e^{u_i} / \sum e^{u_j}$, then

$$\frac{\partial}{\partial u_j} \log p_i = 1_{\{i=j\}} - p_j.$$

Why is $\nabla \log \pi(a|s)$ easy to compute?

Reminder: if $p_i = e^{u_i} / \sum e^{u_j}$, then

$$\frac{\partial}{\partial u_j} \log p_i = 1_{\{i=j\}} - p_j.$$

If $\pi(a|s) \propto \exp(w^T \phi(s,a))$, then it means that $\pi(a|s) = \frac{\exp(w^T \phi(s,a))}{\sum_{a'} \exp(w^T \phi(s,a'))}$.

As a consequence:

$$\nabla_w \pi_w(a|s) = \phi(a,s) - \sum_{a'} \phi(a'|s) \pi_w(a'|s).$$

The REINFORCE algorithm

REINFORCE

1: Initialize w.
2: **while** True **do**
3: Simulate a trajectory (from $t = 1$ to T)
4: **for** $t = T$ to $t = 1$ **do**
5: $G_t := \sum_{t'=t}^{T} R_{t'}$.
6: $\nabla J := G_t \nabla \log \pi(A_t|S_t)$.
7: $\mathsf{w} := \mathsf{w} + \alpha \nabla J$.
8: **end for**
9: **end while**

Recall that $\nabla \log \pi(a|s)$ is easy to compute when $\pi(a|s) \propto w^T \phi(s, a)$.

Variance reduction

Problem: Monte-Carlo sampling can have a large variance.
Ex: if $Q(s, a_1) = 8 \pm 1$ and $Q(s, a_2) = 8.5 \pm 1$, is a_2 better than a_1?

Variance reduction

Problem: Monte-Carlo sampling can have a large variance.
Ex: if $Q(s, a_1) = 8 \pm 1$ and $Q(s, a_2) = 8.5 \pm 1$, is a_2 better than a_1?

Solution: add a baseline $h : \mathcal{S} \to \mathbb{R}$. Indeed, using the same log-trick:

$$\begin{aligned}\mathbb{E}\left[h(s_t)\nabla \log \pi(a_t|s_t)\right] &= \mathbb{E}\left[\sum_{a\in\mathcal{A}} h(s_t)\nabla\pi(a|s_t)\right] \\ &= 0\end{aligned}$$

This shows that for any function h, one has:

$$\nabla_w J(s_0) \propto \sum_t \mathbb{E}\left[(G_t - h(s_t))\nabla \log \pi(a_t|s_t)\right]\}.$$

Choosing a h close to G_t reduces the variance of the estimator.

Table of contents

Classes of learning algorithms

We have seen two classes of RL methods:

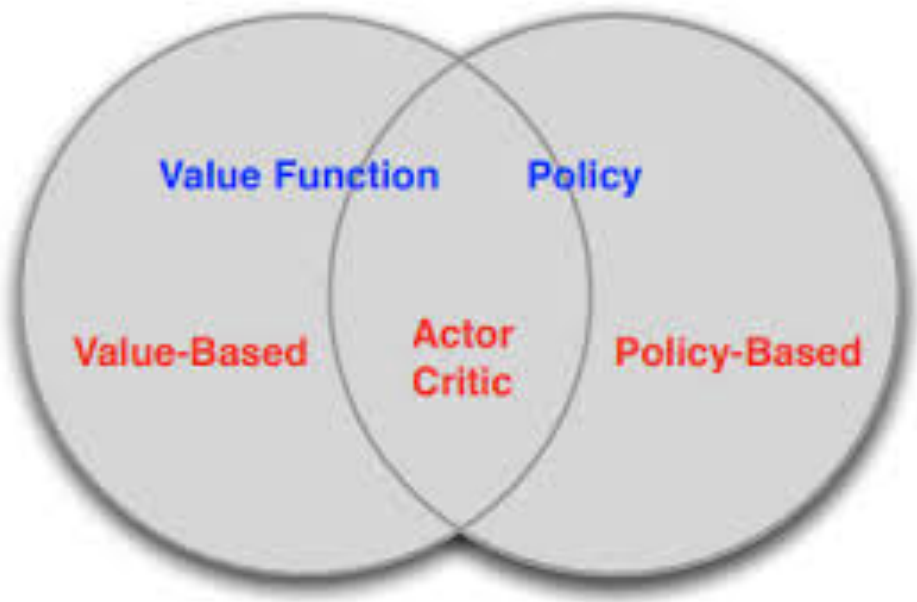
- Value-based (SARSA, Q-learning, Deep QL)
- Policy-based (Policy gradient, REINFORCE)

- Value-based learning can be unstable but uses samples efficiently.
- Policy-based tend to be more robust.

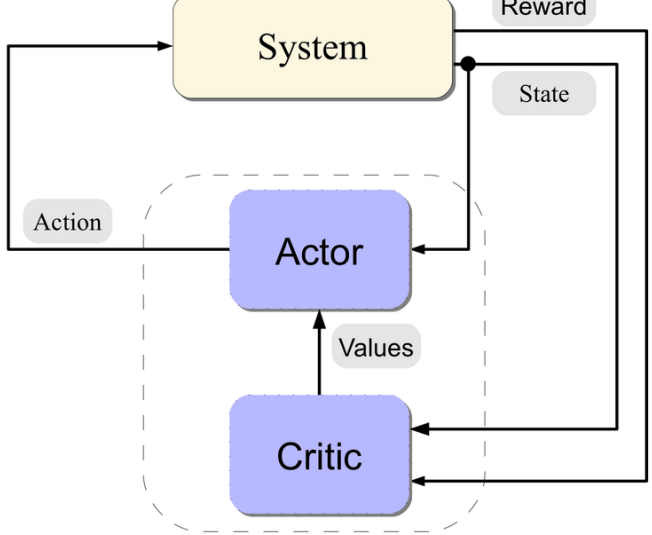
Classes of learning algorithms

We have seen two classes of RL methods:

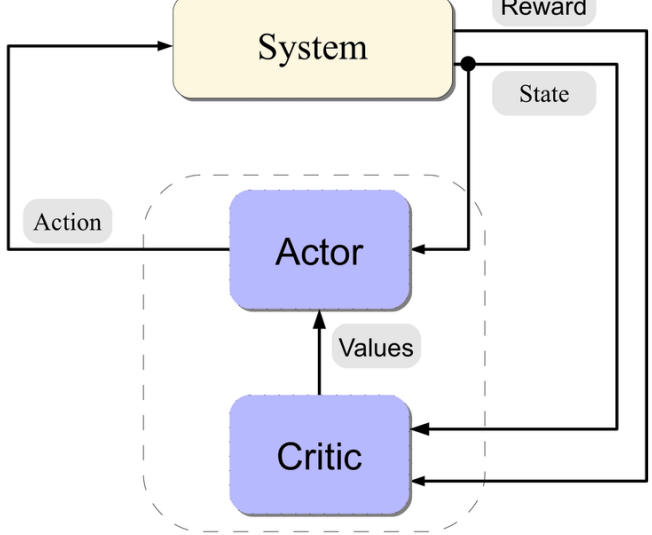
- Value-based (SARSA, Q-learning, Deep QL) =Critic
- Policy-based (Policy gradient, REINFORCE) =Actor
- Value-based learning can be unstable but uses samples efficiently.
- Policy-based tend to be more robust.



Actor Critic method



Actor Critic method



Basic Actor Critic

Initialize parameters $w^{(a)}$ (Actor) and $w^{(c)}$ (Critic)
while True **do**
  Initialize S
  for $t = 1$ to $t = T$ **do**
    $A_t \sim \pi_w(S)$ and simulate R, S'
    $w^{(c)} := w^{(c)} + \alpha^{(c)}(R + \gamma v_{w^{(c)}}(S') - v_{w^{(c)}}(S))$ *# TD-update*
    $w^{(a)} := w^{(a)} + \alpha^{(a)} v_{w^{(c)}}(S) \nabla \log \pi(a_t|s_t)$ *# Policy-gradient*
    S:=S'.
  end for
end while

Going further

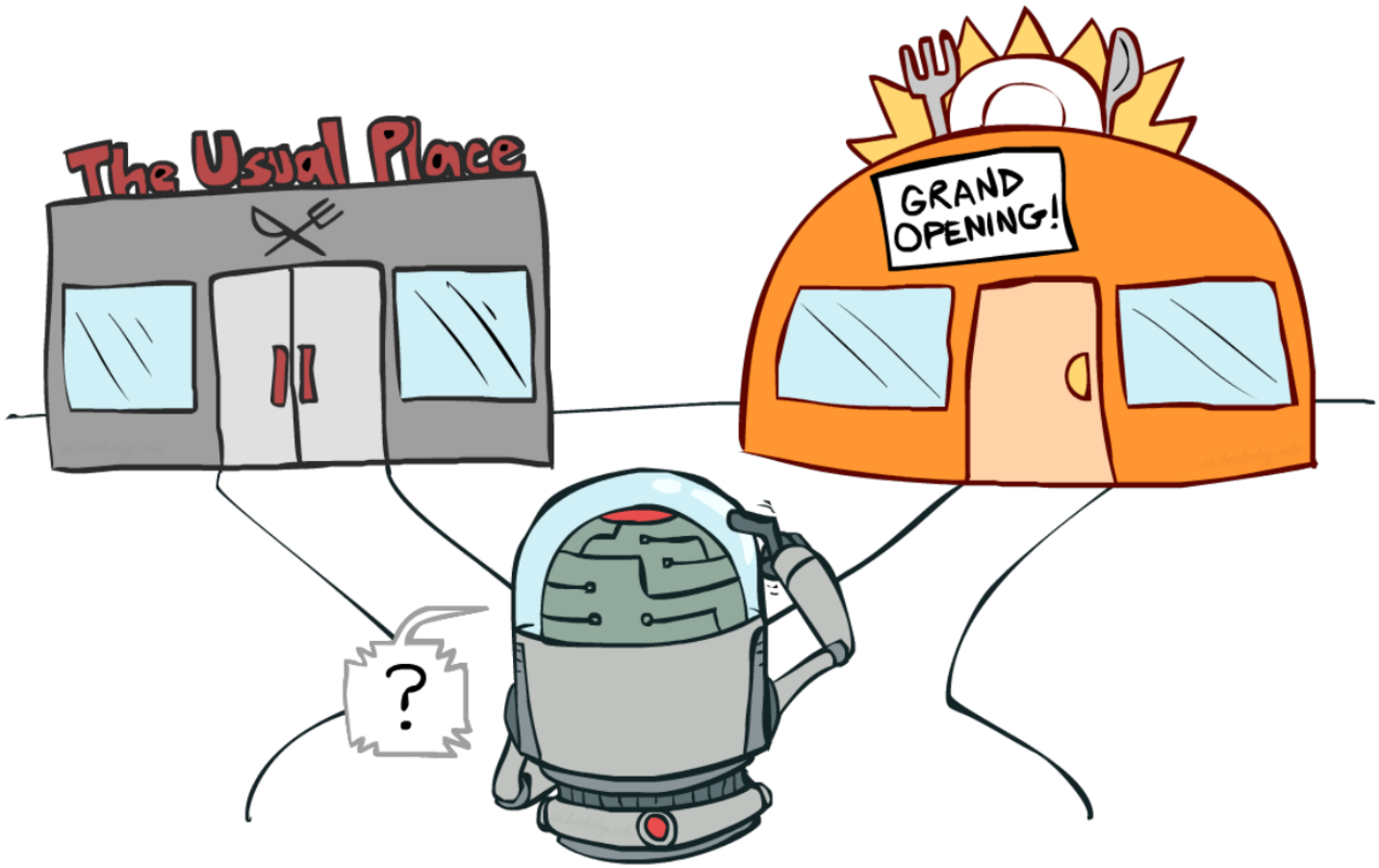
Extra-reading:

- Introduction to Reinforcement Learning (Sutton-Barto, 2018 last ed.)
- Algorithms for Reinforcement Learning (Szepesvari, 2010)
- Deep Reinforcement learning: hands on (Maxim Lapan, 2020)

Next course: some thoughts on exploration / exploitation.

Outline

Reminder: exploration-exploitation dilemma and bandits



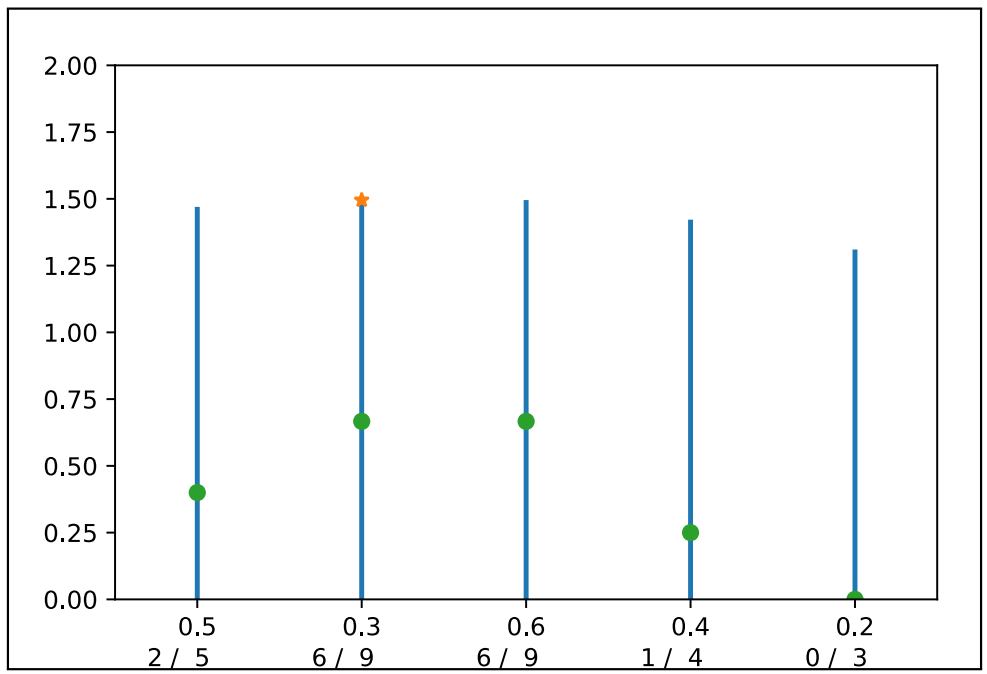
- How useful is this for RL?

Reminder: UCB algorithm

UCB computes a confidence bound $UCB_a(t)$ such that $\mu_a(t) \leq UCB_a(t)$ with high probability. Example : $UCB1$ [Auer et al. 02] uses

$$UCB_a(t) = \hat{\mu}_a(t) + \sqrt{\frac{\alpha \log t}{2N_a(t)}}.$$

- Choose $A_{t+1} \in \arg\max_{a \in \{1 \dots n\}} UCB_a(t)$ (optimism principle).



Can we use optimism for MDPs?

Observe the empirical means $\hat{R}(s,a)$ and $\hat{P}(s' \mid s,a)$.

What bonus should one use?

Can we use optimism for MDPs?

Observe the empirical means $\hat{R}(s,a)$ and $\hat{P}(s' \mid s,a)$.

What bonus should one use?

- UCRL2 (Jaksch 2010) or variant: use bonus on R and P. Let $\delta(s,a) = C\sqrt{t/N_t(s,a)}$ where $N_t(s,a)$ is the number of time that you took action a in state s before time t.

$$\mathcal{R} = \{\text{vector } r \text{ such that for all } s,a\text{: } |r(s,a) - \hat{r}(s,a)| \leq \delta(s,a)\}$$

$$\mathcal{P} = \{\text{trans. matrix } P \text{ s.t. for all } s,a,a' \left| P(s,a,a') - \hat{P}(s,a,a') \right| \leq \delta(s,a$$

Optimism:

  - Apply π that maximizes $V^{\pi}_{r,P\in\mathcal{R},\mathcal{P}}$ (by using extended value iteration) and re-update the policy periodically.

Tree search

For turn-based two players zero sum games

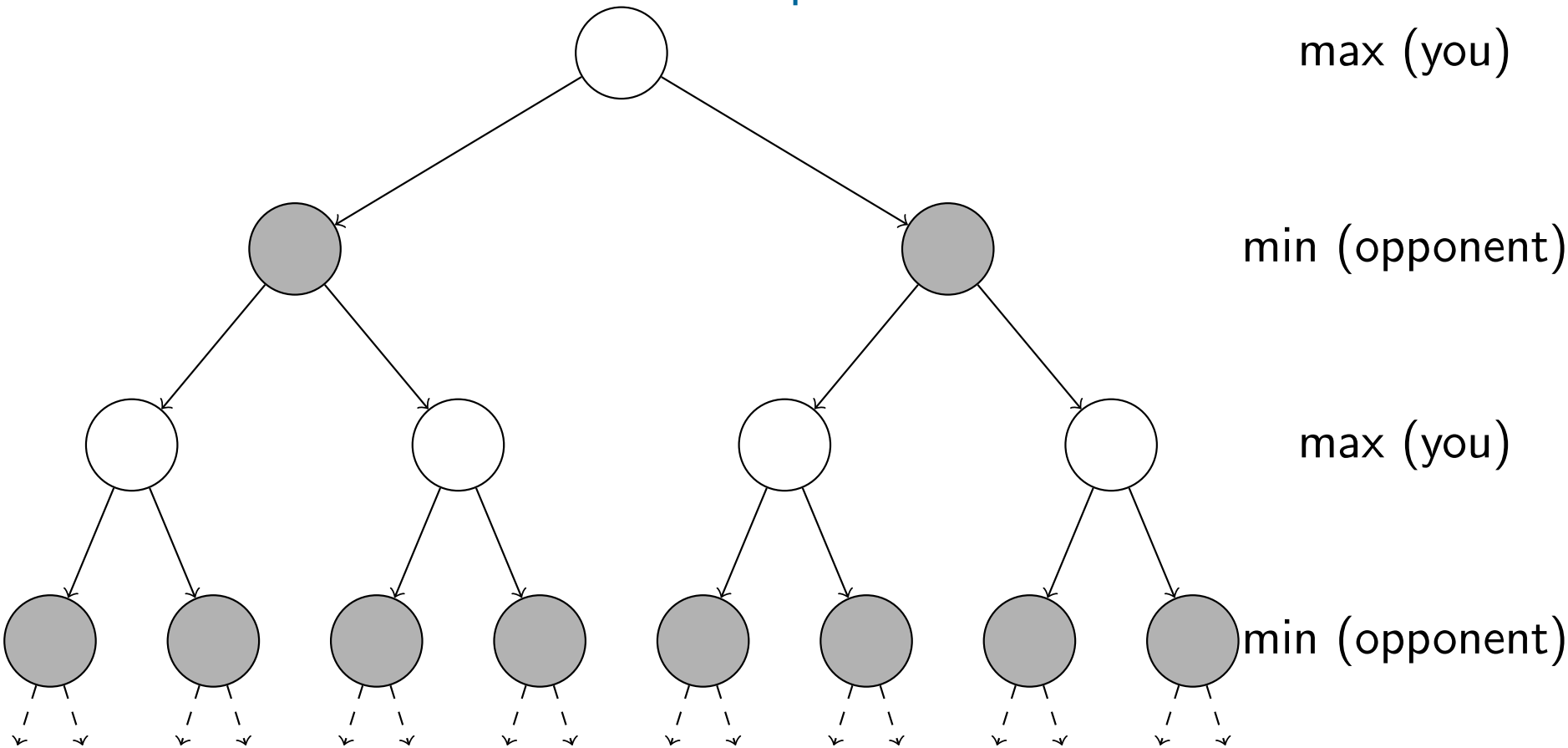
From a given position, takes the best decision.

- Generate a tree of possibilities.
- Explore this tree.

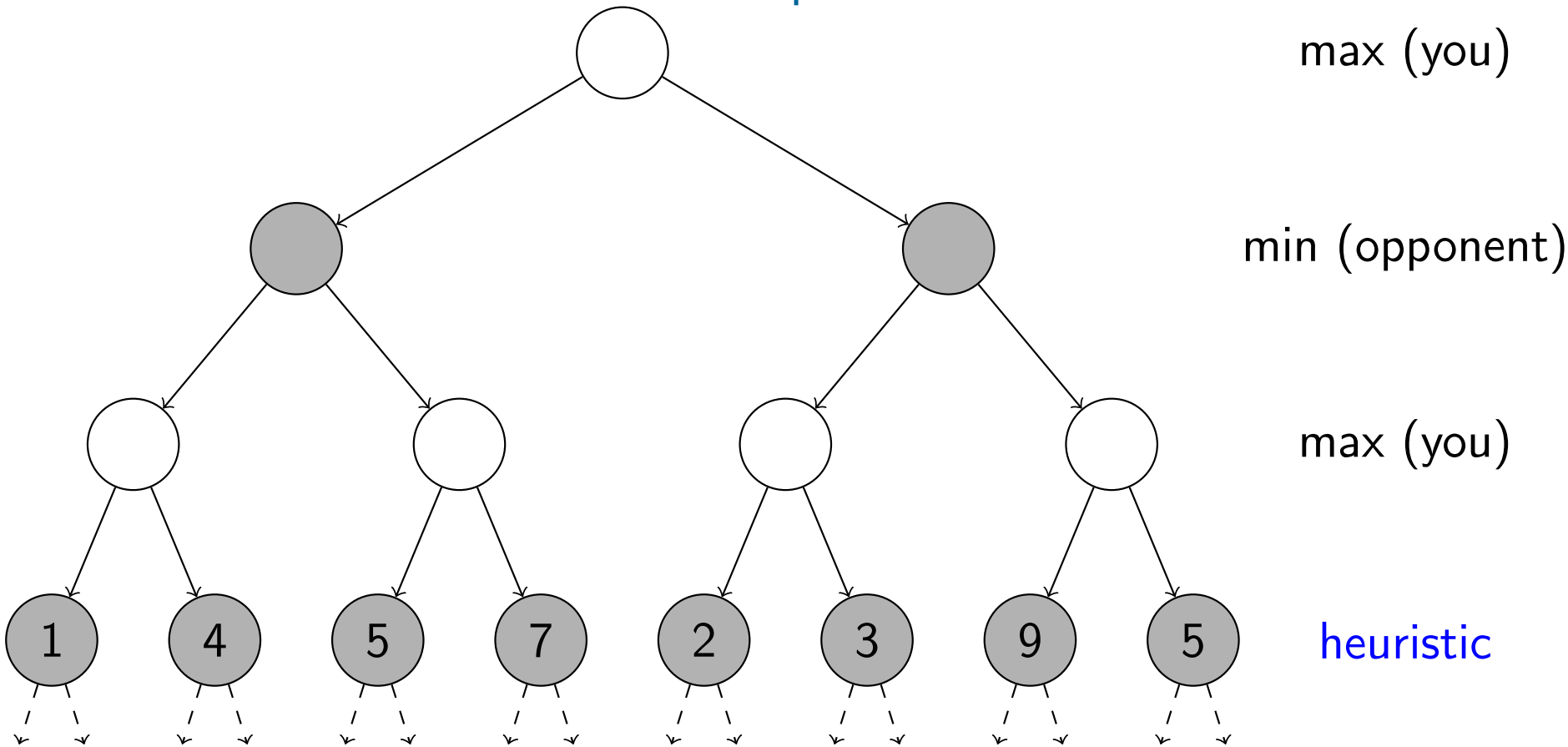
What if the tree is too big?

Table of contents

You can construct the tree of possibilities



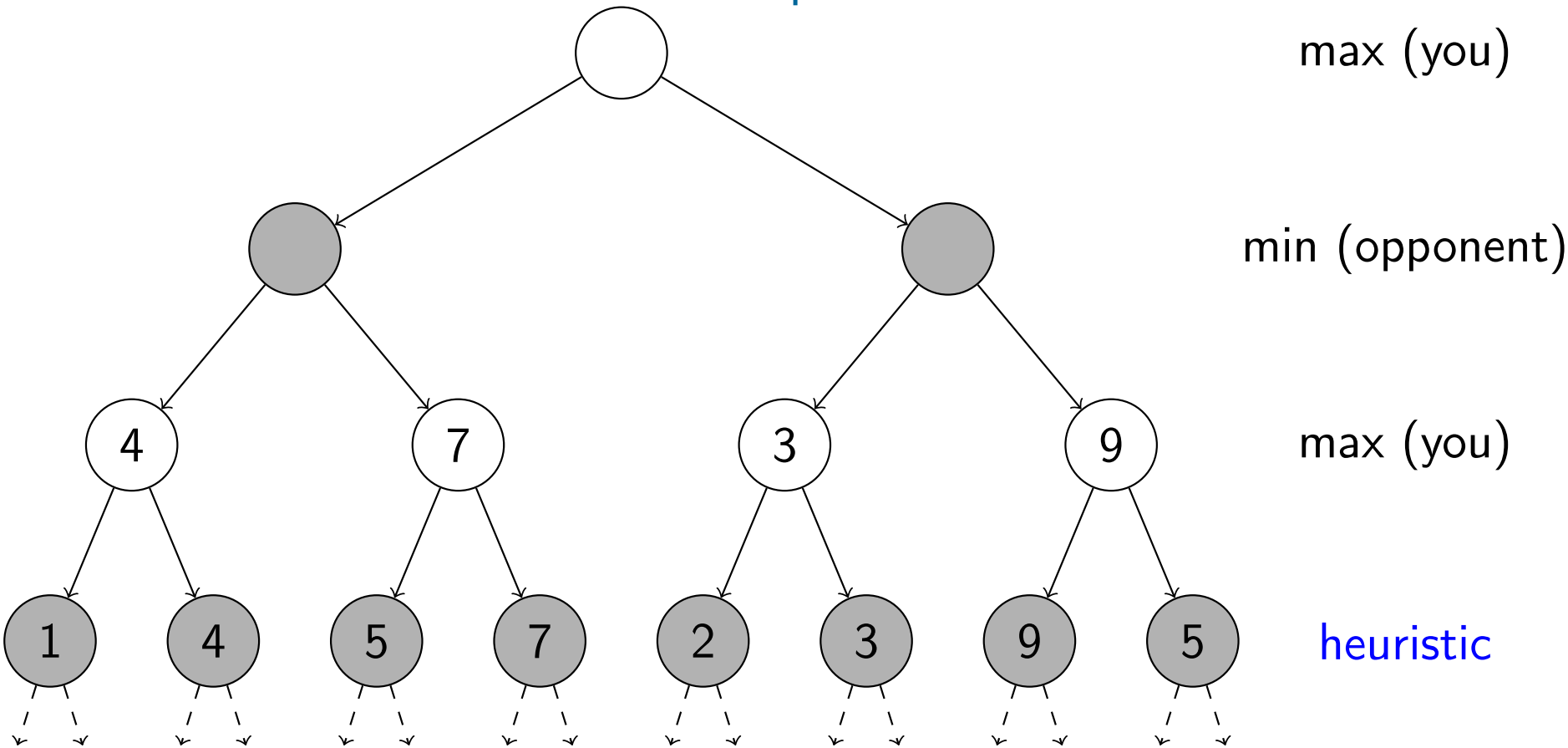
You can construct the tree of possibilities



If the tree is two big, you stop at depth D and use a heuristic.

- You can backtrack with the min-max algorithm.

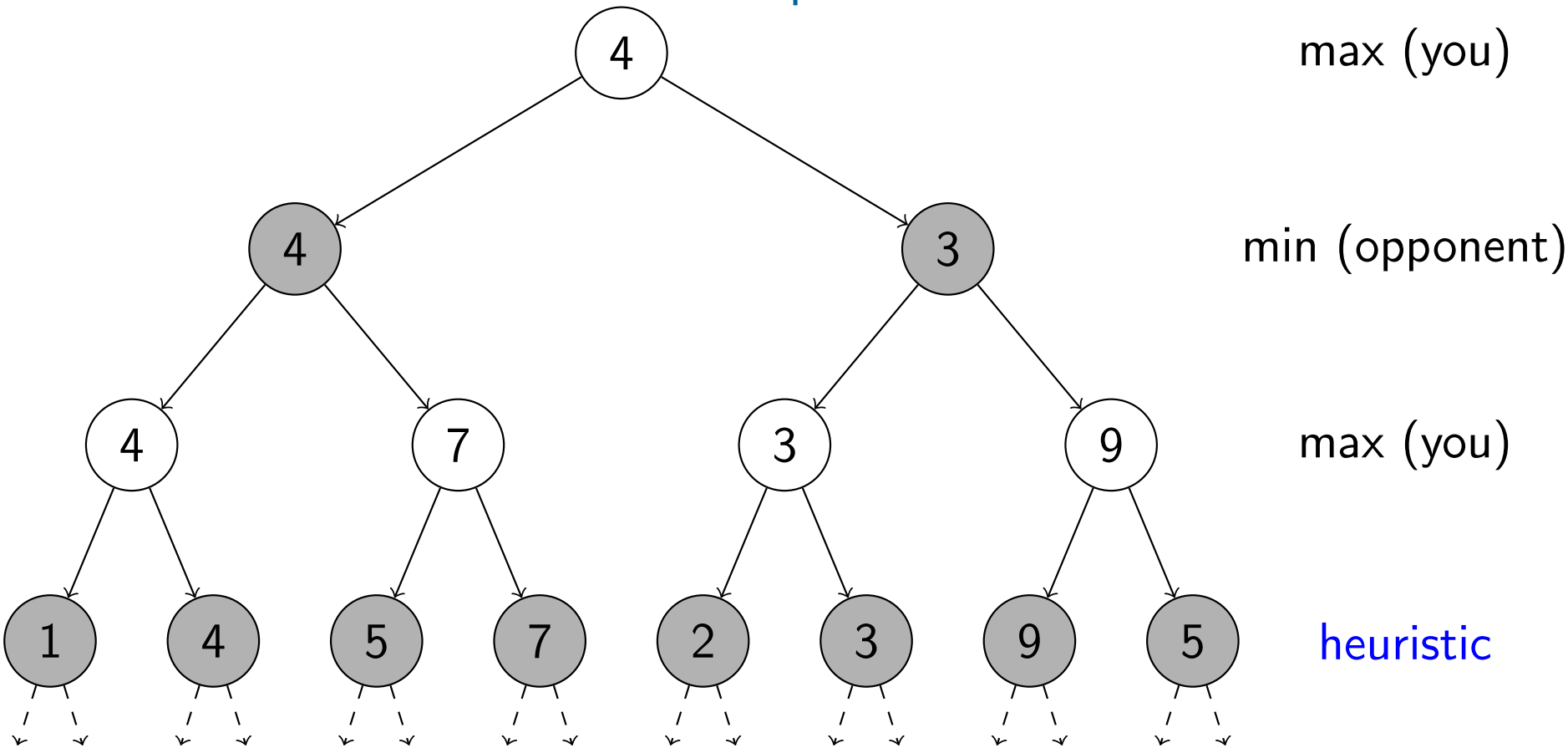
You can construct the tree of possibilities



If the tree is two big, you stop at depth D and use a heuristic.

- You can backtrack with the min-max algorithm.

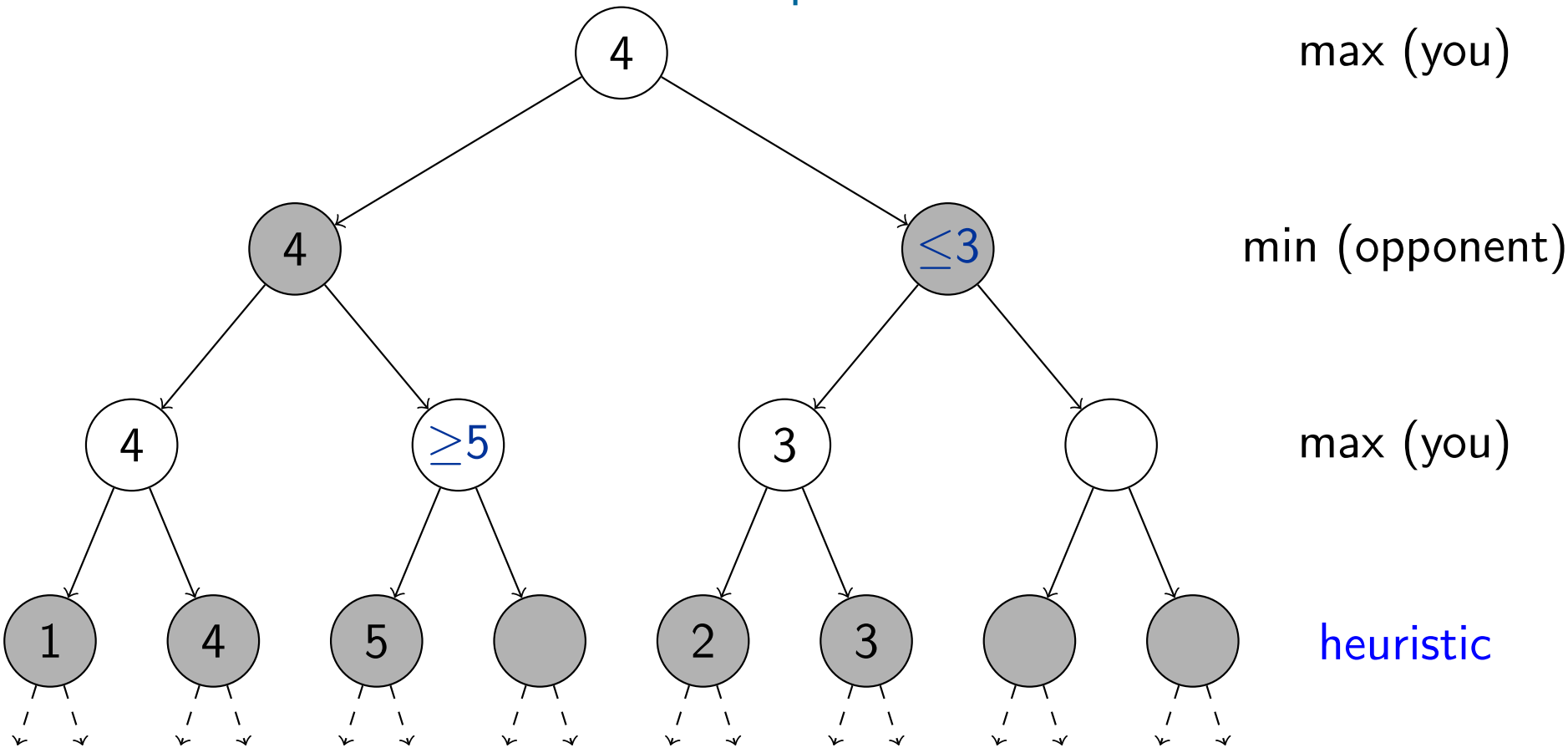
You can construct the tree of possibilities



If the tree is two big, you stop at depth D and use a heuristic.

- You can backtrack with the min-max algorithm.

You can construct the tree of possibilities



If the tree is two big, you stop at depth D and use a heuristic.

- You can backtrack with the min-max algorithm.
- For optimization, you can use alpha-beta pruning.

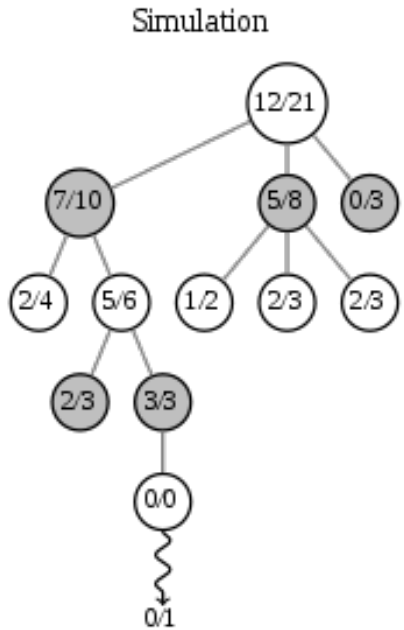
Table of contents

Min-max and alpha-beta perform well (ex: Chess)...

...but can be limited (ex: go)

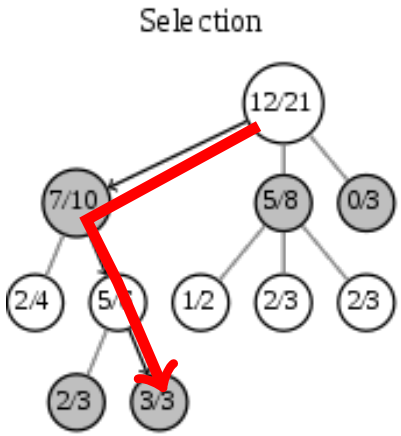
- Tree can still be very big (A^D)
- You need a good heuristic.
  - Result is only available at the end
- You might want to avoid the exploration of not promising parts.
  - For that you need a good heuristic.

MCTS (Monte Carlo Tree Search) uses simulation to conduct the tree search



- Simulate many games and compute how many were won.
- Explore carefully which actions were best.

MCTS (Monte Carlo Tree Search) uses simulation to conduct the tree search

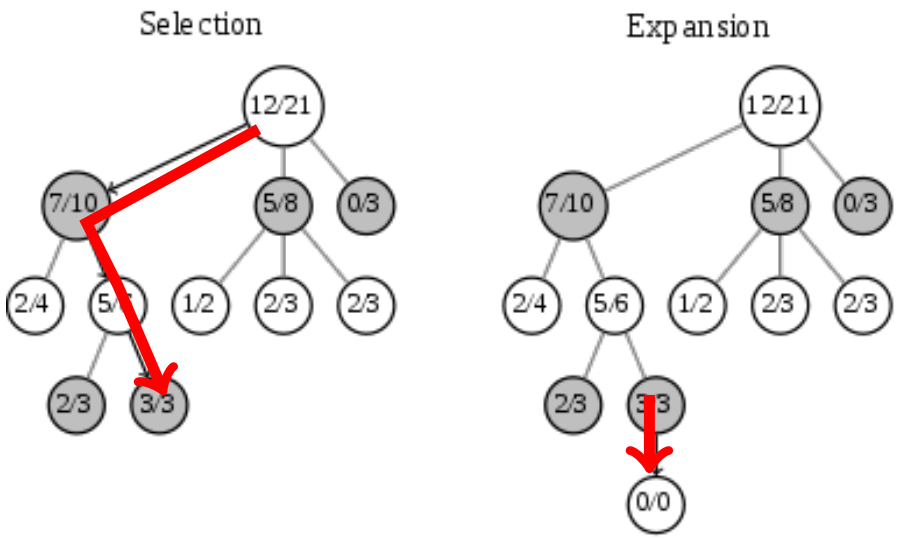


For each child, let $S(c)$ be the number of success and $N(c)$ be the number of time you played c, and $t = \sum_{c'} N(c')$.

- Explore $\arg\max_c \frac{S(c)}{N(c)} + 2\sqrt{\frac{\log t}{N(c)}}$.

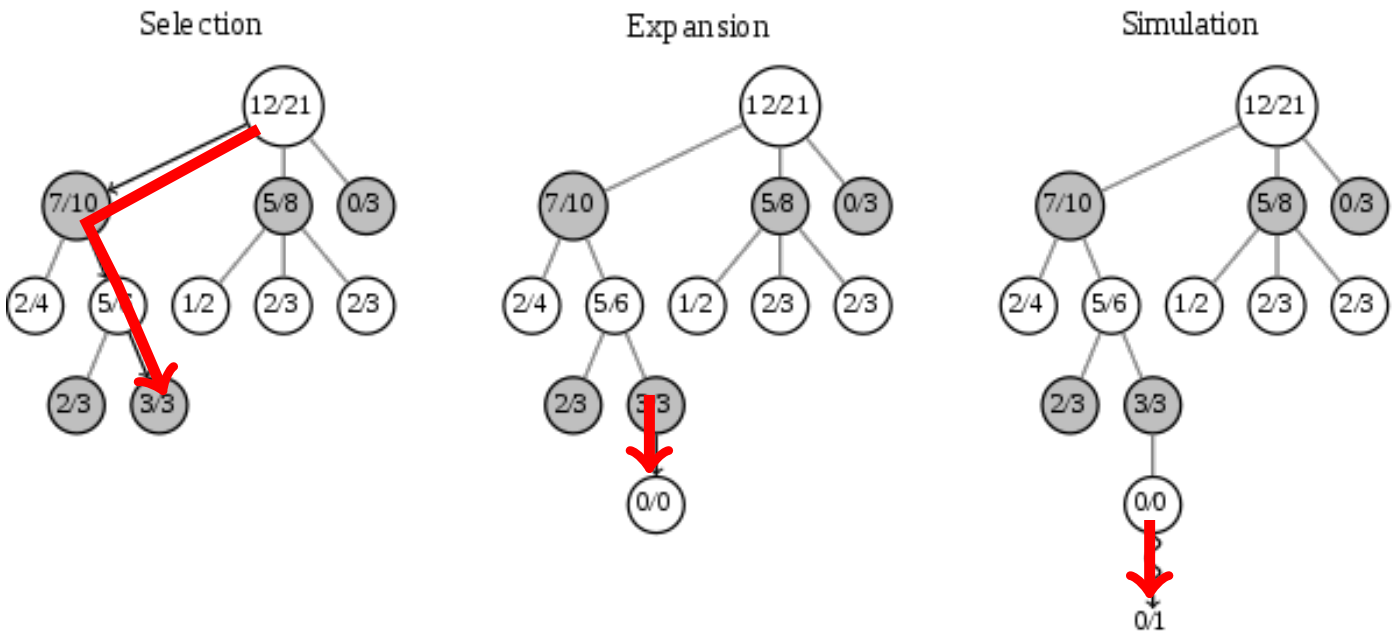
Open question: no guarantee with $\sqrt{\log t / N(c)}$. Is $\sqrt{t}/N(c)$ better?

MCTS (Monte Carlo Tree Search) uses simulation to conduct the tree search



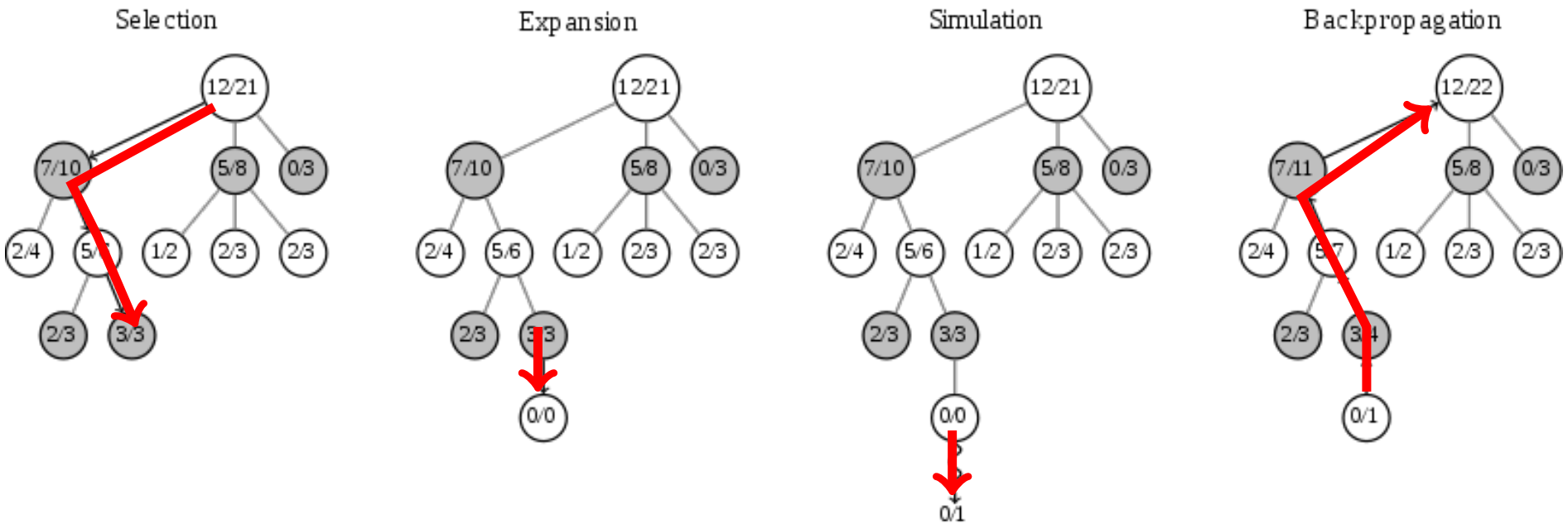
- Create one or multiple children of the leaf.

MCTS (Monte Carlo Tree Search) uses simulation to conduct the tree search



- Obtain a value of the node (e.g. rollout)

MCTS (Monte Carlo Tree Search) uses simulation to conduct the tree search



Rollout policy
(ex: random)

- Backpropagate to the root

MCTS algorithm

MCTS

1: **while** Some time is left **do**
2: Select a leaf node *#UCB-like*
3: Expand a leaf
4: Use rollout (or equivalent) to estimate the leaf *#random sampling*
5: Backpropagate to the root
6: **end while**
7: Return $\arg\max_{c \in \text{children(root)}} N(c)$ *#or* $S(c)/N(c)$.

Demo / exercice

Table of contents

Conclusion

Exploration v.s. exploitation is central in RL

- Bandits and regret help formalizing this idea.
- One important notion is the use of optimism to force exploration.
  - Bayesian sampling can also be used
- Theoretical tools guide practical implementations.