MFML: MDP and Reinforcement Learning

Nicolas Gast

November 30, 2023

Overview of the course

Up to now:

1. Supervised / unsupervised learning.
   - Data $\mapsto$ model
2. Online learning
   - Decision $\mapsto$ Data $\mapsto$ Decisions

Overview of the course

Up to now:

1. Supervised / unsupervised learning.
   - Data $\mapsto$ model
2. Online learning
   - Decision $\mapsto$ Data $\mapsto$ Decisions

End of the course:

3. Reinforcement learning
   - State $\mapsto$ Decision $\mapsto$ Reward and new state

What is Reinforcement Learning?

And why it differs from supervised or unsupervised learning

What is Reinforcement Learning?

And why it differs from supervised or unsupervised learning

- No i.i.d. dataset, but an environmeent.
- No labels, but observation of rewards.
- We design an agent, that maps states to actions.

What is Reinforcement Learning?

And why it differs from supervised or unsupervised learning

- No i.i.d. dataset, but an environmeent.
- No labels, but observation of rewards.
- We design an agent, that maps states to actions.

Challenges:

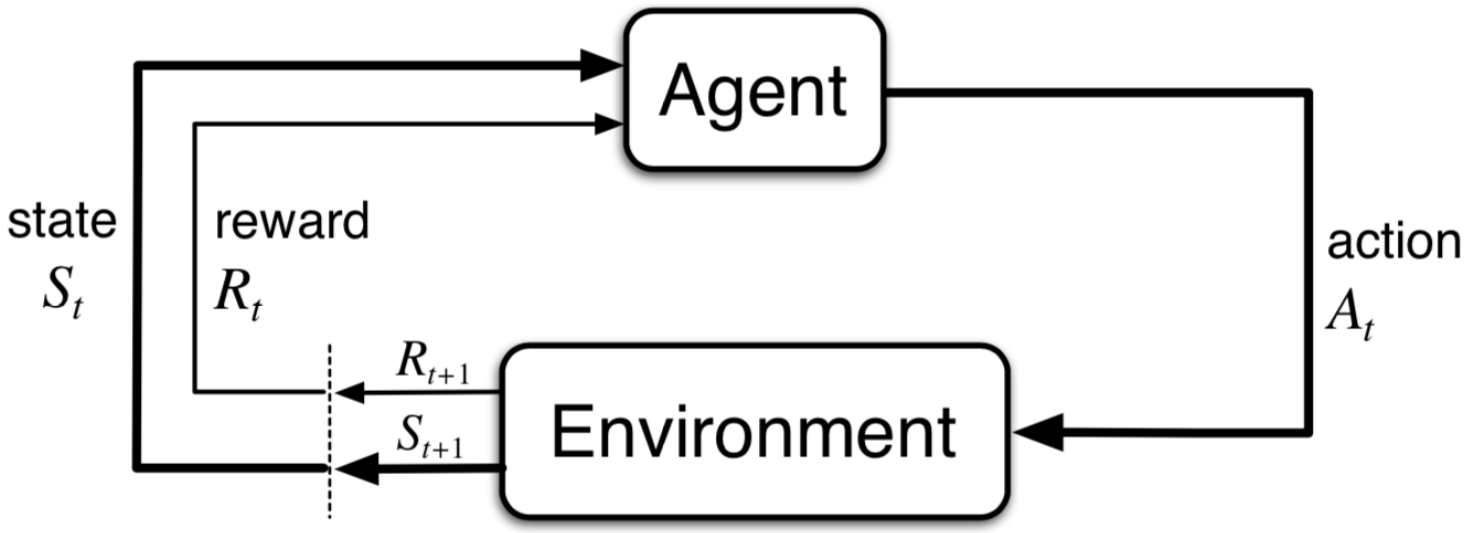
- Many possible states, actions
- Reward can be delayed, or sparse.

Applications

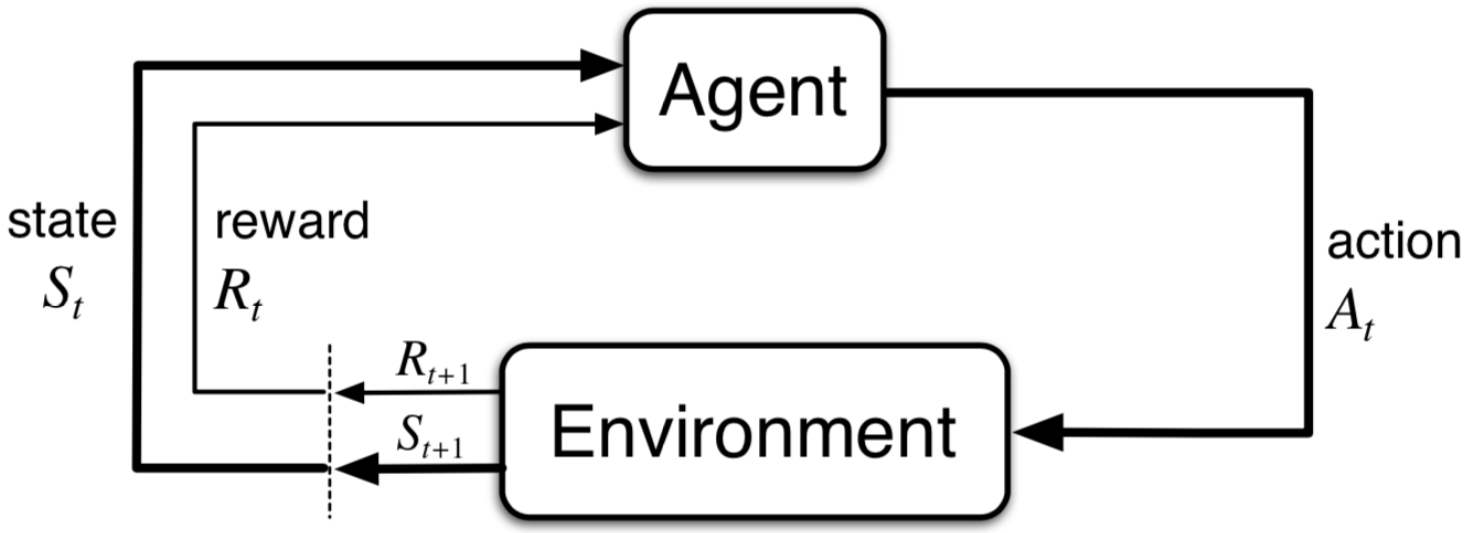
- Games (Go, Atari, StarCraft,...) ▸ StarCraft
- Auto-piloting vehicles ▸ Robots, ▸ Helicopter
- Supply management, energy ▸ data-center
- Trading, bidding ▸ Bidding
- Toy models ▸ AIGym
- ...

The number of application is increasing.

RL is about interacting with an environment



RL is about interacting with an environment



1. Get an observation of the state of the environment
2. Choose an action
3. Obtain a reward

You goal is to select actions to maximize the total reward.

Reward signal

At time t, we observe S_t, take action A_t, and obtain a reward R_{t+1}.

$$S_1, A_1 \qquad R_2, S_2, A_2 \qquad R_3, S_3, A_3 \qquad \dots \qquad R_T, S_T$$

Reward signal

At time t, we observe S_t, take action A_t, and obtain a reward R_{t+1}.

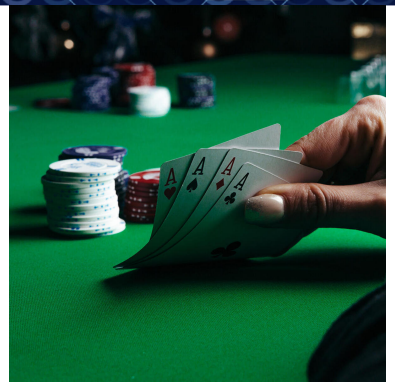
$$S_1, A_1 \qquad R_2, S_2, A_2 \qquad R_3, S_3, A_3 \qquad \dots \qquad R_T, S_T$$

Impact of actions can be delayed.

- On which actions does the reward depend?

Impact of actions can be weak

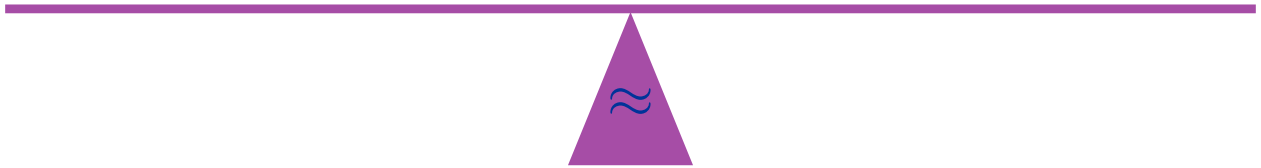
- or noisy



Objective of this course

Theory

Practice

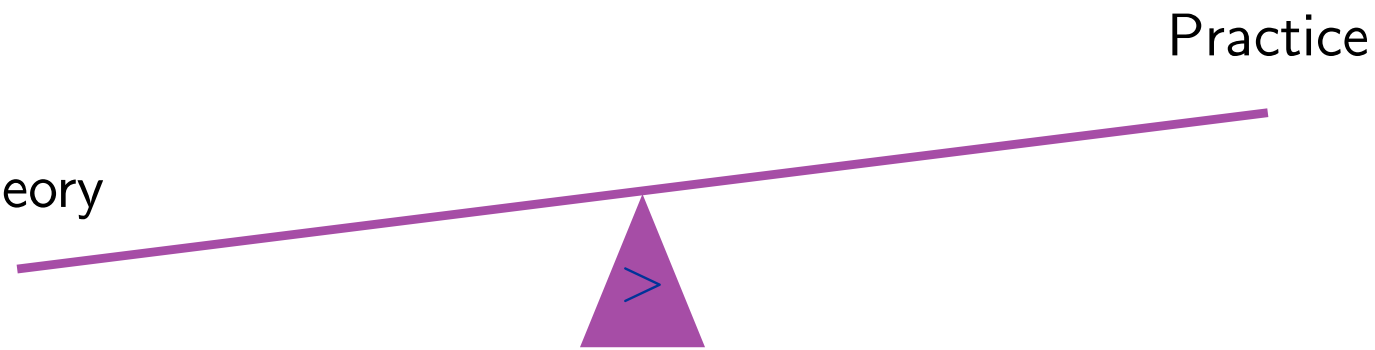


MDPs
Tabular RL
Algorithms guarantees
Regret minimization

Deep RL
Policy gradient

Exercises

Objective of this course



Personal work, a few advice

Week	December 5-6	Dec 12-13	Dec 19-20
Tuesday	Course (MDP)	Course	Course
Wednesday	Exos. (bring laptop)	Course	Course

No project. Final exam in January.

- Question what you learn
- Try to do some exercises.
  - Program, go deeper, ask follow-up questions.
- Ask questions during or after the course.

Personal work, a few advice

Week	December 5-6	Dec 12-13	Dec 19-20
Tuesday	Course (MDP)	Course	Course
Wednesday	Exos. (bring laptop)	Course	Course

No project. Final exam in January.

- Question what you learn
- Try to do some exercises.
  - Program, go deeper, ask follow-up questions.
- Ask questions during or after the course.

Read books (and/or research articles)

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

Content of the course

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)

Outline

1. Markov Decision Processes (MDPs)
   - Example and definition
   - Policies and Returns
   - Value Function and Bellman's Equation (finite horizon)
   - Infinite-horizon discounted problems
   - Conclusion
2. Tabular reinforcement learning
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)

Illustrative example: the wheel of fortune

You can draw a wheel indefinitely. After time t:

- If you draw, the wheel stops on $X_t \in \{1 \dots 10\}$ (uniformly).
- You can draw again or stop and earn X_t.

- You can draw the wheel up to $T = 10$ times. How do you play?

The environement lives in a "space"

- $\mathcal{S}$ – state space.
- $\mathcal{A}$ – action space.
- $\mathcal{R}$ – reward space.

Dynamics:

- (possibly random) evolution of states
- (possibly random) rewards

Markov decision processes

A MDPs is defined by:

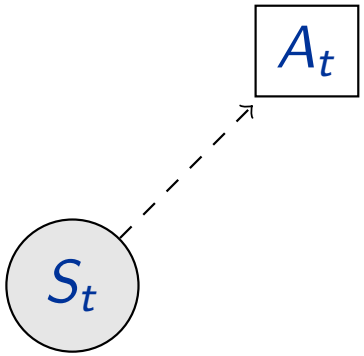
- $\mathcal{S}$ state space
- $\mathcal{A}$ action set
- Evolution is driven by Markovian transitions

$$\mathbb{P}(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a) = P(s', r \mid s, a).$$

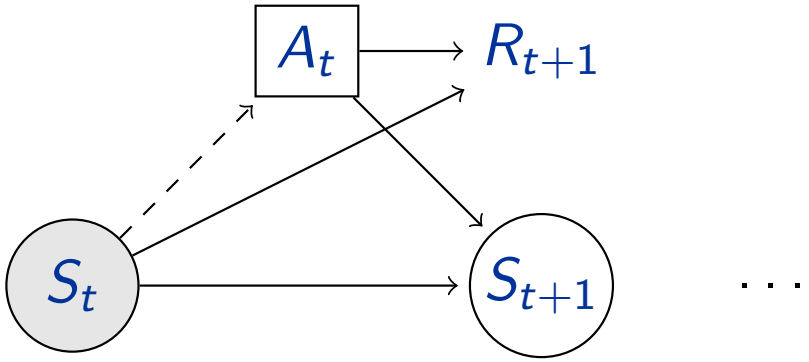
MDP = Markov chain + decisions

Most reinforcement learning problems can be framed as MDPs.

Graphical representation



Graphical representation



Some examples

- Wind production problem (tomorrow's exercise session)

A Wind turbine produces $(W_t)^3 \cos(\theta_t)$ where W_t is the wind speed and θ_t is your angle with respect to wind. Assume that:

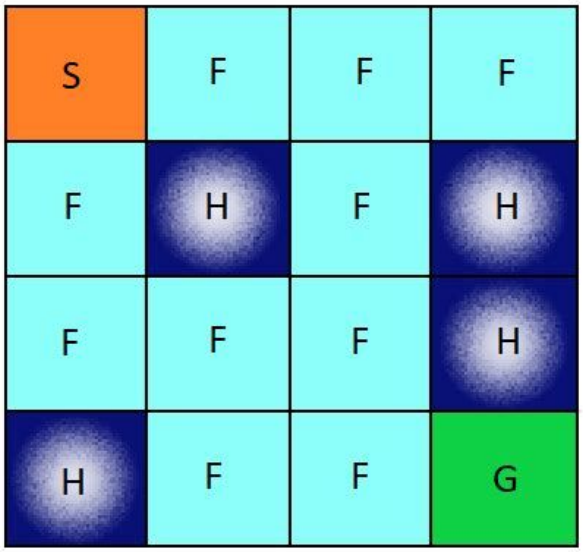
- Wind direction changes of ± 1 degree with probability $1/2$.
- Turning your turbine costs you $a > 0$.

Write the MDP for different models:

- Assuming that W_t is constant.
- Assuming that $W(t)$ evolve over time $W(t+1) = \min(1, \max(0, W(t) \pm b))$.
- Assuming that the direction in which the wind changes stays the same with probabilty 90%.

Some examples

- Wind production problem (tomorrow's exercise session)
- Frozen-lake ▸ Link (this is a gridworld example)



- Set space: $\{(0,0),\ldots,(3,3)\}$.
- Actions: $\{L,R,U,D\}$.
- Transitions: $1/3$ in right direction.
- Rewards: there are Holes and a Goal.
  - ★ Jumping to the goal gives you "1".

- There also some deterministic MDPs
  - Shortest paths probmems
  - Deterministic games (e.g., go, chess)

Table of contents

1. Markov Decision Processes (MDPs)
   - Example and definition
   - Policies and Returns
   - Value Function and Bellman's Equation (finite horizon)
   - Infinite-horizon discounted problems
   - Conclusion
2. Tabular reinforcement learning
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)

Policies

A (deterministic) policy specifies which action to take in a given state:

$$\pi : \mathcal{S} \to \mathcal{A}.$$

It indicates which action to take in a given state: $A_t = \pi(S_t)$. This defines the behavior of the agent.

Policies

A (deterministic) policy specifies which action to take in a given state:

$$\pi : \mathcal{S} \to \mathcal{A}.$$

It indicates which action to take in a given state: $A_t = \pi(S_t)$. This defines the behavior of the agent.

A stochastic policy specifies a distribution over actions:

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1].$$

The agent takes $A_t \sim \pi(\cdot|S_t)$.

Policies

A (deterministic) policy specifies which action to take in a given state:

$$\pi : \mathcal{S} \to \mathcal{A}.$$

It indicates which action to take in a given state: $A_t = \pi(S_t)$. This defines the behavior of the agent.

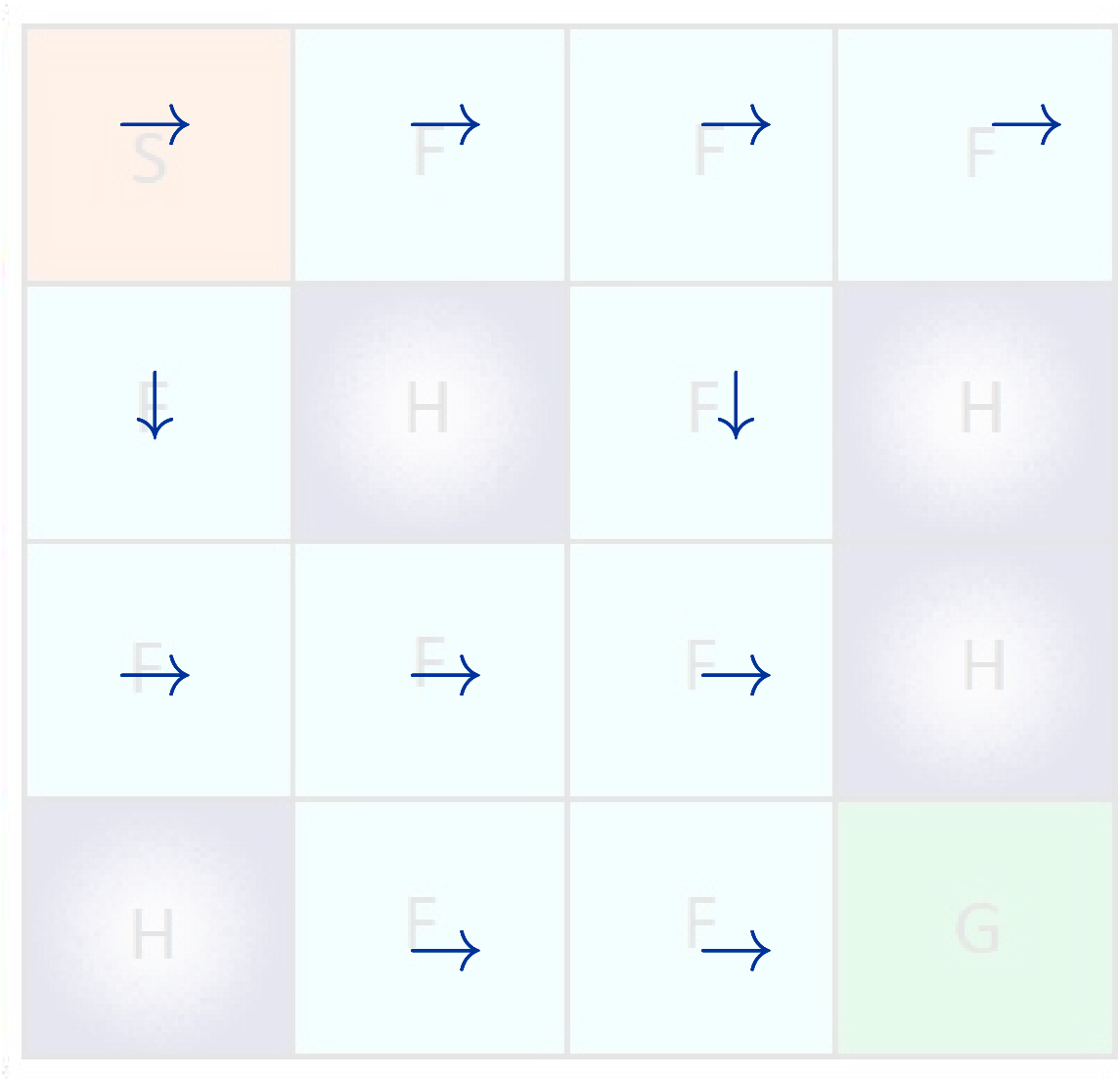
A stochastic policy specifies a distribution over actions:

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1].$$

The agent takes $A_t \sim \pi(\cdot|S_t)$.

Deterministic	Stochastic
Optimal in general	It forces exploration Useful in games / non-Markovian Differentiable

Example of a (deterministic) policy



Return of a policy

We want to compute the best policy... But what is the best policy?

Return of a policy

We want to compute the best policy... But what is the best policy?

Do we choose A_t to optimize:

- R_{t+1}? (no: too greedy)

Return of a policy

We want to compute the best policy... But what is the best policy?

Do we choose A_t to optimize:

- R_{t+1}? (no: too greedy)
- R_T? (only final reward?)

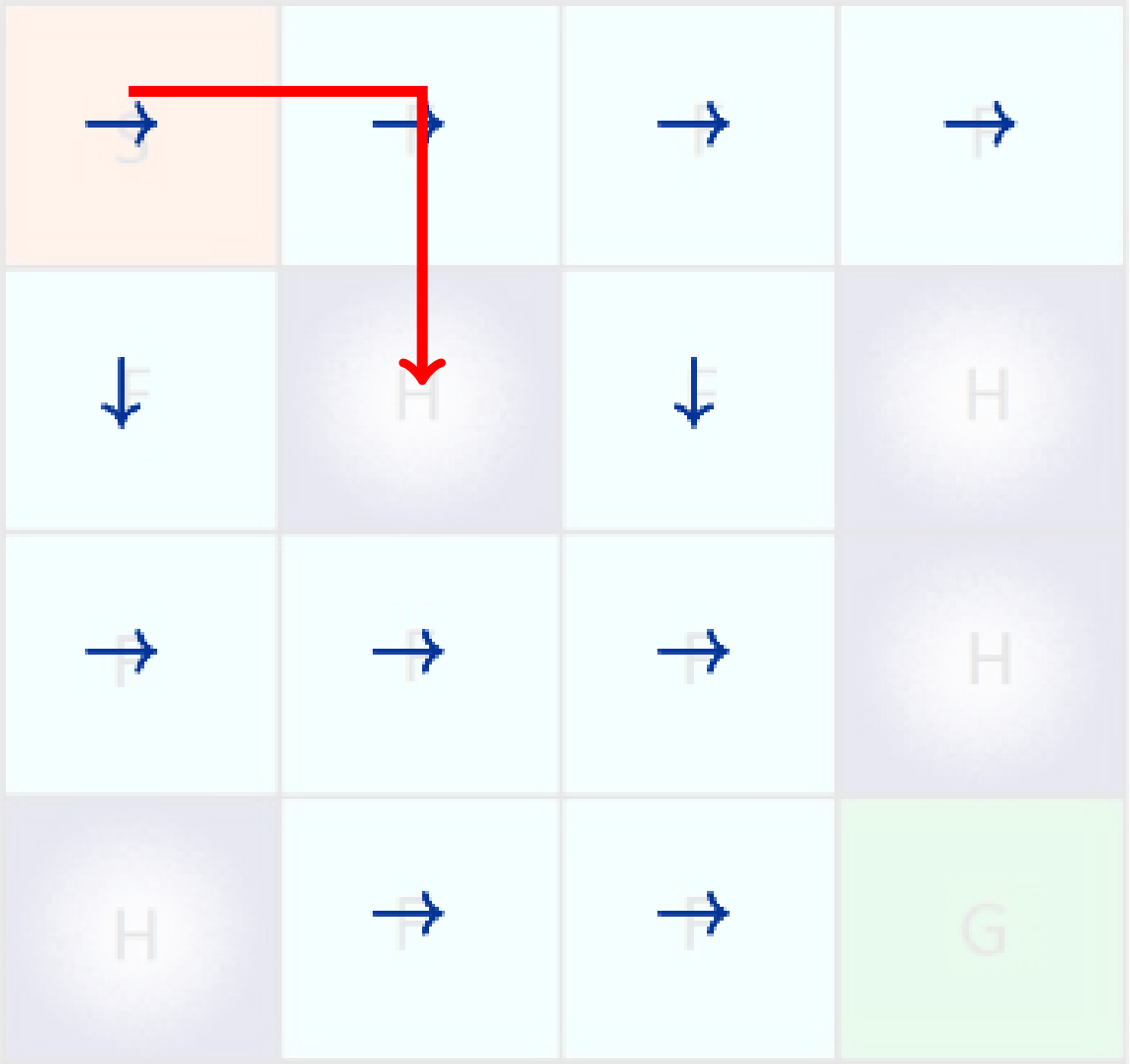
Return of a policy (finite horizon)

Sometimes, a problem has a known finite horizon T. In which case, the return (a.k.a. gain) at time t is:

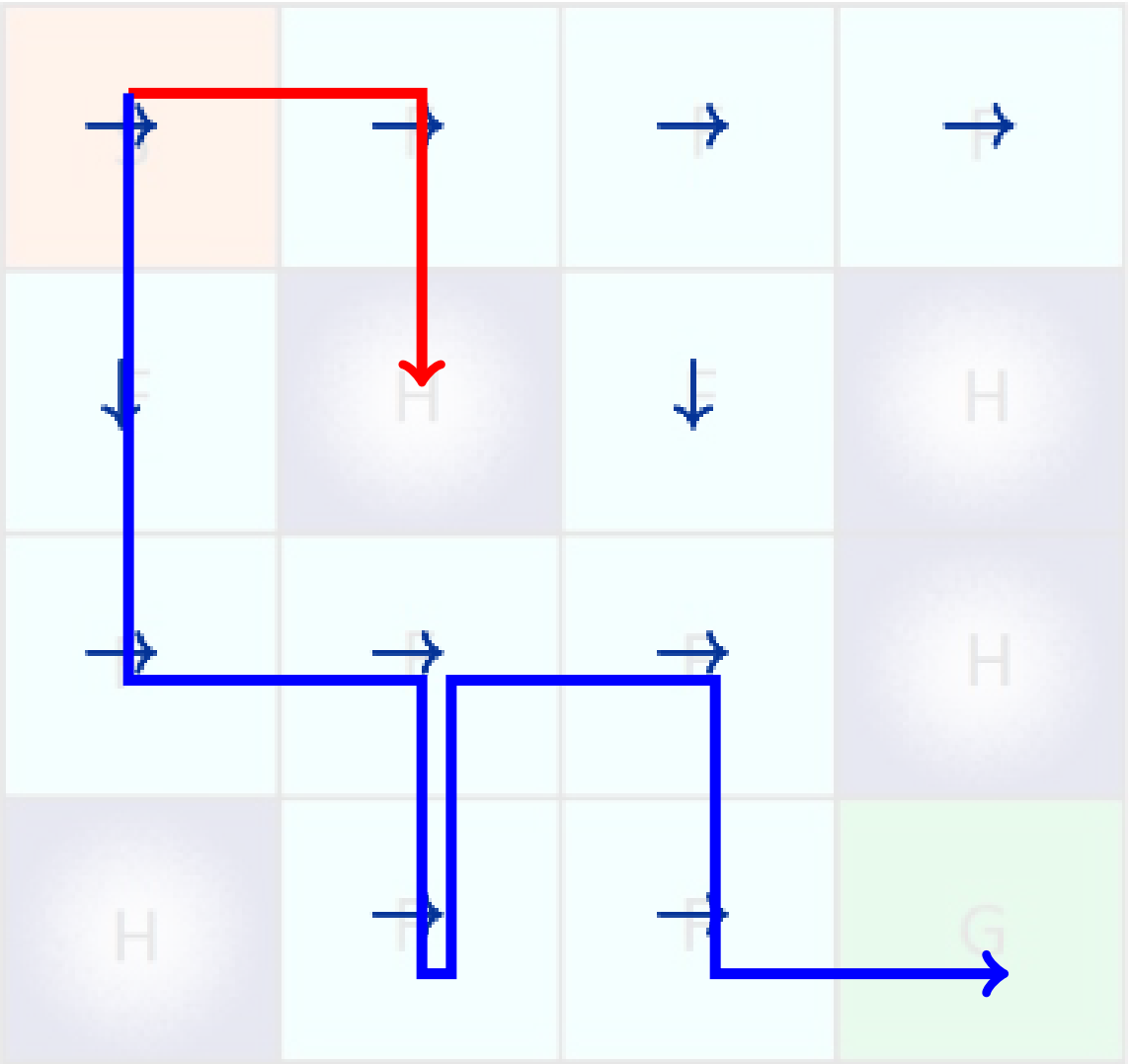
$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T.$$

The return is random.

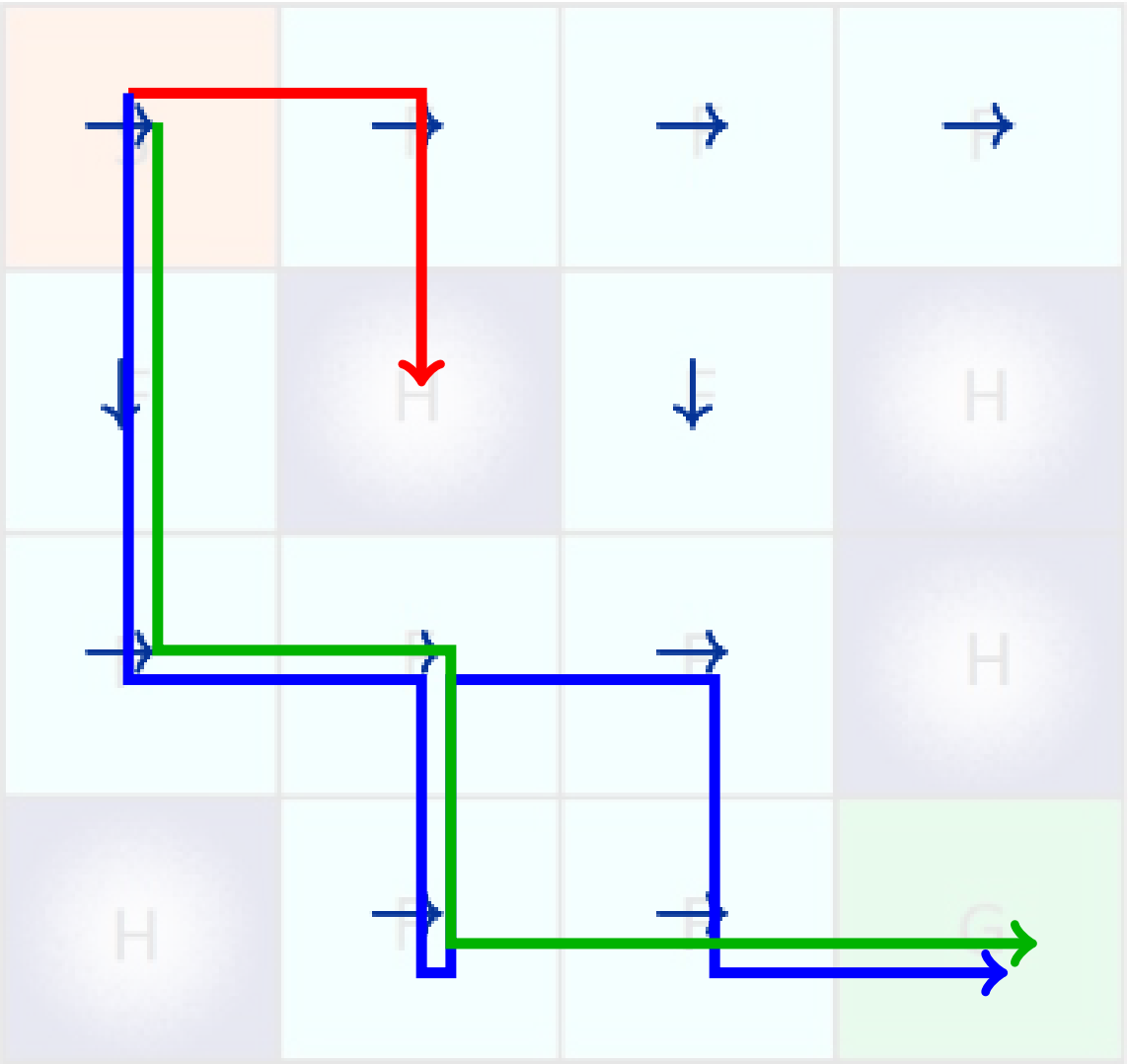
Return: example



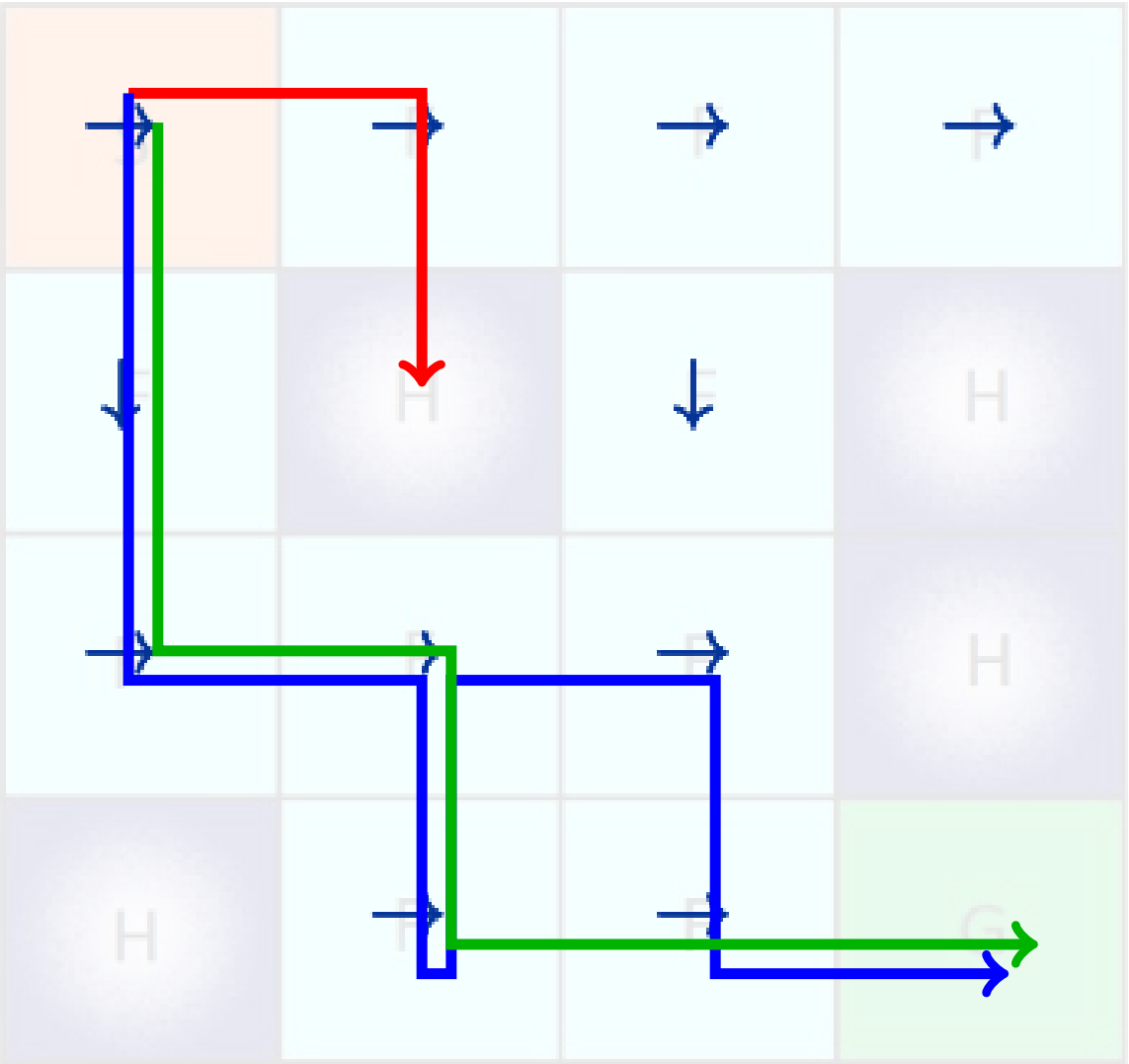
Return: example



Return: example



Return: example



Return(Red) = 0
Return(Green)=1
Return(Blue) = 1.

The return is random.

In practice, we will look at the expected return $\mathbb{E}[G_t]$.

Table of contents

1. Markov Decision Processes (MDPs)
   - Example and definition
   - Policies and Returns
   - Value Function and Bellman's Equation (finite horizon)
   - Infinite-horizon discounted problems
   - Conclusion
2. Tabular reinforcement learning
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)

Value function

The value function of a policy π is

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

where $\mathbb{E}^\pi\left[\cdot\right]$ means $\mathbb{E}\left[\cdot \mid A_{t+k} \sim \pi(S_{t+k}) \quad (k \geq 0)\right]$.

Value function

The value function of a policy π is

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

where $\mathbb{E}^\pi[\cdot]$ means $\mathbb{E}[\cdot \mid A_{t+k} \sim \pi(S_{t+k}) \quad (k \geq 0)]$.

It specifies the expected return. For each t, it is a vector of $|\mathcal{S}|$ values. If $\mathcal{S} = \{s_1 \dots s_4\}$

	s_1	s_2	s_3	s_4
V				

Bellman's Equation (policy evaluation, finite horizon)

We have $V_t^\pi(s) = \mathbb{E}^\pi\left[G_t \mid S_t = s\right]$ and

$$\begin{aligned} G_t &= R_{t+1} + R_{t+2} + \dots R_T \\ &= R_{t+1} + G_{t+1}. \end{aligned}$$

Hence:

$$V_t^\pi(s) =$$

where $\mathrm{r}(s,a) = \sum_{r'} r' p(r' \mid s,a)$.

Bellman's Equation (policy evaluation, finite horizon)

We have $V_t^\pi(s) = \mathbb{E}^\pi\left[G_t \mid S_t = s\right]$ and

$$\begin{aligned} G_t &= R_{t+1} + R_{t+2} + \dots R_T \\ &= R_{t+1} + G_{t+1}. \end{aligned}$$

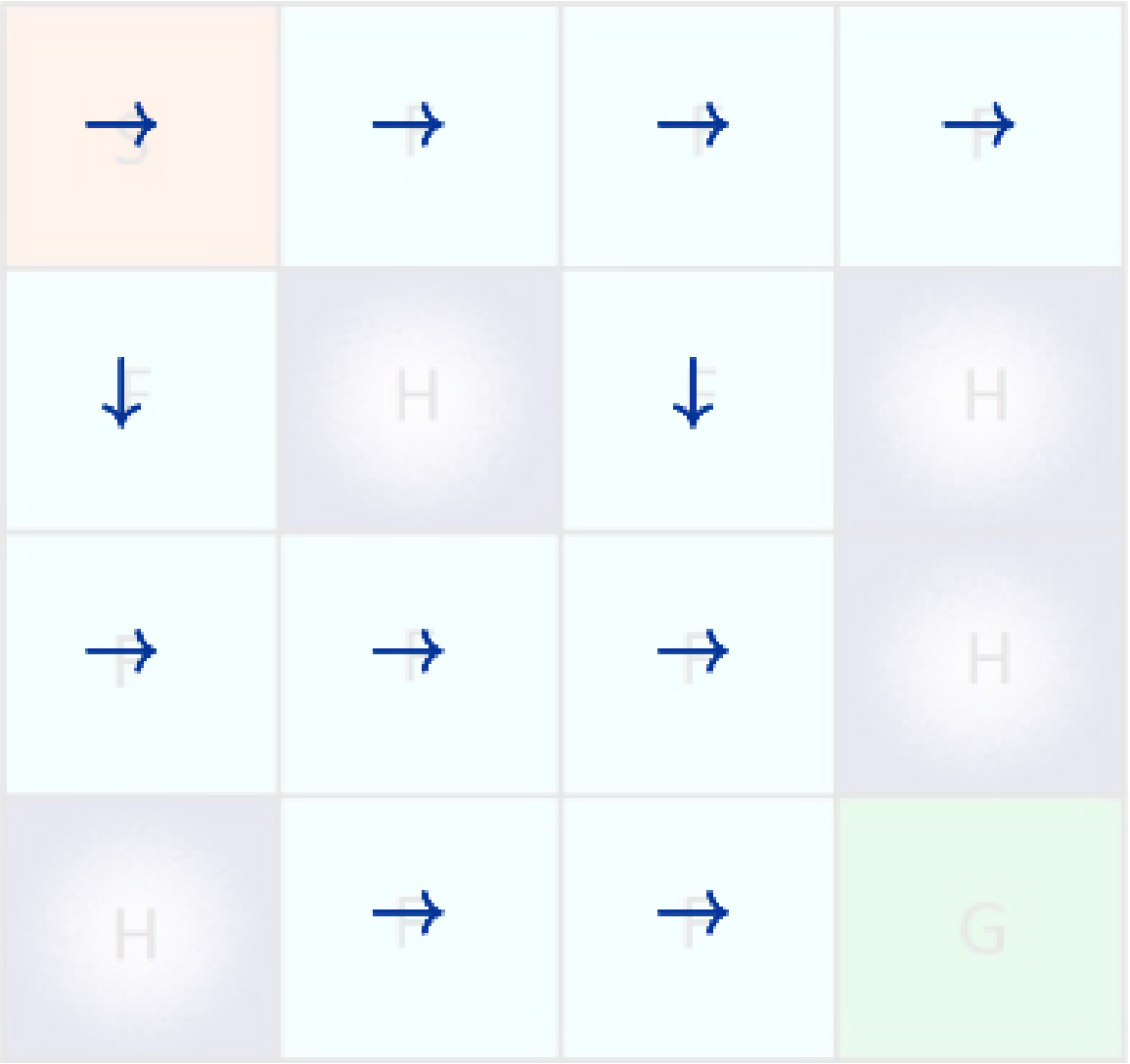
Hence:

$$\begin{aligned} V_t^\pi(s) &= \underbrace{\sum_{s',r'} (r + V^\pi(s'))p(s', r \mid s, a = \pi(s))}_{=Q_{t+1}^\pi(s,\pi(s))}. \\ &= \mathrm{r}(s, \pi(s)) + \sum_{s'} V_{t+1}^\pi(s')p(s' \mid s, a = \pi(s)), \end{aligned}$$

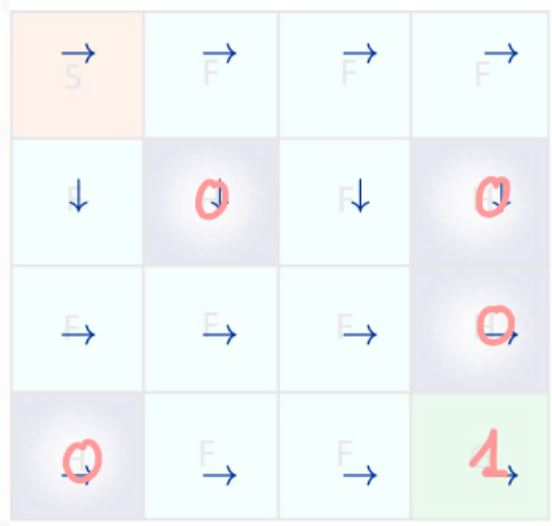
where $\mathrm{r}(s, a) = \sum_{r'} r' p(r' \mid s, a)$.

Algorithm: backward induction

Example : Finite-horizon Bellman's equation (evaluation)



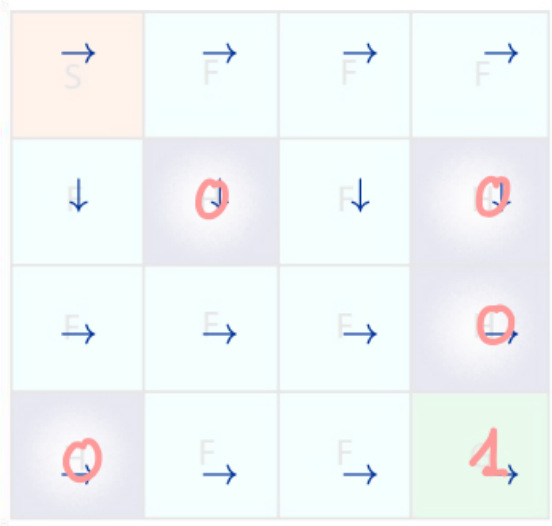
Example : Finite-horizon Bellman's equation (evaluation)



$t = 1$

$t = 2$

$t = 3$



$t = 4$

$t = 5$

$t = T = 6$

Action-Value function

The action-value function of a policy π is

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

Action-Value function

The action-value function of a policy π is

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

It is a table of $|\mathcal{S}| \times |\mathcal{A}|$ values. If $\mathcal{S} = \{s_1 \dots s_4\}$ and $\mathcal{A} = \{a_1, a_2\}$:

Q	a_1	a_2
s_1		
s_2		
s_3		
s_4		

From Q, we can define a greedy policy: $a_t = \arg\max_{a \in \mathcal{A}} Q(s_t, a)$.

Optimal policy

We denote by $V_t^*(s) = \max_\pi V_t^\pi(s)$ and $Q_t^*(s,a) = \max_\pi Q_t^\pi(s)$.
For a finite-horizon T, a policy is a function $\pi : \mathcal{S} \times \{1 \dots T\} \to \mathcal{A}$.

$$V_t^*(s) =$$

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

Optimal policy

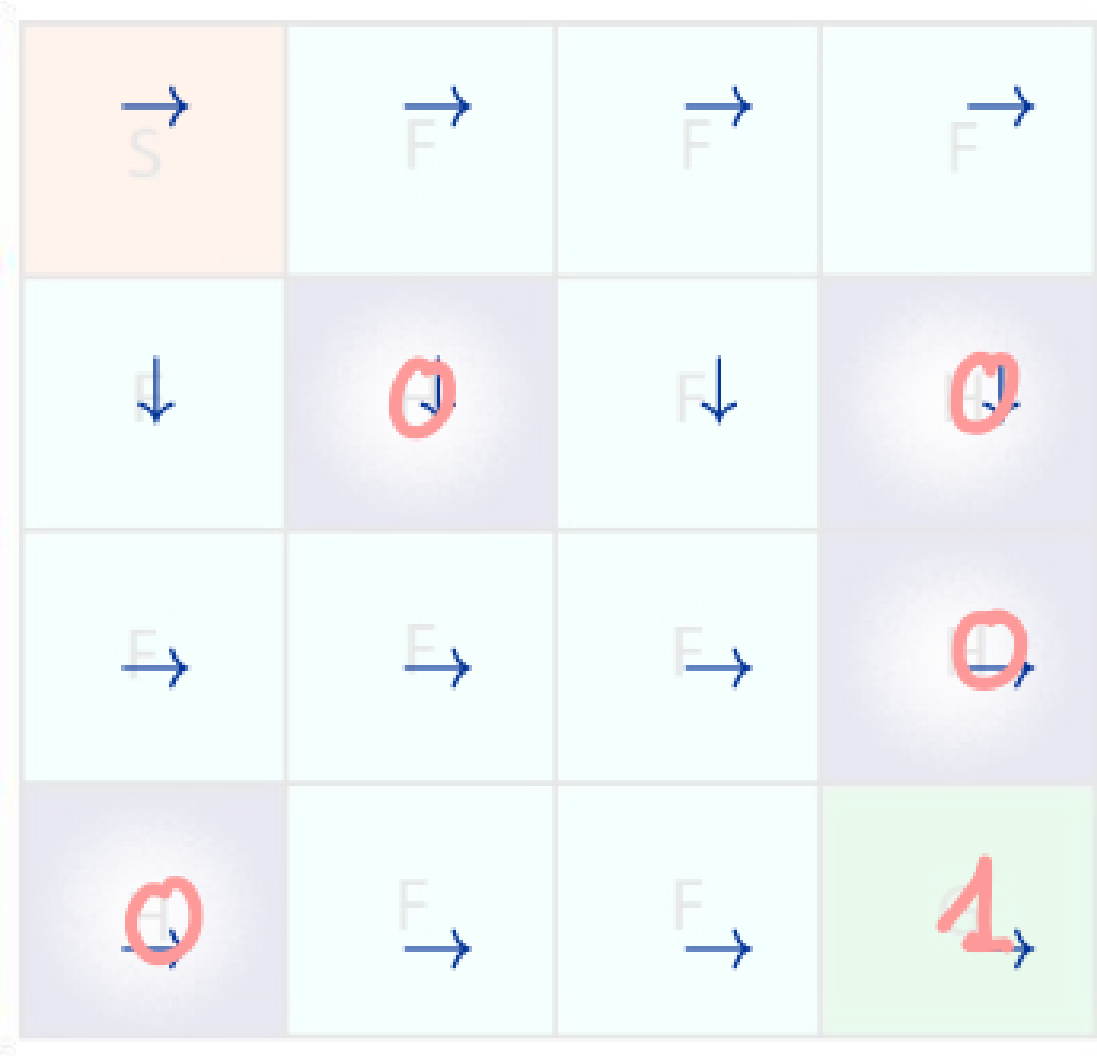
We denote by $V_t^*(s) = \max_\pi V_t^\pi(s)$ and $Q_t^*(s,a) = \max_\pi Q_t^\pi(s)$.
For a finite-horizon T, a policy is a function $\pi : \mathcal{S} \times \{1 \dots T\} \to \mathcal{A}$.

$$V_t^*(s) = \max_a Q_t(s,a)$$
$$Q_t^*(s,a) = \sum_{s',a} (r + V_{t+1}^*(s'))P(s', r \mid s, a)$$

Initial condition:

$$V_T^*(s) = \max_a r(s,a)$$

Optimal policy (finite horizon): illustration



$T = 6$

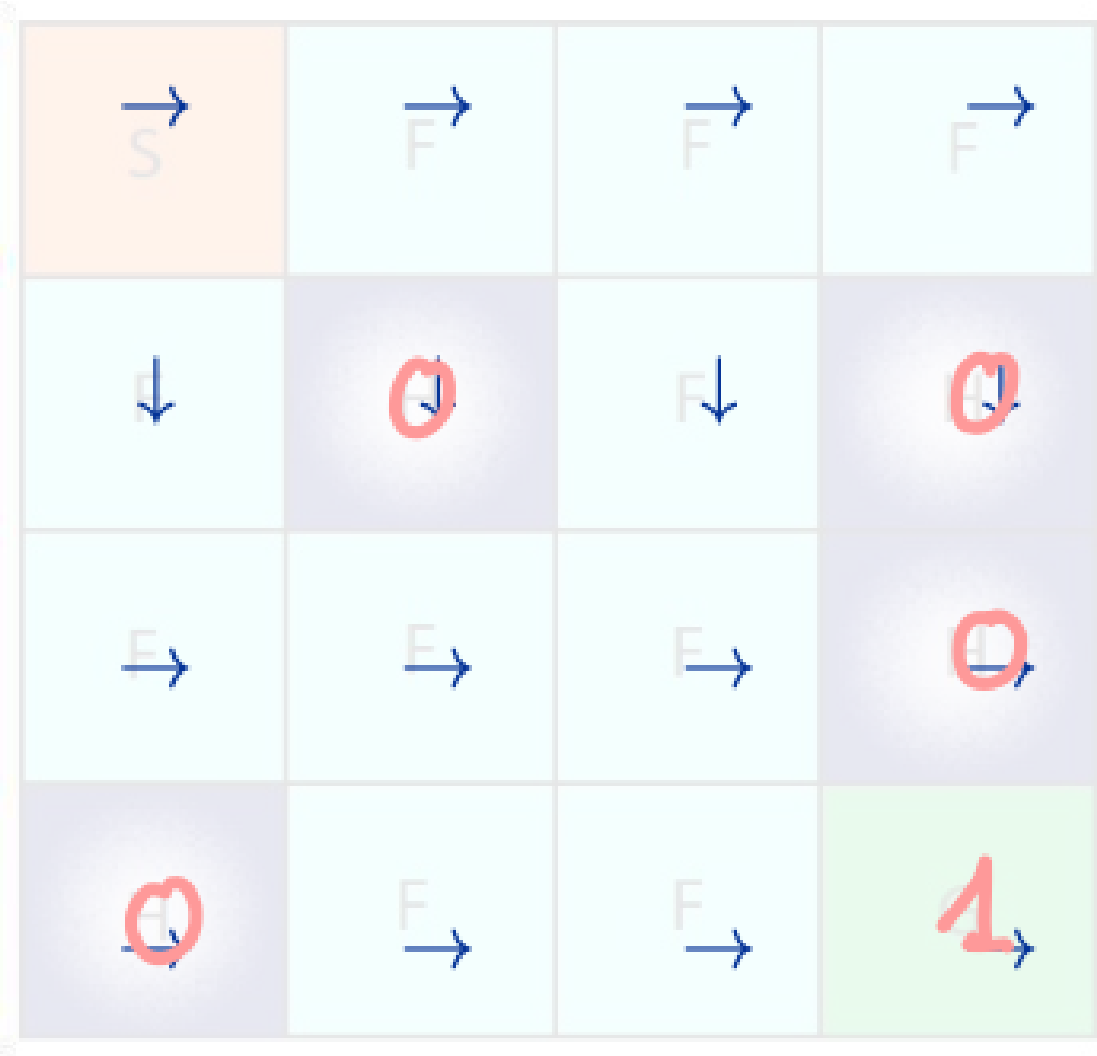
$Q_5^*((2,3), D) =$

$Q_5^*((2,3), R) =$

$Q_5^*((2,3), L) =$

$Q_5^*((2,3), U) =$

Optimal policy (finite horizon): illustration



$T = 6$

$Q_5^*((2,3), D) =$

$Q_5^*((2,3), R) =$

$Q_5^*((2,3), L) =$

$Q_5^*((2,3), U) =$

$Q_4^*((2,3), D) =$

$Q_4^*((2,3), R) =$

$Q_4^*((2,3), L) =$

$Q_4^*((2,3), U) =$

Optimal policy (finite horizon): illustration

```
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.33  1
L L L L
L L L L
L L L L
L L D L
```

```
0.00 0.00 0.01 0.00
0.00 0.00 0.05 0.00
0.02 0.11 0.21 0.00
0.00 0.26 0.57  1
L L L L
L L L L
D D L L
L R D L
```

```
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.11 0.00
0.00 0.11 0.44  1
L L L L
L L L L
L L L L
L D D L
```

```
0.00 0.00 0.02 0.00
0.01 0.00 0.07 0.00
0.05 0.16 0.24 0.00
0.00 0.31 0.61  1
L D L L
L L L L
D D L L
L R D L
```

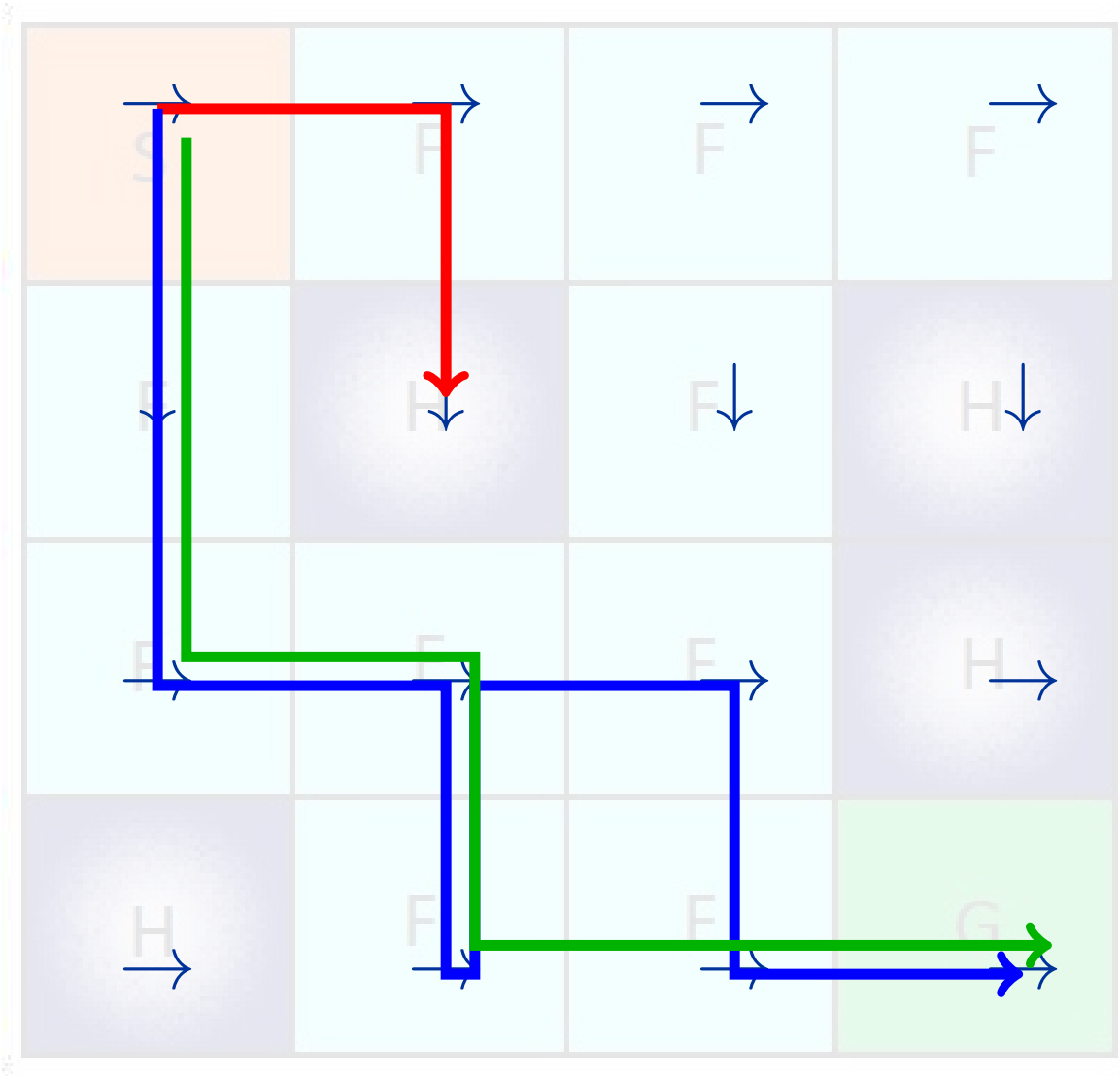
```
0.00 0.00 0.00 0.00
0.00 0.00 0.04 0.00
0.00 0.07 0.15 0.00
0.00 0.19 0.52  1
L L L L
L L L L
L D L L
L D D L
```

```
0.00 0.01 0.03 0.01
0.02 0.00 0.09 0.00
0.07 0.20 0.28 0.00
0.00 0.36 0.64  1
D R L U
L L L L
U D L L
L R D L
```

Table of contents

1. Markov Decision Processes (MDPs)
   - Example and definition
   - Policies and Returns
   - Value Function and Bellman's Equation (finite horizon)
   - Infinite-horizon discounted problems
   - Conclusion
2. Tabular reinforcement learning
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)

Which trajectory is best?



Return of a policy (discounted infinite horizon)

When T is not specified, it is common to look at the discounted return:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}, \end{aligned}$$

with $\gamma \in [0, 1)$.

- $\gamma = 0$: myopic (greedy).
- $\gamma = 1$: total reward.

Value of a policy and value iteration

Call T^π the operator that associates to a vector V the vector $T^\pi V$:

$$T^\pi V(s) = \mathrm{r}(s, \pi(s)) + \gamma \sum_{s'} V(s') p(s' \mid s, a = \pi(s))$$

The value of a policy is the unique vector V^π such that $T^\pi V^\pi = V^\pi$.

Value of a policy and value iteration

Call T^π the operator that associates to a vector V the vector $T^\pi V$:

$$T^\pi V(s) = \mathrm{r}(s, \pi(s)) + \gamma \sum_{s'} V(s') p(s' \mid s, a = \pi(s))$$

The value of a policy is the unique vector V^π such that $T^\pi V^\pi = V^\pi$.

Proof. T^π is contracting for the $\|v\| = \max_s |v(s)|$.

How to compute V^π

Two solutions:

1. Solve the linear system.
2. Initialize $V^{(0)} = 0$ and apply $V^{(k+1)} = T^\pi V^{(k)}$ until convergence.

The optimal policy

We denote by $V^*(s) = \max_\pi V^\pi(s)$ and $Q^*(s,a) = \max_\pi Q^\pi(s,a)$.

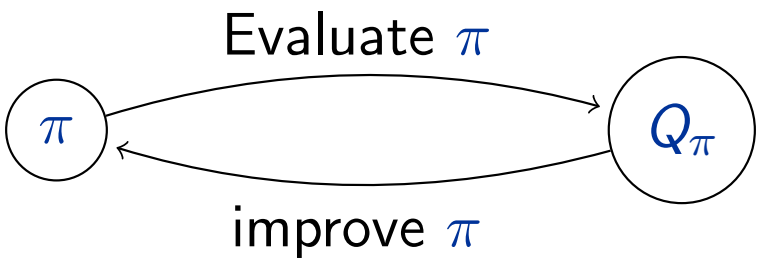
The optimal policy π^* is such that:

$$\pi^* = \arg\max_\pi V^\pi(s) \qquad \forall s \in \mathcal{S}$$

or equivalently:

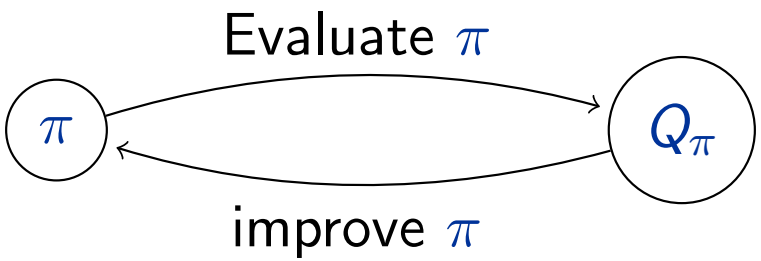
$$\pi^* = \arg\max_\pi Q^\pi(s,a) \qquad \forall s \in \mathcal{S}, s \in \mathcal{A}$$

Iterative solutions



If you know the transitions and reward: value iteration or policy iteration.

Iterative solutions

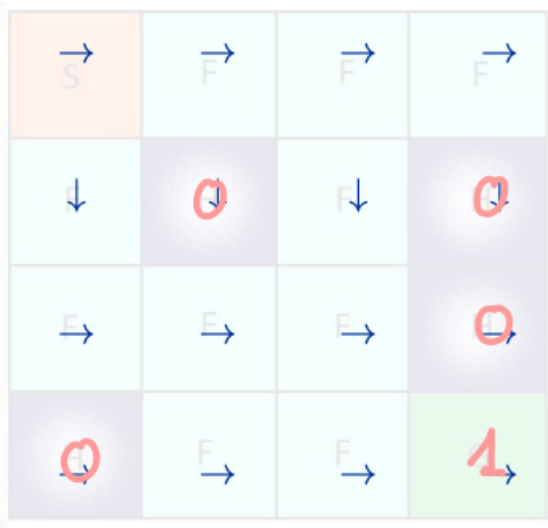


If you know the transitions and reward: value iteration or policy iteration.
Value iteration:

- Initialize V^0 (for instance to 0).
- For $k \geq 0$ and $s \in \mathcal{S}$, do:
  $$V^{k+1}(s) := \max_{a \in \mathcal{A}} \left(\mathrm{r}(s,a) + \gamma \sum_{s'} V^k(s') p(s' \mid s,a) \right)$$

"<u>Theorem</u>": If $\gamma < 1$, then $V^k - V^* = O(\gamma^k)$.

Illustration



```
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00  1
L L L L
L L L L
L L L L
L L L L
```

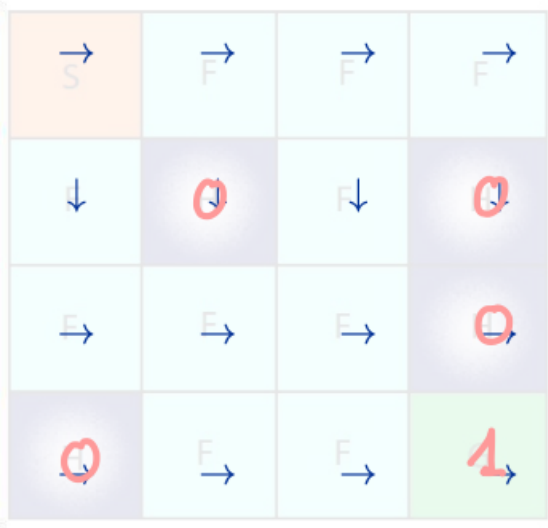
$\gamma = 0.8$

Iteration $k = 0$

```
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00  1
L L L L
L L L L
L L L L
L L L L
```

$\gamma = 0.9$

Illustration



```
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.33  1
L L L L
L L L L
L L L L
L L D L
```

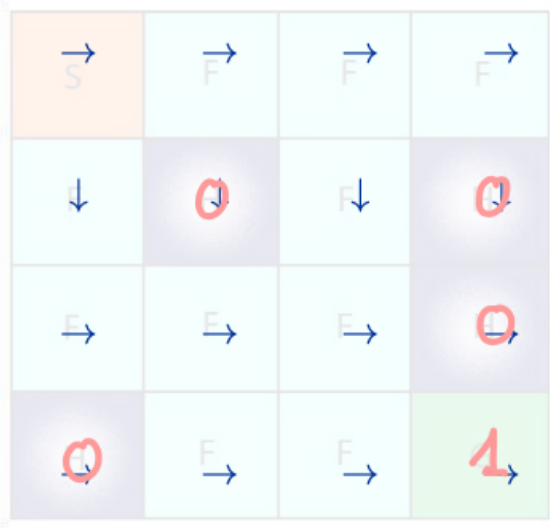
$\gamma = 0.8$

Iteration $k = 1$

```
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.33  1
L L L L
L L L L
L L L L
L L D L
```

$\gamma = 0.9$

Illustration



```
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.09 0.00
0.00 0.09 0.42  1
L L L L
L L L L
L L L L
L D D L
```

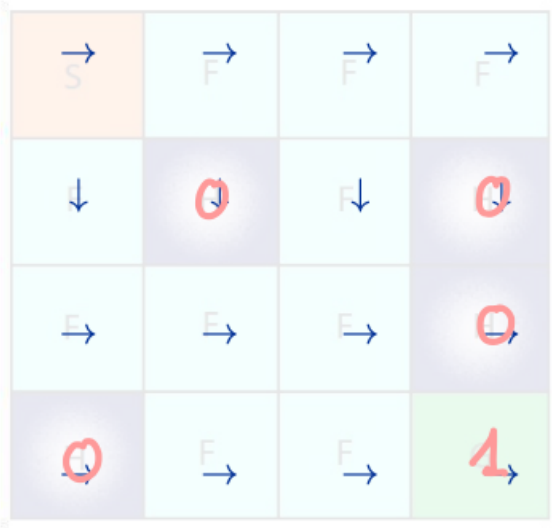
$\gamma = 0.8$

```
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.10 0.00
0.00 0.10 0.43  1
L L L L
L L L L
L L L L
L D D L
```

$\gamma = 0.9$

Iteration $k = 2$

Illustration



0.00	0.00	0.00	0.00
0.00	0.00	0.02	0.00
0.00	0.05	0.11	0.00
0.00	0.14	0.47	1
L	L	L	L
L	L	L	L
L	D	L	L
L	D	D	L

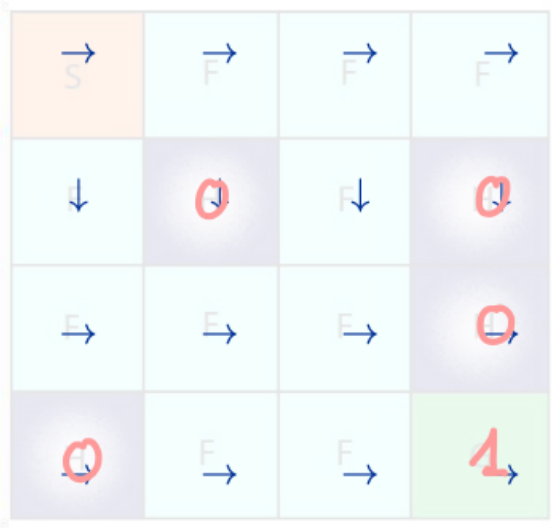
$\gamma = 0.8$

0.00	0.00	0.00	0.00
0.00	0.00	0.03	0.00
0.00	0.06	0.13	0.00
0.00	0.16	0.49	1
L	L	L	L
L	L	L	L
L	D	L	L
L	D	D	L

$\gamma = 0.9$

Iteration $k = 3$

Illustration



0.00	0.00	0.01	0.00
0.01	0.00	0.04	0.00
0.03	0.10	0.17	0.00
0.00	0.21	0.52	1
D	R	L	U
L	L	L	L
U	D	L	L
L	R	D	L

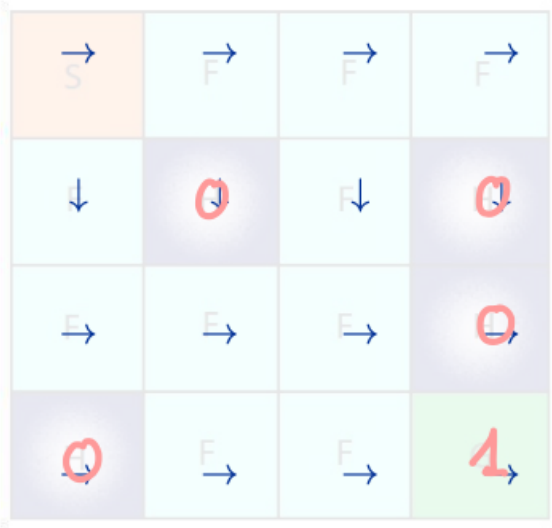
$\gamma = 0.8$

Iteration $k = 4$

0.00	0.01	0.02	0.01
0.01	0.00	0.06	0.00
0.05	0.14	0.22	0.00
0.00	0.28	0.57	1
D	R	L	U
L	L	L	L
U	D	L	L
L	R	D	L

$\gamma = 0.9$

Illustration



0.02	0.02	0.03	0.02
0.03	0.00	0.06	0.00
0.06	0.13	0.20	0.00
0.00	0.25	0.54	1
D	U	R	U
L	L	L	L
U	D	L	L
L	R	D	L

$\gamma = 0.8$

0.07	0.06	0.07	0.06
0.09	0.00	0.11	0.00
0.15	0.25	0.30	0.00
0.00	0.38	0.64	1
L	U	L	U
L	L	L	L
U	D	L	L
L	R	D	L

$\gamma = 0.9$

Iteration $k = 90$

Policy iteration

Policy iteration:

- Initialize π^0 (to some random value).
- For $k \geq 0$:

  Compute Q^{π^k} (=linear system)

  For all $a \in \mathcal{A}$: $\pi^{k+1}(s) := \arg\max_{a \in \mathcal{A}} Q^{\pi^k}(s, a)$.

"<u>Theorem</u>": If $\gamma < 1$, then after a <u>finite</u> number of iterations: $V^k = V^*$.

Exercise: the wheel of fortune

You can draw a wheel indefinitely. After time t:

- If you draw, the wheel stops on $X_t \in \{1 \dots 10\}$ (uniformly). You earn X_t.
- You can draw again or keep $X_{t+1} := X_t$.

How do you play knowing that you want to maximize your discounted reward: $\mathbb{E}\left[\sum_{t=1}^{\infty} \delta^t X_t\right]$ with $\gamma = 0.9$?

- Compare value iteration and policy iteration algorithms.

Table of contents

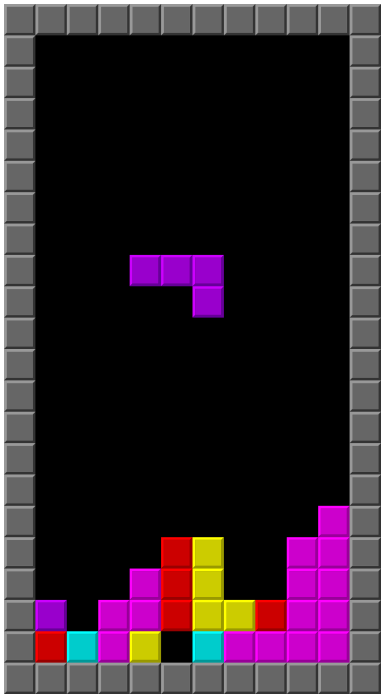
1. Markov Decision Processes (MDPs)
   - Example and definition
   - Policies and Returns
   - Value Function and Bellman's Equation (finite horizon)
   - Infinite-horizon discounted problems
   - Conclusion
2. Tabular reinforcement learning
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)

Important concepts

(to be filled by you!)

Challenges and future courses

- Learning transitions and reward? (course 2)
- If the state space is too large?
  - How do you store Q-values? (course 3)
- Exploration or exploitation (course 4)



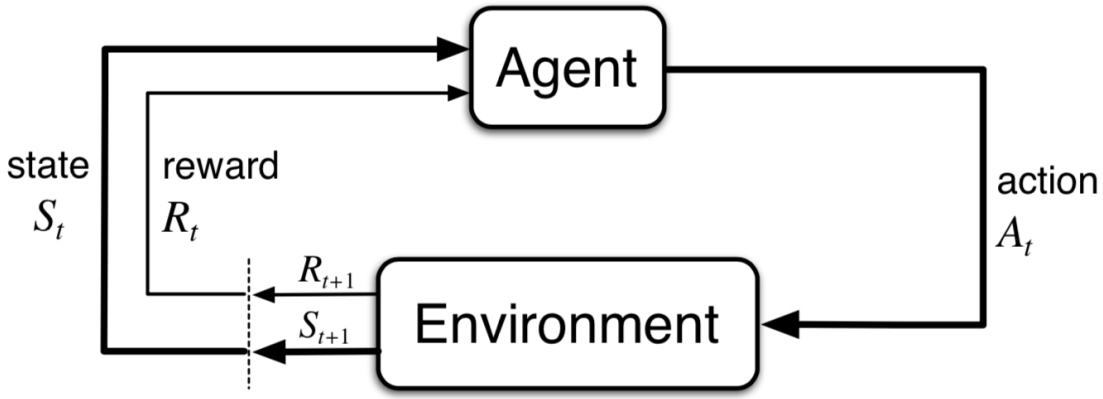
Some of the modern challenges

- Limited samples, convergence guarantees.
- Safety issues, explainable agents.
- Multi-agents (ex: competitive objectives?)
- Delayed or partial observations.

Outline

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
   - Monte-Carlo methods
   - Temporal difference
   - Q-learning and SARSA
   - Conclusion
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)

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

$$V^\pi(x) \qquad \text{and} \qquad Q^\pi(x,a).$$

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

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a)$$
$$Q^*(s,a) = \mathrm{r}(s, \pi(s)) + \gamma \sum_{s'} V^*(s'))p(s' \mid s, a)$$

Two problems:

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

Bellman's equation

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

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

~~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

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
   - Monte-Carlo methods
   - Temporal difference
   - Q-learning and SARSA
   - Conclusion
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)

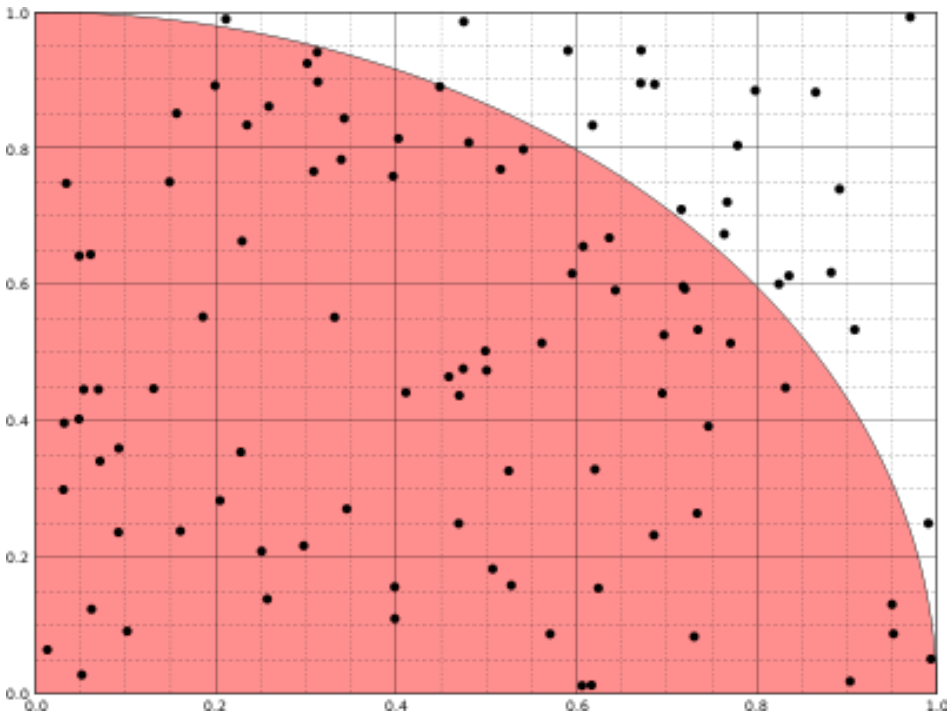
Monte Carlo methods

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

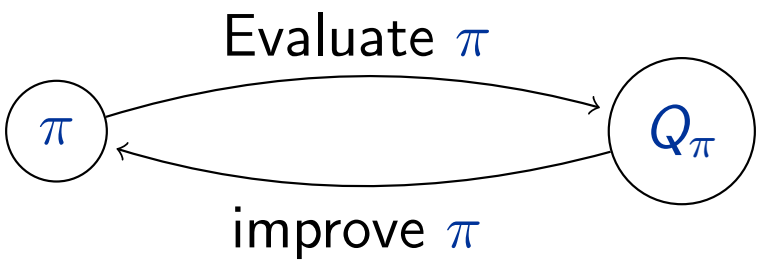
First-visit Monte-Carlo

```
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} + γ G_{t+1}.
        If S_t does not appear in S_0 ... 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 $Q(s, a)$.

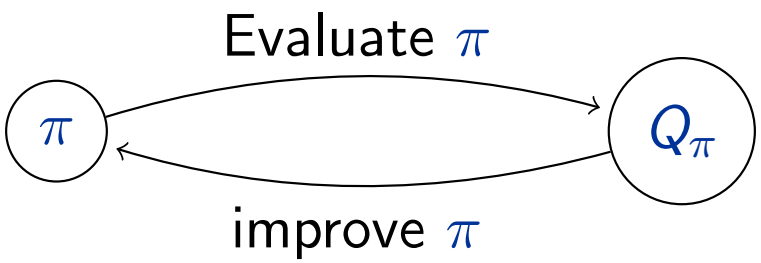


Recall: improve can be done by using greedy:

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

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

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
   - Monte-Carlo methods
   - Temporal difference
   - Q-learning and SARSA
   - Conclusion
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)

The temporal difference (TD) error

Bellman's equation states:

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

The temporal difference (TD) error

Bellman's equation states:

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

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.

TD learning algorithm

TD(0) for evaluating V^π

```
Initialize V(s) arbitrarily.
while True do
    Initialize S
    for While S' is not a terminal state do
        Sample A ~ π(S) and simulate a transition S', R ~ p(· | S, A).
        V(S) := V(S) + α_t(R + γV(S') − V(S)).
        S := S'
    end for
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 $\sum_t \beta_t^2(s) < \infty$.

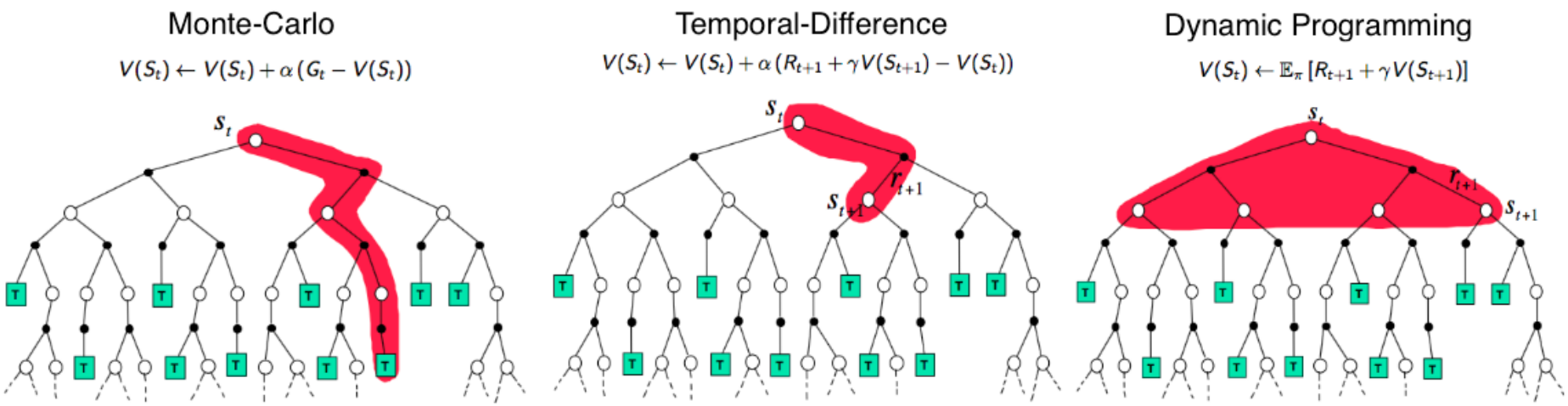
As T^π is contracting, Theorem 1 of On the convergence of stochastic iterative dynamic programming algorithms., Jaakkola, Jordan, Singh, NeurIPS 93 shows that this implies $\lim_{t\to\infty} V_t = V^\pi$ almost surely.

Relation between MC, TD and DP

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

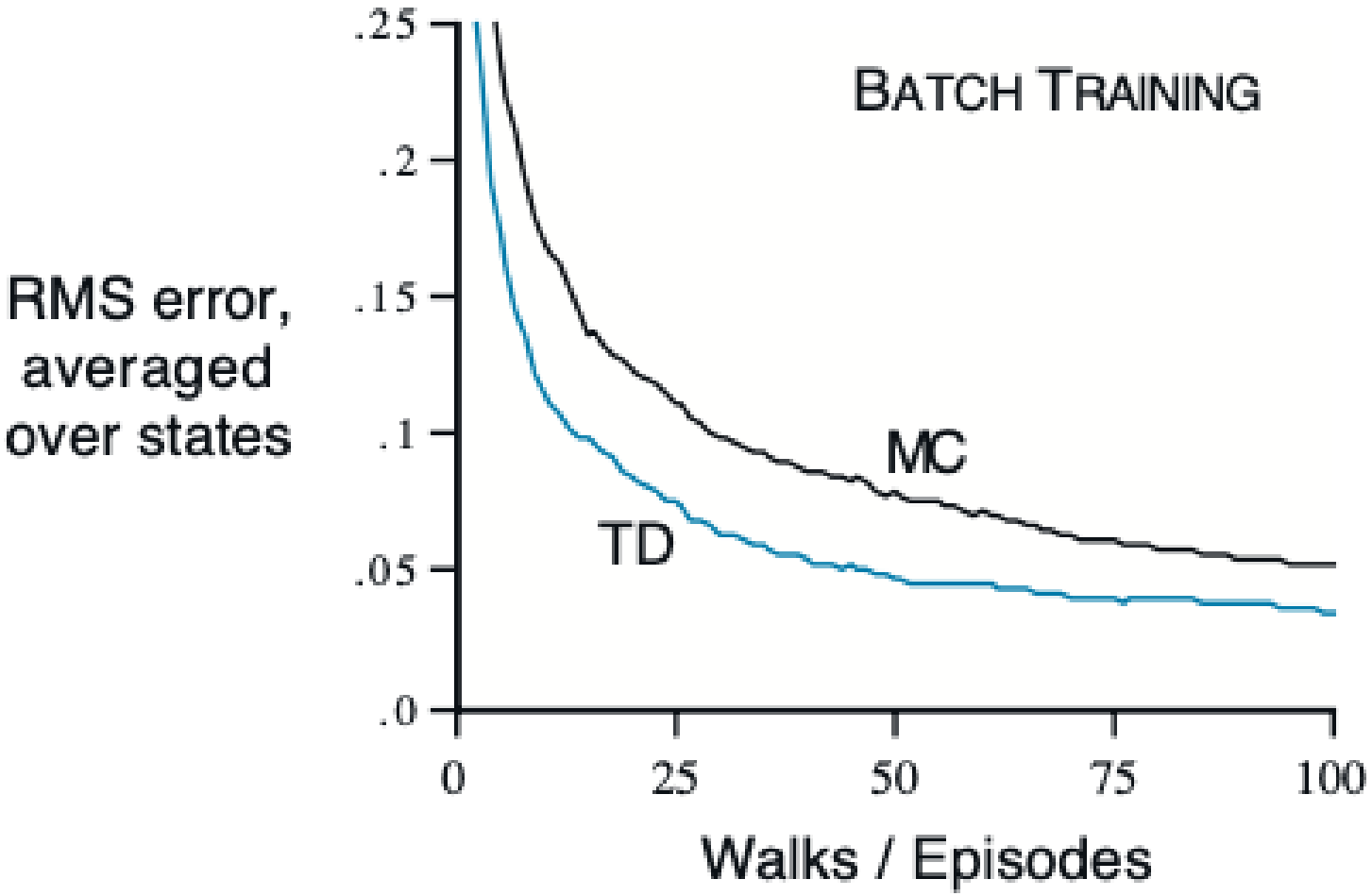
$$V(S_t) = \mathbb{E}\left[R_{t+1} + \gamma V(S_{t+1})\right] \qquad TD$$

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



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

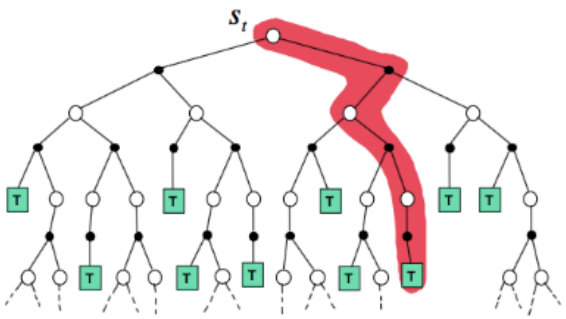
TD vs MC comparison: general case



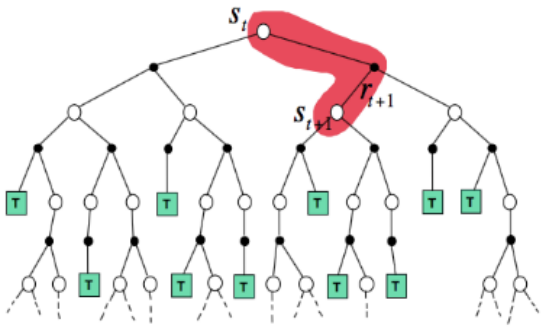
source: Sutton, Barto 2018. For a random-walk example.

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

TD v.s. MC and tradeoffs

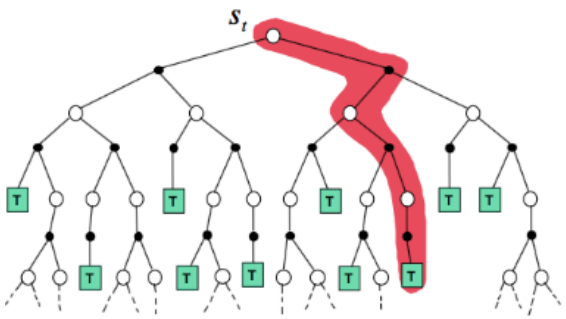


One full trajectory for update

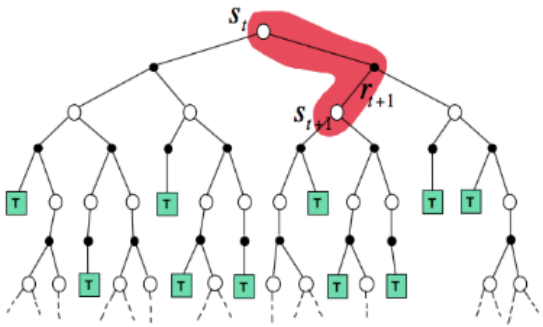


Updates take time to propagate

TD v.s. MC and tradeoffs



One full trajectory for update



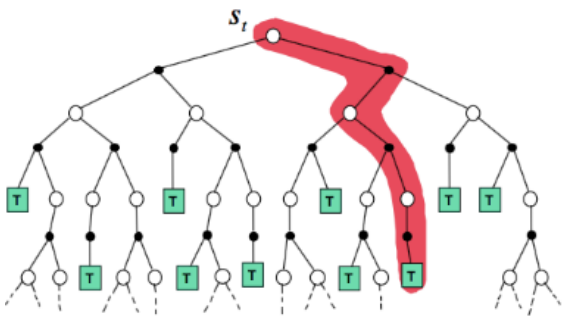
Updates take time to propagate

Tradeoff:

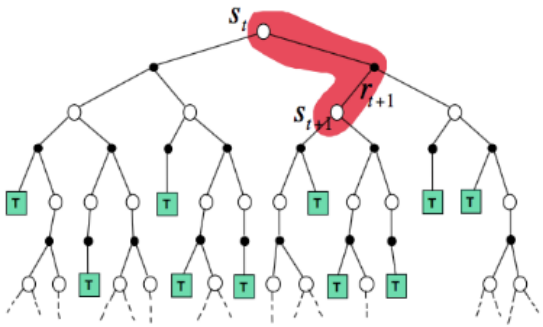
- Use n-step returns (see Sutton-Barto, chapter 7).

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

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

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
   - Monte-Carlo methods
   - Temporal difference
   - Q-learning and SARSA
   - Conclusion
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)

TD learning = policy evaluation. What about optimization?

Bellman's equations are:

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

TD learning = policy evaluation. What about optimization?

Bellman's equations are:

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

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

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.

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

```
Initialize Q(s, a) arbitrarily.
while True do
    Initialize S
    while S' is not a terminal state do
        π = policy derived from Q (e.g. ε-greedy).
        Sample A ~ π(S) and simulate a transition S', R ~ p(· | S, A).
        Q(S, A) := Q(S, A) + α_t(R + γ max_a Q(S', a) − Q(S, A)).
        S := S'
    end while
end while
```

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

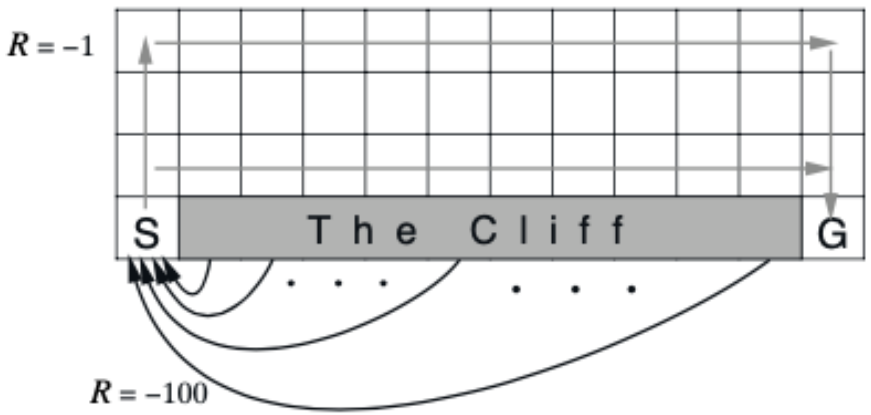
SARSA

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 ~ p(· | S, A) and A' := π(S').
        Q(S, A) := Q(S, A) + α_t(R + γQ(S', A') − Q(S, a)).
        S := S', A := A'
    end while
end while
```

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

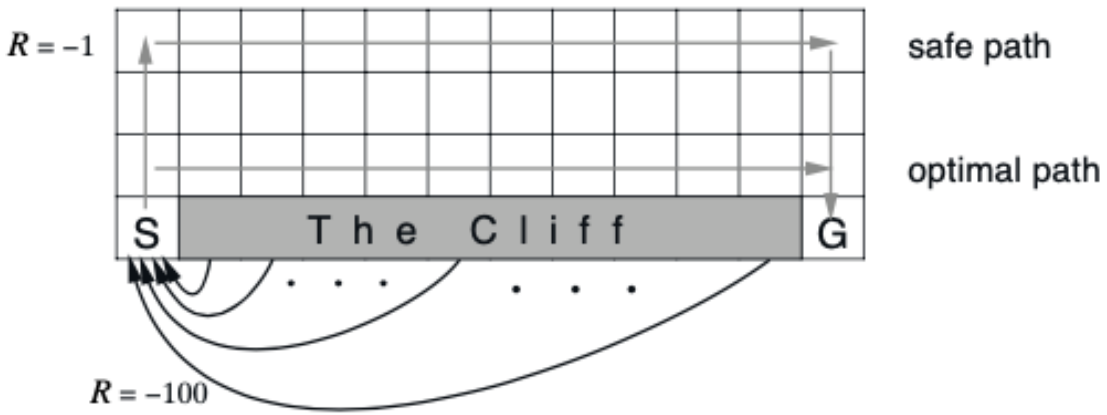
SARSA vs Q-learning



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

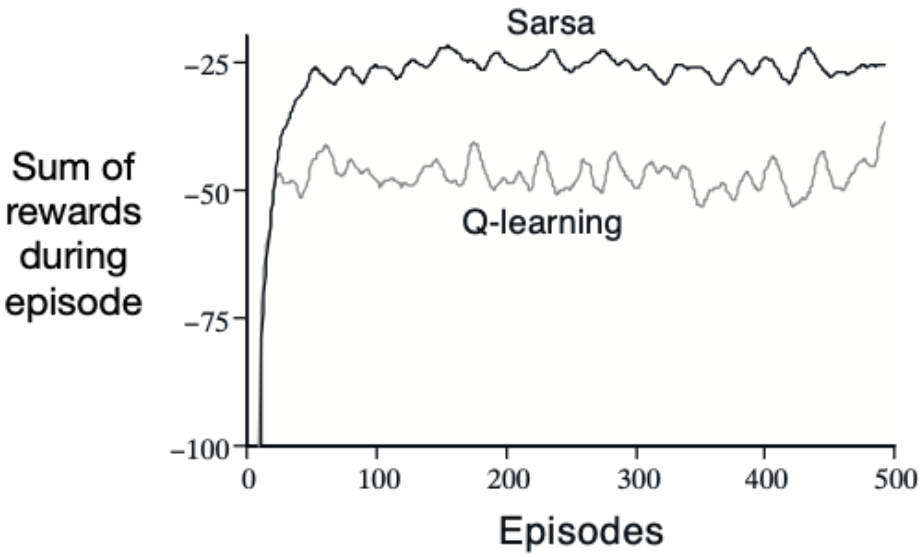
SARSA or Q-learning: what will be the difference?

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 4), and Q-learning with UCB Exploration is Sample Efficient for Infinite-Horizon MDP by Dong et al 2019.

Table of contents

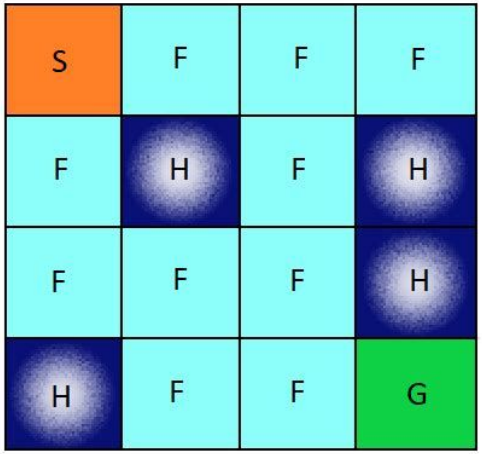
1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
   - Monte-Carlo methods
   - Temporal difference
   - Q-learning and SARSA
   - Conclusion
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)

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	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

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

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
3. Large state-spaces and approximations
   - Value function approximation and Deep Q-Learning
   - Policy gradient
   - Conclusion and other methods
4. Monte-Carlo tree search (MCTS)

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 $Q(s,a) \approx Q^*(s,a)$.

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

Two types of methods:

- MC methods:

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

- TD methods (SARSA / Q-learning)

Does it scale?

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

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

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?)

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?)

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

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
3. Large state-spaces and approximations
   - Value function approximation and Deep Q-Learning
   - Policy gradient
   - Conclusion and other methods
4. Monte-Carlo tree search (MCTS)

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:

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

where w is a vector parameter to be found.

Function approximation

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

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

where w is a vector parameter to be found.

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

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

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

Function approximation

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

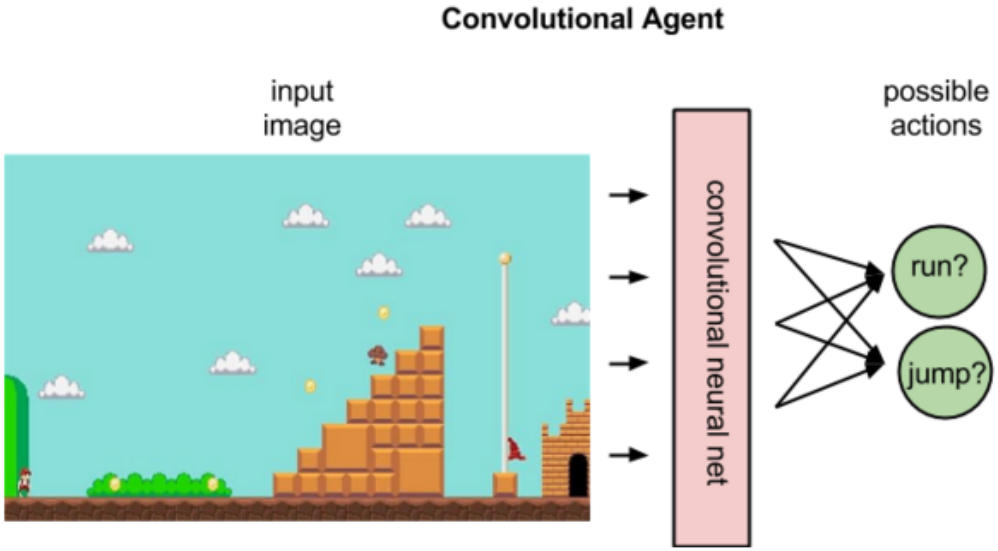
$$Q(S, A) \approx q_{\mathrm{w}}(S, A),$$

where w is a vector parameter to be found.

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

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

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

From Q-learning to deep Q-learning

The original Q-learning uses that:

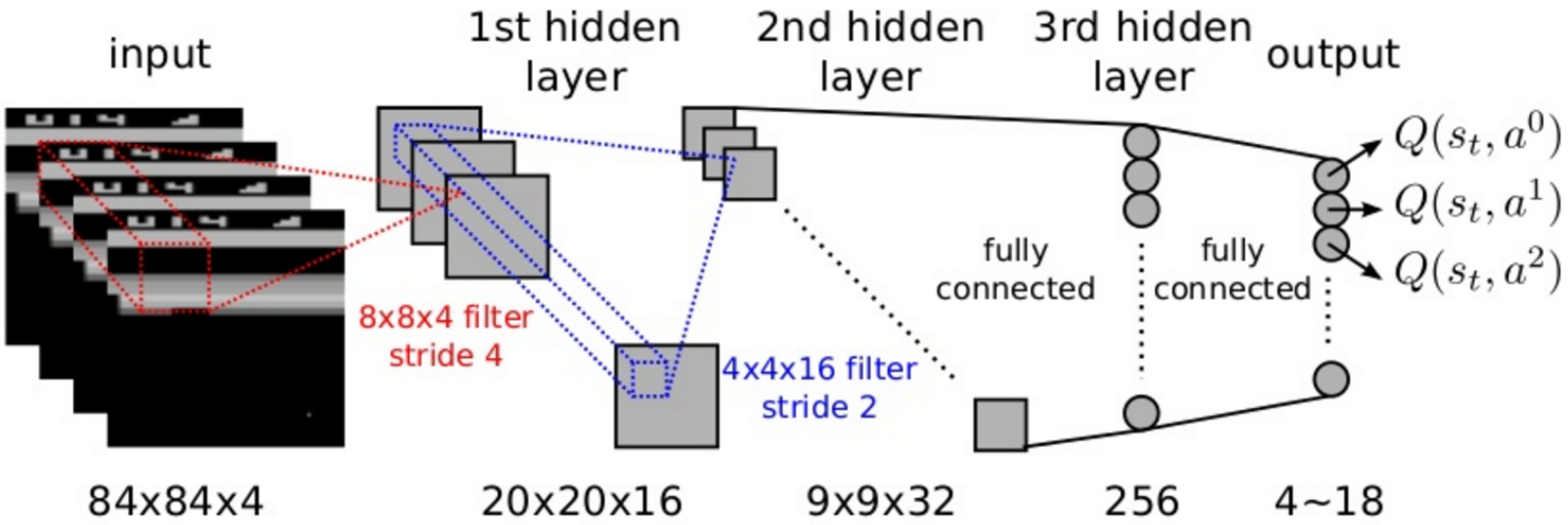
$$Q(S_t, A_t) = \mathbb{E}\left[R_{t+1} + \max_{a \in \mathcal{A}} Q(S_{t+1}, a)\right].$$

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

For that, we do:

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

Problems:

Why is vanilla unstable?

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

For that, we do:

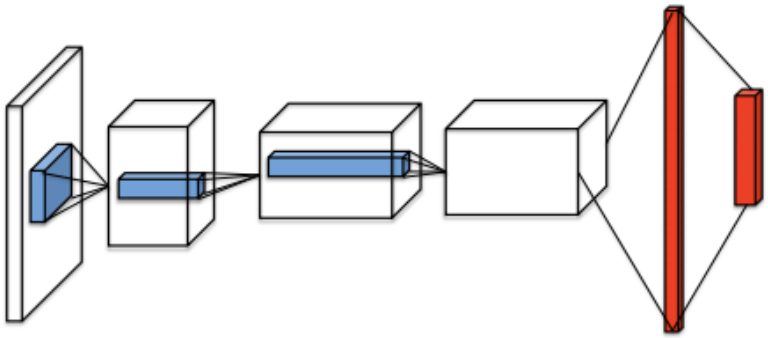
$$\mathrm{w} := \mathrm{w} + \alpha \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} q_{\mathrm{w}}(S_t, a) - q_{\mathrm{w}}(S_t, A_t) \right) \nabla_{\mathrm{w}}(q_{\mathrm{w}}(S_t, A_t)).$$

Problems:

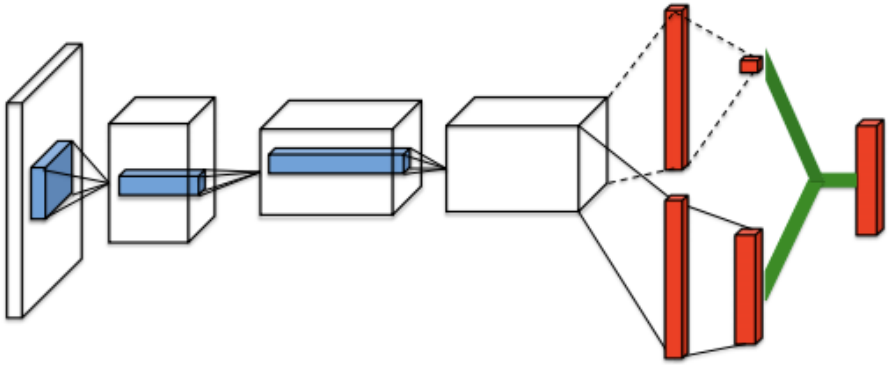
- Target and sources are highly correlated
- Target changes as we learn.
- Exploration is not guaranteed.

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

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
3. Large state-spaces and approximations
   - Value function approximation and Deep Q-Learning
   - Policy gradient
   - Conclusion and other methods
4. Monte-Carlo tree search (MCTS)

Policy search

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

$$\pi_{\mathsf{w}}(a \mid s) \propto \exp(\mathsf{w}^T \phi(s, a)),$$

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

Policy search

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

$$\pi_{\mathrm{w}}(a \mid s) \propto \exp(\mathrm{w}^T \phi(s, a)),$$

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

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

Policy search

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

$$\pi_{\mathsf{w}}(a \mid s) \propto \exp(\mathsf{w}^T \phi(s, a)),$$

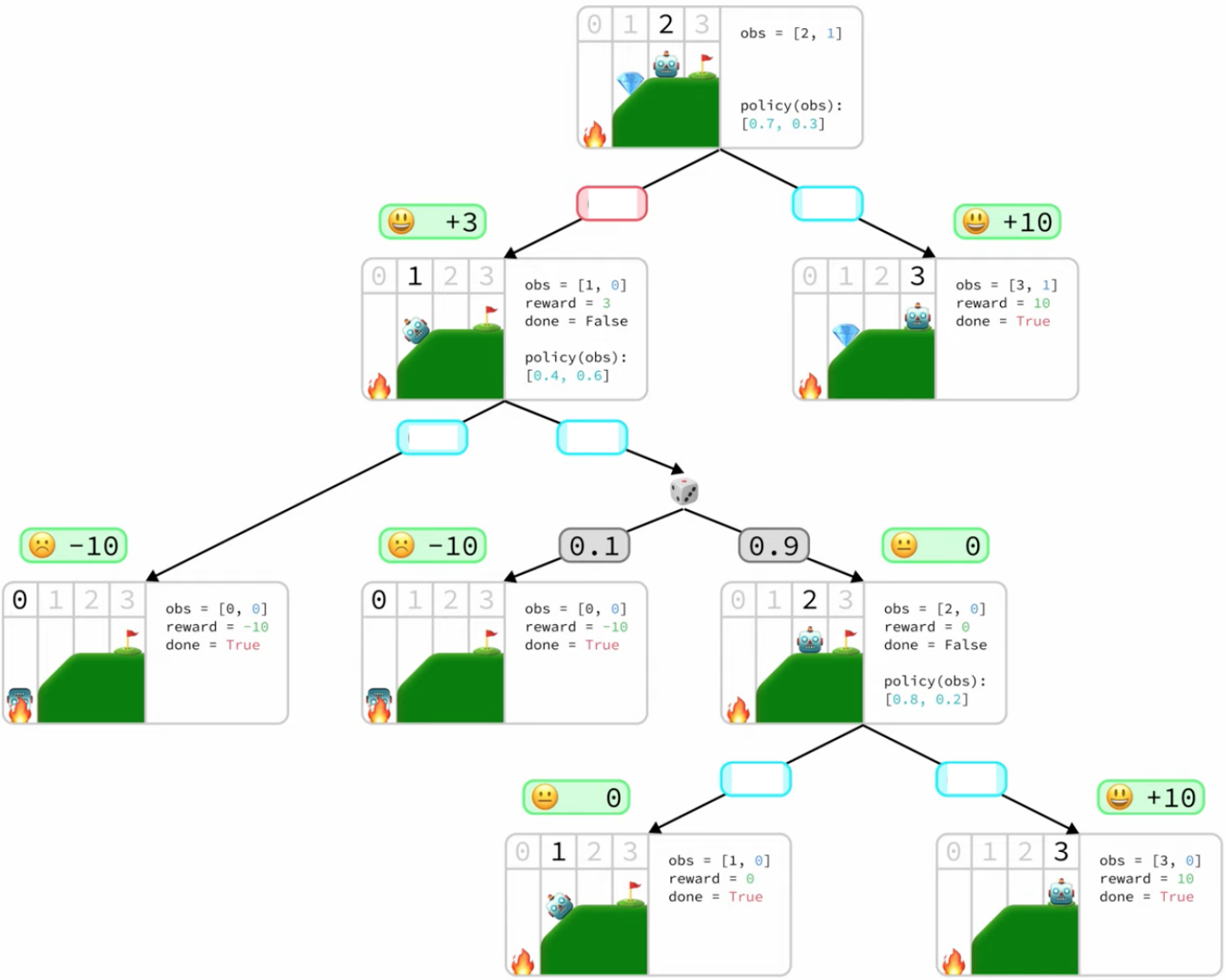
where $\phi(s, a)$ is a feature vector.

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

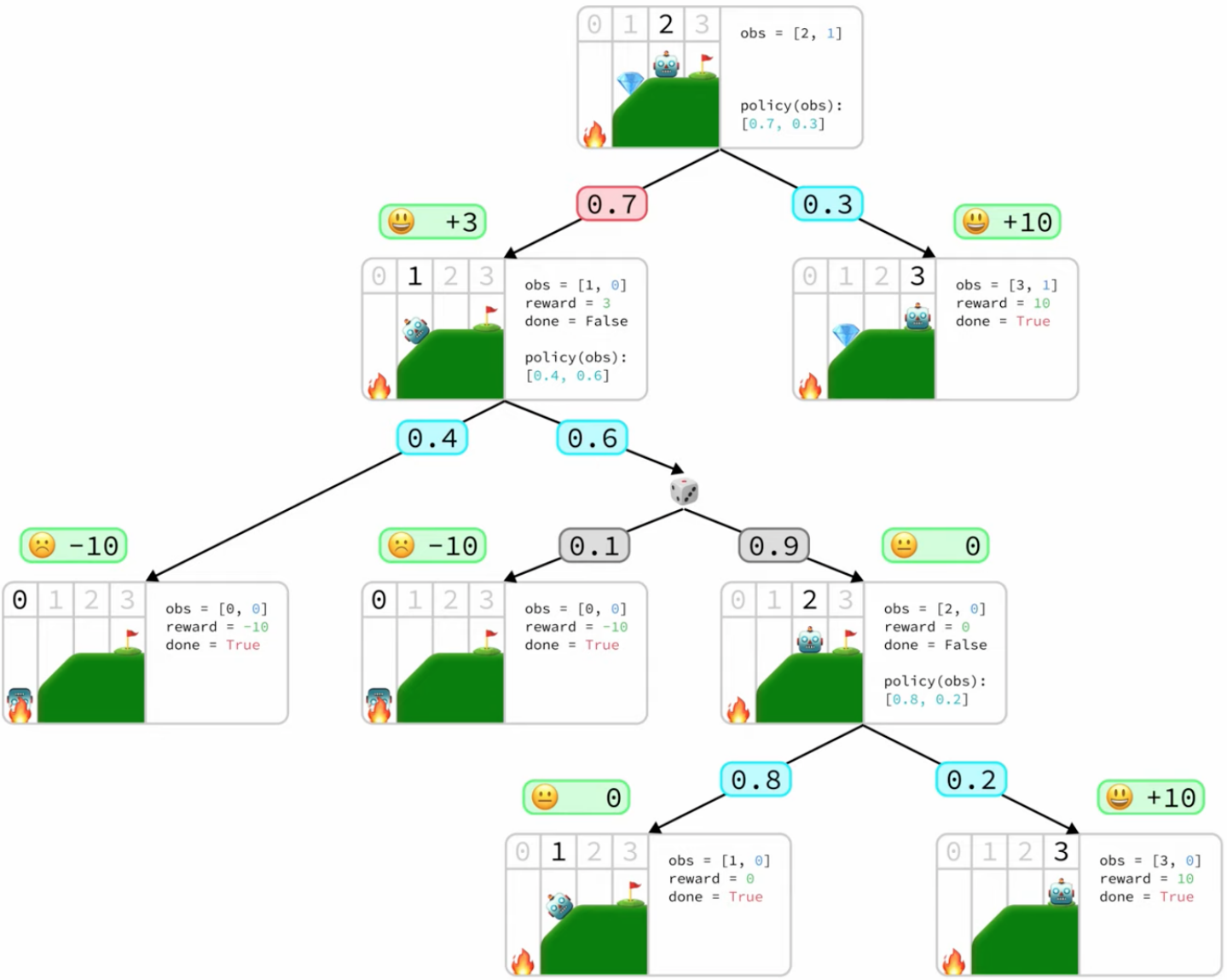
- Sometimes, this works well with direct methods (brute-force)
- We can also use policy gradients:

$$\mathsf{w} := \mathsf{w} + \alpha \nabla_{\mathsf{w}} J(\mathsf{w}).$$

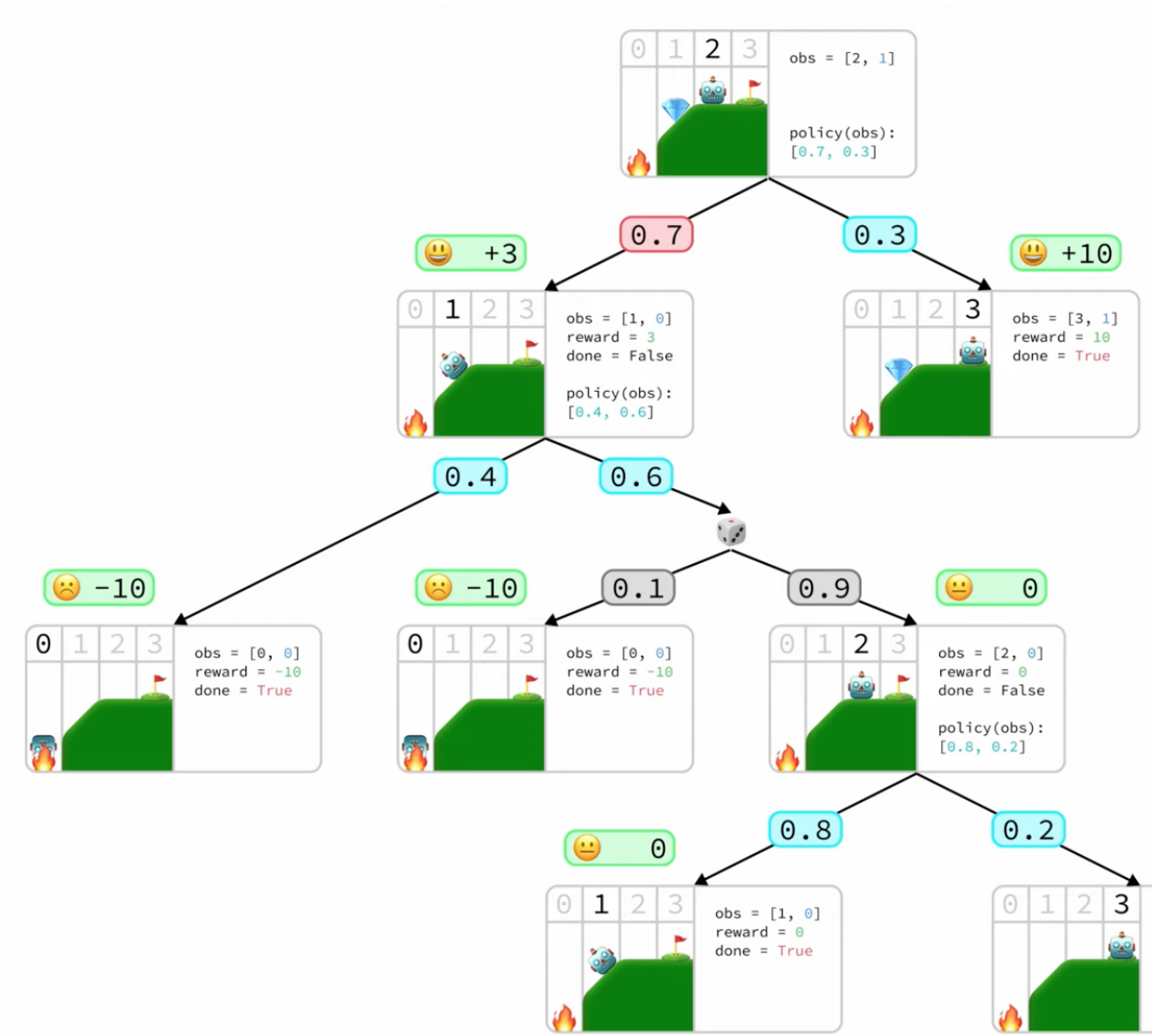
On an example https://www.youtube.com/watch?v=cQfOQcpYRzE



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

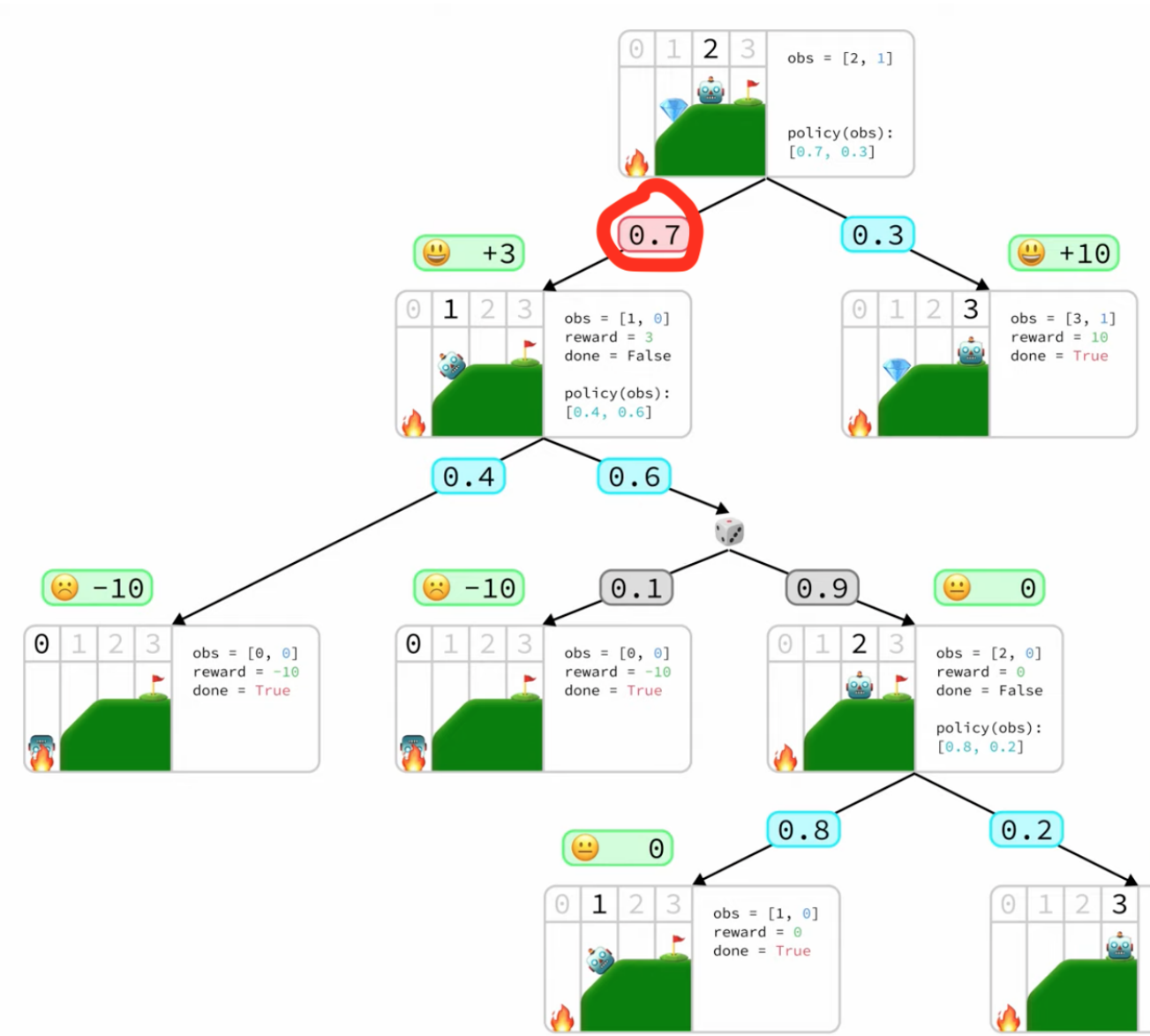


On an example https://www.youtube.com/watch?v=cQfOQcpYRzE



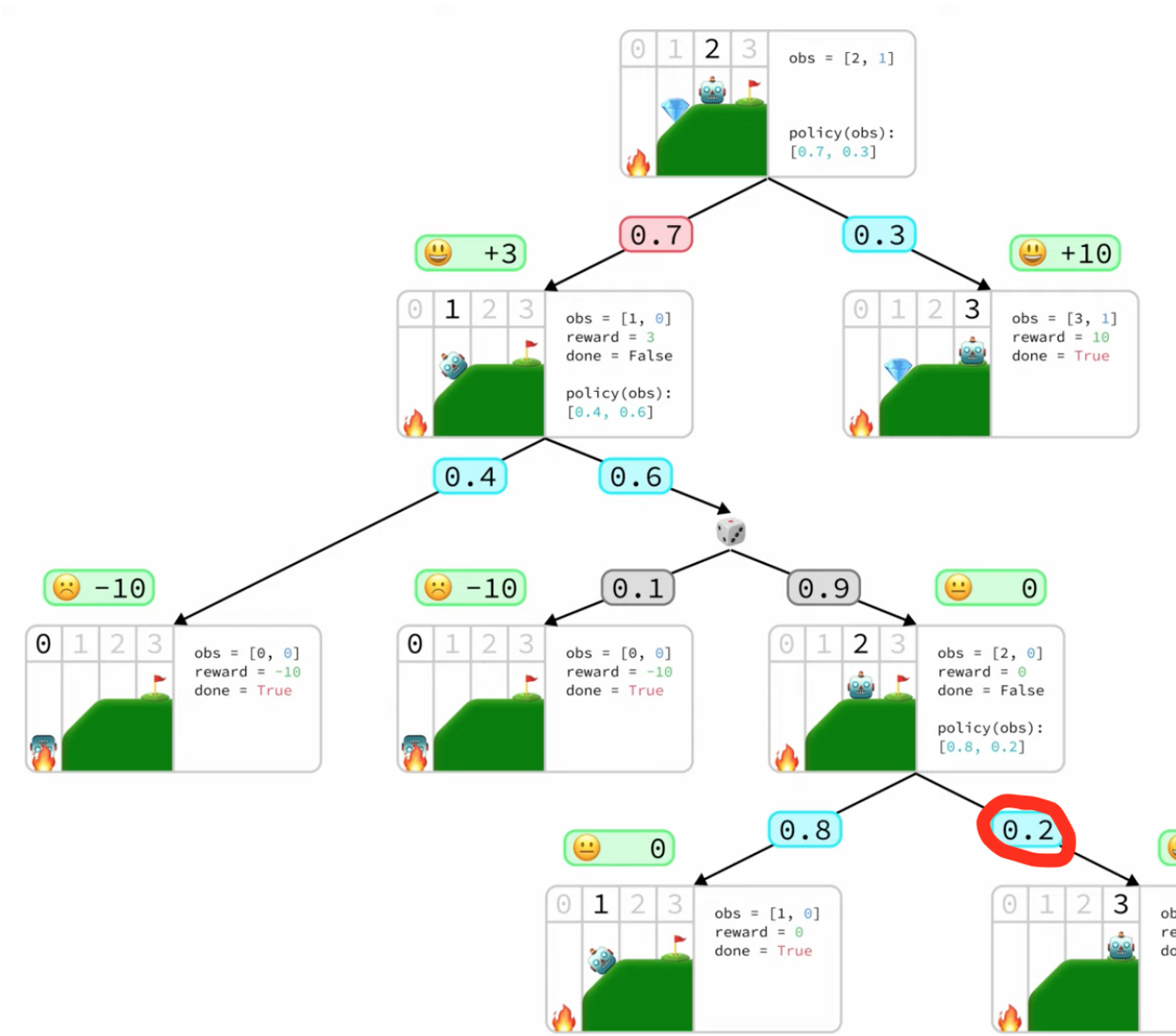

```
Expected Return (G) =

    (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:

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

The policy gradient theorem

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

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

Hence, an unbiased estimate of the gradient $\nabla_{\mathrm{w}}\mathbb{E}\left[G_0\right]$ 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:

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

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

```
Initialize w.
while True do
    Simulate a trajectory (from t = 1 to T)
    for t = T to t = 1 do
        G_t := Σ_{t'=t}^{T} R_{t'}.
        ∇J := G_t ∇ log π(A_t|S_t).
        w := w + α∇J.
    end for
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:

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

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

Table of contents

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
3. Large state-spaces and approximations
   - Value function approximation and Deep Q-Learning
   - Policy gradient
   - Conclusion and other methods
4. Monte-Carlo tree search (MCTS)

Classes of learning algorithms

We have seen two classes of RL methods:

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

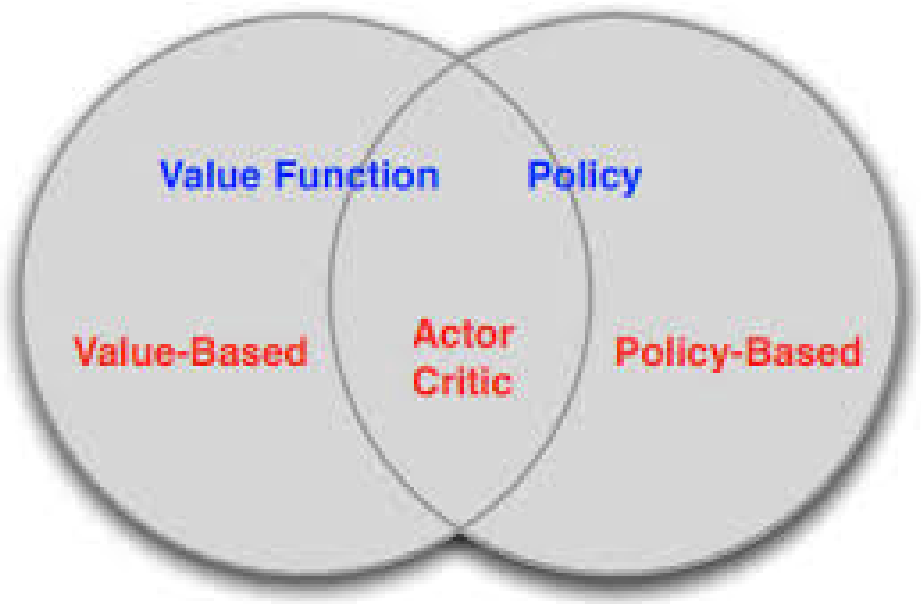
- Value-based learning can be unstable but uses samples efficiently.
- Policy-based tend to be more robust.

Classes of learning algorithms

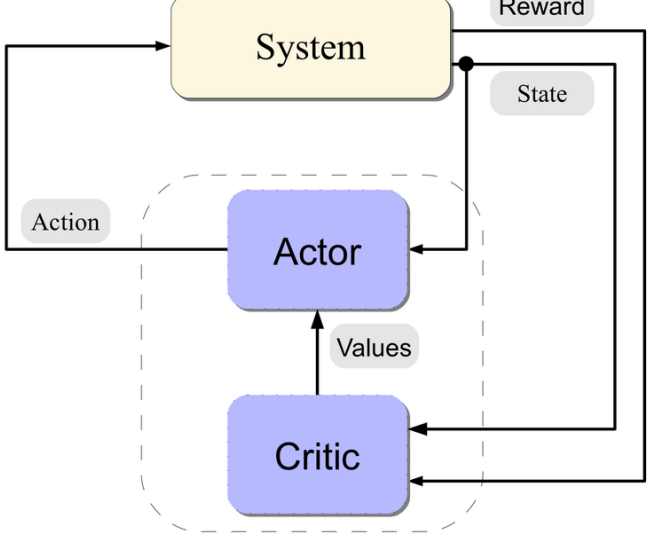
We have seen two classes of RL methods:

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

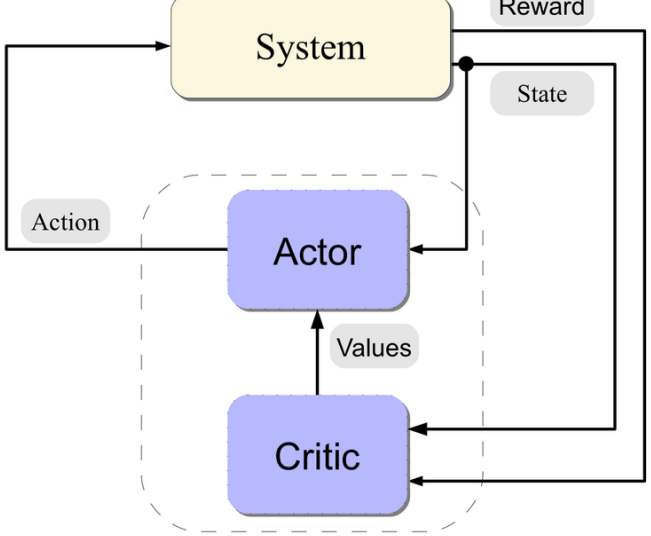
- Value-based learning can be unstable but uses samples efficiently.
- Policy-based tend to be more robust.



Actor Critic method



Actor Critic method



Basic Actor Critic

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

Going further

Extra-reading:

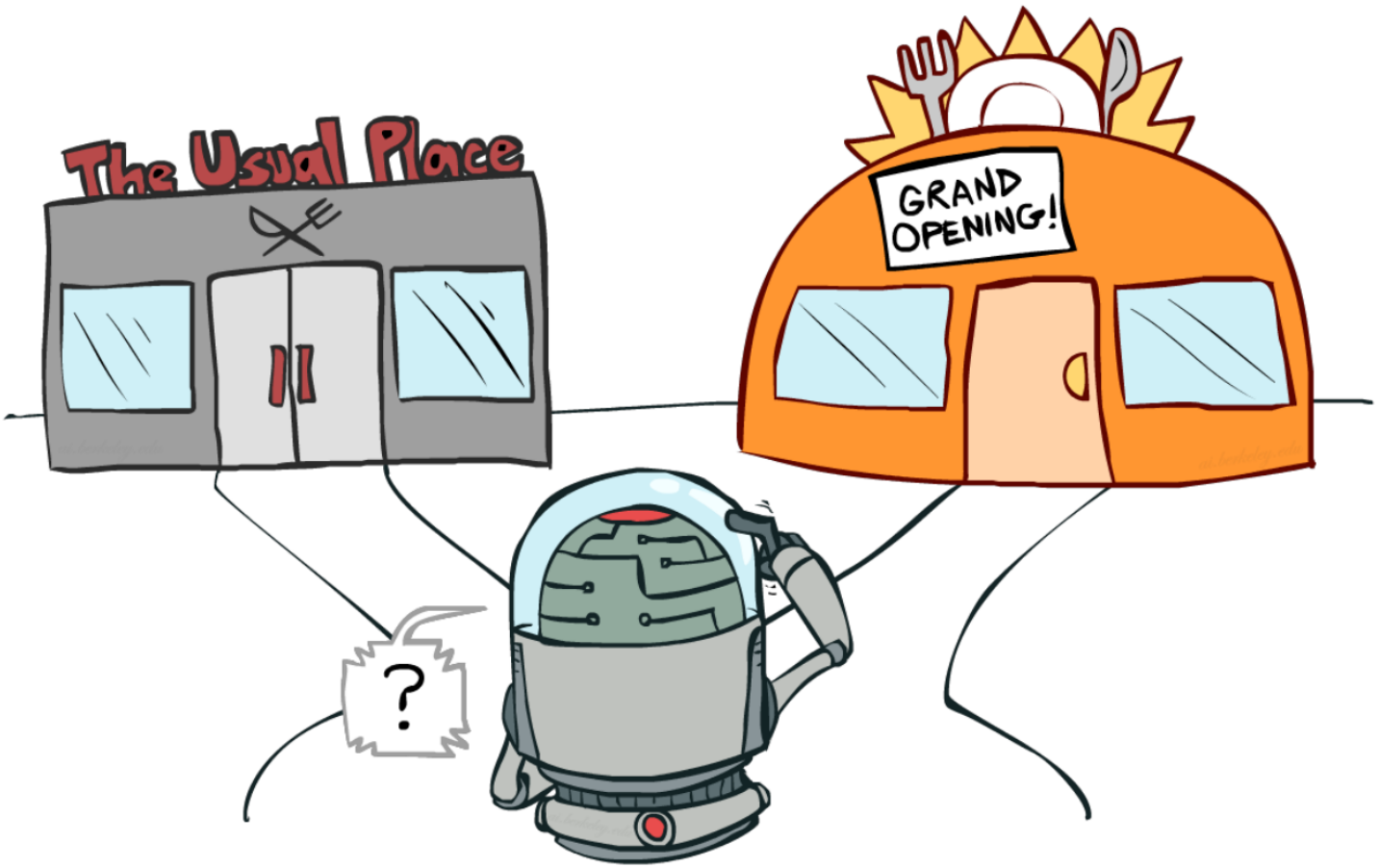
- Introduction to Reinforcement Learning (Sutton-Barto, 2018 last ed.)
- Algorithms for Reinforcement Learning (Szepesvari, 2010)
- Deep Reinforcement learning: hands on (Maxim Lapan, 2020)

Next course: some thoughts on exploration / exploitation.

Outline

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)
   - Min-max and alpha-beta pruning
   - MCTS and exploration
   - Conclusion

Reminder: exploration-exploitation dilemma and bandits



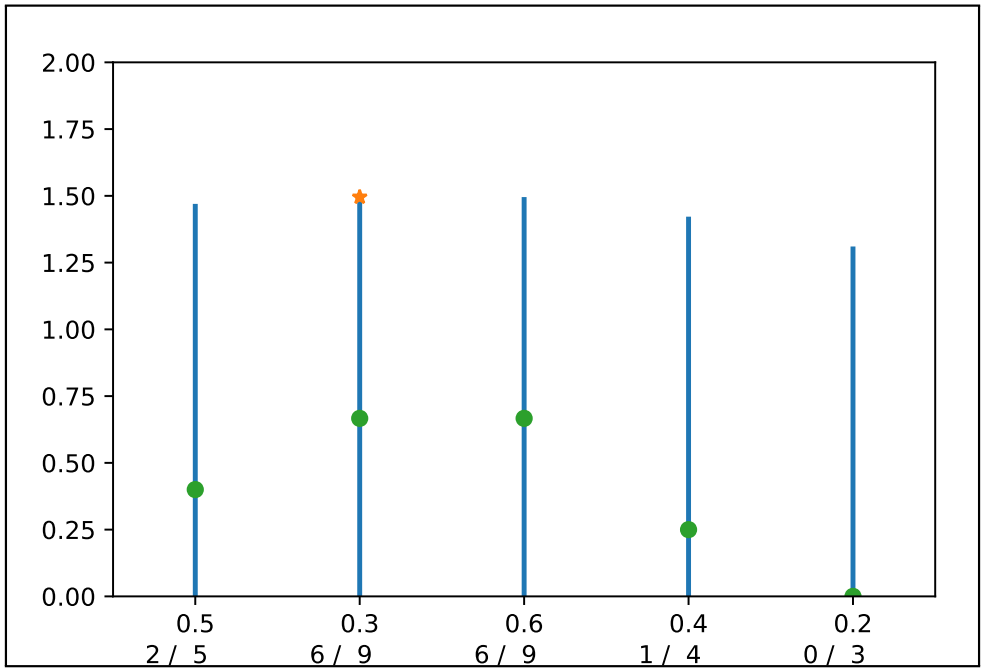
- How useful is this for RL?

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 $A_{t+1} \in \arg\max_{a \in \{1 \ldots n\}} UCB_a(t)$ (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.

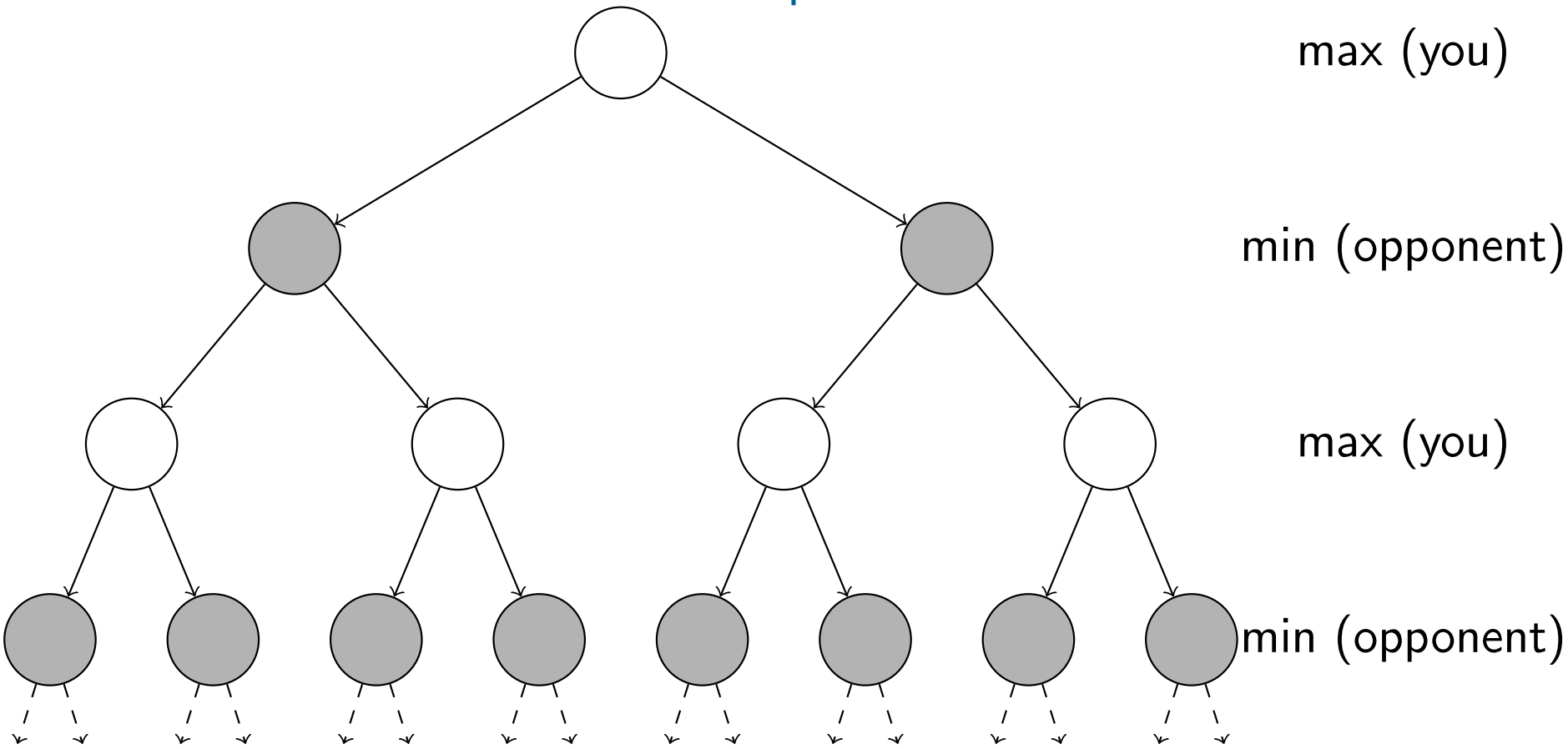
- Generate a tree of possibilities.
- Explore this tree.

What if the tree is too big?

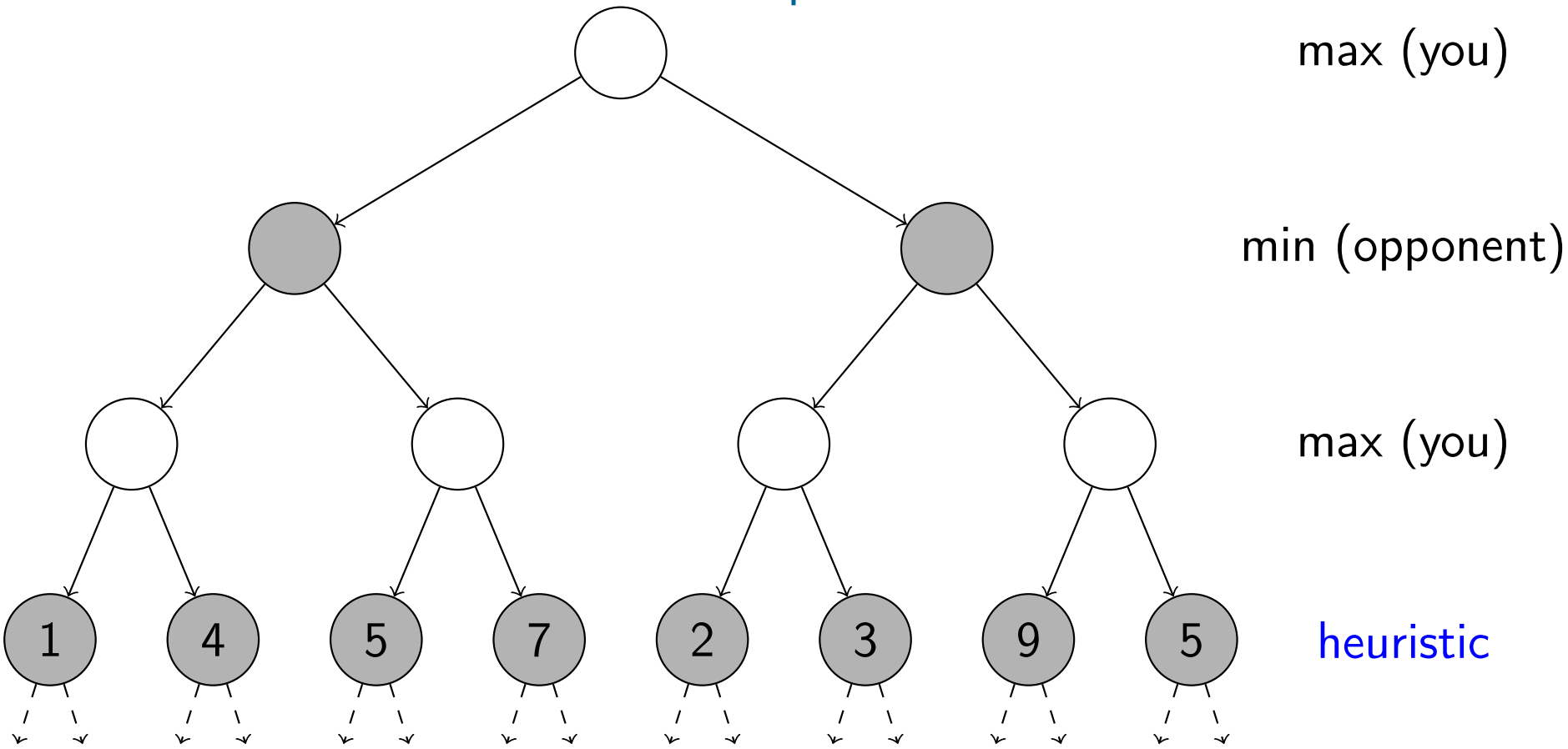
Table of contents

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)
   - Min-max and alpha-beta pruning
   - MCTS and exploration
   - Conclusion

You can construct the tree of possibilities



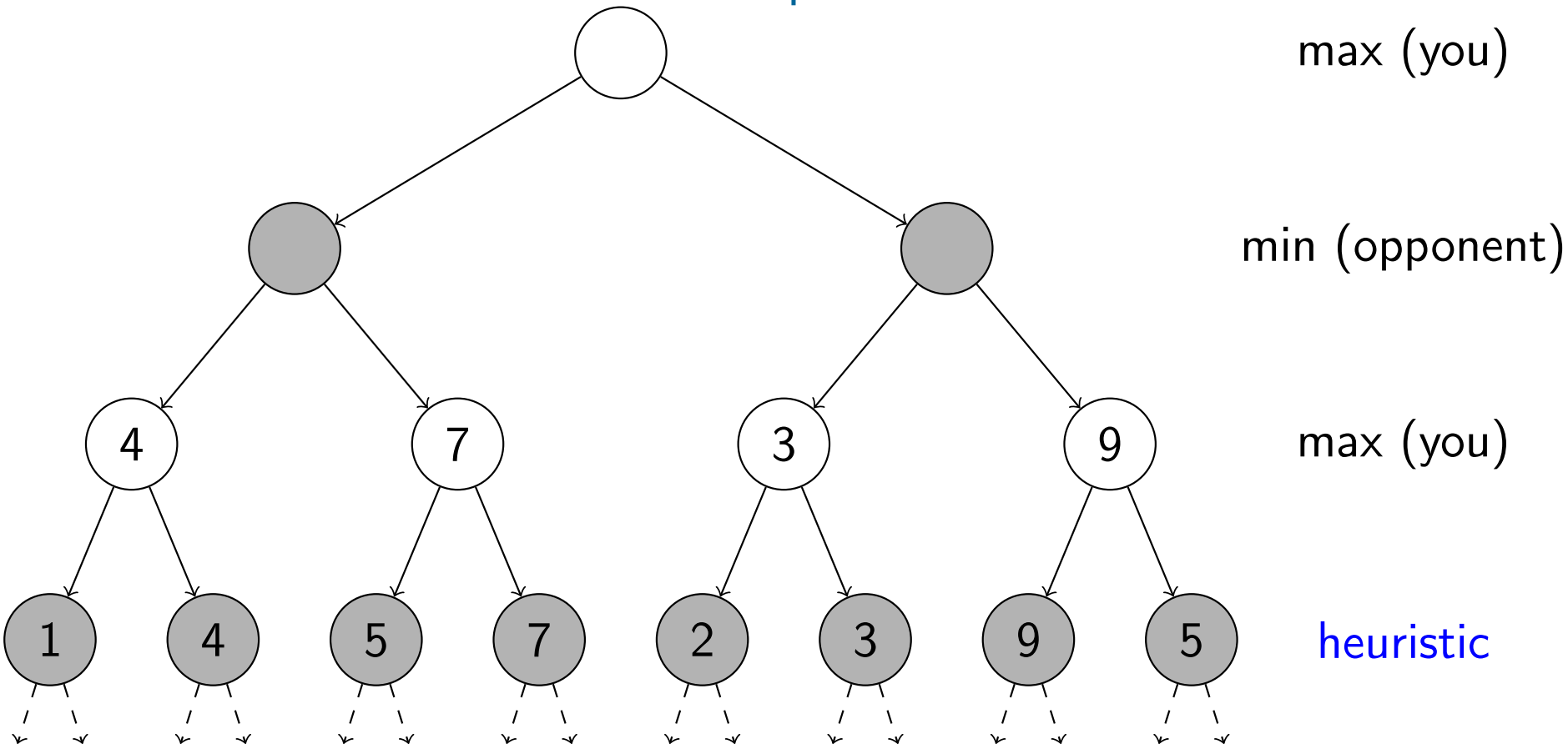
You can construct the tree of possibilities



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

- You can backtrack with the min-max algorithm.

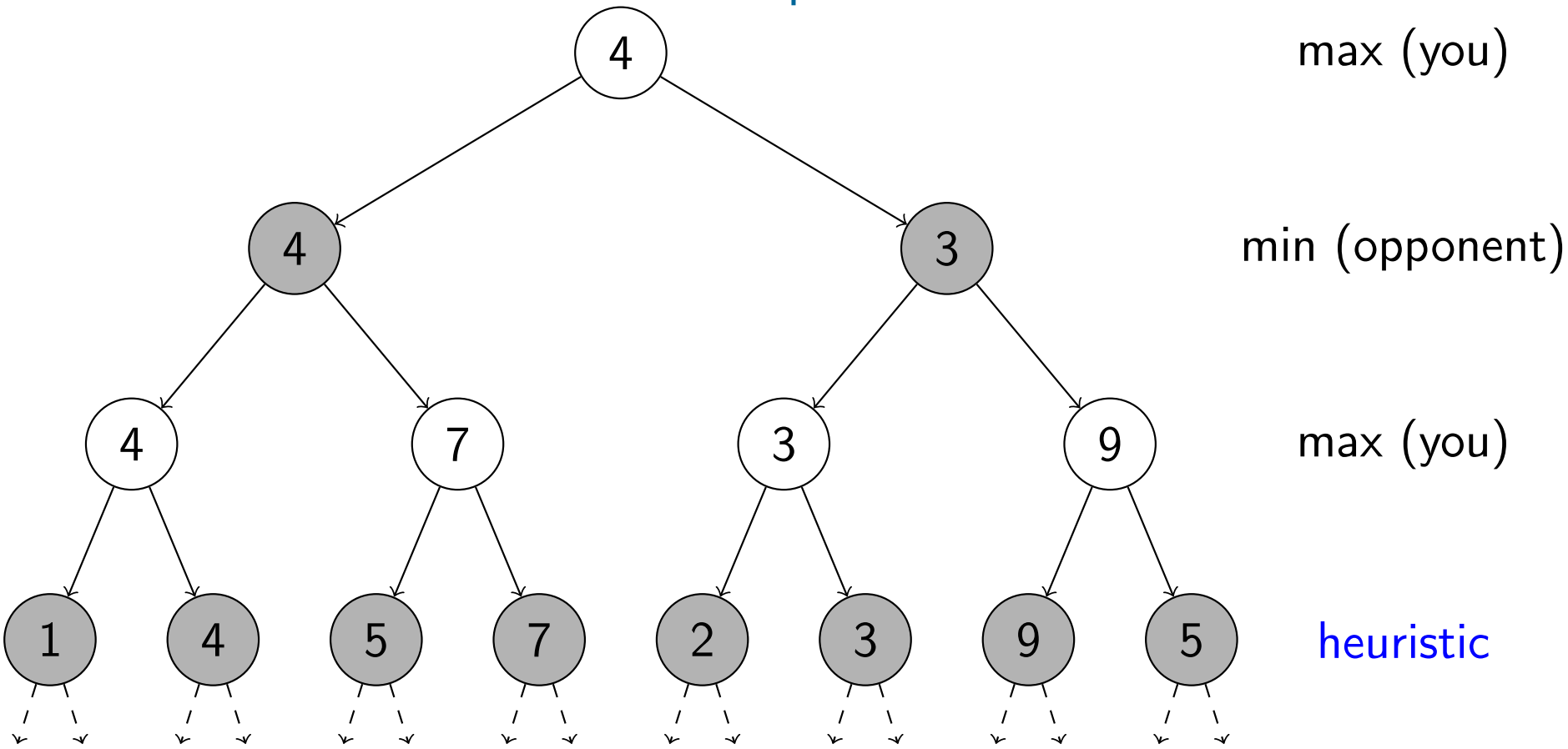
You can construct the tree of possibilities



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

- You can backtrack with the min-max algorithm.

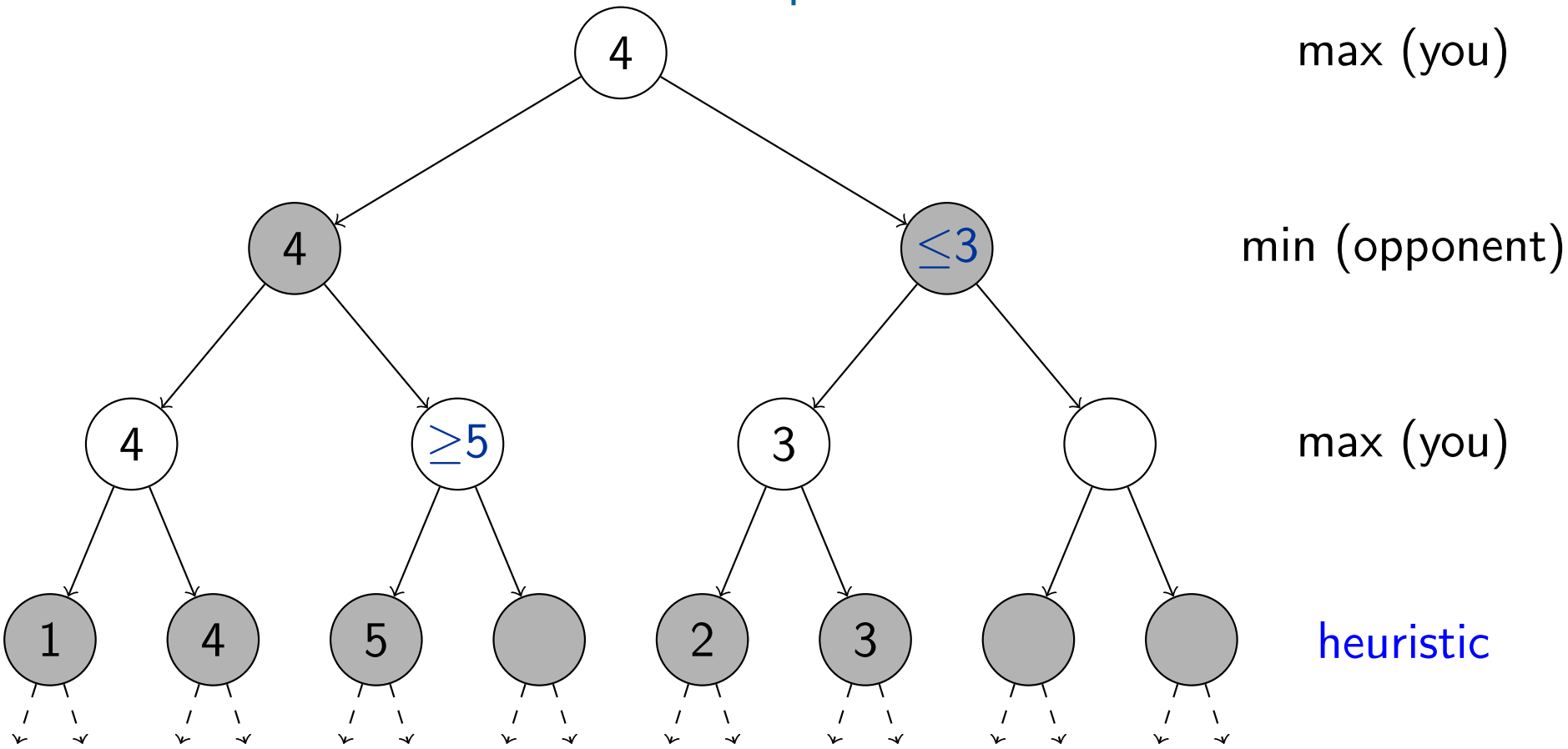
You can construct the tree of possibilities



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

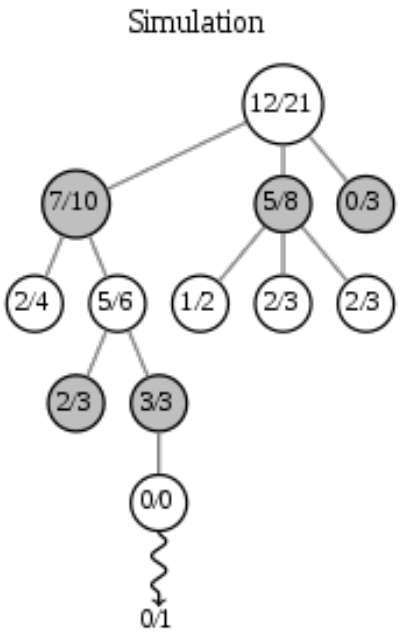
1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)
   - Min-max and alpha-beta pruning
   - MCTS and exploration
   - Conclusion

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

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

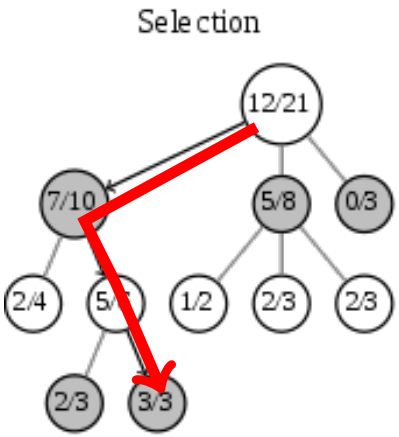
- Tree can still be very big (A^D)
- You need a good heuristic.
  - Result is only available at the end
- You might want to avoid the exploration of not promising parts.
  - For that you need a good heuristic.

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

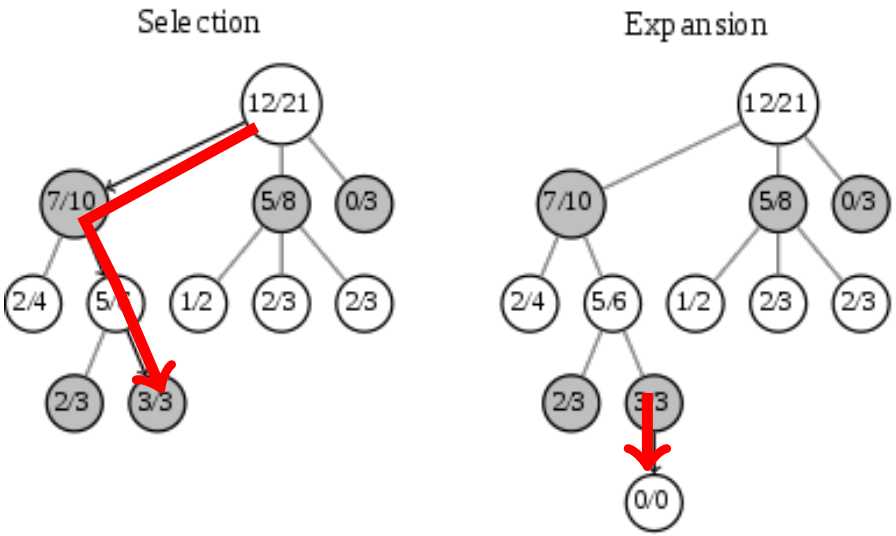


For each child, let $S(c)$ be the number of success and $N(c)$ be the number of time you played c, and $t = \sum_{c'} N(c')$.

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

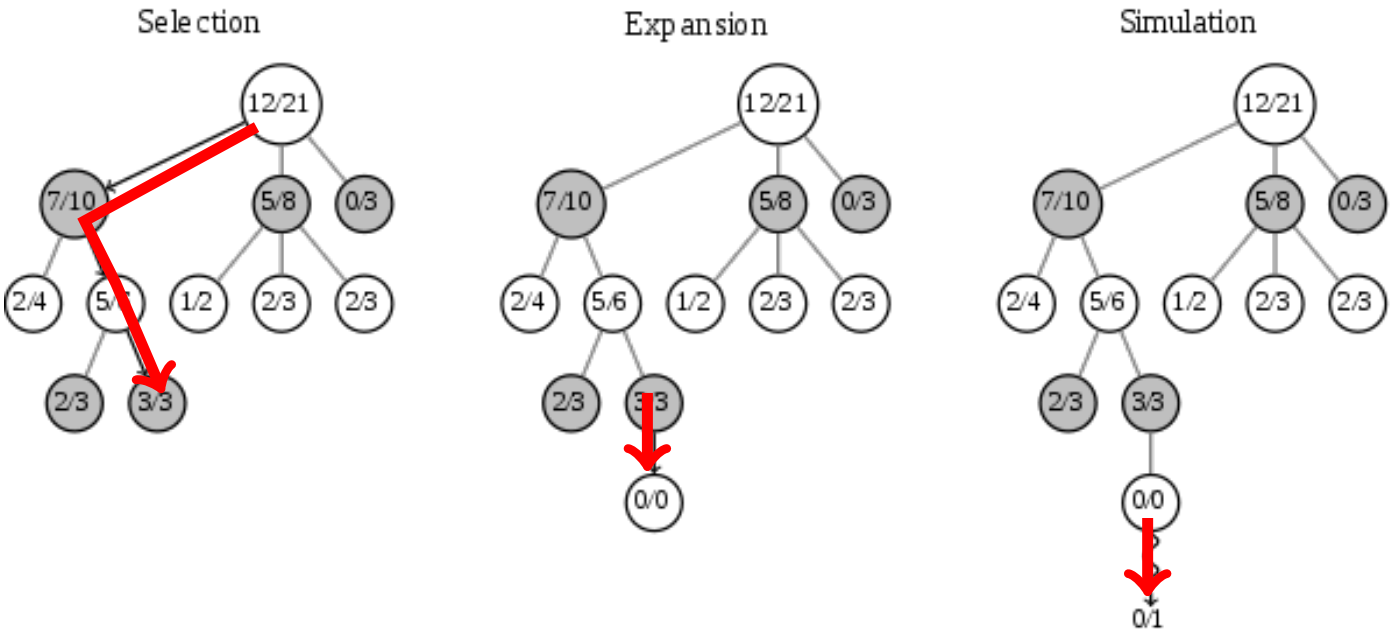
Open question: no guarantee with $\sqrt{\log t / N(c)}$. Is $\sqrt{t}/N(c)$ better?

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



- Create one or multiple children of the leaf.

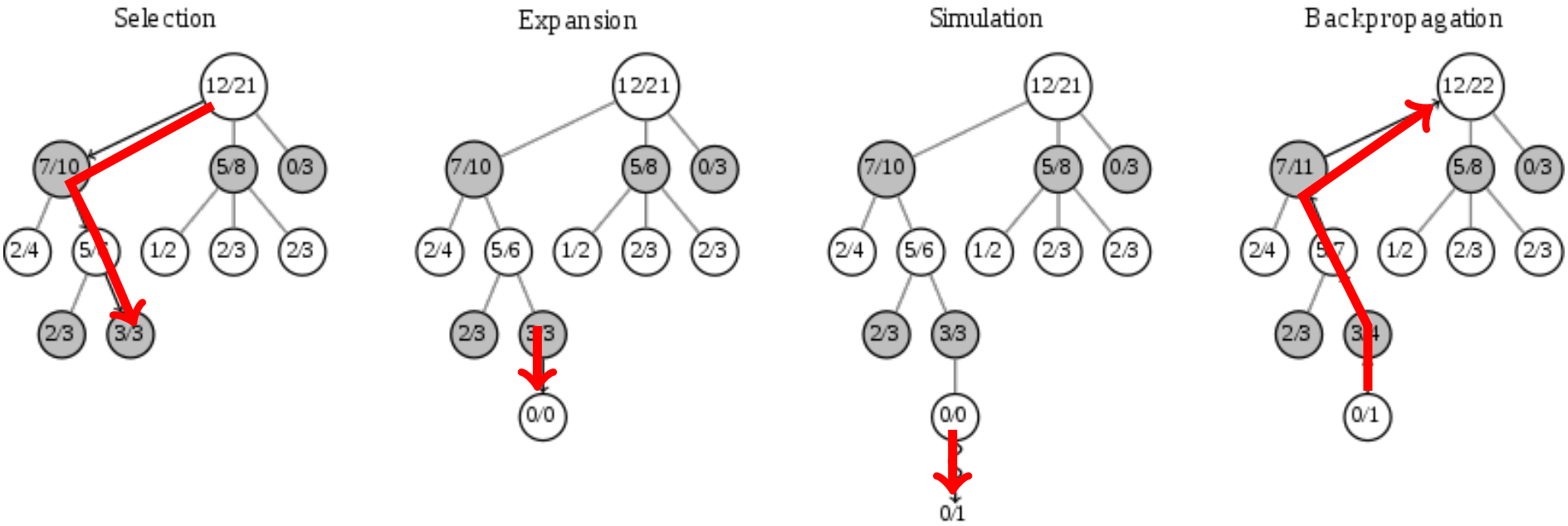
MCTS (Monte Carlo Tree Search) uses simulation to conduct the tree search



Rollout policy
(ex: random)

- 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

```
while Some time is left do
    Select a leaf node                                    #UCB-like
    Expand a leaf
    Use rollout (or equivalent) to estimate the leaf      #random sampling
    Backpropagate to the root
end while
Return arg max_{c∈children(root)} N(c)                    #or S(c)/N(c).
```

Demo / exercice

Table of contents

1. Markov Decision Processes (MDPs)
2. Tabular reinforcement learning
3. Large state-spaces and approximations
4. Monte-Carlo tree search (MCTS)
   - Min-max and alpha-beta pruning
   - MCTS and exploration
   - Conclusion

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.