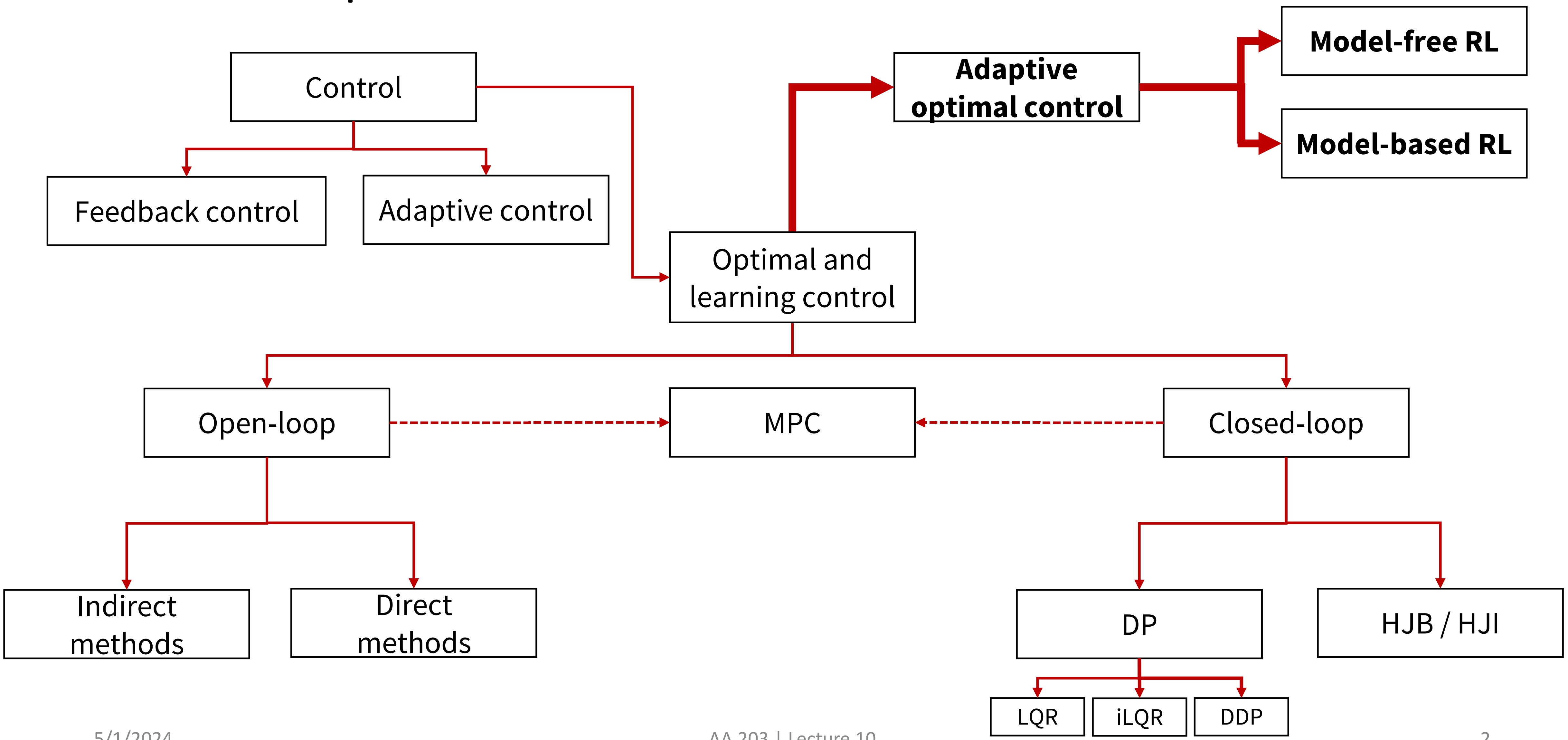
AA203
Optimal and Learning-based Control

Intro to Reinforcement Learning

Stanford University

Autonomous Systems Lab

Roadmap


Control
Feedback control
Adaptive control
Optimal and learning control
Adaptive optimal control
Model-free RL
Model-based RL
Open-loop
MPC
Closed-loop
Indirect methods
Direct methods
DP
HJB / HJI
LQR
iLQR
DDP

Outline

What is Reinforcement Learning? (and the RL setting)

From exact methods to model-free control

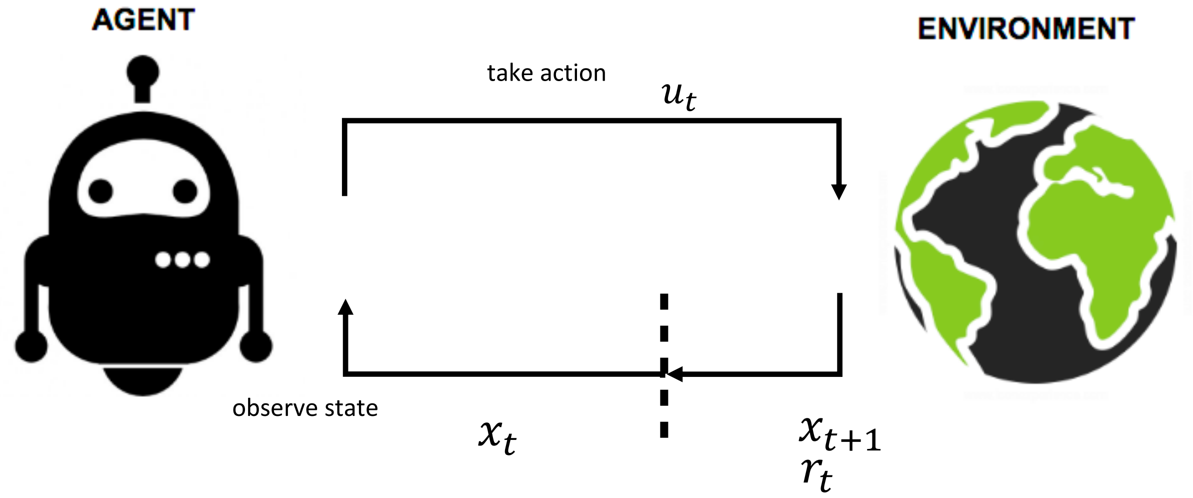
- Monte Carlo Learning
- Temporal-Difference (TD) Learning

A taxonomy of RL algorithms & important trade-offs

What is reinforcement learning?

Fundamentally:

- A mathematical formalism for **learning-based** decision making
- An approach for learning decision making and control **from experience** ⟶ Success is measured by a scalar **reward**



$$\tau = (x_0, u_0, \dots, x_N, u_N)$$

Why reinforcement learning?

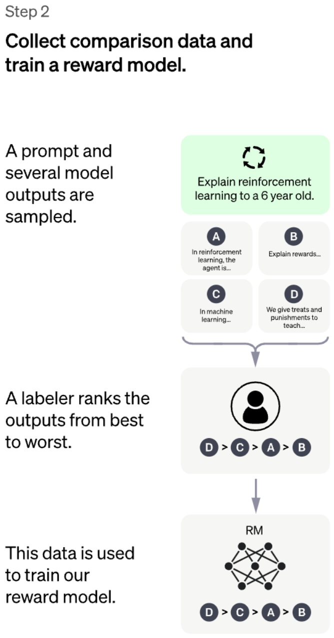
- Only need to specify a reward function and the agent learn everything else!

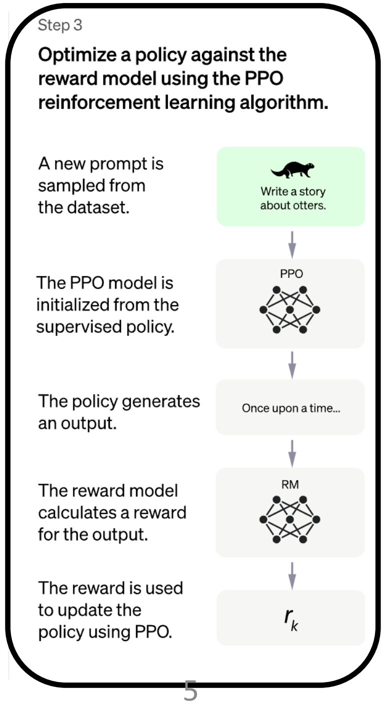
ChatGPT "Alignment"- OpenAI

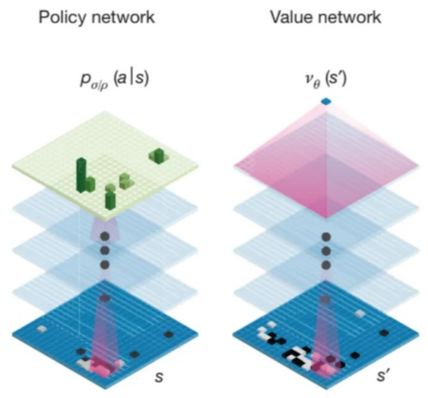
Silver et al. 2016

Levine*, Finn* et al. 2016

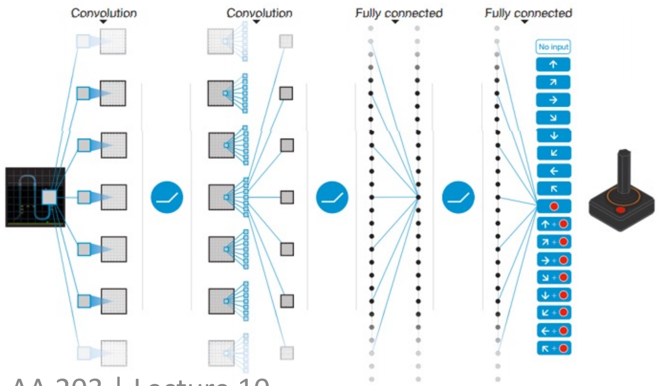
Mnih et al. 2014







Deep Q Network



Characteristics of reinforcement learning?

How does RL differ from other machine learning paradigms?

- No supervision, only a reward signal
- Feedback can be delayed, not instantaneous (i.e., credit assignment)
- Data is **not** i.i.d., earlier decisions affect later interactions (tension between exploration and exploitation)

Markov Decision Process

State: $x \in \mathcal{X}$

Action: $u \in \mathcal{U}$

Transition function / Dynamics: $T(x_t \mid x_{t-1}, u_{t-1}) = p(x_t \mid x_{t-1}, u_{t-1})$

Reward function: $r_t = R(x_t, u_t) \colon \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$

Discount factor: $\gamma \in (0,1)$

Typically represented as a tuple

$$\mathcal{M} = (\mathcal{X}, \mathcal{U}, T, R, \gamma)$$

Goal: choose a policy that maximizes cumulative (discounted) reward

$$\pi^* = \underset{\pi}{\mathrm{arg}max}\ \mathbb{E}_p\left[\sum_{t \geq 0} \gamma^t R\big(x_t, \pi(x_t)\big)\right]$$

Value functions

State-value function: "*the expected total reward if we start in that state and act accordingly to a particular policy*"

$$V_\pi(x_t) = \mathbb{E}_p\left[\sum_{t'\geq t} \gamma^{t'} R\left(x_{t'}, \pi\left(x_{t'}\right)\right)\right]$$

Action-state value function: "*the expected total reward if we start in that state, take an action, and act accordingly to a particular policy*"

$$Q_\pi(x_t, u_t) = \mathbb{E}_p\left[\sum_{t'\geq t} \gamma^{t'} R\left(x_{t'}, u_{t'}\right)\right]$$

Optimal state-value function: $V^*(x) = \max_\pi V_\pi(x)$

Optimal action-state value function: $Q^*(x, u) = \max_\pi Q_\pi(x, u)$

Bellman Equations

The optimal value function satisfies Bellman's equation:

$$V^*(x_t) = \max_u \left(R(x_t, u_t) + \gamma \sum_{x_{t+1} \in X} T(x_{t+1} \mid x_t, u_t)\, V^*(x_{t+1}) \right)$$

Bellman Optimality Equation

For any stationary policy π, the values $\mathrm{V}_\pi(x) := \mathbb{E}\left[\sum_{t \geq 0} \gamma^t R\left(x_t, \pi(x_t)\right)\right]$ are the unique solution to the equation

$$\begin{aligned} \mathrm{V}_\pi(x_t) &= \mathbb{E}_\pi\left[R(x_t, \pi(x_t)) + \gamma \mathrm{V}_\pi(x_{t+1})\right] \\ &= R(x_t, \pi(x_t)) + \gamma \sum_{x_{t+1} \in X} T(x_{t+1} \mid x_t, \pi(x_t))\, \mathrm{V}_\pi(x_{t+1}) \end{aligned}$$

Bellman Expectation Equation

Bellman Equations

The optimal state-action value function (Q function) $Q^*(x, u)$ satisfies Bellman's equation:

$$Q^*(x_t, u_t) = R(x_t, u_t) + \gamma \sum_{x_{t+1} \in X} T(x_{t+1} \mid x_t, u_t) \max_{u_{t+1}} Q^*(x_{t+1}, u_{t+1})$$

Bellman Optimality Equation

For any stationary policy π, the corresponding Q function satisfies

$$Q_\pi(x_t, u_t) = R(x_t, u_t) + \gamma \sum_{x_{t+1} \in X} T(x_{t+1} \mid x_t, u_t) Q_\pi\big(x_{t+1}, \pi(x_{t+1})\big)$$

Bellman Expectation Equation

Solving MDPs

In previous lectures, we resorted to exact methods

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

All of these formulations require a model of the MDP!

To solve unknown MDPs, we'll use interactions with the environment

Limitations of exact methods (such as Policy/Value Iteration):

- Update equations (i.e., Bellman equations) require access to dynamics model $T(x_{t+1} \mid x_t, u_t)$ **Sampling-based approximations**
- Iteration over (and storage of) all states and actions requires small, discrete state-action space **Function approximation**

Outline

What is Reinforcement Learning? (and the RL setting)

From exact methods to model-free control

- Monte Carlo Learning
- Temporal-Difference (TD) Learning

A taxonomy of RL algorithms & important trade-offs

Monte Carlo Reinforcement Learning

- MC methods learn directly from episodes of experience
- MC is model-free: no knowledge of MDP transitions / rewards
- MC uses the simplest possible idea: value = mean return
  - Recall that the return is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

- Caveat: can only apply MC to episodic MDPs
  - All episodes must terminate

Monte Carlo Policy Evaluation

- Let's consider Monte Carlo methods for learning the state-value function $V_\pi(x)$ from episodes of experience under policy π
- Recall that the value function is the expected return

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

$$V_\pi(x_t) = \mathbb{E}\left[\sum_{t' \geq t} \gamma^{t'} R\big(x_{t'}, \pi(x_{t'})\big)\right] = \mathbb{E}[G_t \mid x_t]$$

- Monte-Carlo policy evaluation uses empirical mean return instead of expected return

Monte Carlo Policy Evaluation

First-visit

- To evaluate state x
- The **first** time-step t that state x is visited in an episode
- Increment counter $N(x) \leftarrow N(x) + 1$
- Increment total return $S(x) \leftarrow S(x) + G_t$
- Value is estimated by mean return $\hat{V}(x) = S(x)/N(x)$
- By law of large numbers $\hat{V}(x) \rightarrow V_\pi(x)$ as $N(x) \rightarrow \infty$

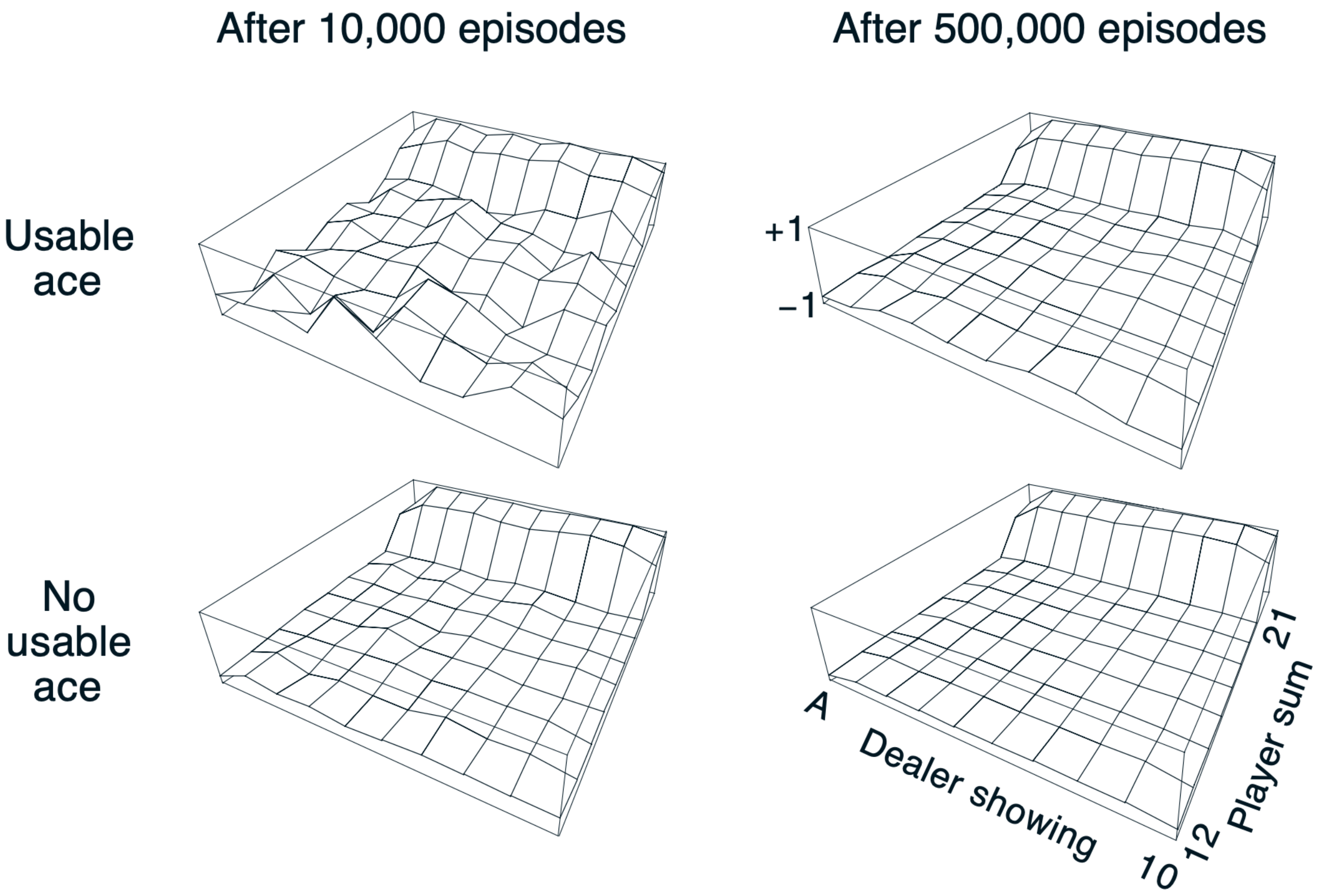
Every-visit

- To evaluate state x
- **Every** time-step t that state x is visited in an episode
- Increment counter $N(x) \leftarrow N(x) + 1$
- Increment total return $S(x) \leftarrow S(x) + G_t$
- Value is estimated by mean return $\hat{V}(x) = S(x)/N(x)$
- By law of large numbers$\hat{V}(x) \rightarrow V_\pi(x)$ as $N(x) \rightarrow \infty$

Example: Blackjack

- **States (200 possible states):**
  - Current sum (12-21)
  - Dealer's showing card (ace-10)
  - Do I have a 'usable' ace (yes-no)
- **Actions:**
  - Stand: stop receiving cards (and terminate)
  - Hit: take another card (no replacement)
- **Reward:**
  - For stand:
    - +1 if sum of cards > sum of dealer cards
    - 0 if sum of cards = sum of dealer cards
    - -1 if sum of cards < sum of dealer cards
  - For hit:
    - -1 if sum of cards > 21 (and terminate)
    - 0 otherwise
- **Transitions:**
  - Automatically hit if sum of cards < 12
- **Policy:**
  - Stand if sum of cards ≥ 20, otherwise hit

Example: Blackjack



Small exercise:

1. Consider the diagrams on the right
   a. Why does the estimated value function jump up for the last two rows in the rear?
   b. Why does it drop off for the whole last row on the left?
2. Would you expect results to be different with EV-MC? Why or why not?

Example: Blackjack

The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j$$

$$= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right)$$

$$= \frac{1}{k}\left(x_k + (k-1)\mu_{k-1}\right)$$

$$= \mu_{k-1} + \frac{1}{k}\left(x_k - \mu_{k-1}\right)$$

- We incrementally update $\hat{V}(x)$ after every episode $\tau = (x_0, u_0, \ldots, x_N, u_N)$
- For each state x_t with return G_t

$$N(x_t) \leftarrow N(x_t) + 1$$

$$\hat{V}(x_t) \leftarrow \hat{V}(x_t) + \frac{1}{N(x_t)}\left(G_t - \hat{V}(x_t)\right)$$

- In non-stationary problems, it is often useful to track a running mean to forget old (and ultimately less relevant) episodes

$$\hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha\left(G_t - \hat{V}(x_t)\right)$$

Outline

What is Reinforcement Learning? (and the RL setting)

From exact methods to model-free control

- Monte Carlo Learning
- Temporal-Difference (TD) Learning

A taxonomy of RL algorithms & important trade-offs

Temporal-Difference Learning

- TD is a combination of Monte Carlo and Dynamic Programming ideas
- Like MC, TD is model-free: no knowledge of MDP transitions / rewards. TD can learn from experience
- Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they **bootstrap**)
- TD updates a guess towards a guess

Temporal-Difference Learning

- To compare MC and TD, let us consider the task of learning V_π from experience under policy π
- Incremental every-visit Monte Carlo:
  - Update value $\hat{V}(x_t)$ toward *actual* return G_t

$$\hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha\left(G_t - \hat{V}(x_t)\right)$$

- Temporal-difference algorithm:
  - Update value $\hat{V}(x_t)$ toward estimated return $R_t + \gamma\hat{V}(x_{t+1})$

$$\hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha\left(R_t + \gamma\hat{V}(x_{t+1}) - \hat{V}(x_t)\right)$$

  - $R_t + \gamma\hat{V}(x_{t+1})$ is called **TD target**
  - $\delta_t = R_t + \gamma\hat{V}(x_{t+1}) - \hat{V}(x_t)$ is called **TD error**

TD methods combine:

1) the sampling of Monte Carlo
2) with the bootstrapping of DP

Advantages and disadvantages of MC vs TD

- TD can learn *before* knowing the final outcome
  - TD can learn online after every step
  - MC must wait until the end of the episode

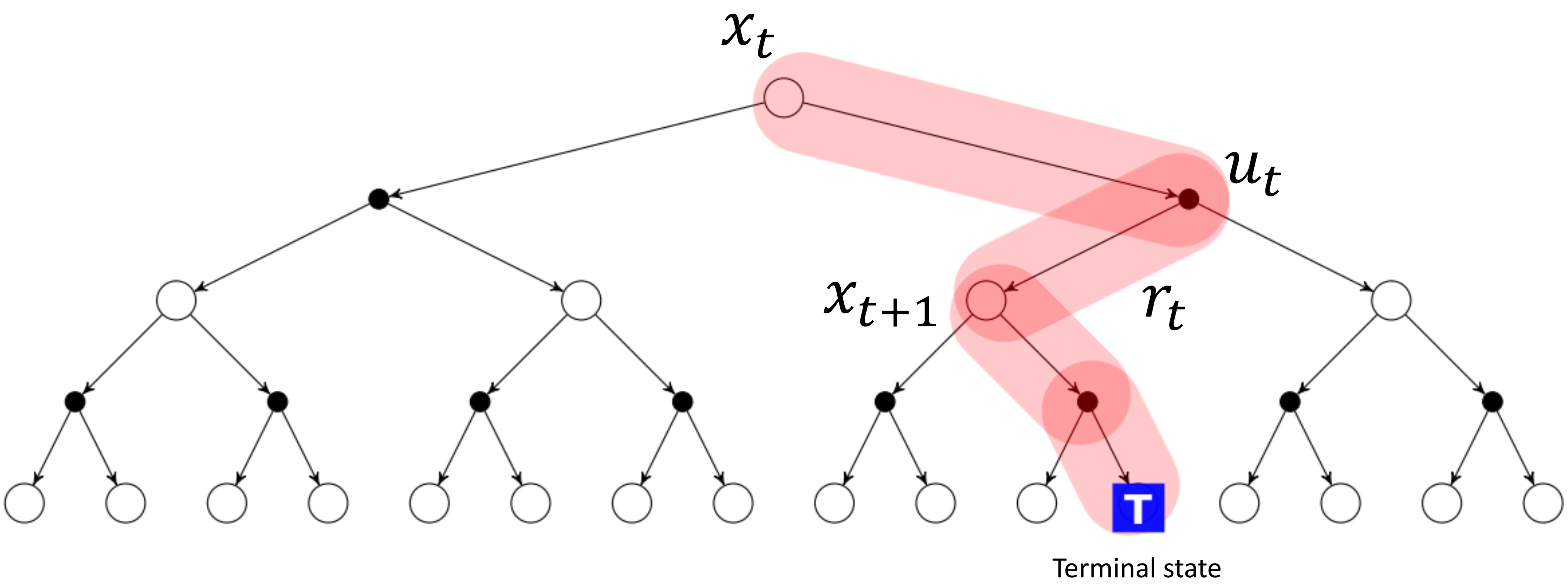
- TD can learn without the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works in episodic (terminating) environments

Bias-Variance Trade-off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$ is an unbiased estimate of $V_\pi(x)$
- In theory, the *true* TD target $R_t + \gamma V(x_{t+1})$ is also an unbiased estimate of $V_\pi(x)$
- TD target $R_t + \gamma \hat{V}(x_{t+1})$is a biased estimate of $V_\pi(x)$
- However, the TD target is much lower variance than the return
  - The return G_t depends on a **full sequence** of random actions, transitions, rewards (i.e., evaluated at the end of the episode)
  - The TD error only depends on **one** random action, transition, reward

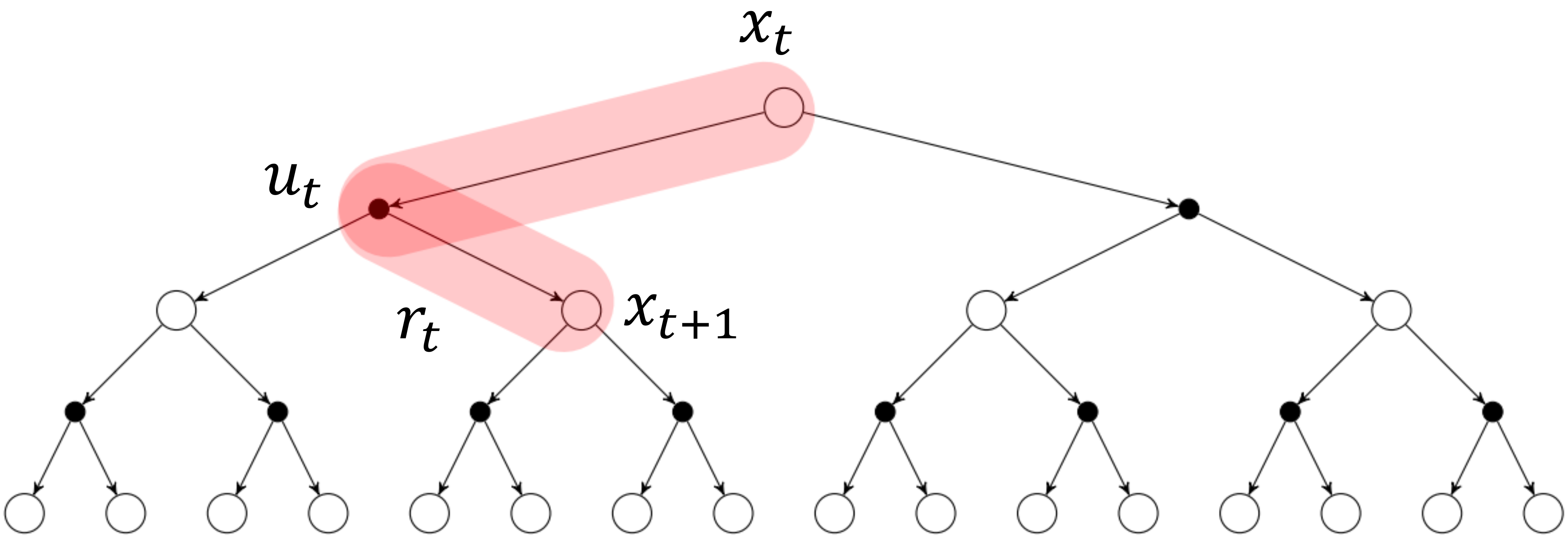
Monte-Carlo Backup

$$\hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha\left(G_t - \hat{V}(x_t)\right)$$



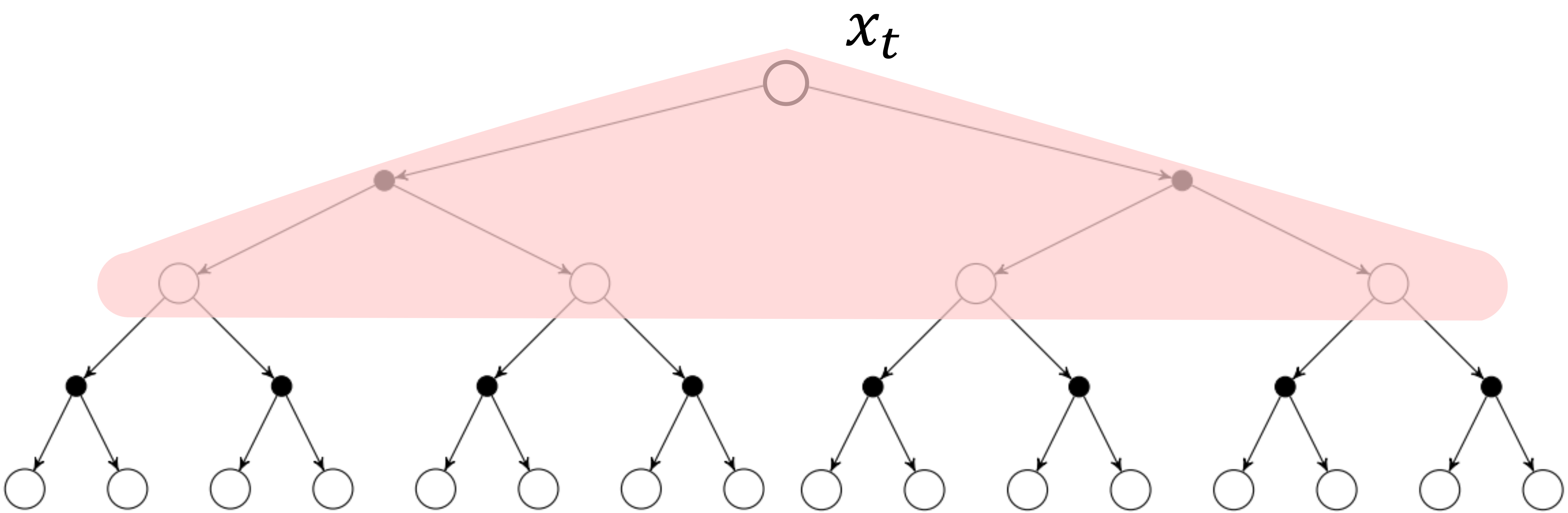
Temporal-Difference Backup

$$\hat{V}(x_t) \leftarrow \hat{V}(x_t) + \alpha\left(R_t + \gamma\hat{V}(x_{t+1}) - \hat{V}(x_t)\right)$$



Dynamic Programming Backup

$$\hat{V}(x_t) \leftarrow \mathbb{E}\left[R_t + \gamma \hat{V}(x_{t+1})\right]$$

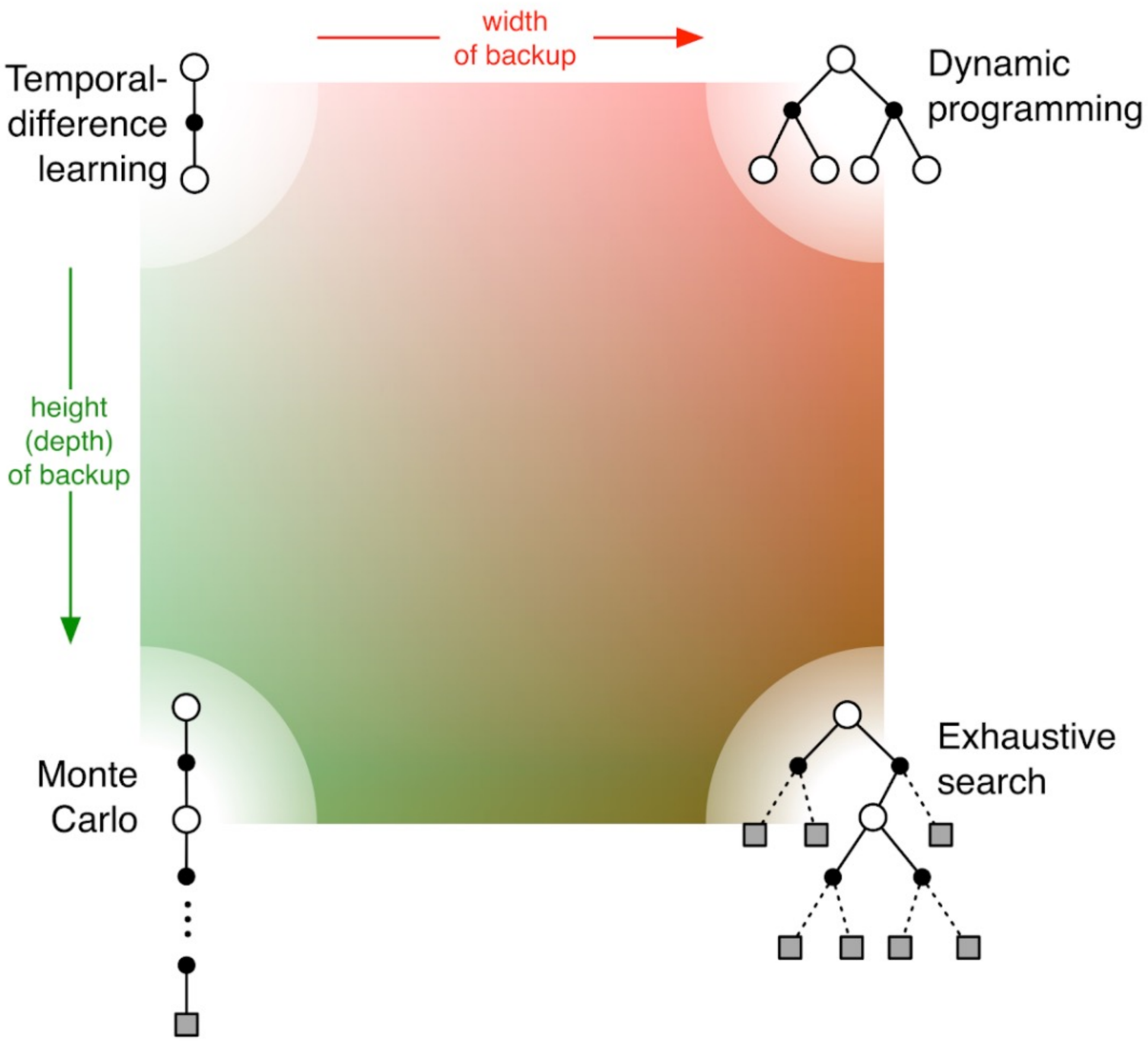


Bootstrapping and sampling

- **Sampling:** define the update through samples to approximate expectations
  - MC samples
  - TD samples
  - DP does not sample

- **Bootstrapping:** define the update through an estimate
  - MC does not bootstrap
  - TD bootstraps
  - DP bootstraps

A unifying view of RL



Outline

What is Reinforcement Learning? (and the RL setting)

From exact methods to model-free **control**

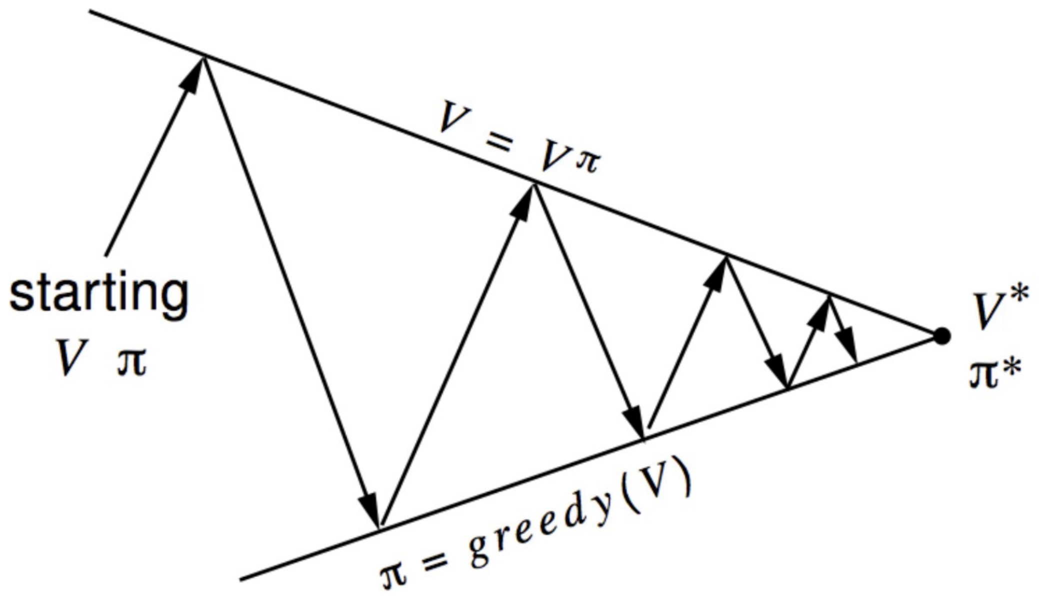
- Monte Carlo Learning
- Temporal-Difference (TD) Learning

A taxonomy of RL algorithms & important trade-offs

(Review) Generalized Policy Iteration

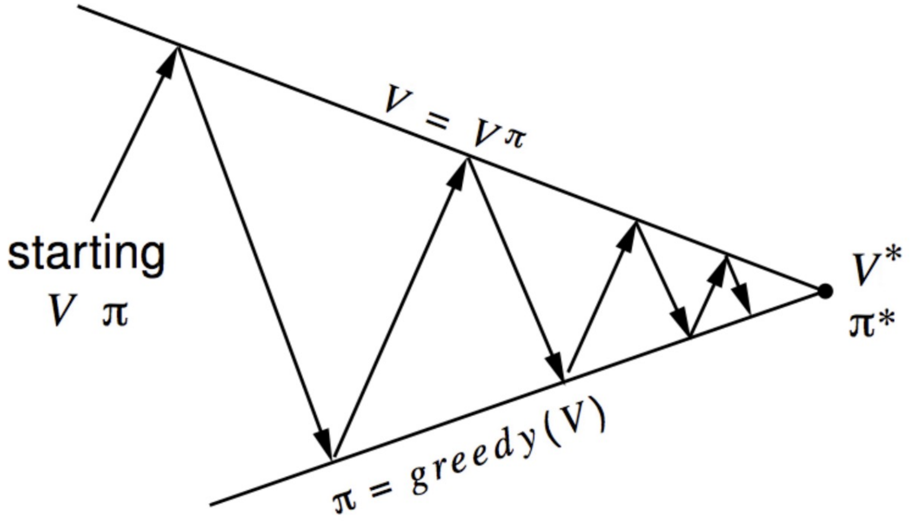
In previous lectures, we discussed Policy Iteration as consisting of two simultaneous, interactive processes: Policy **Evaluation** and Policy **Improvement**

We use the term *generalized policy iteration* (GPI) to refer to the general idea of letting policy-evaluation and policy improvement processes interact, independent of the granularity and other details of the two processes.



Policy **Evaluation:** Iterative policy evaluation
Policy **Improvement:** Greedy policy improvement

GPI with Monte-Carlo Evaluation



Policy **Evaluation:** Monte-Carlo policy evaluation of $V(x)$?

Policy **Improvement:** Greedy policy improvement?

Problem:

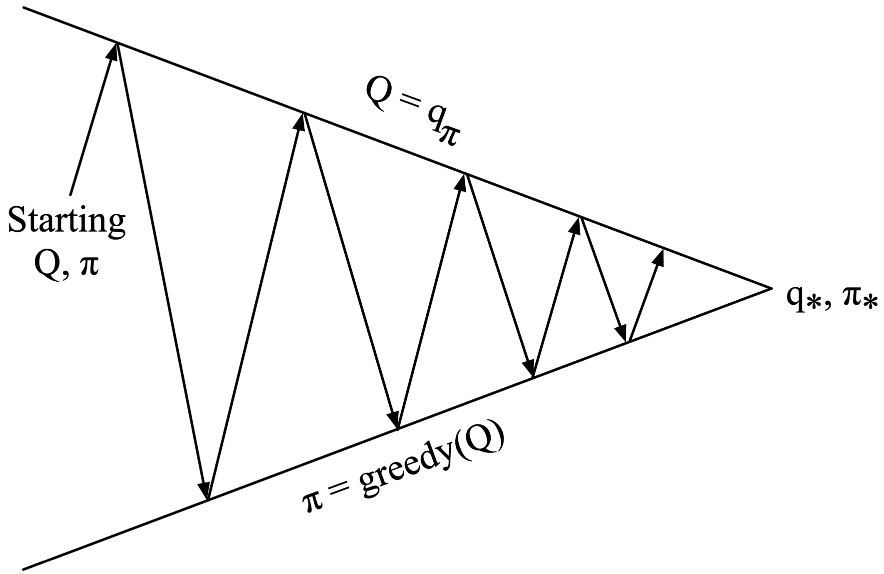
Greedy policy improvement over $V(x)$ requires a model of the MDP!

$$\pi_{k+1}(x) = \underset{u}{\mathrm{argmax}} \left(R(x,u) + \gamma \sum_{x_{t+1} \in \mathcal{X}} T(x_{t+1} \mid x_t, u_t) V_{k+1}(x_{t+1}) \right)$$

On the other hand, greedy policy improvement over $Q(x,u)$ does not

$$\pi_{k+1}(x) = \underset{u}{\mathrm{argmax}} Q(x,u)$$

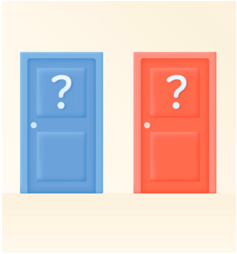
GPI with state-action value function



Policy **Evaluation:** Monte-Carlo policy evaluation of $Q(x, u)$

Policy **Improvement:** Greedy policy improvement?

Problem:



Exploration! Let's consider an example:

- Need to choose among two possible doors:
- You open the left door: $R = 0, V(left) = 0$
- You open the right door: $R = 1, V(right) = 1$
- You open the right door: $R = 3, V(right) = 2$
- You open the right door: $R = 2, V(right) = 2$
- …

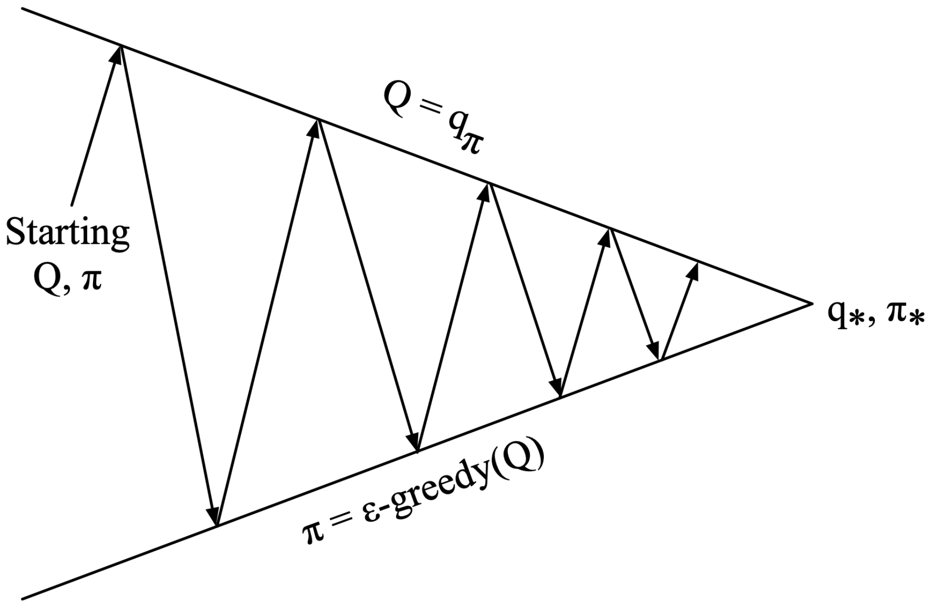
To estimate state-action values through samples, every **state-action pair** needs to be visited (opposed to each state as in MC estimation of $V(x)$)

Deterministic policies do not allow this exploration

A simple (but effective) strategy: ϵ-Greedy Exploration

- With probability $1-\epsilon$, choose the greedy action
- With probability ϵ, choose a random action
- Ensures that all m actions are tried with non-zero probability

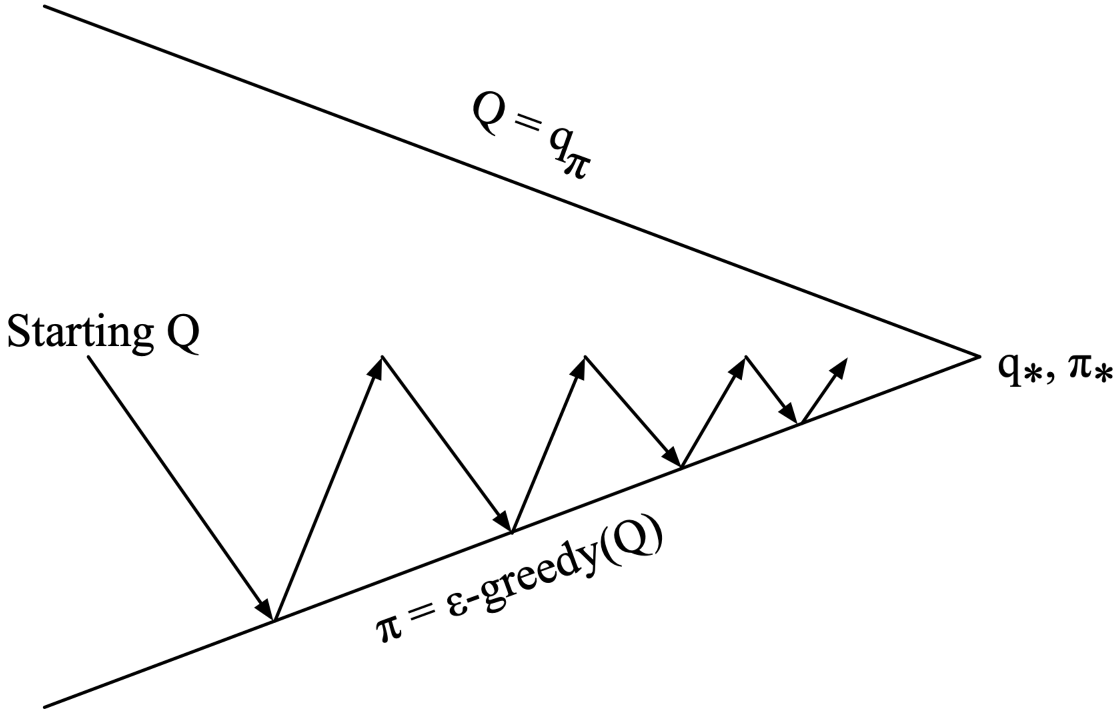
$$\pi(u \mid x) = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon & \text{if } u^* = \underset{u \in \mathcal{U}}{\operatorname{argmax}}\, Q(x,u) \\ \frac{\epsilon}{m} & \text{otherwise} \end{cases}$$



Policy **Evaluation:** Monte-Carlo policy evaluation of $Q(x,u)$

Policy **Improvement:** ϵ-Greedy policy improvement?

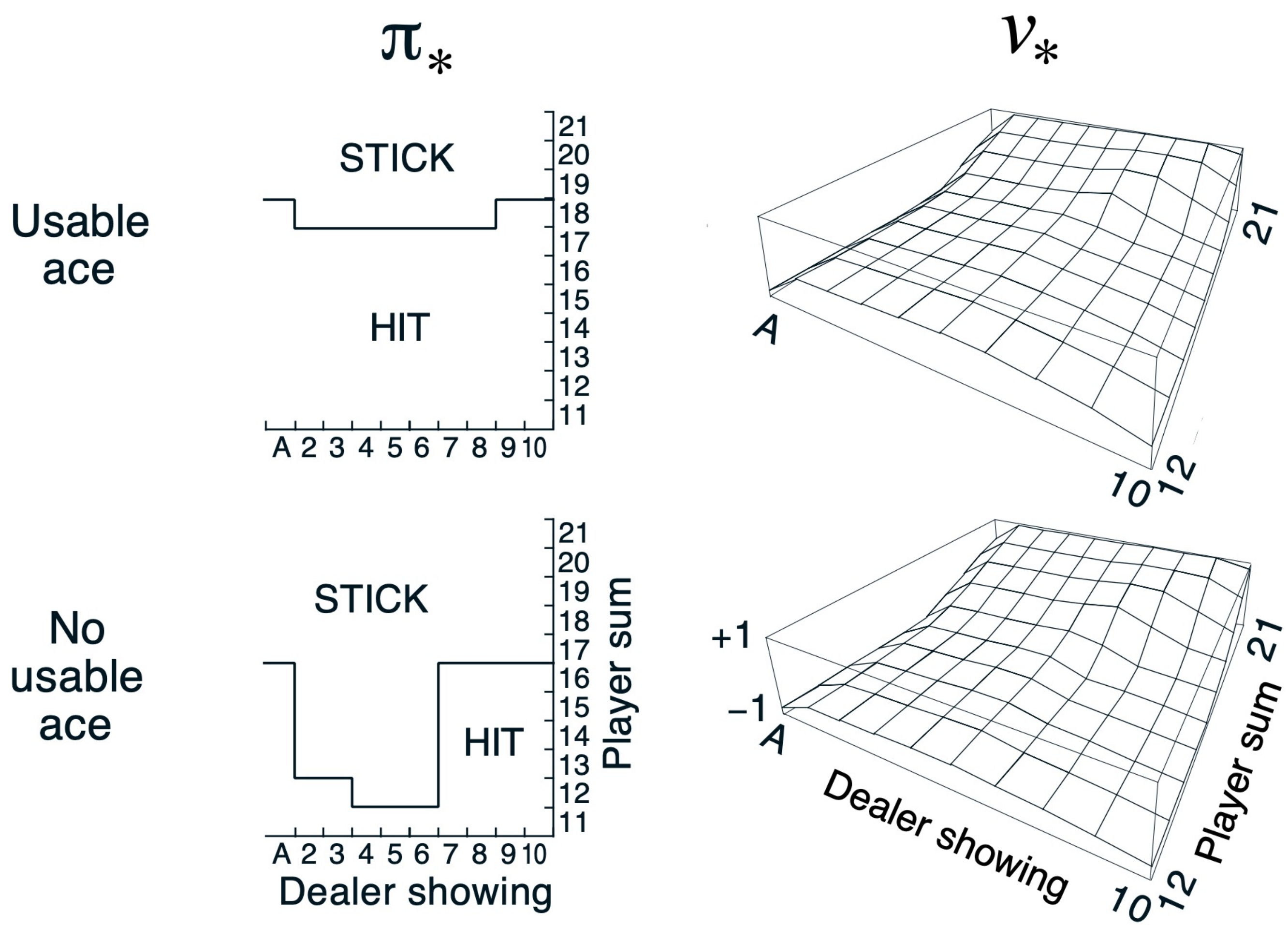
Monte-Carlo Control



Policy **Evaluation:** Monte-Carlo policy evaluation of $\hat{Q}(x, u) \approx Q(x, u)$

Policy **Improvement:** ϵ —Greedy policy improvement

Example: Blackjack

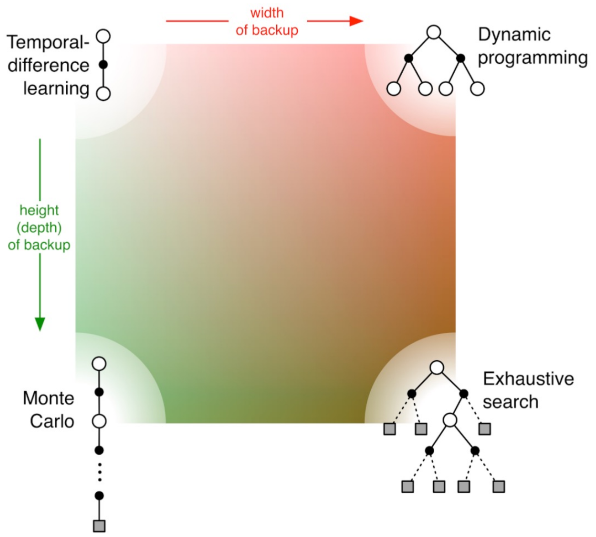


To recap…

We discussed the main limitations of exact methods (such as Policy/Value Iteration):

- Update equations (i.e., Bellman equations) require access to dynamics model $T(x_{t+1} \mid x_t, u_t)$ **Sampling-based approximations**
- Iteration over (and storage of) all states and actions requires small, discrete state-action space **Function approximation**

We introduced core ideas such as Monte-Carlo and Temporal-Difference Learning and derived ways to solve unknown MDPs



However, we did not discuss methods to deal with high-dimensional state/action spaces… more on this later!

Outline

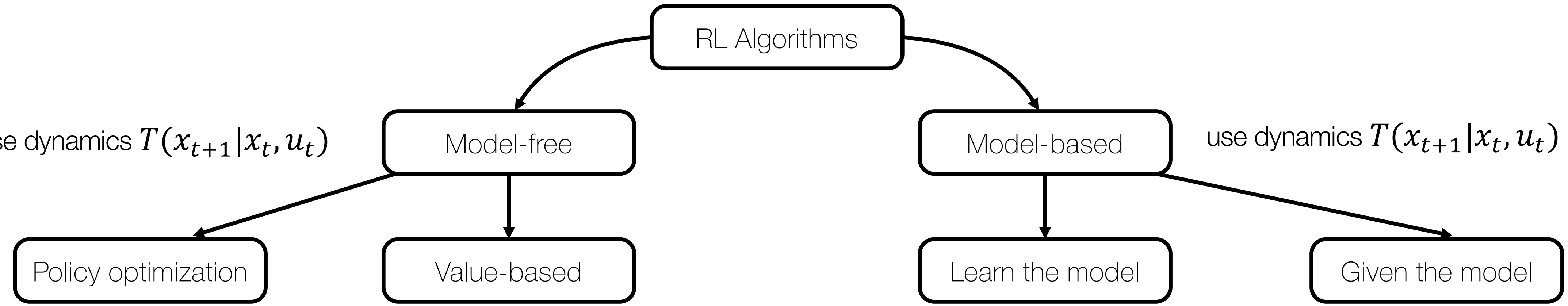
What is Reinforcement Learning? (and the RL setting)

From exact methods to model-free control

- Monte Carlo Learning
- Temporal-Difference (TD) Learning

A taxonomy of RL algorithms & important trade-offs

A taxonomy of RL



do not use dynamics $T(x_{t+1}|x_t, u_t)$

use dynamics $T(x_{t+1}|x_t, u_t)$

Policy optimization: directly maximize the RL objective

$$\mathbb{E}_{\tau\sim p_\pi(\tau)}\left[\sum_{t=0}^{H}\gamma^t r(x_t, u_t)\right]$$

Value-based: policy implicitly defined via $V(x)$ or $Q(x, u)$

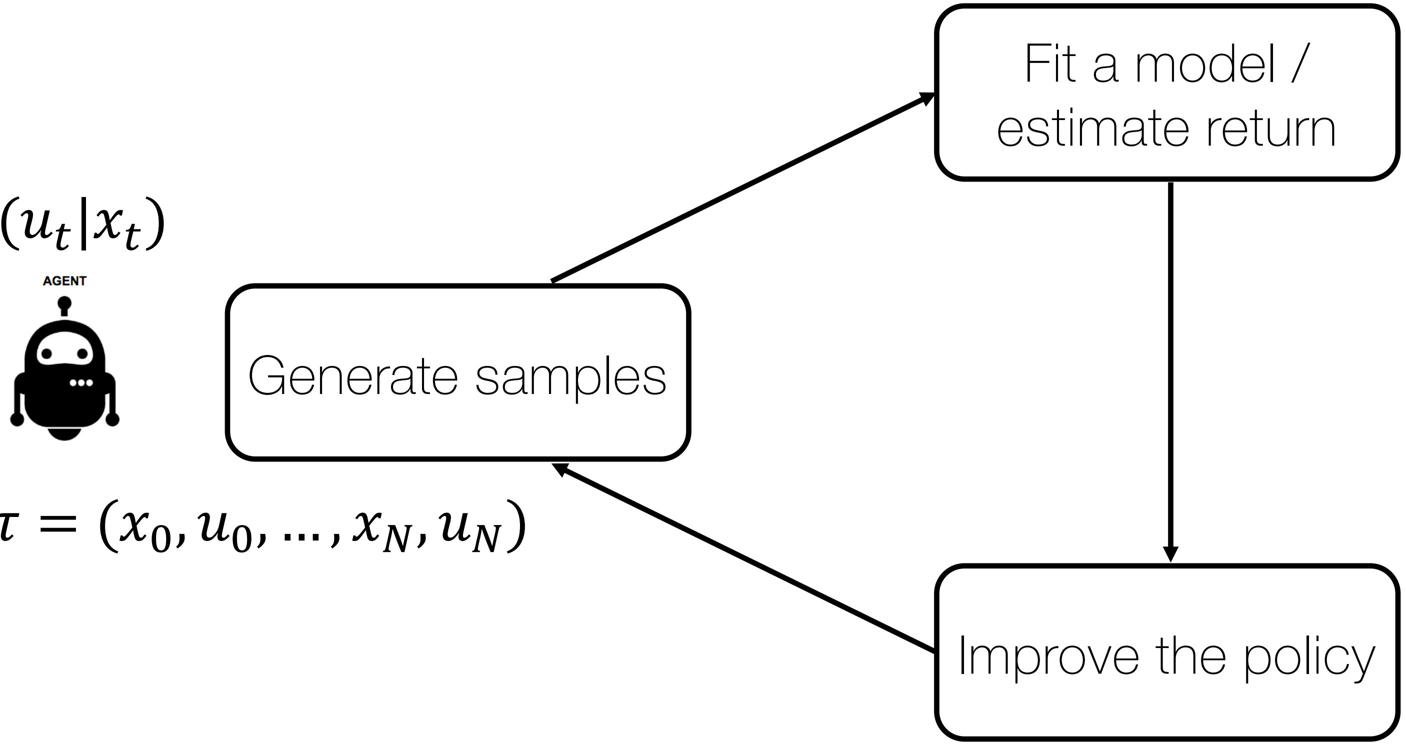
$$\pi(x_t) = \arg\max_u Q(x_t, u_t)$$

Learn the model: Estimate $f_\theta \approx T(x_{t+1}|x_t, u_t)$

Given the model: $T(x_{t+1}|x_t, u_t)$ is known

The skeleton of an RL algorithm

$\pi(u_t|x_t)$



$\tau = (x_0, u_0, \ldots, x_N, u_N)$

$f_\theta(x_t) \approx V^\pi(x_t)$

$f_\theta(x_t, u_t) \approx Q^\pi(x_t, u_t)$

$f_\theta(x_t, u_t) \approx P(x_{t+1} \mid x_t, u_t)$

set $\pi(x_t) = \arg\max_a Q(x_t, u_t)$ (e.g., Q-learning, DQN)

$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}\left[\sum_t r(x_t, u_t)\right]$ (e.g., PG, A2C, A3C)

Why so many RL algorithms?

- **Different tradeoffs:**
  - Sample efficiency
  - Stability & easy of use

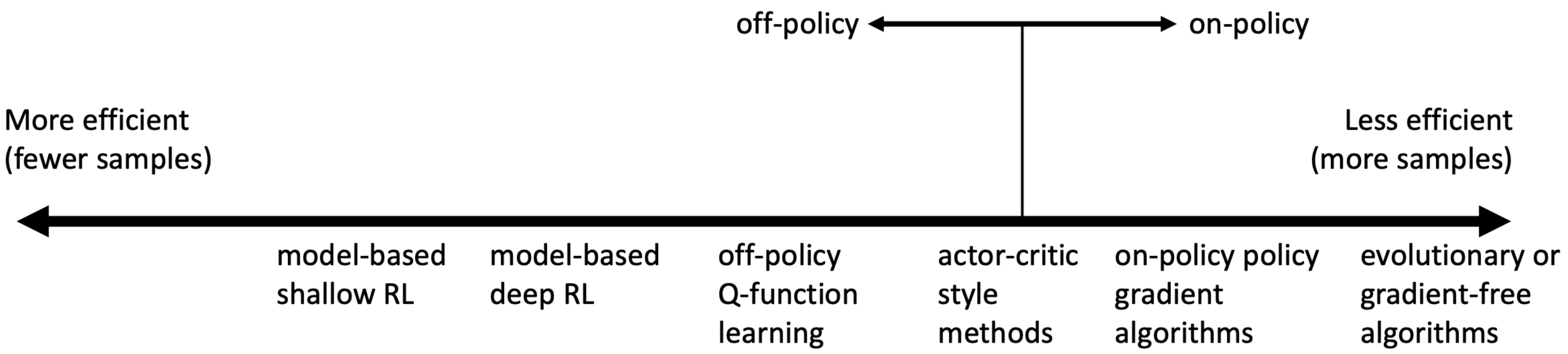
- **Different assumptions:**
  - Stochastic or deterministic
  - Continuous or discrete
  - Episodic or infinite horizon

- **Different things are easy or hard in different settings:**
  - Easier to represent the policy?
  - Easier to represent the model?

Comparison: sample efficiency

- Sample efficiency = how many samples do we need to get a good policy?

- Crucial question: is the algorithm *off policy?*
  - **Off policy**: able to improve the policy without generating new samples from the current policy
  - **On policy**: each time the policy is changed, even a little bit, we need to generate new samples



Why even bother using less efficient algorithms? Wall-clock time is not the same as efficiency!

Comparison: stability and ease of use

- Does it converge?
- And if it does, to what?
- Does it *always* converge?

- Supervised learning: almost always gradient descent
- Reinforcement learning: often not gradient descent
  - Q-learning: fixed point iteration
  - Model-based RL: model estimator is not optimized for expected reward

Next time

- HJB, HJI, and reachability analysis