Reinforcement Learning

CS 59300: RL1

September 18, 2025

Joseph Campbell
Department of Computer Science

Today's lecture

1. Overestimation bias in Q-learning

2. Other tricks and Rainbow

3. Policy Gradient

Some content inspired by David Silver's UCL RL course and Katerina Fragkiadaki's CMU 10-403

Recap: Max as inner optimization

The ”simple” method for finding $\max_{a' \in \mathcal{A}}$ is to perform optimization

Cross-Entropy Method

- Start with a randomly initialized normal distribution
- Sample actions from it
- Select top-K actions sorted by $Q(s, a)$
- Fit distribution to top-K samples
- Repeat

Recap: QT-Opt

Goal: use stochastic optimization to find target $Q_T(s_{t+1}, a')$

for episode = 1…M do
  for t=1…T do
    Perform CEM to find $a_t = \max_a \hat{Q}\ (s_t, a; \theta)$
    With probability ϵ select random action else a_t
    Execute action a_t and observe r_t and s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D and sample random minibatch
    Set $y_j = r_j$ if episode ends else $r_j + \gamma \max_{a'} \hat{Q}\ \{\ (s_{j+1}, a'; \theta)$
    Perform a gradient step on $(y_j - \hat{Q}(\phi_j, a_j; \theta))^2$

Kalashnikov et al, *QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation*, 2018.

Recap: A more sophisticated solution

We can derive Q-values using a different equation!

Advantage

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

The advantage represents how good action a is relative to π

- $A_\pi(s,a) > 0$: a is better than what I would get with π
- $A_\pi(s,a) < 0$: a is worse

Recap: How does this apply to continuous actions?

Start with dueling DQN, and further decompose the advantage

$$A_\pi(s,a) = -\frac{1}{2}\left(a-\mu(s)\right)^T P(s)\left(a-\mu(s)\right)$$

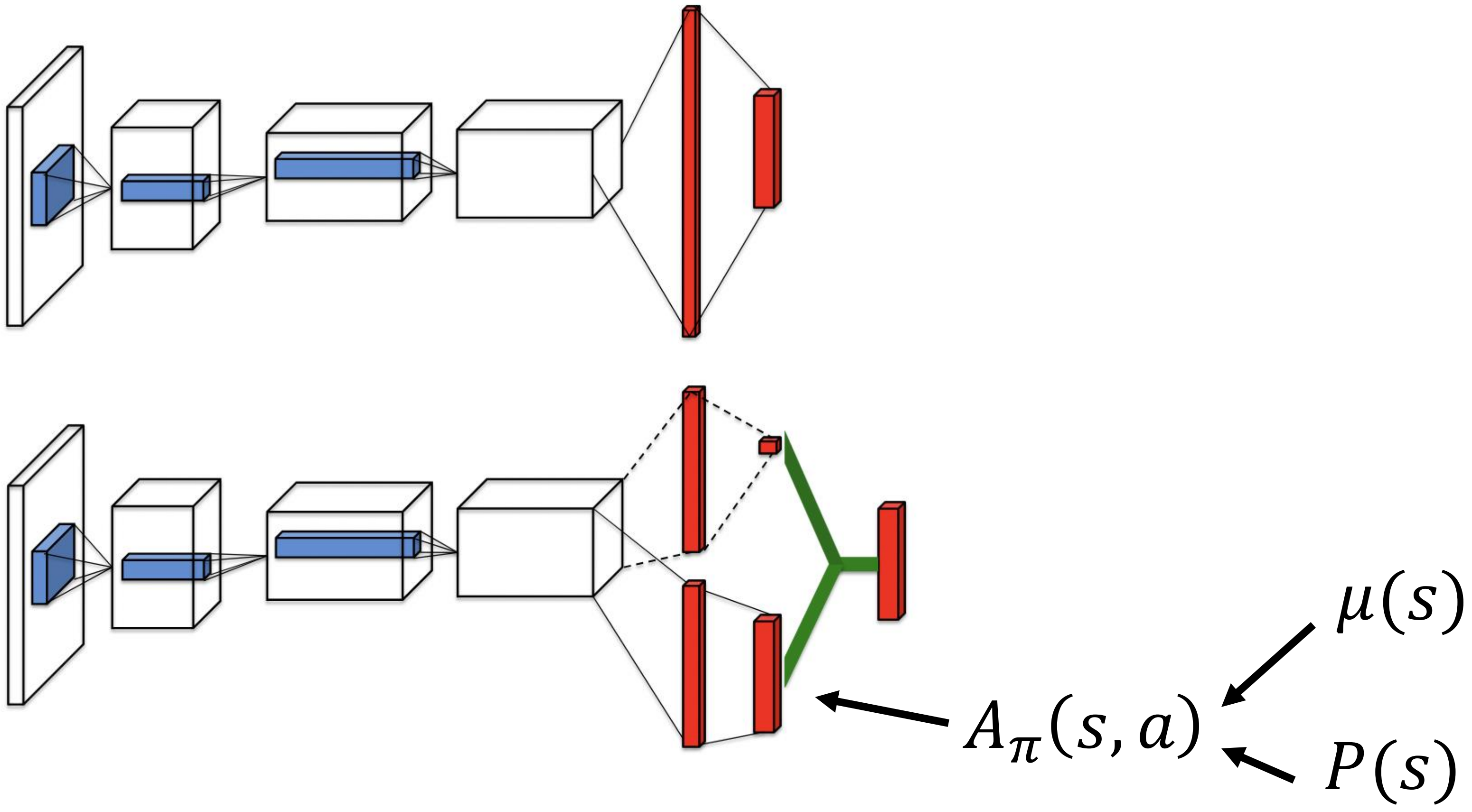
Positive-definite square matrix.
Obtained via Cholesky decomposition: $L(s)L(s)^T$

Assumption: quadratic dynamics and linear rewards

- The advantage is parameterized as a quadratic function

Gu et al, Continuous Deep Q-Learning with Model-based Acceleration, 2016.

Recap: Decomposed advantage



Overestimation bias in Q-learning

Recall that in Q-learning…

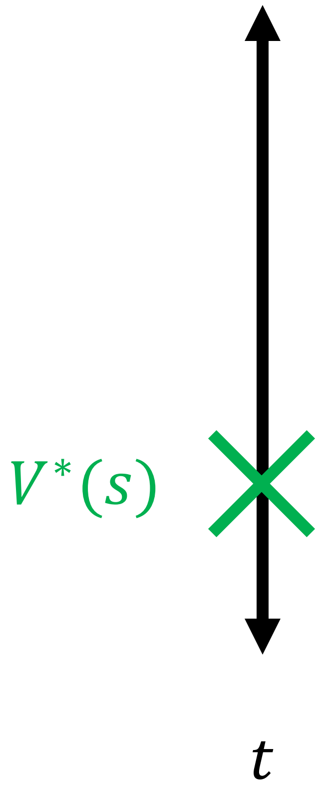
We take the action with the maximum Q-value

$$Q(s, a) = Q(s, a) + \alpha\,(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a') - Q(s, a))$$

What happens if there is noise in our estimate?

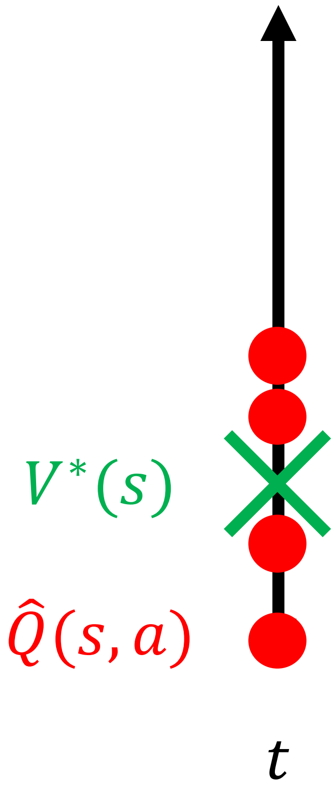
Consider uniformly random errors

Errors drawn from $\mathcal{U}[-1,1]$



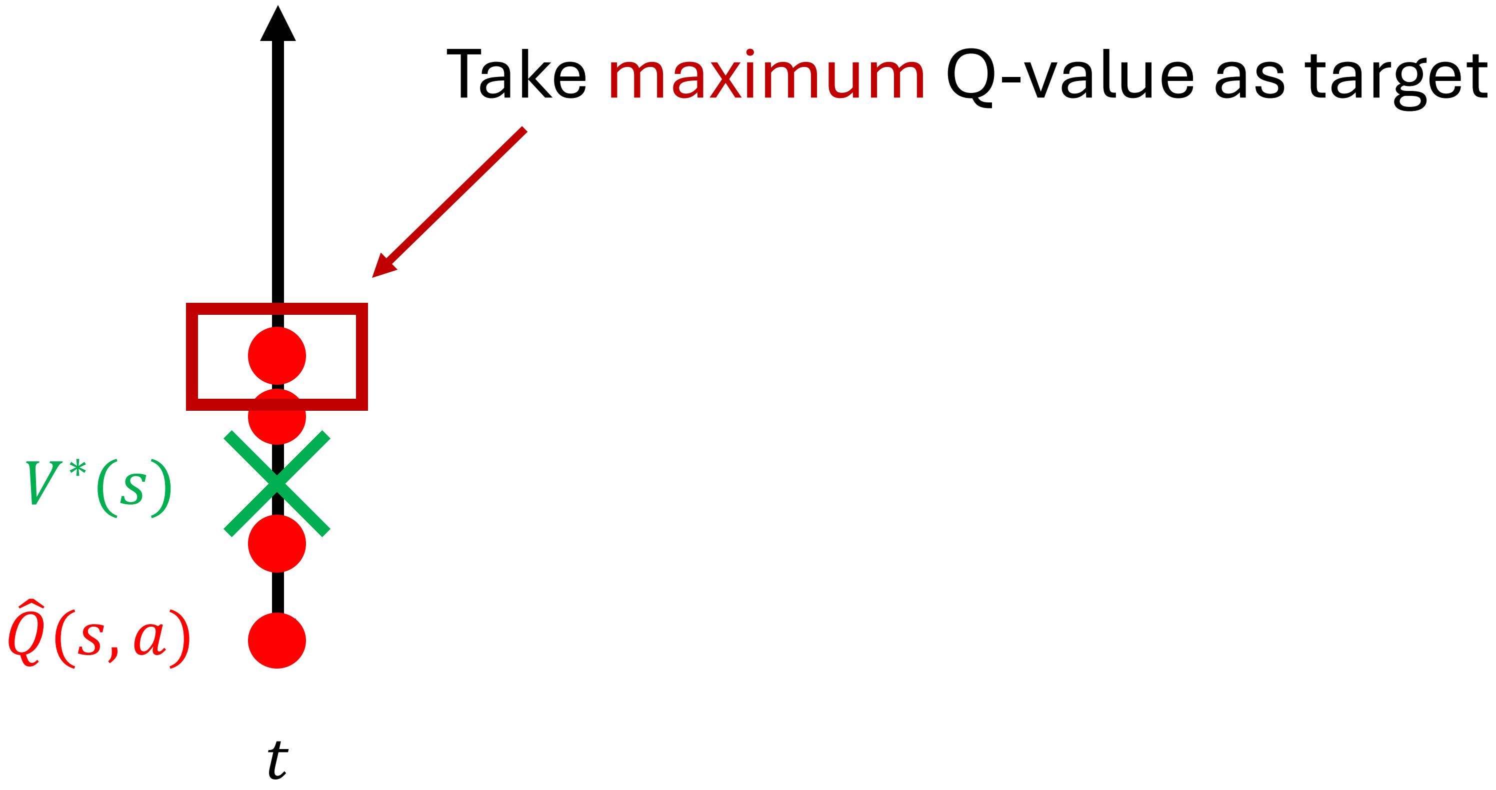
Consider uniformly random errors

Errors drawn from $\mathcal{U}[-1,1]$



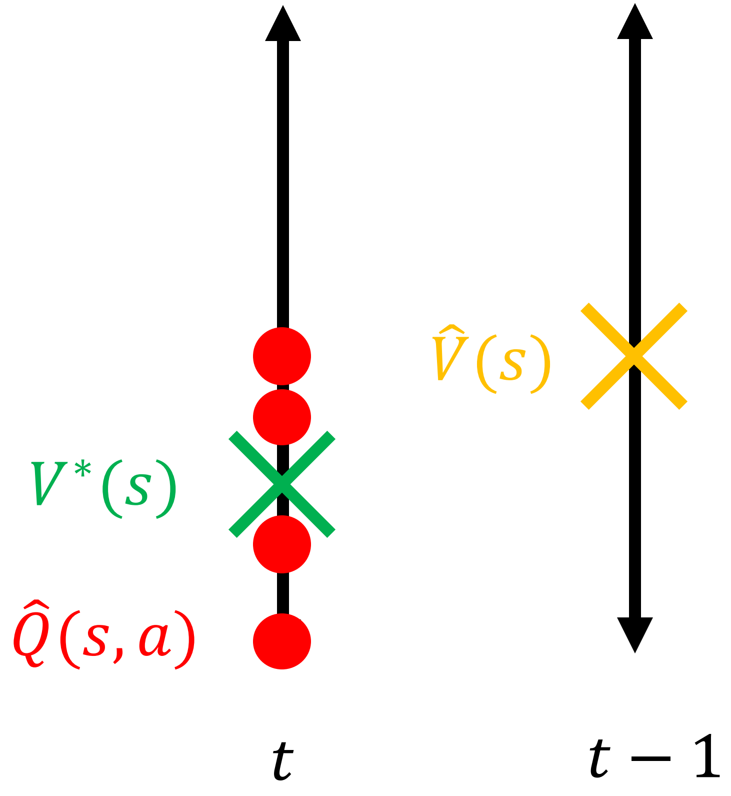
Consider uniformly random errors

Errors drawn from $\mathcal{U}[-1,1]$



Consider uniformly random errors

Errors drawn from $\mathcal{U}[-1,1]$

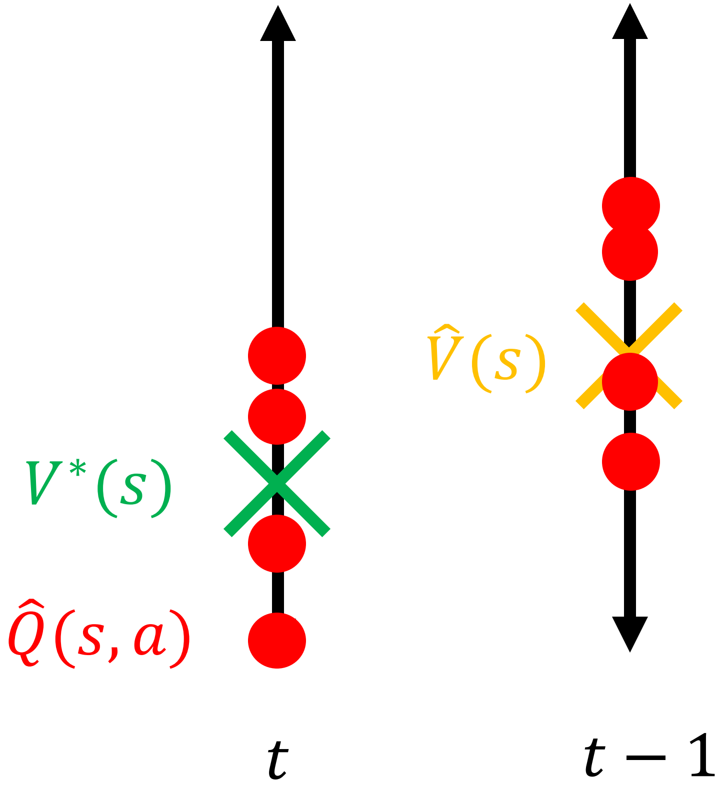


We bootstrap off of this value!

This means we think the best Q-value is higher than it actually is...

Consider uniformly random errors

Errors drawn from $\mathcal{U}[-1,1]$

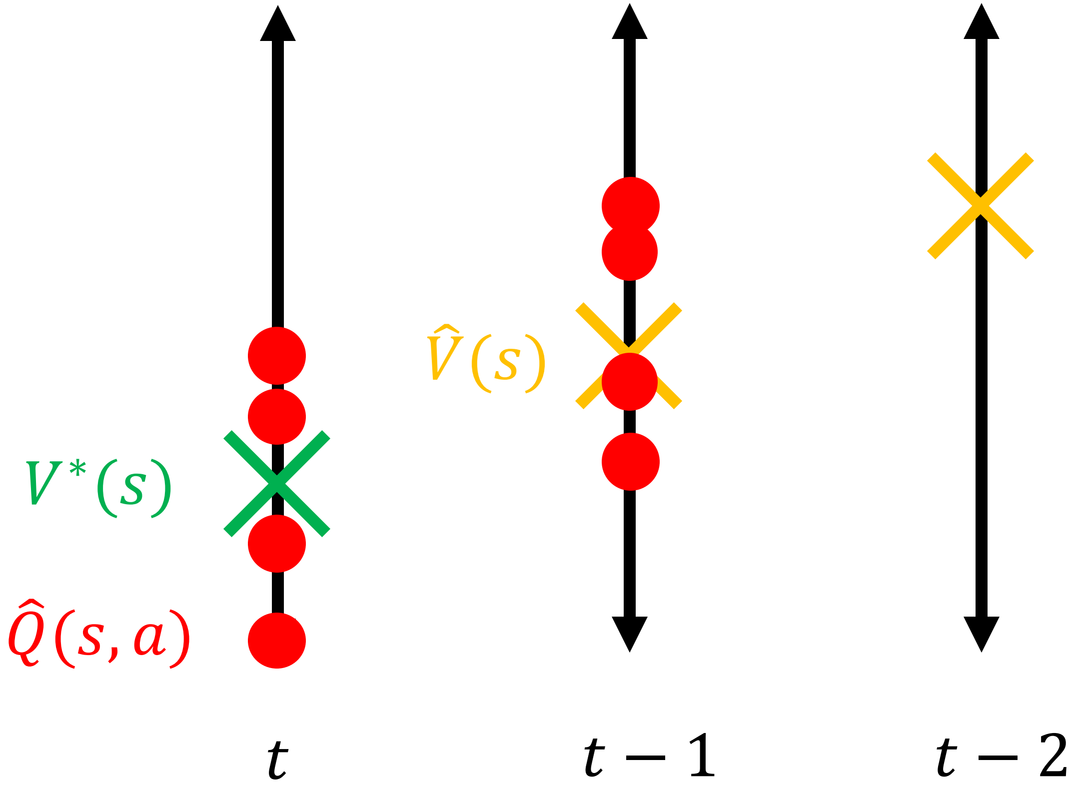


We bootstrap off of this value!

This means we think the best Q-value is higher than it actually is…

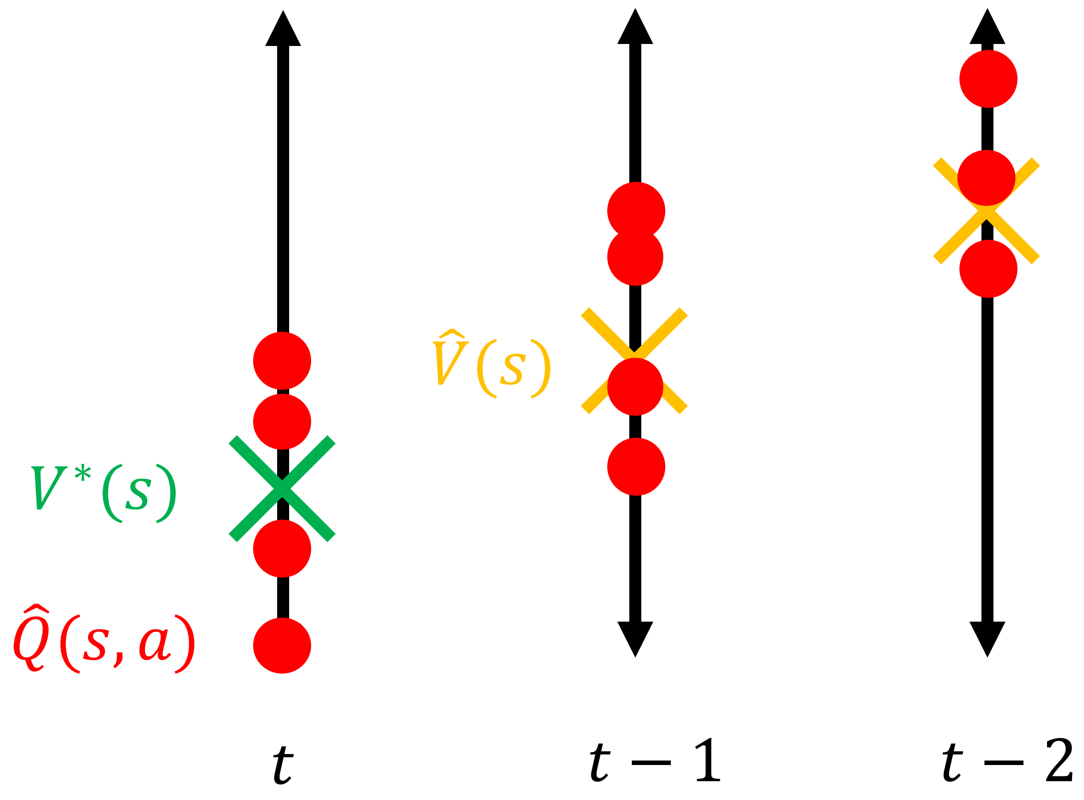
Consider uniformly random errors

Errors drawn from $\mathcal{U}[-1,1]$



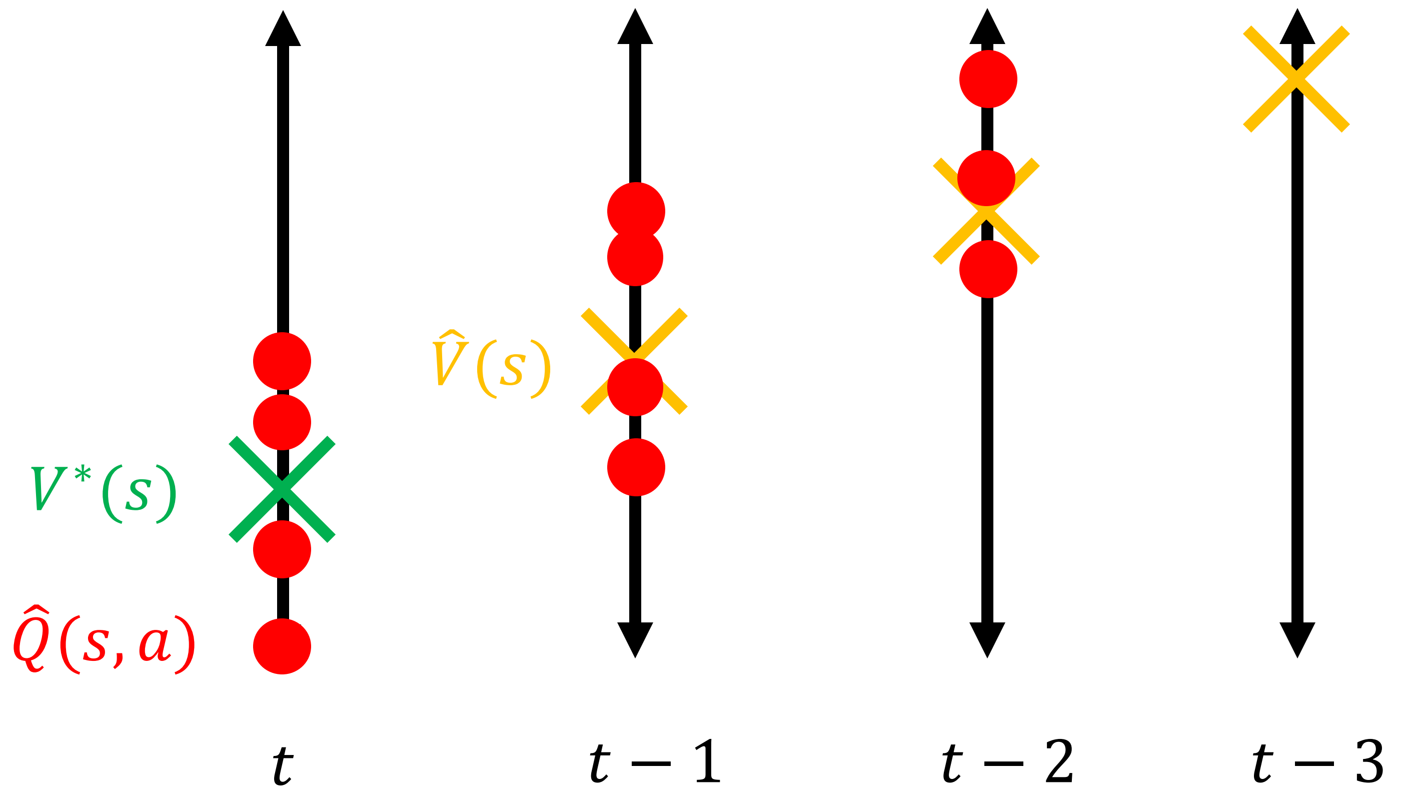
Consider uniformly random errors

Errors drawn from $\mathcal{U}[-1,1]$



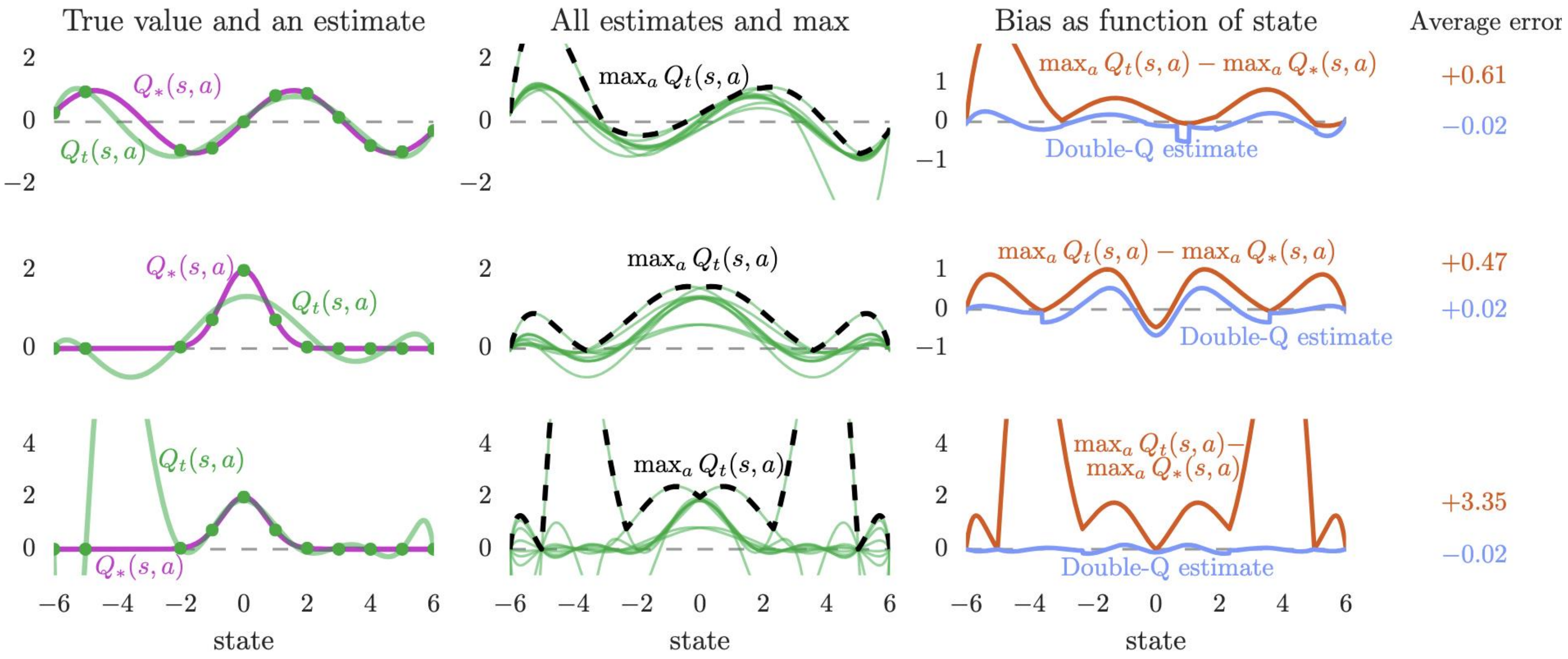
Consider uniformly random errors

Errors drawn from $\mathcal{U}[-1,1]$



Uniformly random errors lead to overestimation

This is exacerbated by bootstrapping in TD updates



For uniformly random errors in $[-\epsilon, \epsilon]$

Over-estimation error is $\frac{m-1}{m+1}$ for m actions

- The error increases as m increases. **Why?**

Where do these errors come from?

- Environmental noise
- Function approximation
- Reward non-stationarity
- …

If all states/actions are *uniformly* over-estimated, not an issue

- Because relative ranking of states/actions is preserved

But in practice errors differ across states/actions

Double DQN

Decompose $\max$ into action selection and action evaluation

$$Q(s,a) = Q(s,a) + \alpha\left(\underbrace{r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q^{\mathrm{Tar.}}(s_{t+1}, a')}_{\text{TD Target}} - Q(s,a)\right)$$

Double DQN

Decompose max into action selection and action evaluation

$$Q(s,a) = Q(s,a) + \alpha\left(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q^{\text{Tar.}}(s_{t+1}, a') - Q(s,a)\right)$$

$$\underbrace{r_{t+1} + \gamma Q^{\text{Tar.}}(s_{t+1}, \overbrace{\operatorname*{argmax}_{a' \in \mathcal{A}} Q(s_{t+1}, a')}^{\text{Action selection}})}_{\text{Action evaluation}}$$

van Hasselt et al, Deep Reinforcement Learning with Double Q-learning, 2015.

Double DQN

$$r_{t+1} + \gamma Q^{\text{Tar.}}(s_{t+1}, \operatorname*{argmax}_{a' \in \mathcal{A}} Q(s_{t+1}, a'))$$

Uses target network

Uses regular network

Intuition: when action selection and action evaluation use the same estimator, there is provable positive error bias.

If the two steps use independent estimators, evaluation is unbiased.

Decorrelating error terms

A single estimator is biased the same noise chooses the “winning” action we take and how much we value it.

- We choose the maximum of noisy estimates

Using two independent estimators decorrelates the noise!

- The “winning” action is evaluated using separate noise values
- One estimator might over-value one action, but the other might under-value it, because noise is uncorrelated
- Reduces to a weighted estimate of unbiased expected values

Decorrelating error terms

A single estimator is biased the same noise chooses the “winning” action we take and how much we value it

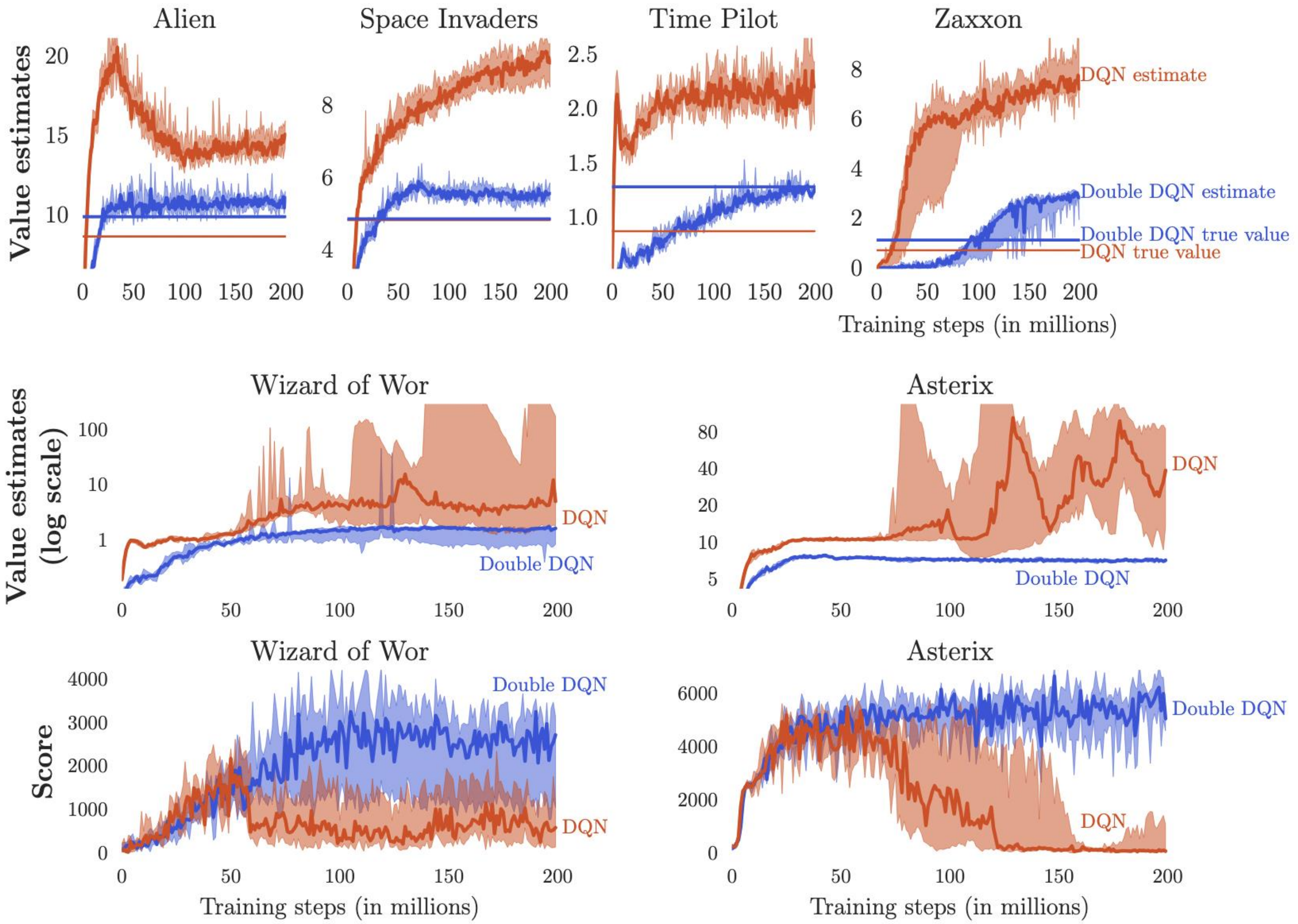
- We ch

Proof can be found in:

Hado van Hasselt, *Double Q-learning*, 2010.

Using tw

- The “w ues
- One estimator might over-value one action, but the other might under-value it, because noise is uncorrelated
- Reduces to a weighted estimate of unbiased expected values



No fun videos unfortunately

van Hasselt et al, Deep Reinforcement Learning with Double Q-learning, 2015.

Other tricks and Rainbow

Recap: Experience replay

Instead of using consecutive values, store all samples in a buffer

During training, we randomly sample experiences from the buffer

- Helps approximate i.i.d. assumption (any 2 samples are unlikely to be strongly correlated)
- Breaks temporal correlations and stabilizes training

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights

for episode = 1…M do

    Initialize sequence $s_1 = \{x_1\}$ and pre. seq. $\phi_1 = \phi(s_1)$

    for t=1…T do

        With probability ϵ select a random action a_t

            otherwise $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action a_t and observe r_t and x_{t+1}

        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and pre. $\phi_{t+1} = \phi(s_{t+1})$

        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

        Sample random minibatch from D

        Set $y_j = r_j$ if episode ends else $r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta)$

        Perform a gradient step on $(y_j - Q(\phi_j, a_j; \theta))^2$

Previous experience is uniformly sampled

Is this a good idea?

Why should I treat all previous experience as equal?

What if I mastered one skill, but not another?

Idea: Prioritize states that have high error

When updating our Q-network, we only care about minimizing error

- If error for a certain state-action is 0, why sample it from buffer?

Suppose want to weight our sampling distribution to avoid this

How?

- *Hint: what readily-available error can we use?*

Prioritized Experience Replay

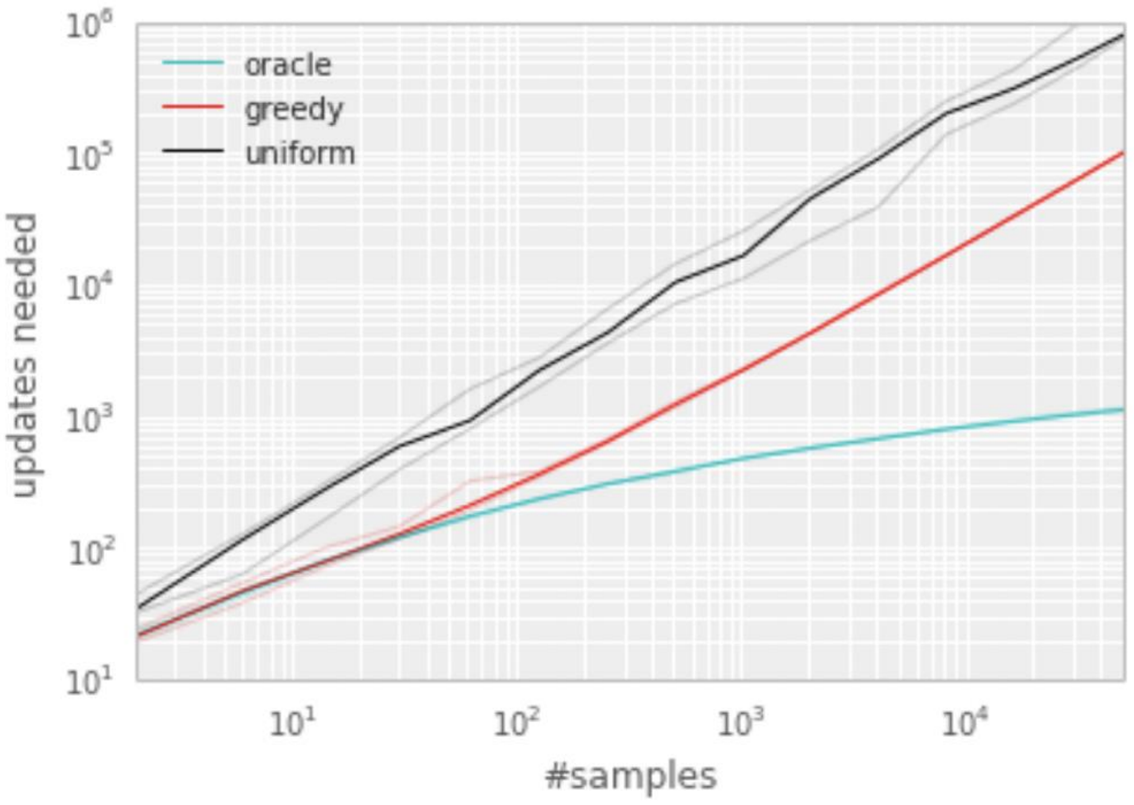
Solution: weigh sampling probability by its TD error

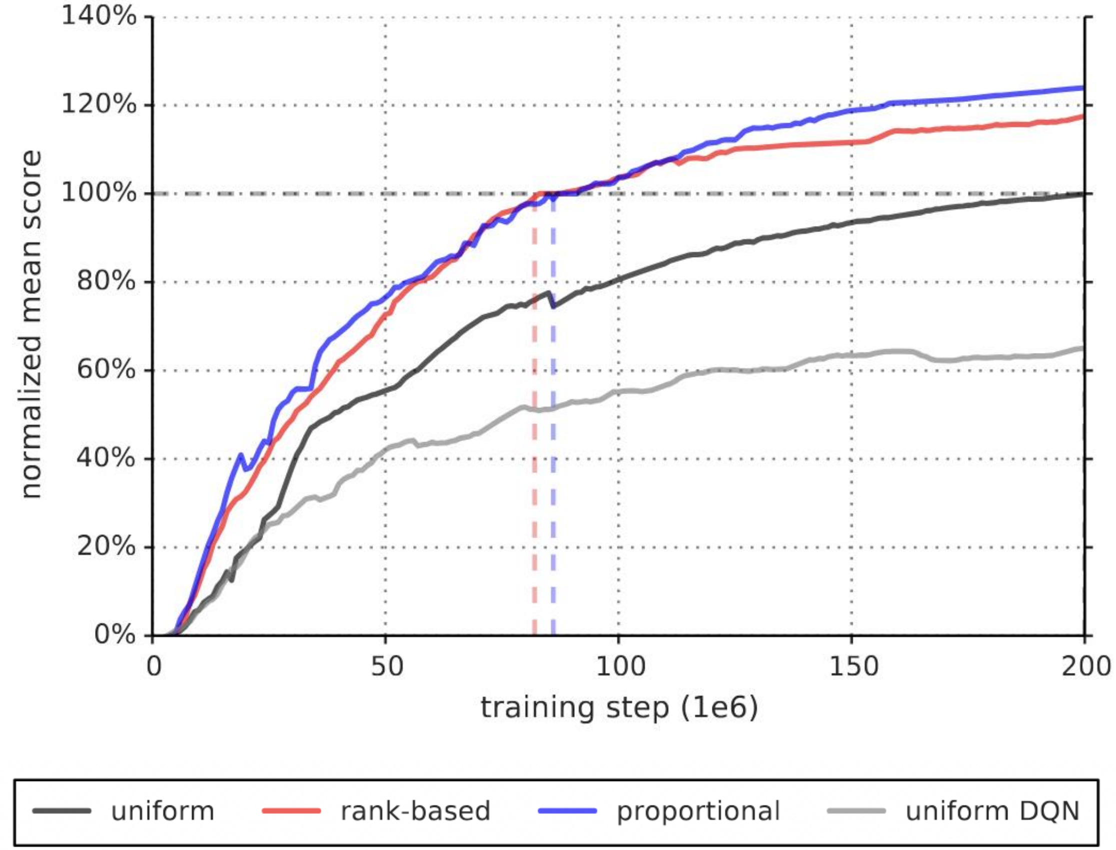
$$\text{Priority} = |\underbrace{r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q\,(s_{t+1}, a')}_{\text{TD Target}} - Q(s,a)|$$

$$\text{Probability} = \frac{\text{Priority}_i}{\sum_k \text{Priority}_k}$$

Schaul et al, Prioritized Experience Replay, 2016

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode = 1…M do
    Initialize sequence $s_1 = \{x_1\}$ and pre. seq. $\phi_1 = \phi(s_1)$
    for t=1…T do
        With probability ϵ select a random action a_t
            otherwise $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t and observe r_t and x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and pre. $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        Sample random minibatch from D according to probability
        Set $y_j = r_j$ if episode ends else $r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta)$
        Perform a gradient step on $(y_j - Q(\phi_j, a_j; \theta))^2$





Schaul et al, Prioritized Experience Replay, 2016

Rainbow

Combines 6 different DQN improvements

- Double Q-learning
- Prioritized replay
- Dueling networks
- Multi-step learning
- Distributional RL
- Noisy nets

The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)

Rainbow: Combining Improvements in Deep Reinforcement Learning

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, David Silver

DeepMind

Abstract

The deep reinforcement learning community has made several independent improvements to the DQN algorithm. However, it is unclear which of these extensions are complementary and can be fruitfully combined. This paper examines six extensions to the DQN algorithm and empirically studies their combination. Our experiments show that the combination provides state-of-the-art performance on the Atari 2600 benchmark, both in terms of data efficiency and final performance. We also provide results from a detailed ablation study that shows the contribution of each component to overall performance.

Introduction

The many recent successes in scaling reinforcement learning (RL) to complex sequential decision-making problems were kick-started by the Deep Q-Networks algorithm (DQN; Mnih et al. 2013, 2015). Its combination of Q-learning with convolutional neural networks and experience replay enabled it to learn, from raw pixels, how to play many Atari games at human-level performance. Since then, many extensions have been proposed that enhance its speed or stability.

Double DQN (DDQN; van Hasselt, Guez, and Silver 2016) addresses an overestimation bias of Q-learning (van Hasselt 2010), by decoupling selection and evaluation of the bootstrap action. Prioritized experience replay (Schaul et al. 2015) improves data efficiency, by replaying more often transitions from which there is more to learn. The dueling network architecture (Wang et al. 2016) helps to generalize across actions by separately representing state values and action advantages. Learning from multi-step bootstrap targets (Sutton 1988; Sutton and Barto 1998), as used in A3C (Mnih et al. 2016), shifts the bias-variance tradeoff and helps to propagate newly observed rewards faster to earlier visited states. Distributional Q-learning (Bellemare, Dabney, and Munos 2017) learns a categorical distribution of discounted returns, instead of estimating the mean. Noisy DQN (Fortunato et al. 2017) uses stochastic network layers for exploration. This list is, of course, far from exhaustive.

Each of these algorithms enables substantial performance improvements in isolation. Since they address radically different issues, and since they build on a shared framework, they could plausibly be combined. In some cases this has been done: Prioritized DDQN and Dueling DDQN both use double Q-learning, and Dueling DDQN was also combined with prioritized replay. In this paper we propose to study an agent that combines all the aforementioned ingredients. We show how these different ideas can be integrated, and that they are indeed complementary. In fact, their combination results in new state-of-the-art results on the benchmark suite of 57 Atari 2600 games from the Arcade Learning Environment (Bellemare et al. 2013), both in terms of data efficiency and of final performance. Finally, we show results from ablation studies to help understand the contributions of the individual components.

Figure 1: Median human-normalized performance across 57 Atari games. We compare Rainbow (rainbow-colored) to DQN and six published baselines. We match DQN's best performance after 7M frames, surpass any baseline in 44M frames, reaching substantially improved final performance. Curves are smoothed with a moving average of 5 points.

We have covered these!

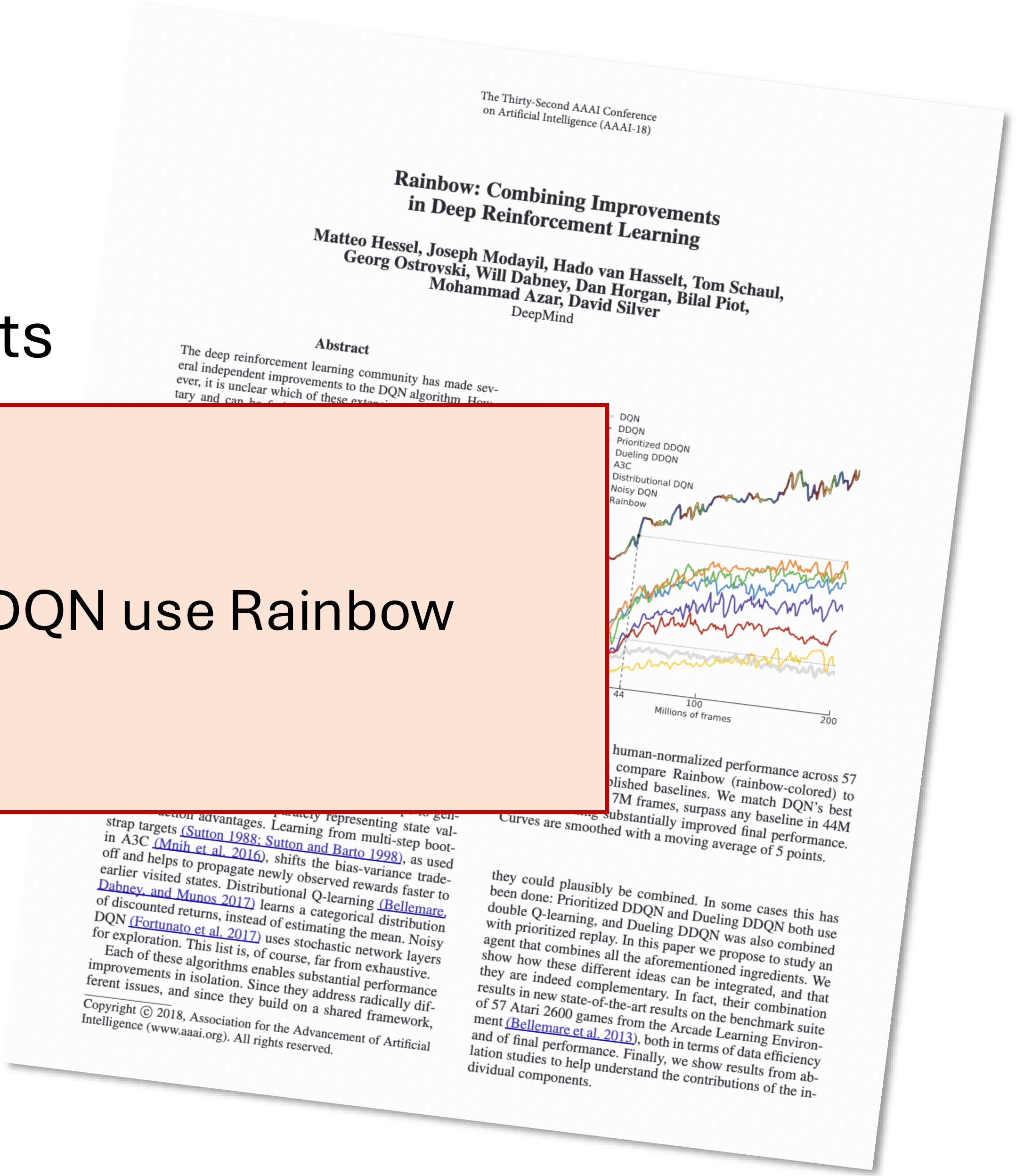
Rainbow

Combines 6 different DQN improvements

- Doubl
- Prioriti
- Duelin
- Multi-s
- Distributional RL
- Noisy nets

For most scenarios, if you use DQN use Rainbow

We have covered these!



The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)

Rainbow: Combining Improvements in Deep Reinforcement Learning

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, David Silver
DeepMind

Abstract

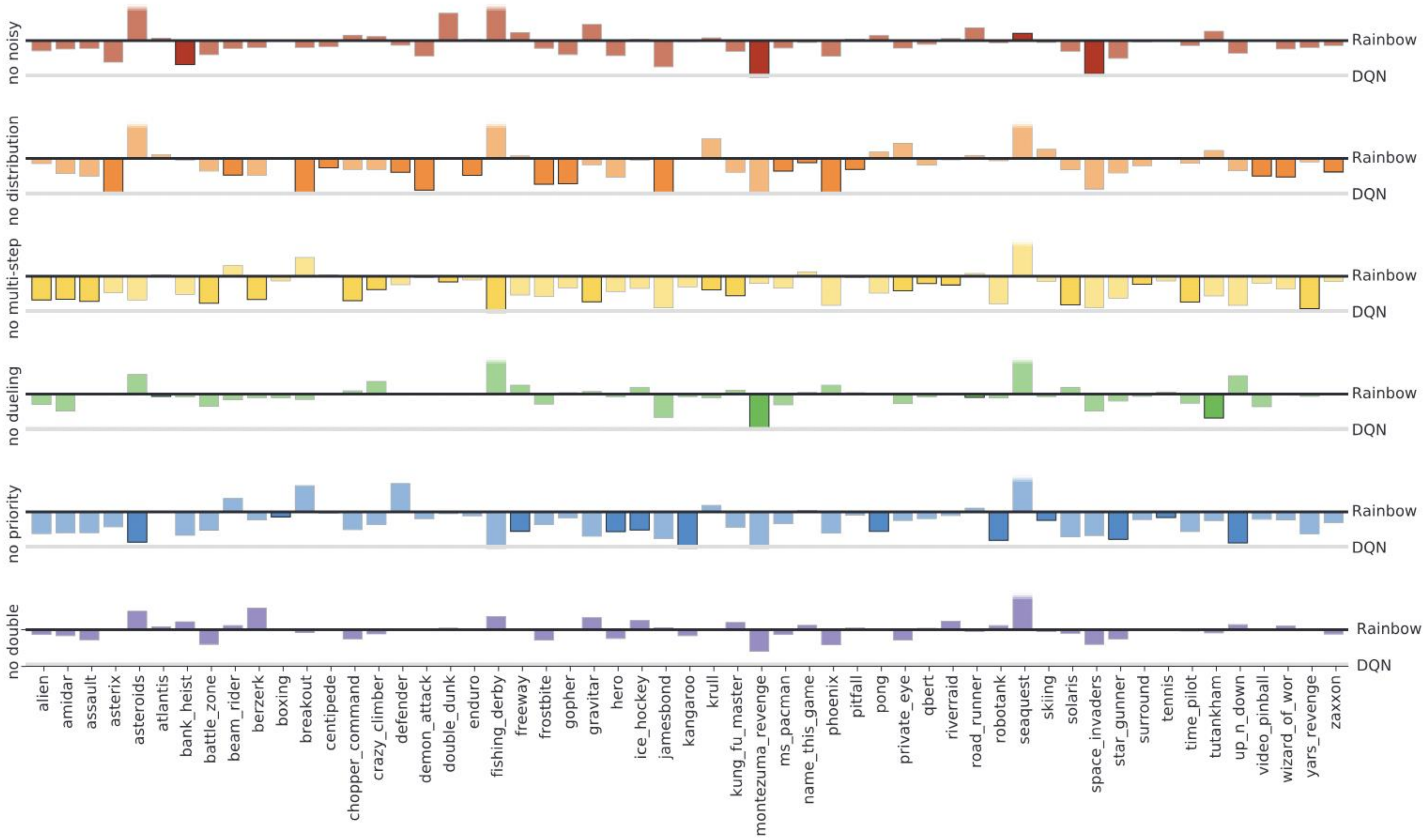
The deep reinforcement learning community has made several independent improvements to the DQN algorithm. However, it is unclear which of these exten

human-normalized performance across 57 compare Rainbow (rainbow-colored) to lished baselines. We match DQN's best 7M frames, surpass any baseline in 44M substantially improved final performance. Curves are smoothed with a moving average of 5 points.

action advantages. Learning from multi-step bootstrap targets (Sutton 1988; Sutton and Barto 1998), as used in A3C (Mnih et al. 2016), shifts the bias-variance trade-off and helps to propagate newly observed rewards faster to earlier visited states. Distributional Q-learning (Bellemare, Dabney, and Munos 2017) learns a categorical distribution of discounted returns, instead of estimating the mean. Noisy DQN (Fortunato et al. 2017) uses stochastic network layers for exploration. This list is, of course, far from exhaustive.

Each of these algorithms enables substantial performance improvements in isolation. Since they address radically different issues, and since they build on a shared framework,

they could plausibly be combined. In some cases this has been done: Prioritized DDQN and Dueling DDQN both use double Q-learning, and Dueling DDQN was also combined with prioritized replay. In this paper we propose to study an agent that combines all the aforementioned ingredients. We show how these different ideas can be integrated, and that they are indeed complementary. In fact, their combination results in new state-of-the-art results on the benchmark suite of 57 Atari 2600 games from the Arcade Learning Environment (Bellemare et al. 2013), both in terms of data efficiency and of final performance. Finally, we show results from ablation studies to help understand the contributions of the individual components.



Hessel et al, Rainbow: Combining Improvements in Deep Reinforcement Learning, 2018.

Policy gradient

Remember continuous action spaces?

$$Q(s,a) = Q(s,a) + \alpha\left(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a') - Q(s,a)\right)$$

QT-Opt

- The ”simple” method for finding $\max_{a' \in \mathcal{A}}$ is to perform optimization
- Computationally inefficient

Normalized advantage flows

- Start with dueling DQN, and further decompose the advantage
- Quadratic assumption, unimodal and “deterministic”

Policy-based reinforcement learning

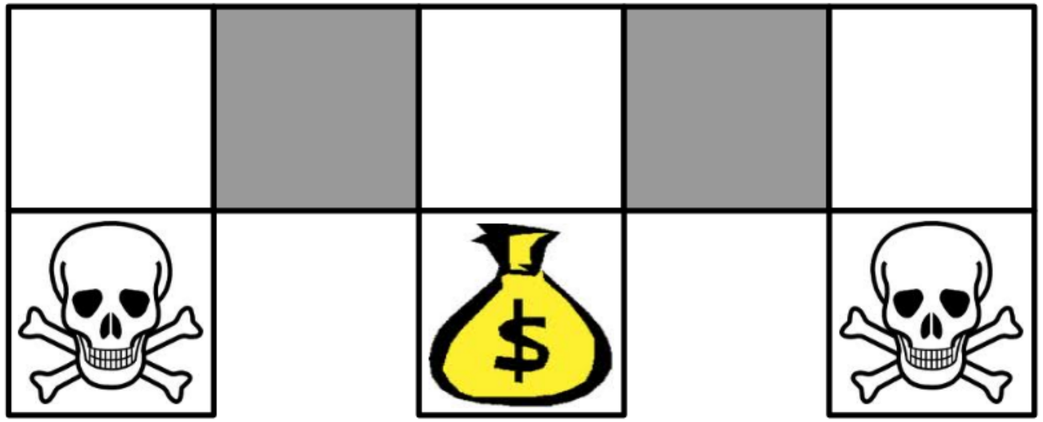
In previous lectures, we primarily focused on modeling and approximating value functions

- So far, the policy can be obtained from the value function
  - Pure-greedy, ϵ-greedy, …

(Usually) deterministic greedy policies

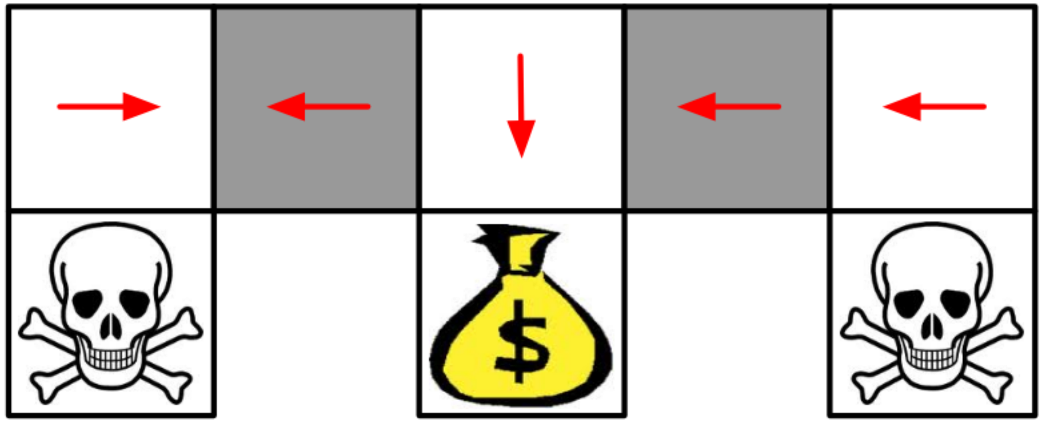
What if instead we model the policy directly?

Example: Gridworld



- Features = wall immediately to N,E,S,W. Actions = move N,E,S,W
- Agent cannot differentiate the gray states from each other
- Consider value-based approximation $Q_\theta(s, a)$ and policy-based approximation $\pi_\theta(a|s)$

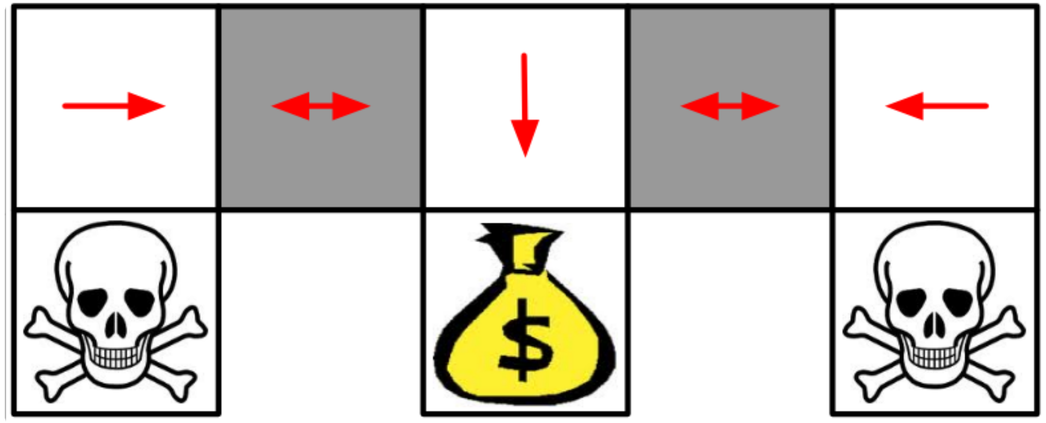
Example: Gridworld



With a deterministic policy as learned in value-based RL, the policy gets stuck and never reaches the treasure

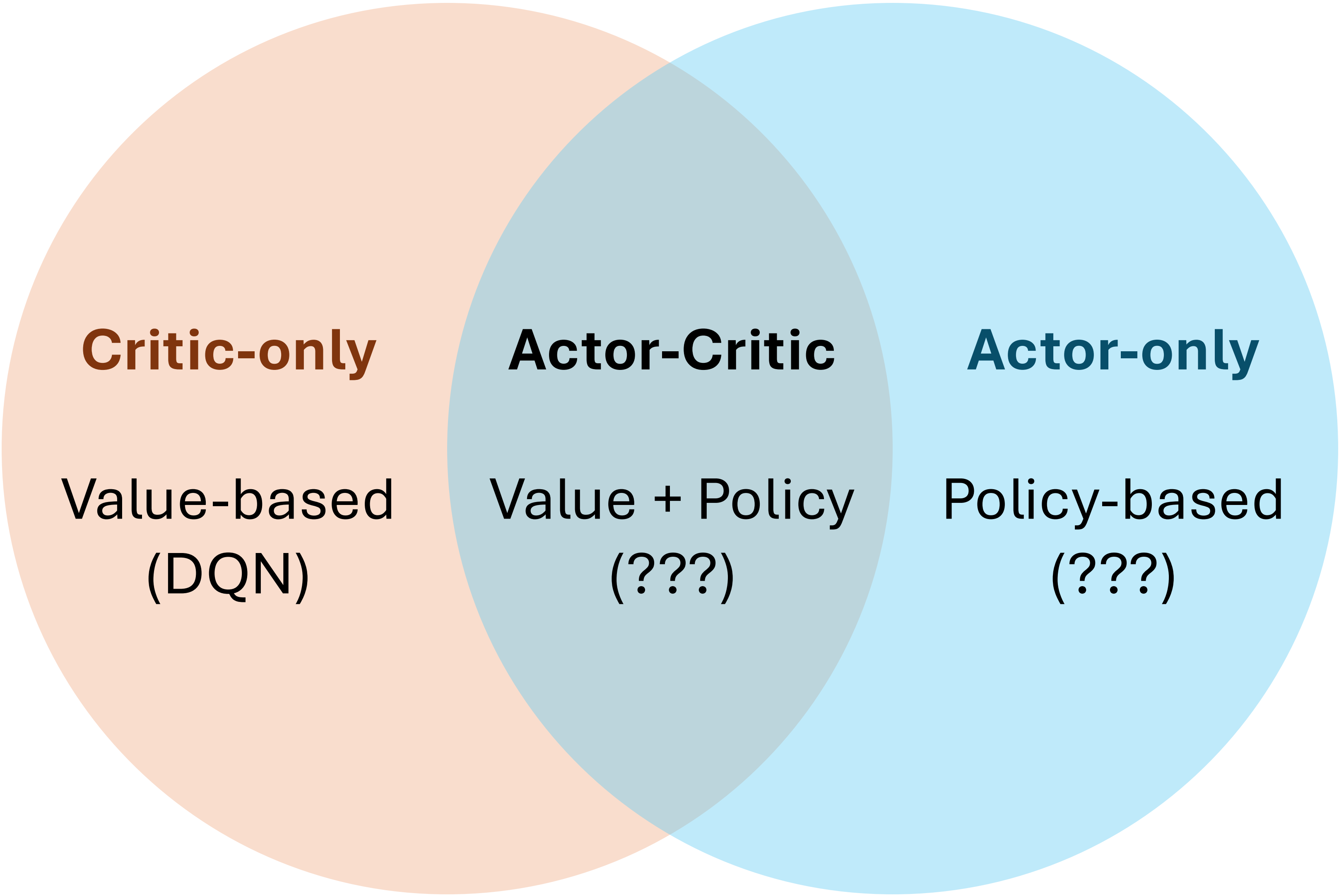
- Move W in both states or move E in both states
- Why?

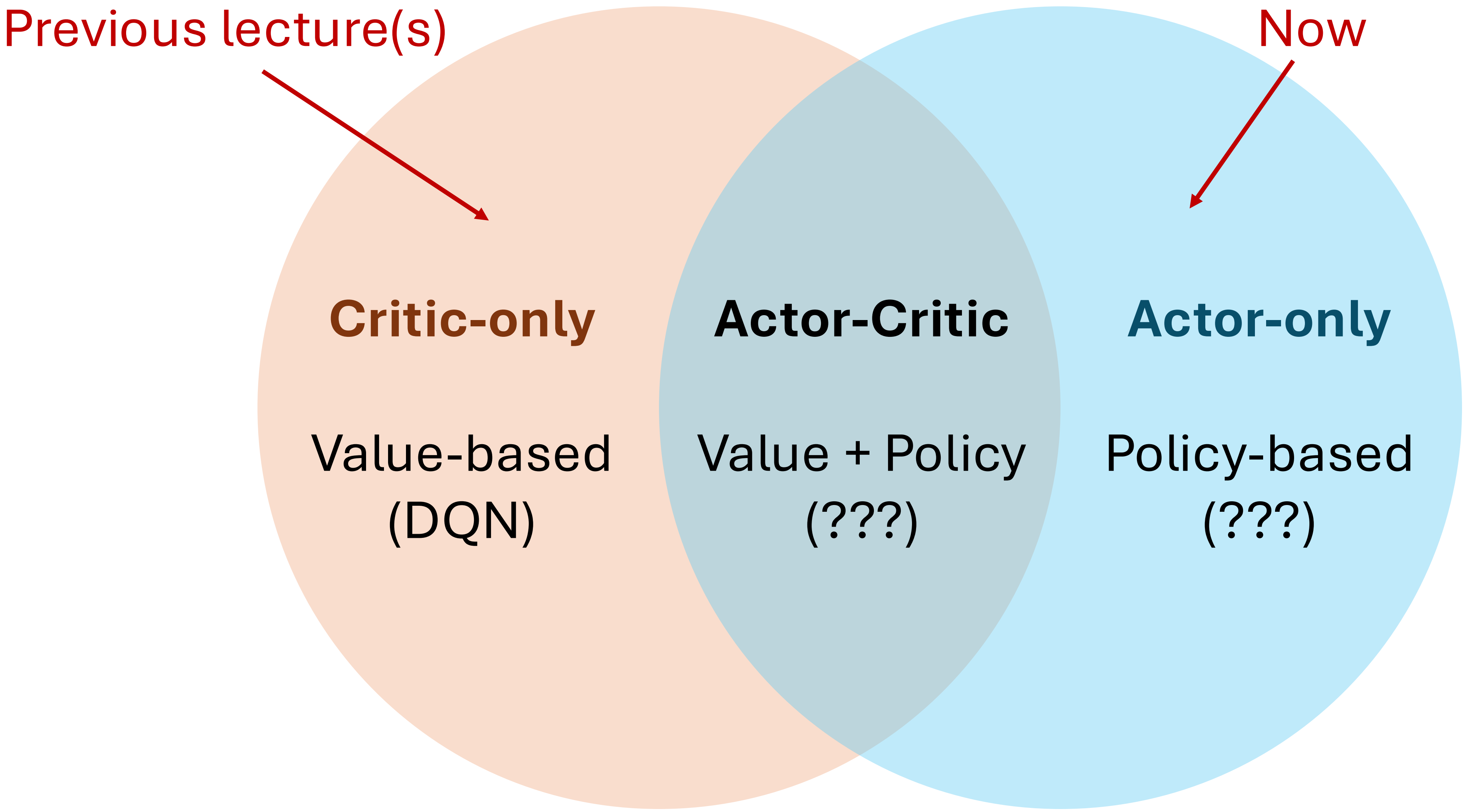
Example: Gridworld



An optimal stochastic policy randomly moves W or E in gray states

- $\pi_\theta(\text{move E} \mid \text{wall to N/S}) = 0.5$
- $\pi_\theta(\text{move W} \mid \text{wall to N/S}) = 0.5$
- Reaches treasure eventually


Critic-only
Value-based
(DQN)
Actor-Critic
Value + Policy
(???)
Actor-only
Policy-based
(???)


Previous lecture(s)
Now
Critic-only
Value-based
(DQN)
Actor-Critic
Value + Policy
(???)
Actor-only
Policy-based
(???)

Why use policy-based RL?

Pros:

- Better convergence properties
- Effective in high-dimensional and continuous action spaces
- Can learn stochastic policies (probabilistic policy)

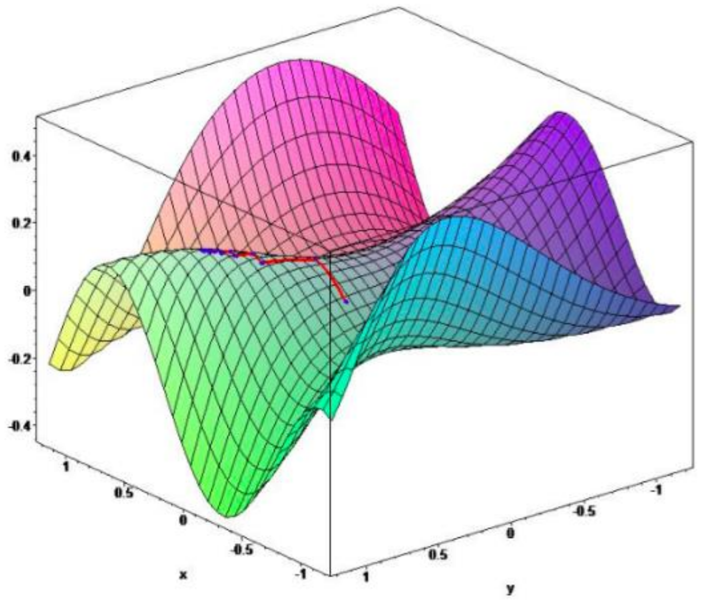
Cons:

- Typically converges to locally not globally optimal policy
- Policy evaluation has high variance (because no value function)

Policy gradient

Let $J(\theta)$ be a differentiable function of parameter vector θ

- Define the gradient as
- $\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$
- To find local minimum, adjust θ in the direction of negative gradient



Policy objectives

Policy gradient = gradient of objective with respect to parameters

- So what is a good choice of policy objective?

Policy objectives

Policy gradient = gradient of objective with respect to parameters

- So what is a good choice of policy objective?

$$\max_{\theta} J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} \gamma^t r_{t+1}] = \mathbb{E}_{\pi_\theta}[G_t]$$

In other words, find policy parameters that max the expected return

- Easiest way to do this? Monte Carlo estimation of returns!

REINFORCE

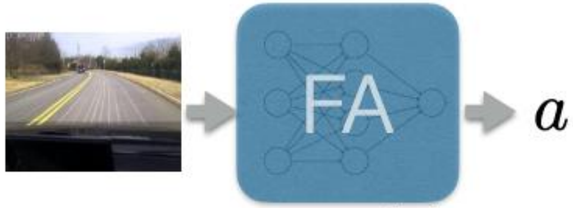
REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Types of policy functions

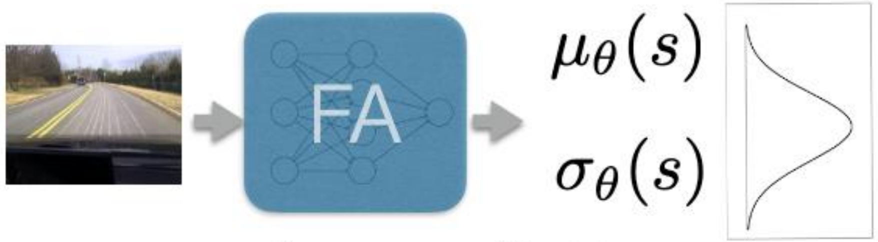
deterministic continuous policy



$$a = \pi_\theta(s)$$

e.g. outputs a steering angle directly

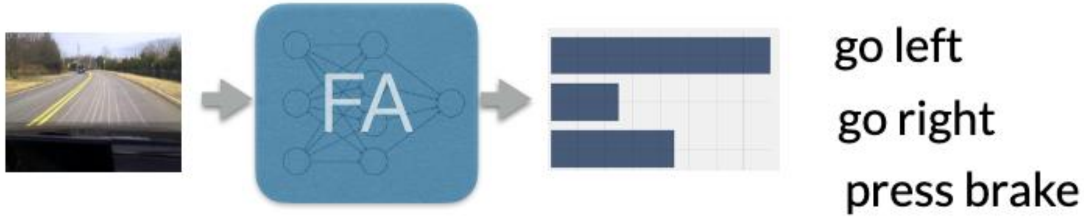
stochastic continuous policy



$$a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta^2(s))$$

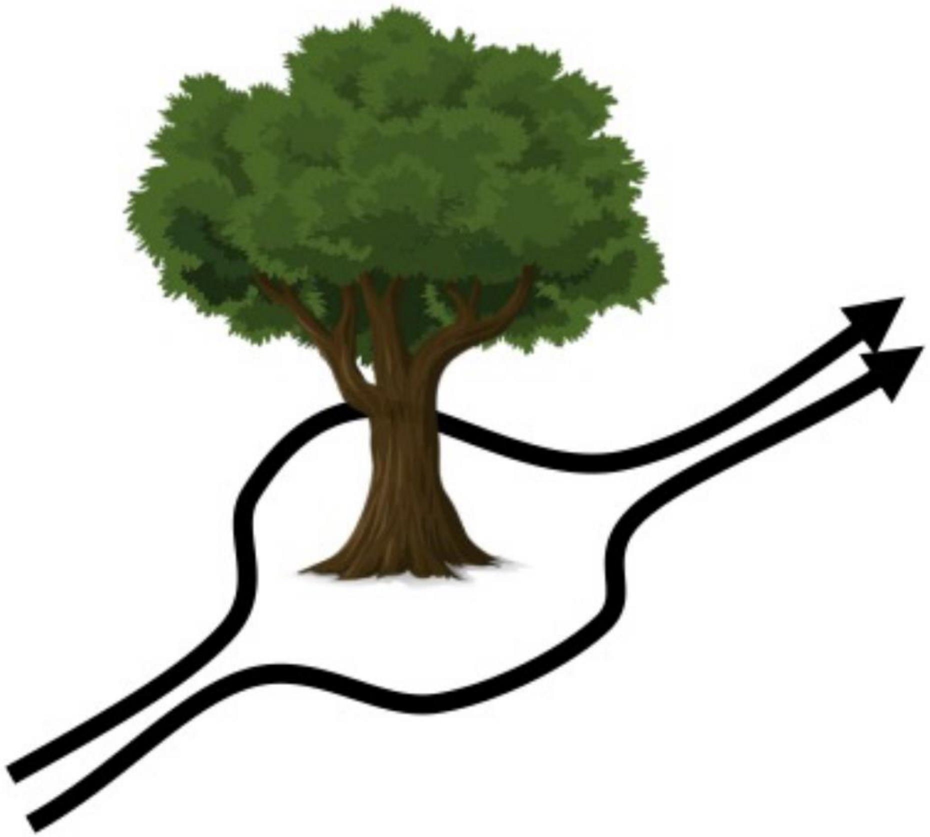
FA for stochastic multimodal continuous policies is an active area of research

(stochastic) policy over discrete actions

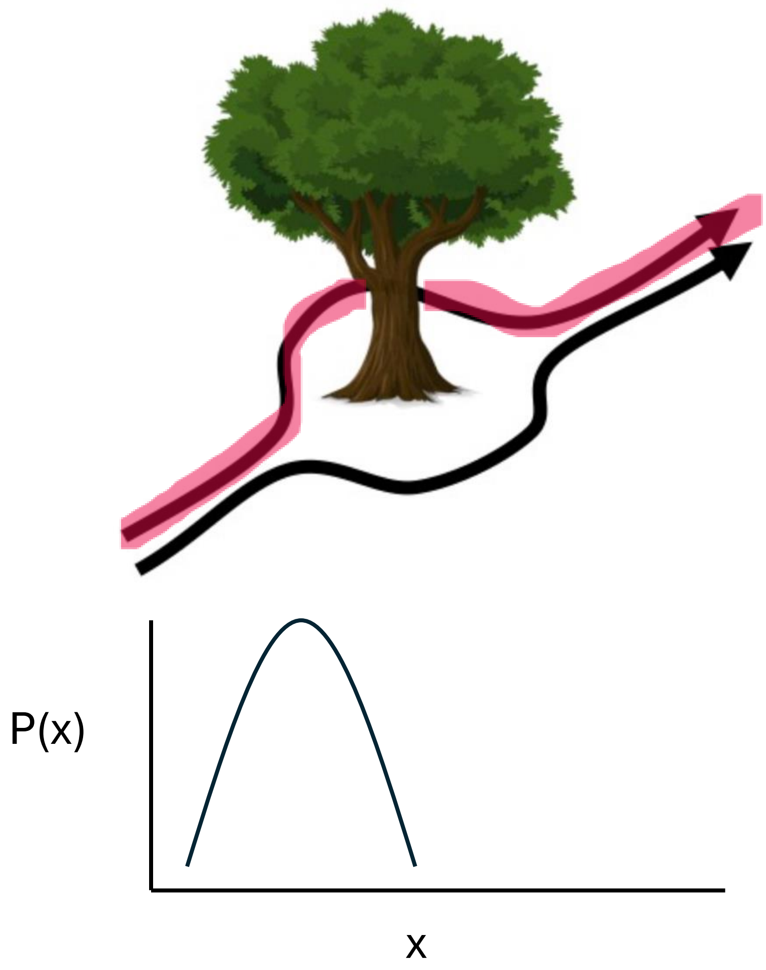


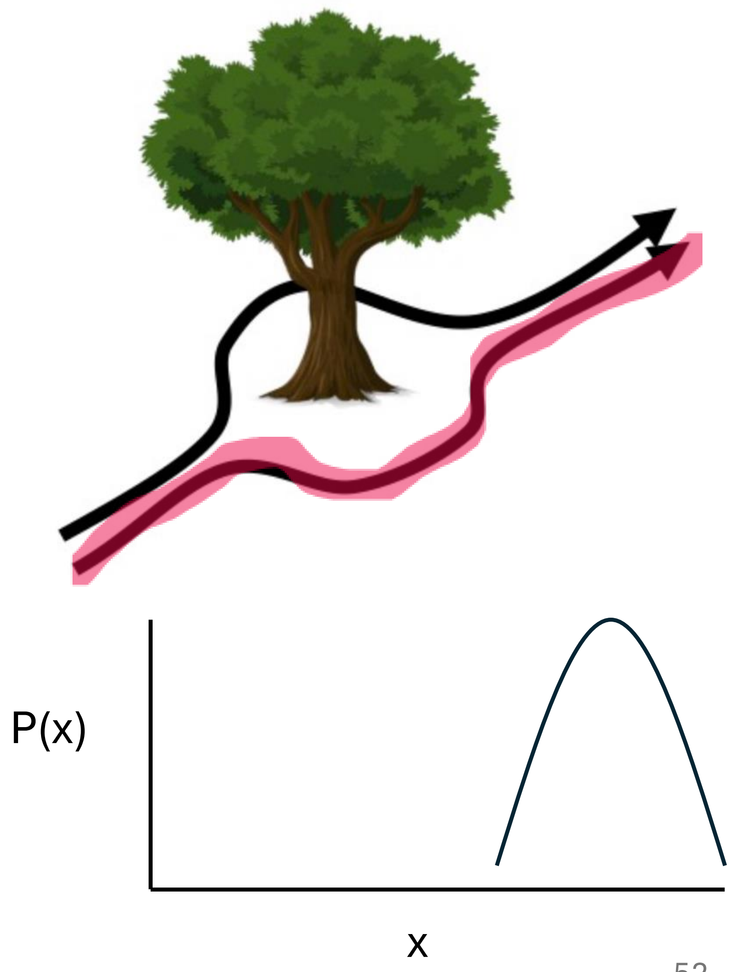
Outputs a distribution over a discrete set of actions

Multi-modality in learning



Multi-modality in learning

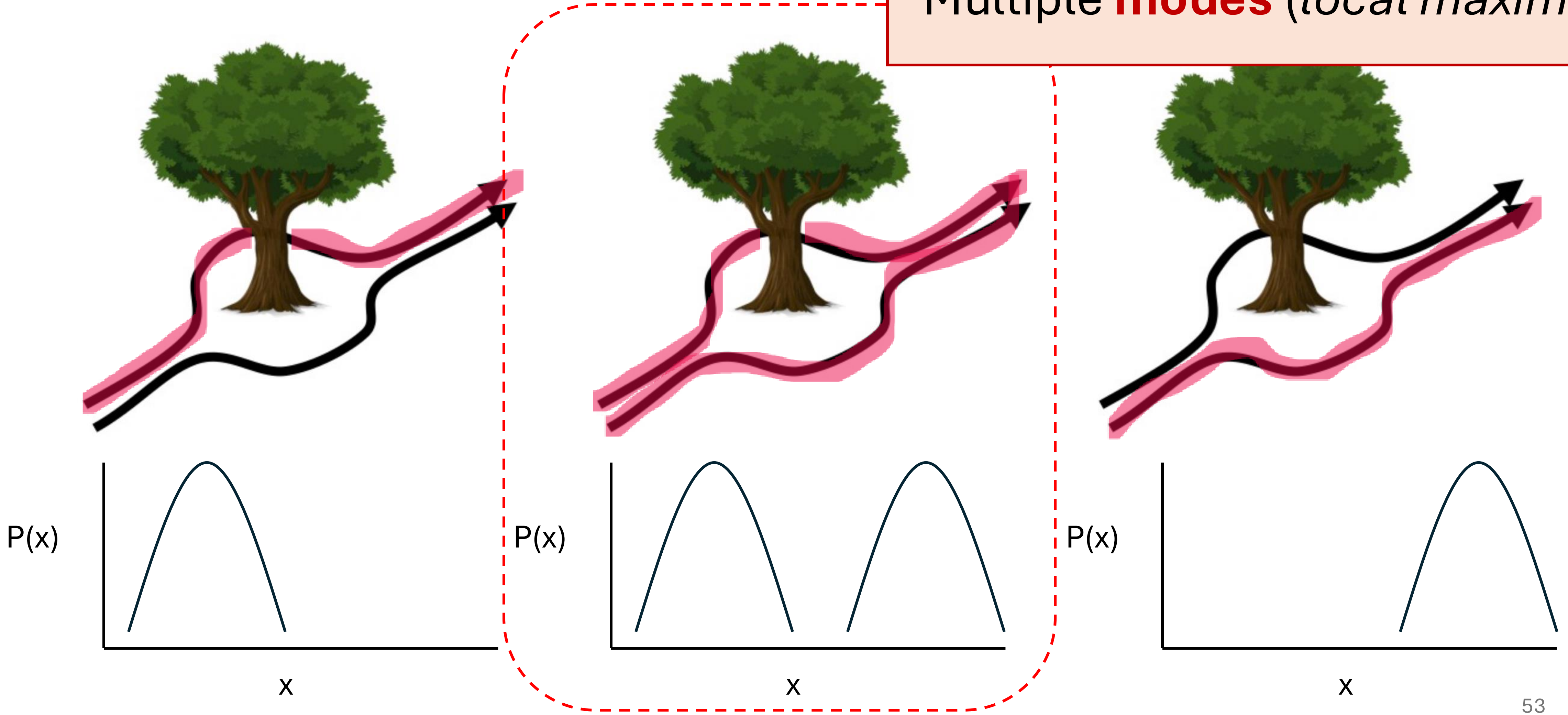




Multi-modality in learning

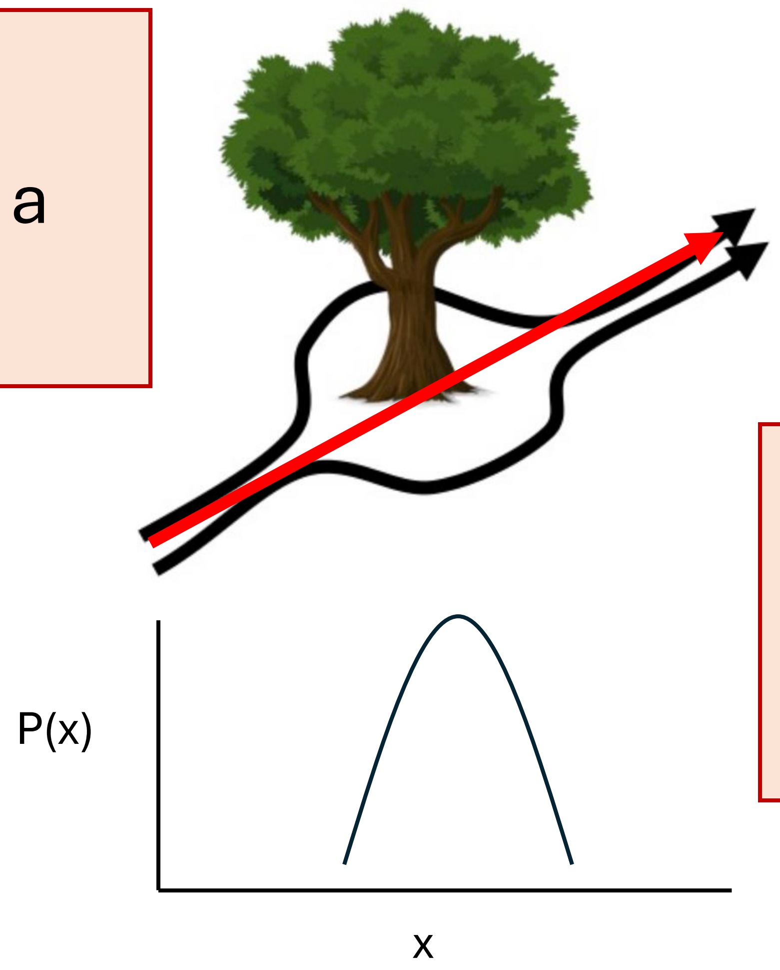
What we want!

Multiple **modes** (*local maxima*)!



But with a unimodal distribution…

The **mean action** is predicted resulting in a single peak



Assumption: continuous action space and MSE loss

The problem with policy-based RL

Monte Carlo value estimates have a problem: **what is it?**

The problem with policy-based RL

Monte Carlo value estimates have a problem: **what is it?**

- Value estimates have a high variance!

In Temporal Difference learning, we got around this by bootstrapping off our own value estimate

- This was used in value-based RL such as Q-learning

Can we combine value-based and policy-based approaches?

Actor-Critic methods

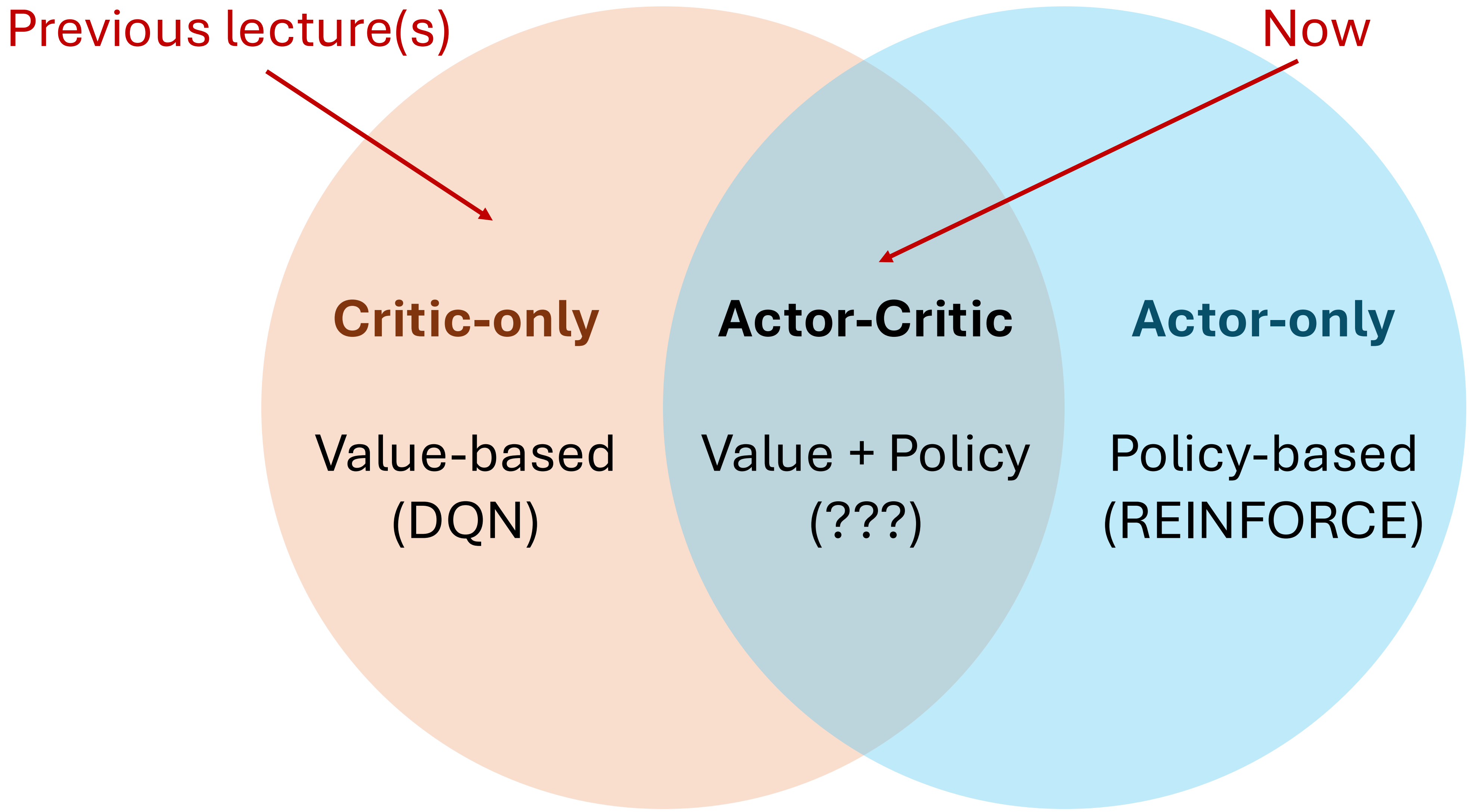
The solution is an actor-critic!

- We approximate both the policy and the value function

We maintain two sets of parameters!

Critic = approximated value function $Q_\phi(s, a) \approx Q_{\pi_\theta}(s, a)$

Actor = approximated policy function $\pi_\theta(a|s)$


Previous lecture(s)
Now
Critic-only
Value-based
(DQN)
Actor-Critic
Value + Policy
(???)
Actor-only
Policy-based
(REINFORCE)

Actor-Critic intuition

We update our policy according to an *approximate policy gradient*

- Approximate gradient provided by critic instead of MC return

Two step iterative approach:

1. Update critic parameters using TD (or similar) update
2. Update policy parameters in direction suggested by critic

Generic one-step actor-critic

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\delta\nabla\ln\pi(A|S, \boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Take-aways

DQN (and Q-learning in general) is biased (positive error)

- Double Q-learning can help mitigate this over-estimation
- Rainbow combines many of the updates we have talked about

Policy gradient estimates policies directly instead of values

Actor-Critic methods combine value and policy estimates