Welcome!

The Plan for Today

1. Course goals & logistics
2. Why study deep reinforcement learning?
3. Intro to modeling behavior and reinforcement learning

Key learnings goals:

- what is deep reinforcement learning??
- how to represent behavior
- how to formulate a reinforcement learning problem

What do we mean by deep reinforcement learning?

Sequential decision-making problems

A system needs to make *multiple* decisions based on stream of information.

observe, take action, observe, take action, …

AND the solutions to such problems

- imitation learning
- model-free & model-based RL
- offline & online RL
- multi-task & meta RL
- RL for LLMs
- RL for robots

and more!

Emphasis on solutions that scale to deep neural networks

How does deep RL differ from other ML topics?

Supervised learning

Given labeled data: $\{(x_i, y_i)\}$, learn $f(x) \approx y$

- directly told what to output
- inputs x are independently, identically distributed (i.i.d.)

Reinforcement learning

Learn *behavior* $\pi(a \mid s)$.

- from experience, indirect feedback
- data **not** i.i.d.: actions affect the future observations.

Behavior can include:

motor control chat bots game playing driving web agents

We can’t cover everything in deep RL.

We’ll focus on:

- core concepts behind deep RL methods
- implementation of algorithms
- examples in robotics, control, language models (but techniques generalize broadly)
- topics that we think are most useful & exciting!

For more theory & other applications, see CS234!

Core goal: Able to understand and implement existing and emerging methods.

Why study deep reinforcement learning?

1. Going beyond supervised (x, y) examples
   - AI model predictions have consequences!
   - When direct supervision isn't available

How can we take them into account?

Learn from *any* objective.

2. Widely used and deployed for performant AI systems

3. Learning from experience seems fundamental to intelligence
   - RL can **discover new solutions**

4. Plenty of exciting open research problems

Why study deep reinforcement learning?

Beyond supervised learning from (x, y) examples

Decision-making problems are everywhere!

a. Any sort of AI agent: robots, autonomous vehicles, web assistants
b. What if you want your AI system to interact with people? chatbots, recommenders
c. What if deploying your system affects future outcomes & observations? "feedback loops"
d. What if don't have labels or your objective isn't just accuracy? (and isn't differentiable!)

Why study deep reinforcement learning?

Widely used for performant AI systems

Learning complex physical tasks: legged robots

https://www.youtube.com/watch?v=9G9E-TIKGM8

https://www.youtube.com/watch?v=xG7WkPU8tgs

Why study deep reinforcement learning?

Widely used for performant AI systems

Learning complex physical tasks: robot manipulation

Physical Intelligence π_0

https://www.youtube.com/watch?v=gg9AYgtYoNs

Google Gemini Robotics

Why study deep reinforcement learning?

Widely used for performant AI systems

Learning to play complex games

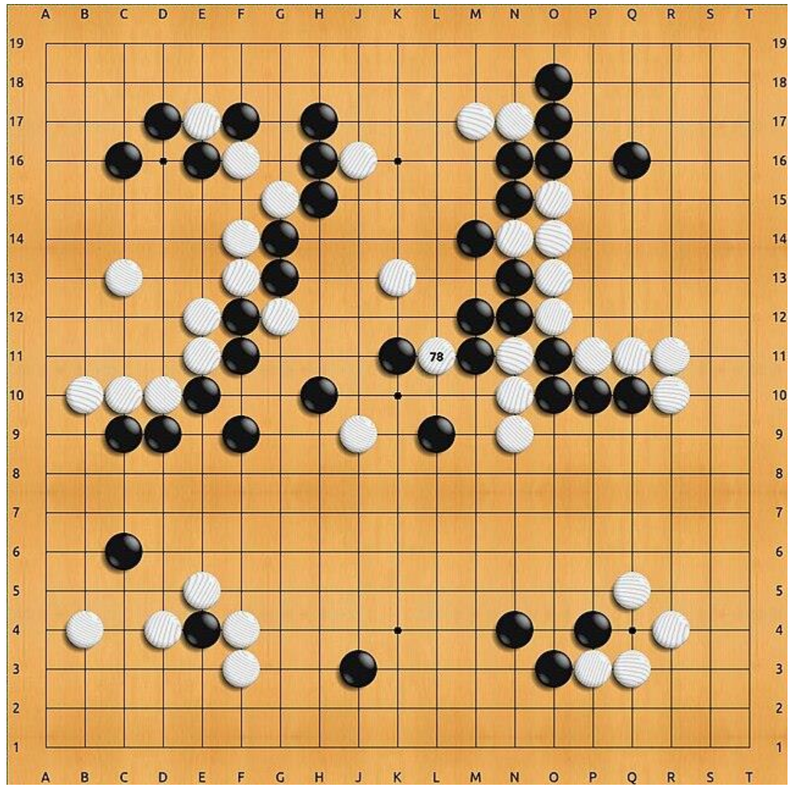


Image from Wikimedia CC-BY 4.0

Ability to **discover** new solutions:

“Move 37” in Lee Sedol AlphaGo match surprises everyone

Why study deep reinforcement learning?

Widely used for performant AI systems

Not just robots and games!

Nearly all modern language models use some form of RL for post-training.

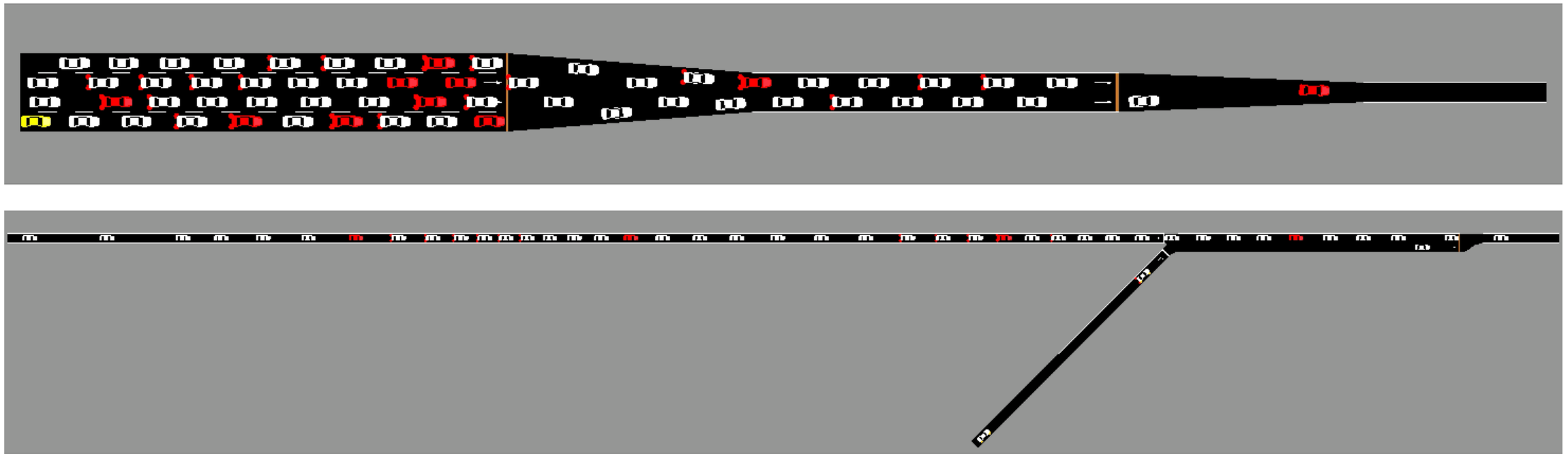
especially for more advanced reasoning.

Why study deep reinforcement learning?

Widely used for performant AI systems

Not just robots and games!

Research on traffic control



Source: https://mit-wu-lab.github.io/automatic_vehicular_control/

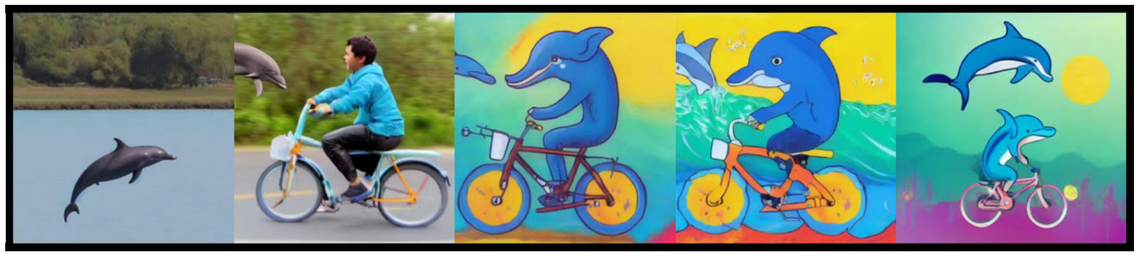
Why study deep reinforcement learning?

Widely used for performant AI systems

Not just robots and games!

Training generative image models to follow their prompt

a dolphin riding a bike →



an ant playing chess →

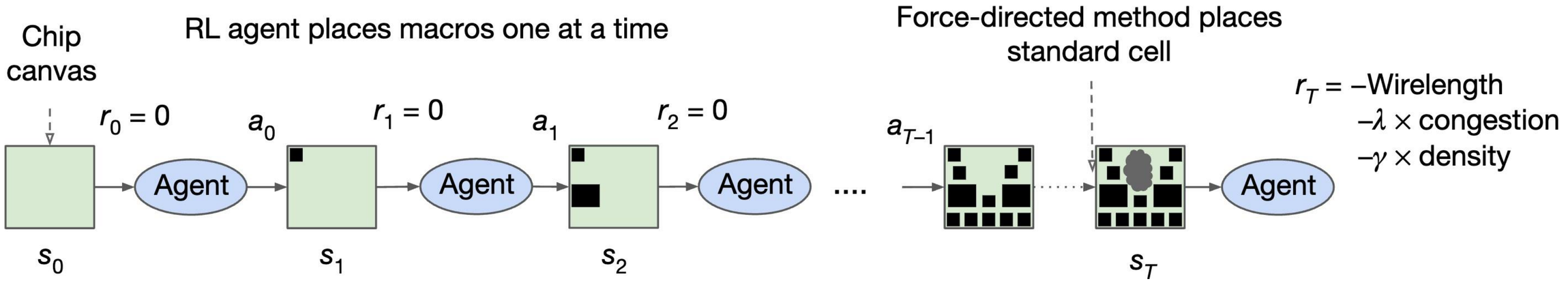
Source: https://rl-diffusion.github.io/

Why study deep reinforcement learning?

Widely used for performant AI systems

Not just robots and games!

Chip design, in Google's production TPU chips



Source: https://research.google/blog/chip-design-with-deep-reinforcement-learning/

Why study deep reinforcement learning?

Fundamental aspect of intelligence

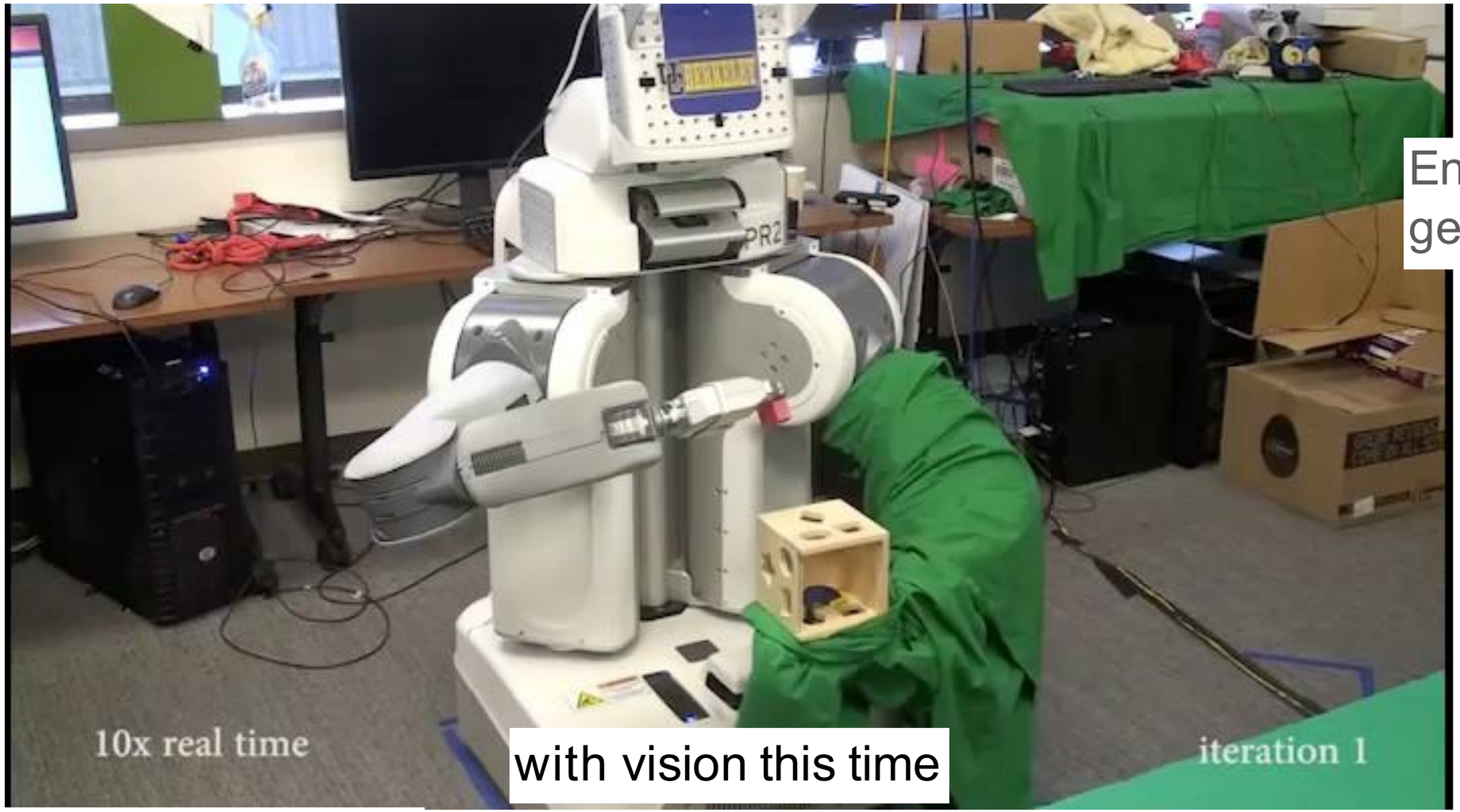
Enables the ability to get better with practice

The robot has its eyes closed.

Levine et al. ICRA ‘15

Why study deep reinforcement learning?

Fundamental aspect of intelligence



Enables the ability to get better with practice

Levine*, Finn* et al. JMLR ‘16

Why study deep reinforcement learning?

Fundamental aspect of intelligence

Enables the ability to get better with practice

Levine*, Finn* et al. JMLR ‘16

Why study deep reinforcement learning?

Still lots of exciting research problems!

How does robot learn to represent what is good or bad for the task? —> reward learning

How can an agent generalize its behavior to many different scenarios?

(Can we apply such a system at scale?)

Leverage large, diverse datasets —> offine RL

Transfer from other tasks, goals —> multitask RL, meta-RL

Can use RL to learn long-horizon tasks, like cooking a meal? —> hierarchy, reasoning

Can robots practice fully autonomously? —> reset-free RL

Behind the scenes of RL…

Yevgen

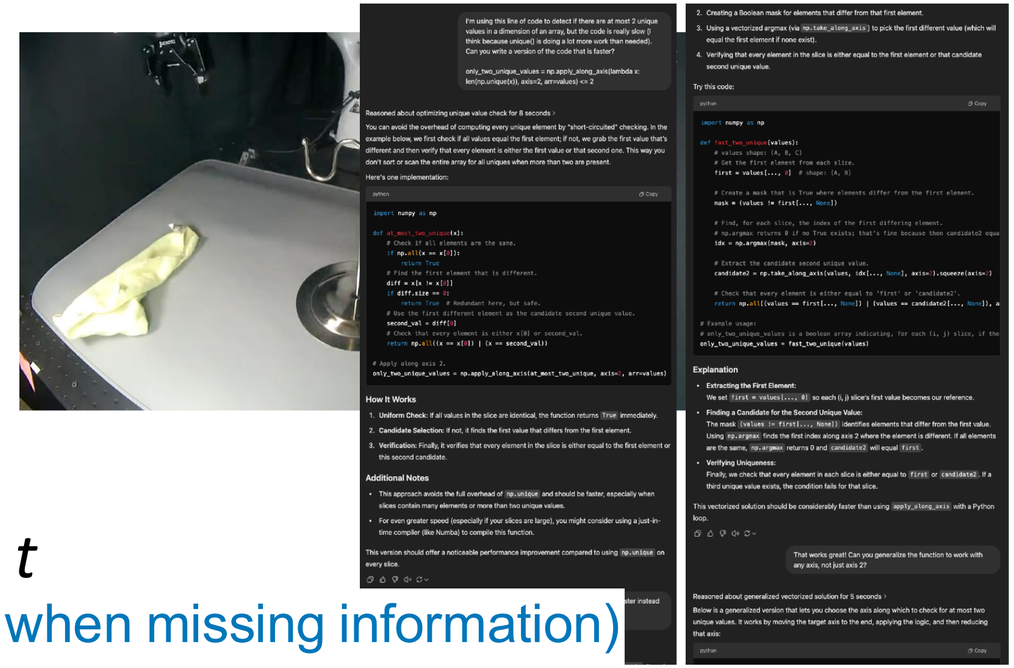
Yevgen is doing more work than the robot!

It’s not practical to collect a lot of data this way.

The Plan for Today

1. Course goals & logistics
2. Why study deep reinforcement learning?
3. **Intro to modeling behavior and reinforcement learning**

How to represent experience as data?



state $\mathbf{s}_t$ - the state of the "world" at time *t*

observation $\mathbf{o}_t$ - what the agent observes at time *t*

(only used when missing information)

action $\mathbf{a}_t$ - the decision taken at time *t*

trajectory - sequence of states/observations and actions

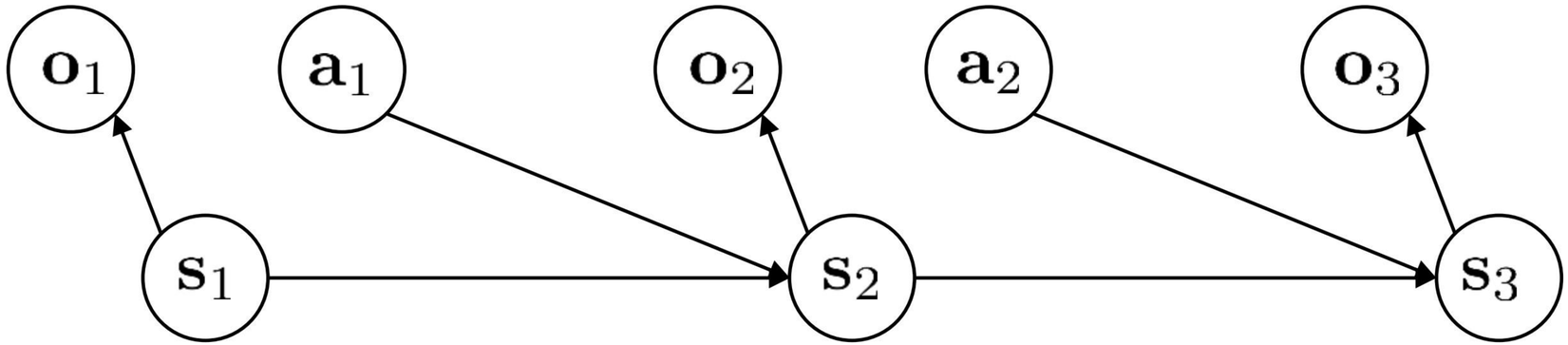
$$(\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \ldots, \mathbf{s}_T, \mathbf{a}_T)$$

(could be length T=1!)

reward function $r(\mathbf{s}, \mathbf{a})$ - how good is $\mathbf{s}$, $\mathbf{a}$?

States vs. observations

Next state is purely a function of the current state and action (and randomness)

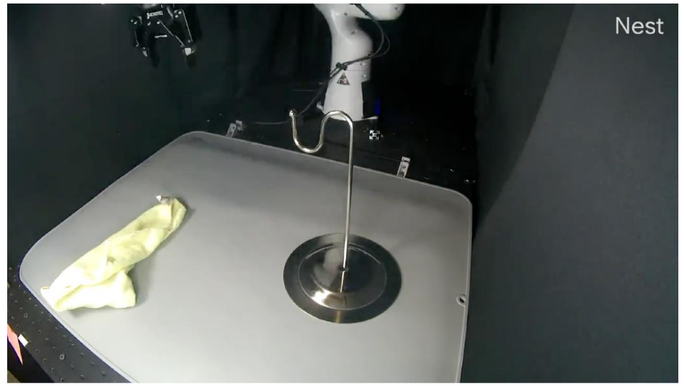


unknown dynamics $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$

independent of $\mathbf{s}_{t-1}$

“Markov property”

Examples



state $\mathbf{s}$ - RGB images, joint positions, joint velocities

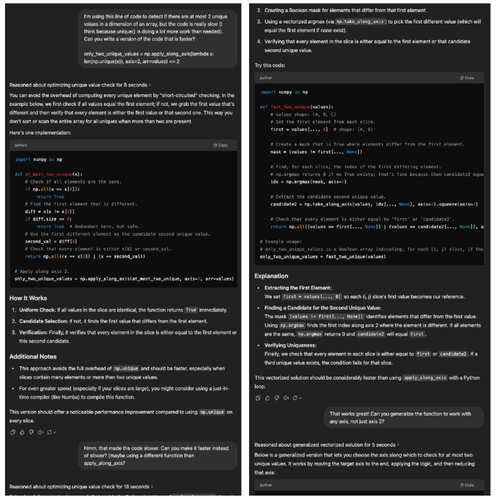
action $\mathbf{a}$ - commanded next joint position

trajectory τ - 10-sec sequence of camera, joint readings, controls at 20 Hz

$$(\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \ldots, \mathbf{s}_T, \mathbf{a}_T), \quad = 200$$

reward $r(\mathbf{s}, \mathbf{a})$ = 1 if the towel is on the hook in state

0 otherwise



observation $\mathbf{o}$ - the user's most recent message

action $\mathbf{a}$ - chatbot's next message

trajectory τ - variable length conversation trace

$$(\mathbf{o}_1, \mathbf{a}_1, \mathbf{o}_2, \mathbf{a}_2, \ldots, \mathbf{o}_T, \mathbf{a}_T)$$

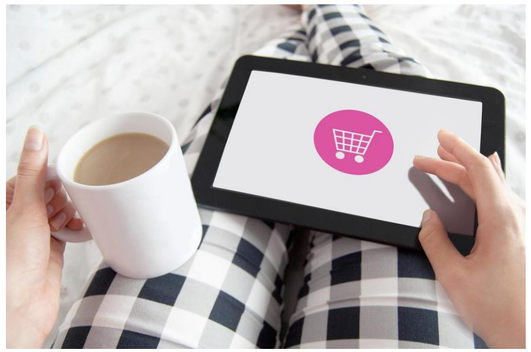
reward $r(\mathbf{s}, \mathbf{a})$ = 1 if the user gives upvote

-10 if the user downvotes

0 if no user feedback

Think: how to represent another example?

autonomous driving



web agent

poker player

choose your own!

Define

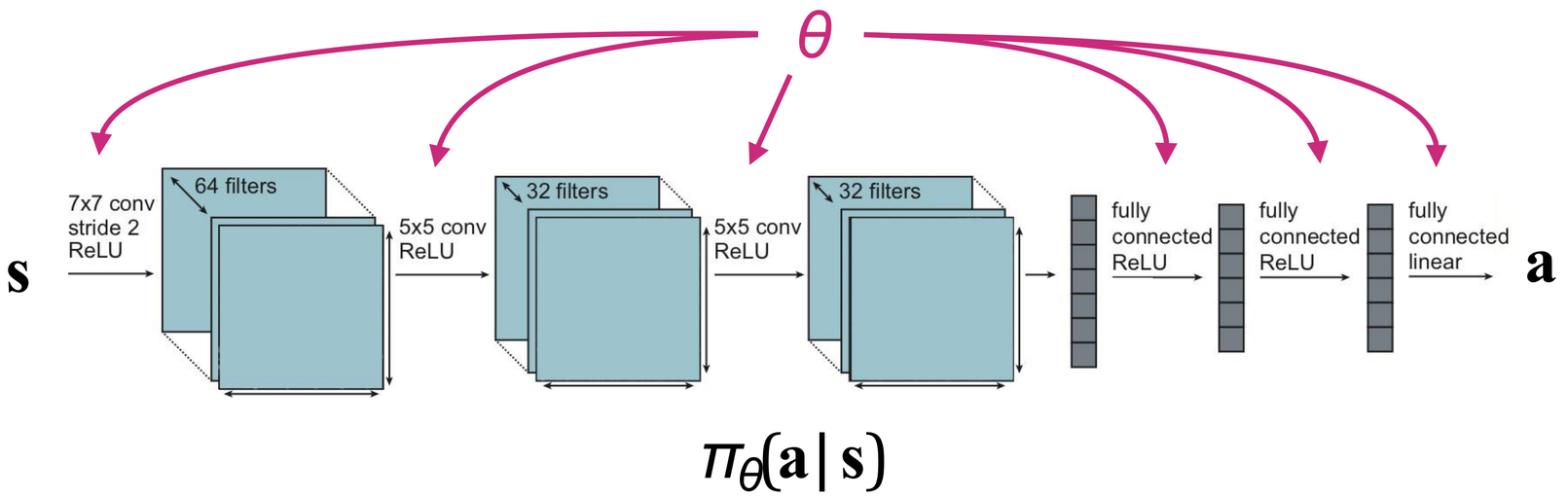
state $\mathbf{s}$ or *observation* $\mathbf{o}$

action $\mathbf{a}$

trajectory τ

reward $r(\mathbf{s}, \mathbf{a})$

How to represent behavior with a neural network?



Observe state $\mathbf{s}_t$

Take action $\mathbf{a}_t$ (e.g. by sampling from *policy* $\pi_\theta(\cdot|\mathbf{s}_t)$)

Observe next state $\mathbf{s}_{t+1}$ sampled from unknown world dynamics $p(\cdot|\mathbf{s}_t, \mathbf{a}_t)$

Result: a *trajectory* $\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T$, also called a policy *roll-out* or an *episode*

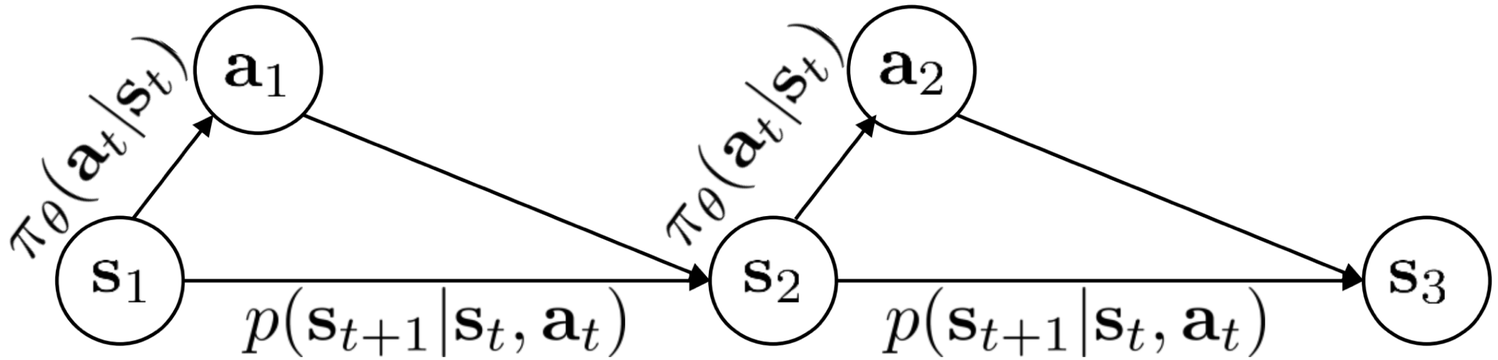
If you only have observations $\mathbf{o}$, give the policy memory: $\pi_\theta(\mathbf{a}_t|\mathbf{o}_{t-m}, \ldots, \mathbf{o}_t)$

What is the goal of reinforcement learning?

maximize sum of rewards: $\max \sum_{t}^{T} r(\mathbf{s}_t, \mathbf{a}_t)$

but this is not a deterministic quantity!

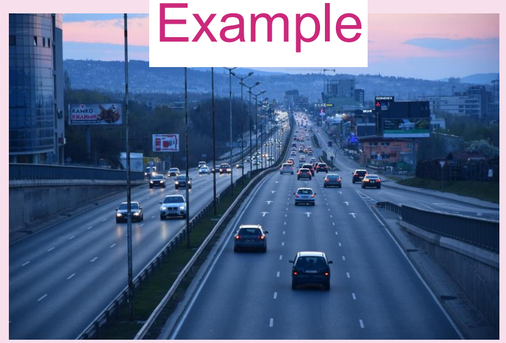
Question: what are the sources of variability?



$$p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$$

$p_\theta(\tau)$

Example



1. the world is stochastic
2. the car may not make the same decision every time.

What is the goal of reinforcement learning?

~~maximize sum of rewards: $\max \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$~~

maximize *expected* sum of rewards: $\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$

$$\underbrace{p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$$

Aside: why stochastic policies?

1. Exploration: to learn from your own experience, must try different things.
2. Modeling stochastic behavior: existing data will exhibit varying behaviors

We can leverage tools from generative modeling!

—> generative model over *actions* given states/observations

What is the goal of reinforcement learning?

maximize *expected* sum of rewards: $\max_{\theta} \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$

$$\underbrace{p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t \,|\, \mathbf{s}_t) p(\mathbf{s}_{t+1} \,|\, \mathbf{s}_t, \mathbf{a}_t)$$

How good is a particular policy?

value function $V^\pi(\mathbf{s})$ - future expected reward starting at $\mathbf{s}$ and following

Q-function $Q^\pi(\mathbf{s}, \mathbf{a})$ - future expected reward starting at $\mathbf{s}$, taking $\mathbf{a}$, then following π

Types of algorithms

maximize *expected* sum of rewards: $\max_{\theta} \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t^T r(\mathbf{s}_t, \mathbf{a}_t) \right]$

1. Imitation learning: mimic a policy that achieves high reward
2. Policy gradients: directly differentiate the above objective
3. Actor-critic: estimate value of the current policy and use it to make the policy better
4. Value-based: estimate value of the optimal policy
5. Model-based: learn to model the dynamics, and use it for planning or policy improvement

Why so many algorithms?

Algorithms make different trade-offs, thrive under different assumptions.

- How easy / cheap is it to collect data with policy? (e.g. simulator vs. hand-written)
- How easy / cheap are different forms of supervision? (demos, detailed rewards)
- How important is stability and ease-of-use?
- Action space dimensionality, continuous vs. discrete
- Is it easy to learn the dynamics model?

Recap of definitions

state $\mathbf{s}_t$ - the state of the "world" at time t

(or *observation* $\mathbf{o}_t$ - what the agent observes at time t)

action $\mathbf{a}_t$ - the decision taken at time t

reward function $r(\mathbf{s}, \mathbf{a})$ - how good is $\mathbf{s}$, $\mathbf{a}$?

initial state distr. $p(\mathbf{s}_1)$, *unknown dynamics* $p(\mathbf{s_{t+1}} \mid \mathbf{s}_t, \mathbf{a}_t)$

} (partially-observed) Markov decision process

MDP, POMDP

Recap of definitions

trajectory τ - sequence of states/observations and actions $(\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \ldots, \mathbf{s}_T, \mathbf{a}_T)$

policy π - represents behavior, selecting actions based on states or observations

Goal: learn policy π_θ that maximizes *expected* sum of rewards:

$$\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t^T r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

value function $V^\pi(\mathbf{s})$ - future expected reward starting at **s** and following π

Q-function $Q^\pi(\mathbf{s}, \mathbf{a})$ - future expected reward starting at **s**, taking **a**, then following π

Imitation Learning

CS 224R

Partial Recap

state $\mathbf{s}_t$ - the state of the "world" at time

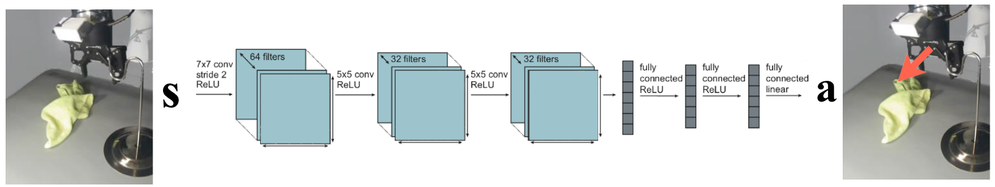
observation $\mathbf{o}_t$ - what the agent observes at time

action $\mathbf{a}_t$ - the decision taken at time

trajectory - sequence of states/observations and actions

$$(\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \ldots, \mathbf{s}_T, \mathbf{a}_T)$$

reward function $r(\mathbf{s}, \mathbf{a})$ - how good is $\mathbf{s}$, $\mathbf{a}$?



$\pi_\theta(\mathbf{a} \mid \mathbf{s})$

policy $\pi(\mathbf{a}_t \mid \mathbf{s}_t)$ or $\pi(\mathbf{a}_t \mid \mathbf{o}_{t-m:t})$ - behavior, usually what we are trying to learn

can be represented using a generative model

The plan for today

Imitation Learning

1. Imitation learning basics
2. Learning expressive policy distributions
3. Learning from online interventions
4. Time permitting: how to collect demonstrations

Key learning goals:

- how to represent distributions with neural networks
- why expressive distributions matter for imitation learning
- what are compounding errors and how to address them

What is the goal of imitation learning?

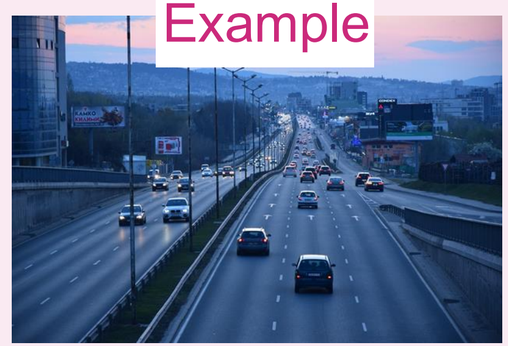
Data: Given trajectories collected by an expert

"*demonstrations*" $\mathcal{D} := \{(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T)\}$

(sampled from some unknown policy π_{expert})

Goal: Learn a policy $_\theta$ that performs at the level of the expert policy, by mimicking it.

Example



- Dataset from human drivers
- Sensor readings + steering commands

Imitation learning - version 0

Deterministic policy

0. Given demonstrations collected by an expert $\mathcal{D} := \{(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T)\}$

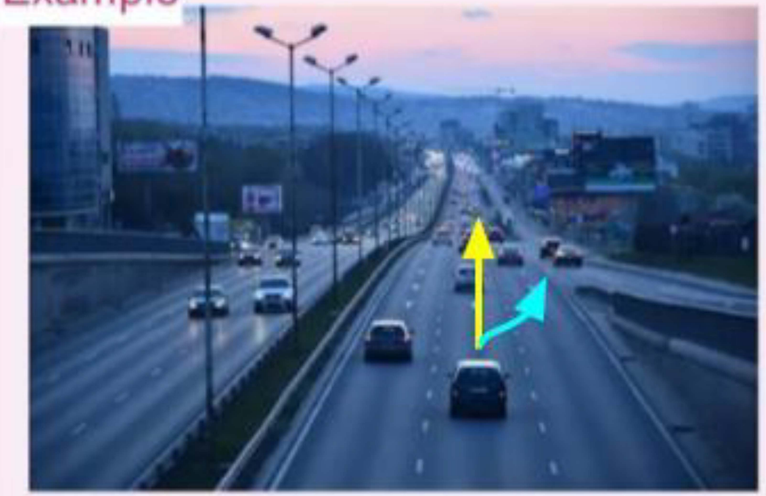
1. For deterministic policy, supervised regression to the expert's actions

$$\min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s},\mathbf{a})\in\mathcal{D}} \|\mathbf{a} - \hat{\mathbf{a}}\|^2 \qquad \text{where } \hat{\mathbf{a}} = \pi_\theta(\mathbf{s})$$

2. Deploy learned policy θ

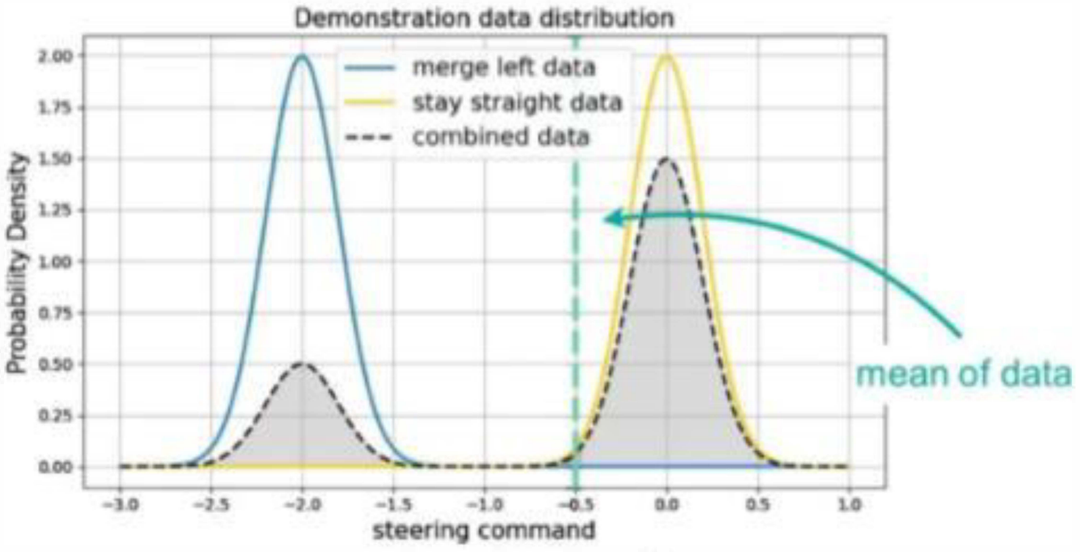
What could go wrong?

Example



- Dataset from human drivers
- Sensor readings + steering commands

Question: what might policy trained with ℓ_2-regression do?

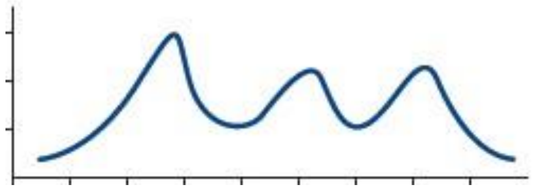


How can we represent more than the mean?

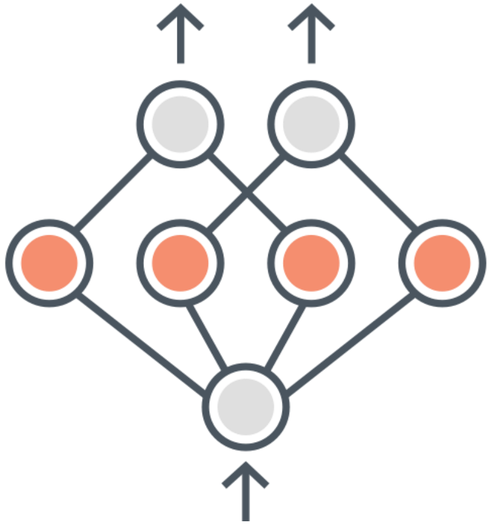
How often does this happen in practice? All the time!

Esp. when data is collected by multiple people.

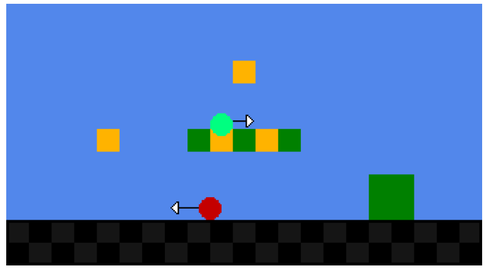
Learning *distributions* with neural networks

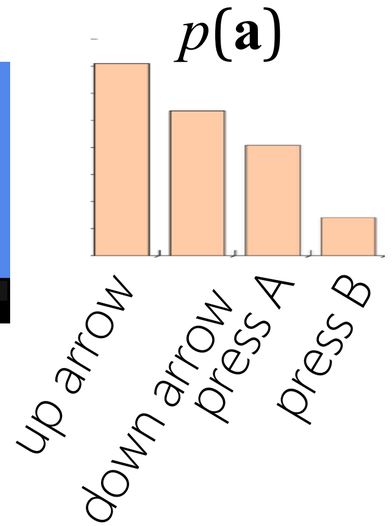


parameters of distribution



1D discrete actions

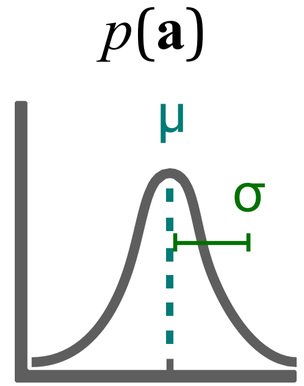




Neural net outputs $p(\text{up}), p(\text{down}), \ldots$ represent categorical distribution.

Maximally expressive

Continuous actions



next steering angle

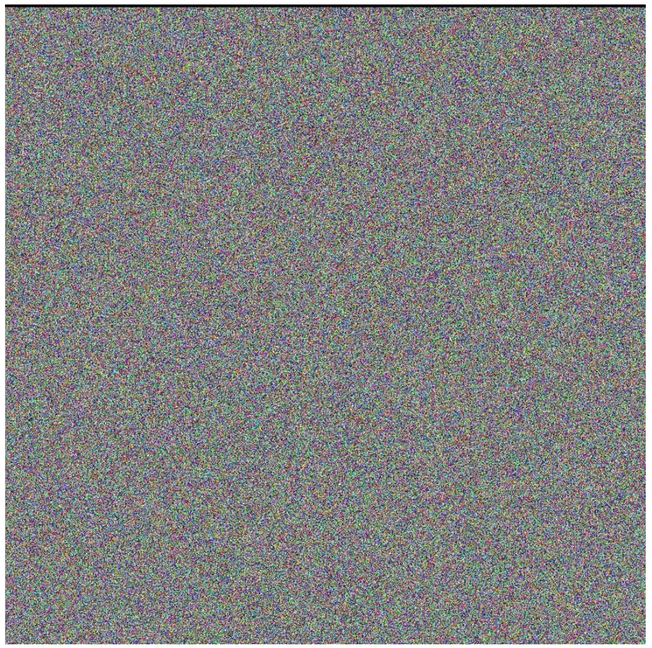
Neural net outputs , to represent Gaussian distribution.

Not very expressive!

Learning *distributions* with neural networks

💡Can we use generative modeling?

image diffusion models



autoregressive models

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

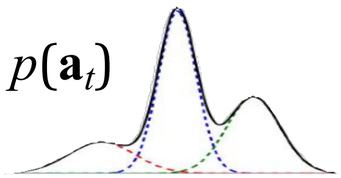
learning $p(\text{image} \mid \text{text description})$

learning $p(\text{next word} \mid \text{words so far})$

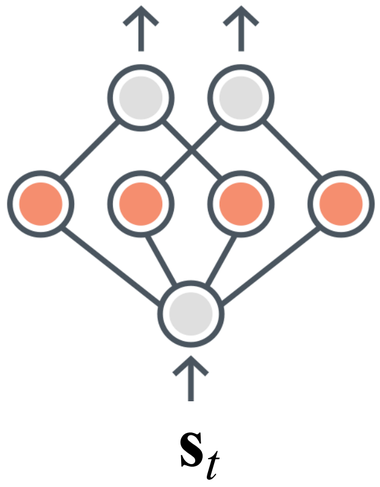
Our goal: learning $p(\text{action} \mid \text{observations})$

Generative models for policies (approximating $p(\mathbf{a} \mid \mathbf{s})$)

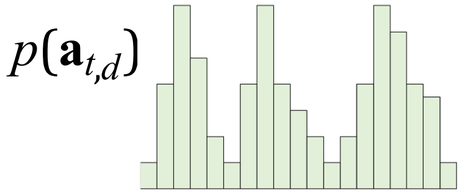
Mixture of Gaussians



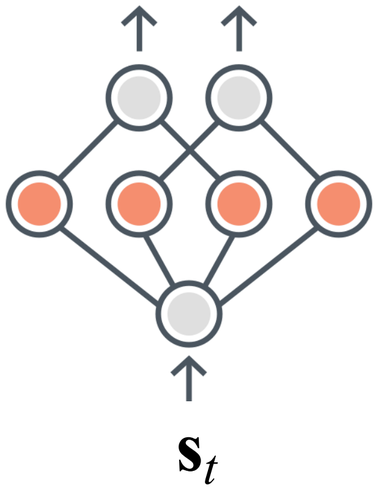
output $\mu_1, \sigma_1, w_1, \mu_2, \sigma_2, w_2, \ldots$



Discretize + Autoregressive

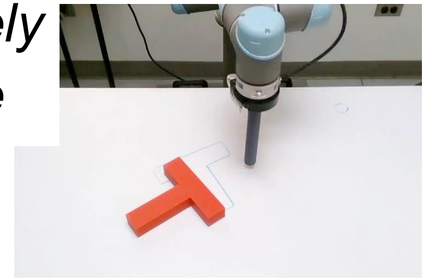


output $p(\mathbf{a}_{t,1}), p(\mathbf{a}_{t,2} \mid \hat{\mathbf{a}}_{t,1}), p(\mathbf{a}_{t,3} \mid \hat{\mathbf{a}}_{t,1:2}), \ldots$

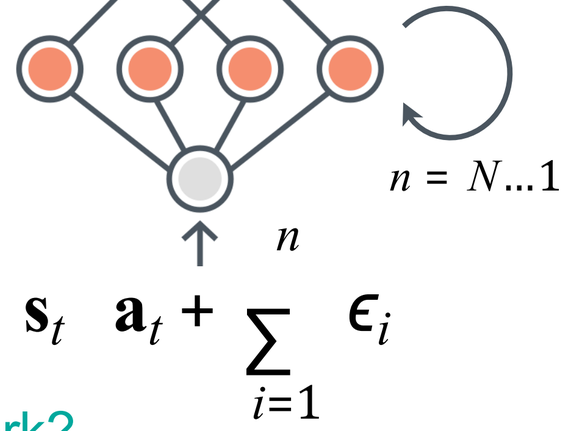


Diffusion

iteratively denoise



output ϵ_n



Question: how are these different from ℓ_2 loss with larger network?

Important Note: Neural network expressivity is often distinct from distribution expressivity!

Imitation learning - version 1

Expressive policies

0. Given demonstrations collected by an expert $\mathcal{D} := \{(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T)\}$

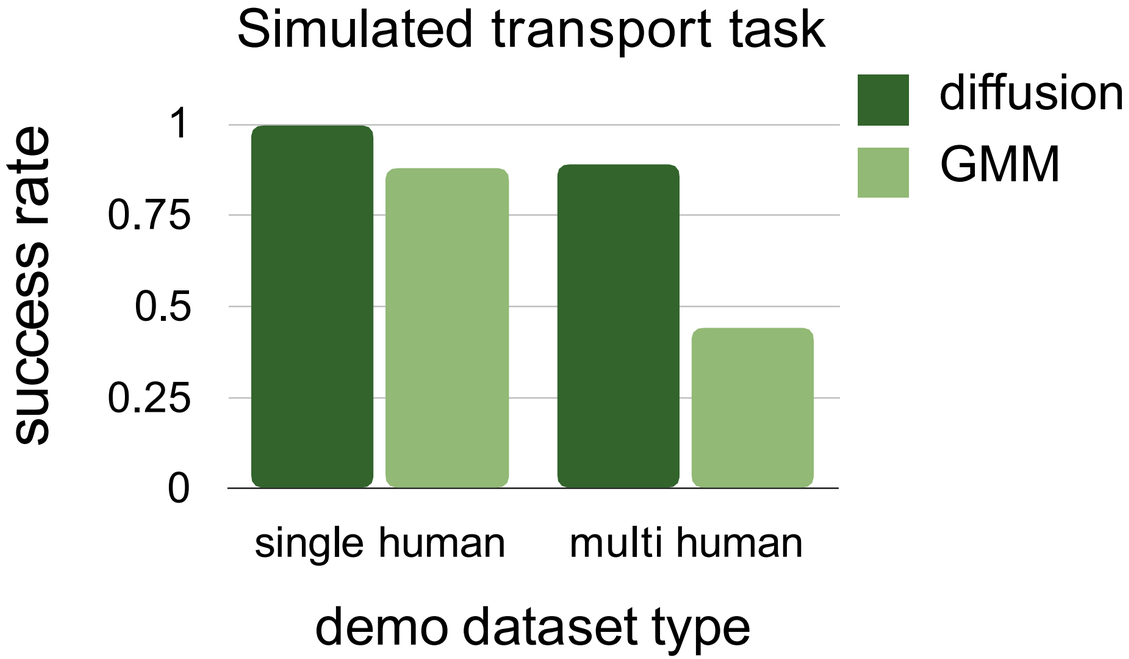
1. Train **generative model** of the expert's actions

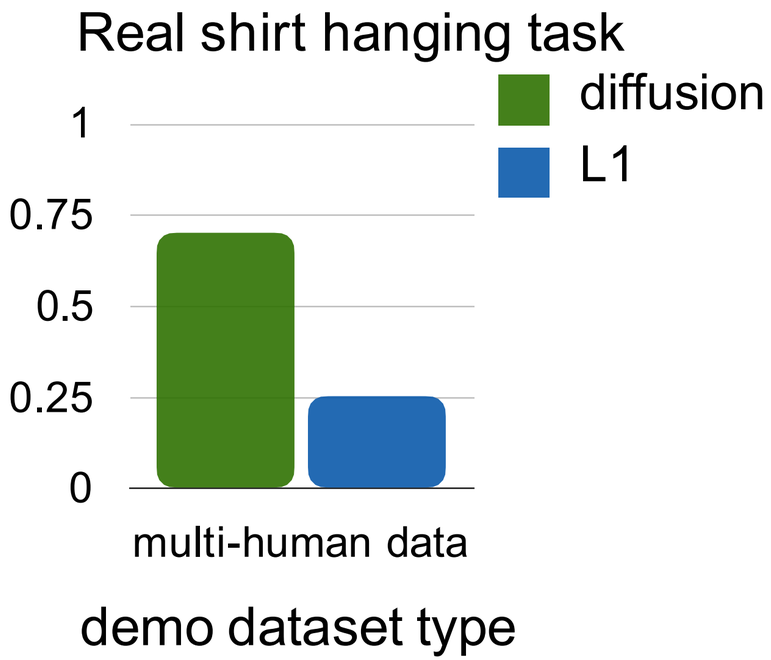
$$\min_\theta - \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\mathcal{D}}[\log \pi_\theta(\mathbf{a}|\mathbf{s})]$$ with expressive distribution $\pi(\cdot|\mathbf{s})$

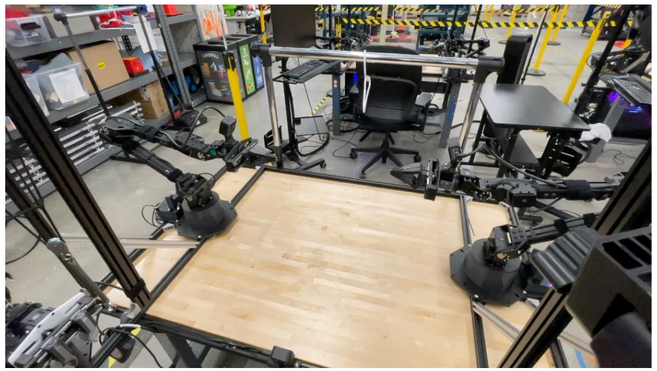
maximize the log probability of the demo actions under the policy

2. Deploy learned policy θ

Imitation learning - version 1 vs version 0







https://diffusion-policy.cs.columbia.edu/
https://aloha-unleashed.github.io/

Robotics: Imitation learning + expressive policies

Physical Intelligence π0

https://www.youtube.com/watch?v=YyXCMhnb_lU

diffusion

NVIDIA Gr00t N1

https://youtu.be/unnkmL6QJ78?feature=shared

diffusion

Figure Helix

https://youtu.be/Z3yQHYNXPws?feature=shared

diffusion

OpenVLA

https://openvla.github.io/

discretize + autoregressive prediction

Autonomous driving: Imitation learning + expressive policies

Waymo EMMA

https://waymo.com/blog/2024/10/introducing-emma/

discretize + autoregressive prediction

Wayve LINGO-2

https://wayve.ai/thinking/lingo-2-driving-with-language/

discretize + autoregressive prediction

Summary so far

Data from one consistent demonstrator	Multimodal data, e.g. from multiple demonstrators
Unimodal policy distribution is enough	Need expressive generative model for policy

Summary so far

- Key idea: Train expressive policy class via generative modeling on dataset of demonstrations.
  - Algorithm is fully *offline*

Definitions.

offline: using only an existing dataset, no new data from learned policy

online: using new data from learned policy

\+ no need for data from policy (online data can be unsafe, expensive to collect)

\+ no need to define a reward function

\- may need **a lot** of data for reliable performance

The plan for today

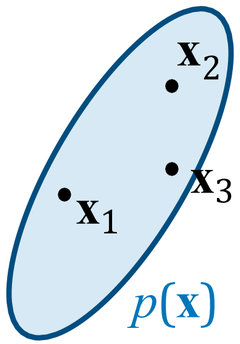
Imitation Learning

1. Imitation learning basics
2. Learning expressive policy distributions
3. Learning from online interventions
4. Time permitting: how to collect demonstrations

What can go wrong in imitation learning?

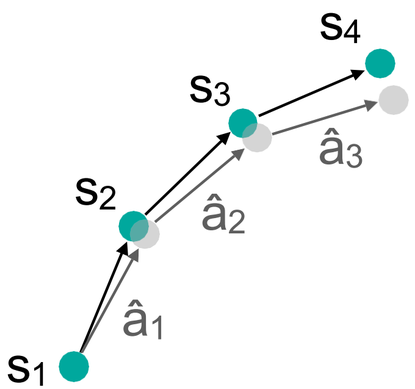
Compounding errors

Supervised learning



Inputs independent of predicted labels $\hat{\mathbf{y}}$

Supervised learning of behavior



Predicted actions affect next state.

Errors can lead to drift away from the data distribution!

Errors can then compound!

$$p_{expert}(\mathbf{s}) \neq p_{\pi}(\mathbf{s})$$

states visited by expert

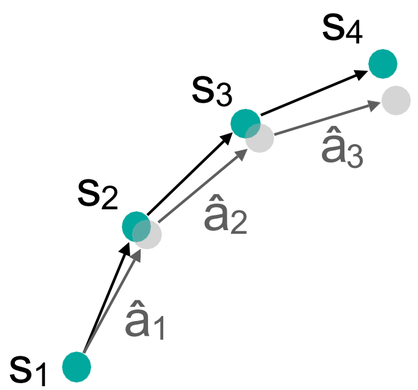
states visited by learned policy

"covariate shift"

What can go wrong in imitation learning?

Compounding errors

Supervised learning of behavior



Predicted actions affect next state.

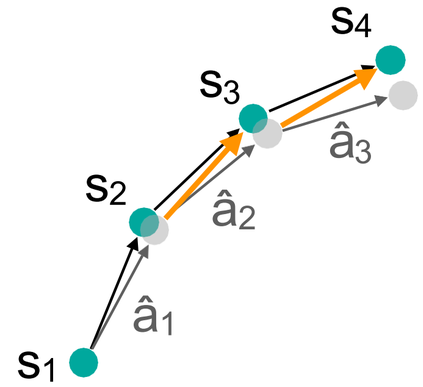
Errors can lead to drift away from the data distribution!

Errors can then compound!

$$p_{expert}(\mathbf{s}) \neq p_{\pi}(\mathbf{s})$$

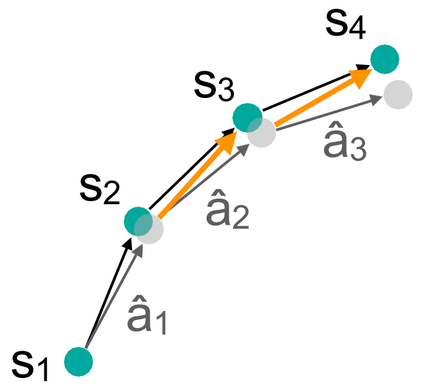
Solutions?

1. Collect A LOT of demo data & hope for the best.
2. Collect corrective behavior data



Addressing Compounding Errors with Interventions

Collect corrective behavior data



1. Roll out learned policy π_θ: $\mathbf{s}'_1, \hat{\mathbf{a}}_1, \ldots, \mathbf{s}'_T$
2. Query expert action at visited states $\mathbf{a}^* \sim \pi_{expert}(\cdot \mid \mathbf{s}')$
3. Aggregate corrections with existing data $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}' \mathbf{a}^*)\}$
4. Update policy $\min_\theta \mathcal{L}(\pi_\theta, \mathcal{D})$

"dataset aggregation" (DAgger)

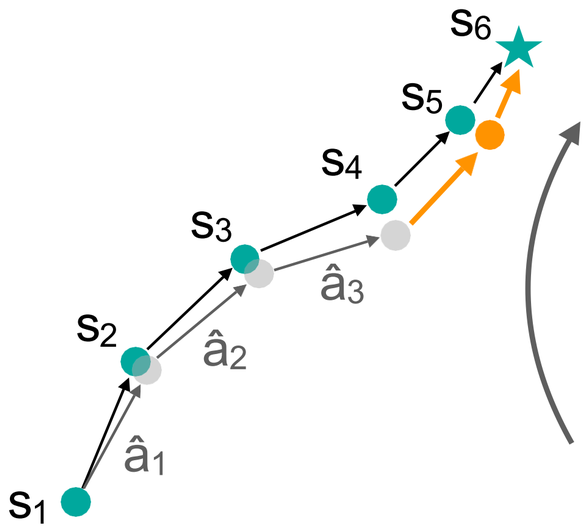
+ data-efficient way to learn from an expert

- can be challenging to query expert when agent has control

Is there another way to collect corrective data?

Addressing Compounding Errors with Interventions

Collect corrective behavior data while *taking full control*



1. Start to roll-out learned policy π_θ: $\mathbf{s}'_1, \hat{\mathbf{a}}_1, \ldots, \mathbf{s}'_t$
2. Expert intervenes at time when policy makes mistake
3. Expert provides (partial) demonstration $\mathbf{s}'_t, \mathbf{a}^*_t, \ldots, \mathbf{s}'_T$
4. Aggregate new demos with existing data $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}'^i \mathbf{a}^{*i})\}; i \geq t$
5. Update policy $\min_\theta \mathcal{L}(\pi_\theta, \mathcal{D})$

"human gated DAgger"

+ (much) more practical interface for providing corrections

- can be hard to catch mistakes quickly in some application domains

Question: could you automatically detect when intervention is needed?

Policy Gradients

CS 224R

Recap

state $\mathbf{s}_t$ - the state of the "world" at time t

action $\mathbf{a}_t$ - the decision taken at time t

trajectory τ - sequence of states/observations and actions

$$(\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \ldots, \mathbf{s}_T, \mathbf{a}_T)$$

reward function $r(\mathbf{s}, \mathbf{a})$ - how good is $\mathbf{s}, \mathbf{a}$?

policy $\pi(\mathbf{a}|\mathbf{s})$ or $\pi(\mathbf{a}|\mathbf{o})$ - behavior, usually what we are trying to learn

Goal: learn policy π_θ that maximizes *expected* sum of rewards:

$$\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Where: $p(\underbrace{\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T}_{p_\theta(\tau)}) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s_{t+1}}|\mathbf{s}_t, \mathbf{a}_t)$

Imitation learning from demonstrations $\mathcal{D}$

$$\min_{\theta} - \mathbb{E}_{(s,a)\sim\mathcal{D}}[\log \pi_\theta(a|s)]$$

+ Simple, scalable approach for learning performant behavior

- Cannot outperform demonstrator, doesn't allow improvement from practice

Definitions.

***offline*: using only an existing dataset, no new data from learned policy**

***online*: using new data from learned policy**

The plan for today

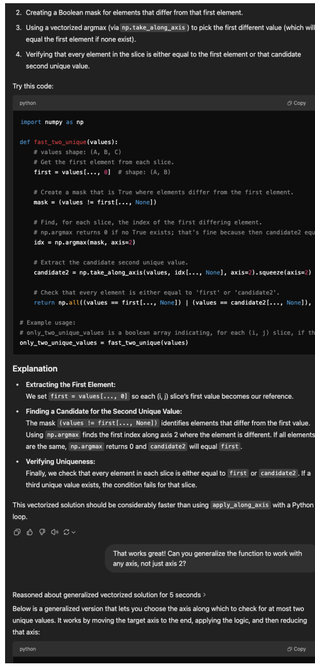
the basis for:

https://www.youtube.com/watch?v=xG7WkPU8tgs

Policy gradients: our first online RL algorithm

1. On-policy policy gradient
   a. Derivation and intuition of policy gradients
   b. Full algorithm
   c. How to make it better - causality and baselines
2. Off-policy policy gradients
   a. Importance sampling
   b. KL constraints

Source: Unitree


2. Creating a Boolean mask for elements that differ from that first element.
3. Using a vectorized argmax (via `np.take_along_axis`) to pick the first different value (which will equal the first element if none exist).
4. Verifying that every element in the slice is either equal to the first element or that candidate second unique value.

Try this code:

```python
import numpy as np

def fast_two_unique(values):
    # values shape: (A, B, C)
    # Get the first element from each slice.
    first = values[..., 0]  # shape: (A, B)

    # Create a mask that is True where elements differ from the first element.
    mask = (values != first[..., None])

    # Find, for each slice, the index of the first differing element.
    # np.argmax returns 0 if no True exists; that's fine because then candidate2 equ
    idx = np.argmax(mask, axis=-1)

    # Extract the candidate second unique value.
    candidate2 = np.take_along_axis(values, idx[..., None], axis=-1).squeeze(axis=-1)

    # Check that every element is either equal to 'first' or 'candidate2'.
    return np.all((values == first[..., None]) | (values == candidate2[..., None]),

# Example usage:
# only_two_unique_values is a boolean array indicating, for each (i, j) slice, if th
only_two_unique_values = fast_two_unique(values)
```

Explanation

- **Extracting the First Element:**
  We set `first = values[..., 0]` so each (i, j) slice's first value becomes our reference.
- **Finding a Candidate for the Second Unique Value:**
  The mask `(values != first[..., None])` identifies elements that differ from the first value. Using `np.argmax` finds the first index along axis 2 where the element is different. If all elements are the same, `np.argmax` returns 0 and `candidate2` will equal `first`.
- **Verifying Uniqueness:**
  Finally, we check that every element in each slice is either equal to `first` or `candidate2`. If a third unique value exists, the condition fails for that slice.

This vectorized solution should be considerably faster than using `apply_along_axis` with a Python loop.

That works great! Can you generalize the function to work with any axis, not just axis 2?

Reasoned about generalized vectorized solution for 5 seconds

Below is a generalized version that lets you choose the axis along which to check for at most two unique values. It works by moving the target axis to the end, applying the logic, and then reducing that axis:

```python
```

Key learning goals:

- Key intuition behind policy gradients
- How to implement, when to use policy gradients

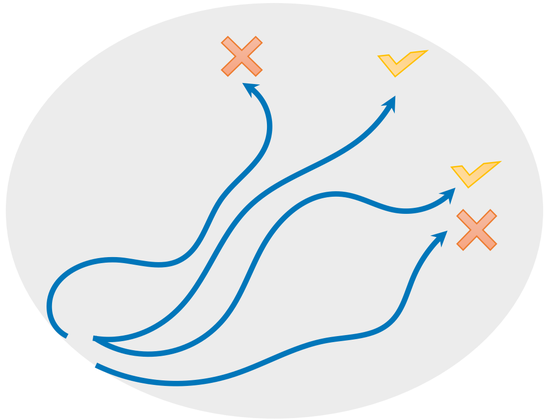
Online RL Outline

First: Initialize the policy (randomly, with imitation learning, with heuristics)

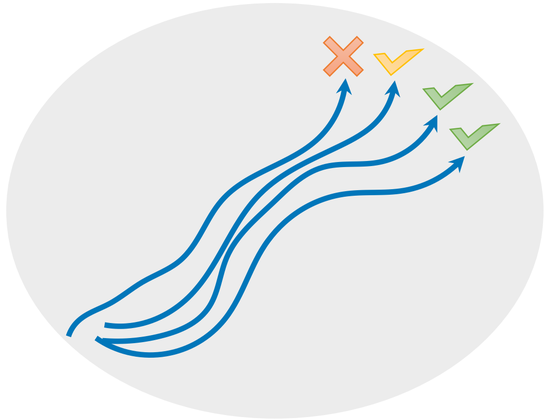
Run policy to collect batch of data

Improve policy using batch of data

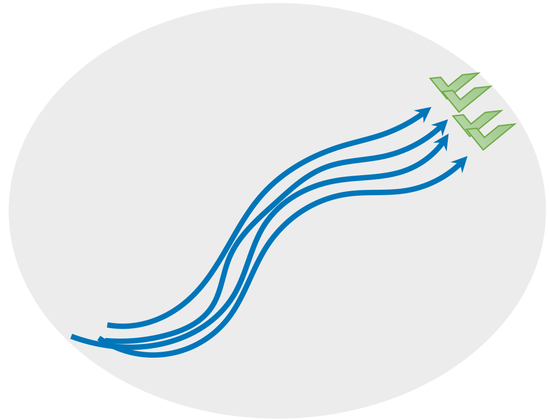
iteration 1



iteration 2

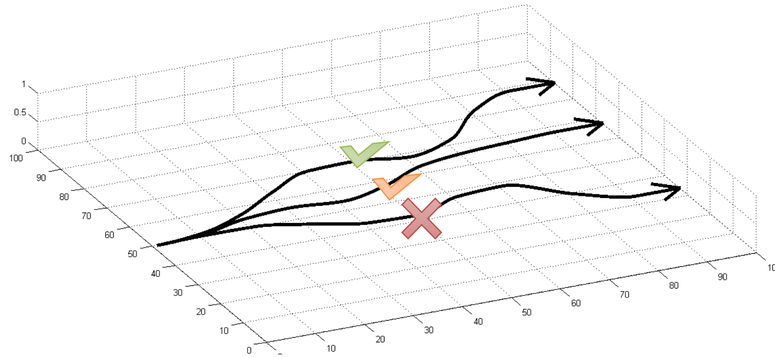


iteration 3



Evaluating the RL objective

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]}_{J(\theta)}$$



$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i\sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

sum over samples from π_θ

Slide adapted from Sergey Levine

Can we get the gradient of the RL objective?

Let's start in terms of trajectories.

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]}_{J(\theta)}$$

a convenient identity

$$p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) = p_\theta(\tau)\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = \nabla_\theta p_\theta(\tau)$$

$$J(\theta) = E_{\tau\sim p_\theta(\tau)}[\underbrace{r(\tau)}_{\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)}] = \int p_\theta(\tau) r(\tau) d\tau$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) r(\tau) d\tau = E_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$$

Slide adapted from Sergey Levine

Can we get the gradient of the RL objective?

From trajectories to final form.

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$$

$$\nabla_\theta \left[\log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$p_\theta(\tau) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\log p_\theta(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

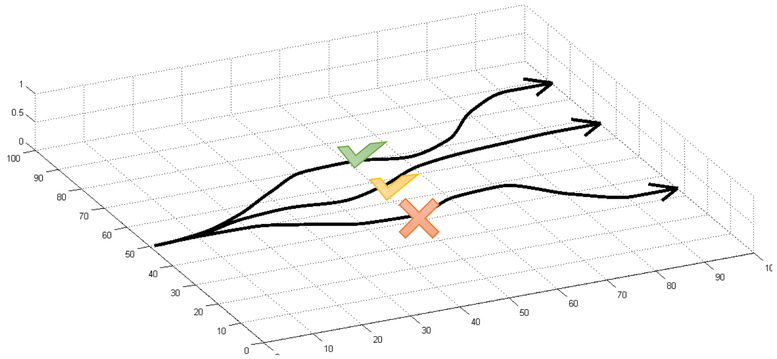
Slide adapted from Sergey Levine

Estimating the gradient

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$



Run policy to collect batch of data

Improve policy using batch of data

Full algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i | \mathbf{s}_t^i) \right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

“REINFORCE algorithm”, vanilla policy gradient

Slide adapted from Sergey Levine

What does the gradient mean?

$$\nabla_\theta J(\theta) \approx \left(\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} r(s_{i,t}, a_{i,t})\right)\left(\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} \mid s_{i,t})\right)$$

Imitation gradient, but weighted by reward

Recall: imitation learning

$$\min_\theta -\mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\mathcal{D}}\left[\log \pi_\theta(\mathbf{a}|\mathbf{s})\right]$$

$$\nabla_\theta J_{\mathrm{BC}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t-1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)$$

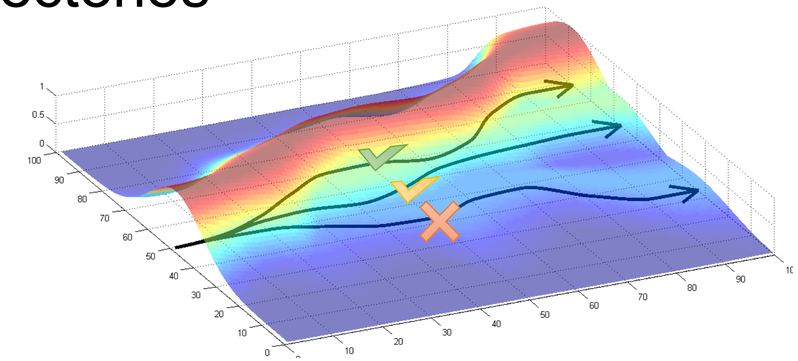
Intuition:

Increase likelihood of actions you took in high reward trajectories.

Decrease likelihood of actions you took in negative reward trajectories

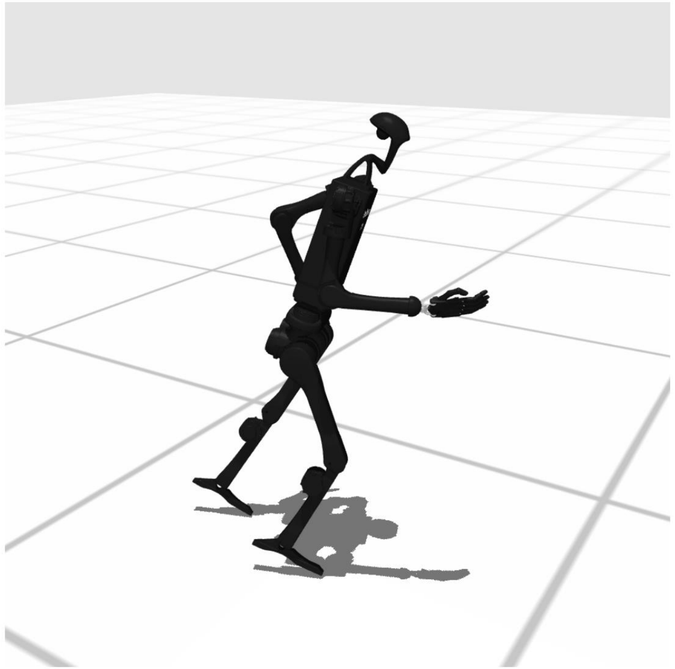
i.e. do more of the good stuff, less of the bad stuff.

formalization of “trial-and-error”



What does the gradient do?

Example: learning humanoid walking in simulation



reward: $r(\mathbf{s}, \mathbf{a}) = \textit{forward velocity of robot}$

(can be negative if robot goes backwards)

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

τ^1: falls backwards ✗

τ^2: falls forwards ✓

τ^3: manages to stand still ✓

τ^4: one small step forward then falls backwards ✗

τ^5: one large step backwards then small step forwards ✗

2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$

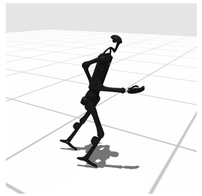
-> will encourage policy to fall forward, and not take step forward 😨

Policy gradient is noisy / high-variance

Improving the gradient

Using causality

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$



τ^5: one large step backwards then small step forwards ✖

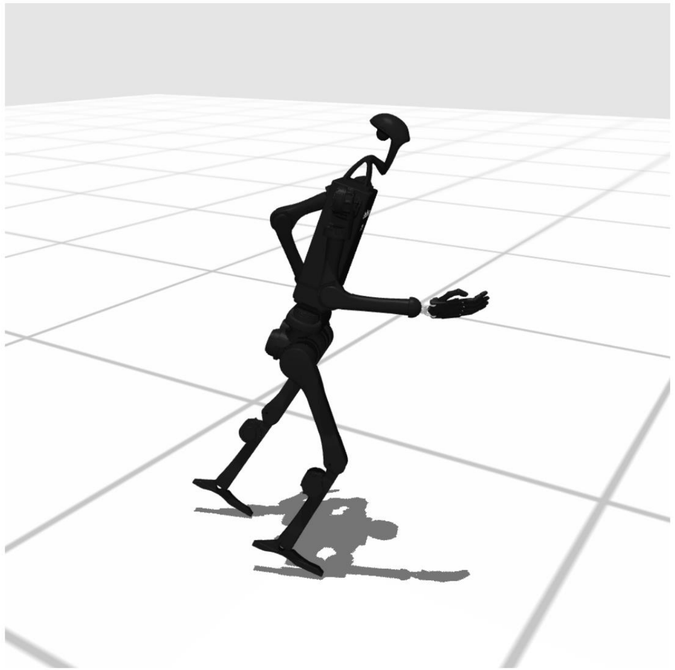
Policy behavior at time t does not affect rewards at time $t' < t$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

sum of future rewards

What does the gradient do?

Example: learning humanoid walking in simulation



reward: $r(\mathbf{s}, \mathbf{a}) = \textit{forward velocity of robot}$

(can be negative if robot goes backwards)

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

τ^1: falls forwards ✔

τ^2: slowly stumbles forwards ✔

τ^3: steadily walks forwards ✔

τ^4: runs forwards ✔

2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$

-> encourages policy to fall, stumble forward some of the time 😨

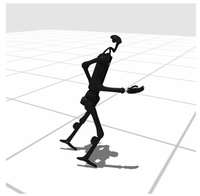
Policy gradient is noisy / high-variance

sensitive to reward scale

Improving the gradient

Introducing baselines

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) r(\tau) \right]$$



τ^1: falls forwards ✔

τ^4: runs forwards ✔

Improving the gradient

Introducing baselines

a convenient identity

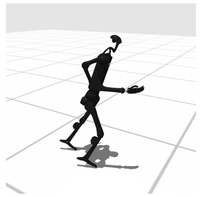
$$p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) = \nabla_\theta p_\theta(\tau)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\nabla_\theta \log p_\theta(\tau)\,(r(\tau) - b)\right]$$

$$b = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$$

subtracting a constant

"baseline"



τ^1: falls forwards ✔

τ^4: runs forwards ✔

If we subtract average reward, we get negative gradients for below-average behavior. 🥳

But, can we even do that? 🤔

$$E[\nabla_\theta \log p_\theta(\tau) b] = \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) b\, d\tau = \int \nabla_\theta p_\theta(\tau) b\, d\tau = b\nabla_\theta \int p_\theta(\tau) d\tau = b\nabla_\theta 1 = 0$$

Subtracting a constant baseline is **unbiased** (change to the gradient is 0 in expectation)

and can reduce **variance** of the gradient

Average reward is a pretty good baseline.

What does the gradient do?

Example: learning to fold a jacket

https://www.youtube.com/watch?v=e2aMSzMYUh0

reward: $r(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{neatly folded} \\ 0.5 & \text{folded with some wrinkles} \\ 0 & \text{not folded} \end{cases}$

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

τ^1: doesn't touch the jacket

τ^2: folds only the sleeves

τ^3: flattens the jacket but does not fold it

τ^4: folds the jacket

2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) - b\right)$

-> gradient is constant for all but one trajectory 😨

Policy gradient is still noisy / high-variance

Best with dense rewards, large batches.

How to implement policy gradients?

Our gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} \mid s_{i,t}) \left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) - b \right)$$

Computing these individually is inefficient.
(N*T backwards passes)

Can we use automatic differentiation on full objective?

Implement "surrogate objective" whose gradient is the same as ∇J

$$\tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

weighted maximum likelihood

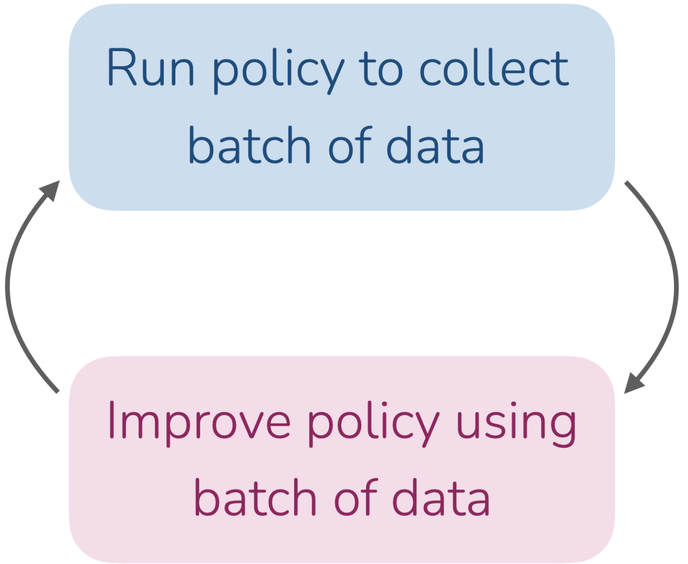
Cross-entropy for discrete action policy, squared error for Gaussian policy

Summary so far

Estimating gradient of RL objective
- log gradient trick
- weigh policy likelihood by future rewards
- subtract baseline (e.g. average reward)
- even with tricks, gradient is noisy

First reinforcement learning algorithm
- collect batch of data, improve policy by applying gradient
- formalizes trial-and-error learning

Key intuition: do more high reward stuff, less low reward stuff



What else is troublesome about policy gradients?

Latest version of our gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) - b \right) \right]$$

assumes samples from current policy π_θ

Full algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$
2. compute $\nabla_\theta J(\theta)$ using $\{\tau^i\}$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ <- we change θ right here

Need to recollect data every gradient step! 😱

Vanilla policy gradient is **on-policy**.

attribute of online RL algorithms

Definitions.

on-policy: update uses only data from current policy

off-policy: update can reuse data from other, past policies

Off-policy version of policy gradient?

Importance sampling

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)]$$

What if we want to use samples from $\bar{p}(\tau)$?

(e.g. previous policy)

$$J(\theta) = E_{\tau \sim \bar{p}(\tau)}\left[\frac{p_\theta(\tau)}{\bar{p}(\tau)} r(\tau)\right]$$

$$\frac{p_\theta(\tau)}{\bar{p}(\tau)} = \frac{\cancel{p(\mathbf{s}_1)} \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}}{\cancel{p(\mathbf{s}_1)} \prod_{t=1}^{T} \bar{\pi}(\mathbf{a}_t|\mathbf{s}_t) \cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}} = \frac{\prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T} \bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)}$$

Importance sampling

Using proposal distribution q

$$\begin{aligned} E_{x \sim p(x)}[f(x)] &= \int p(x) f(x) dx \\ &= \int \frac{q(x)}{q(x)} p(x) f(x) dx \\ &= \int q(x) \frac{p(x)}{q(x)} f(x) dx \\ &= E_{x \sim q(x)}\left[\frac{p(x)}{q(x)} f(x)\right] \end{aligned}$$

Note: Important for q to have non-zero support for high probability $p(x)$

Off-policy version of policy gradient?

Importance sampling

$$\frac{p_\theta(\tau)}{\bar{p}(\tau)} = \frac{\prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T} \bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)}$$

Say we want to update our latest policy $\pi_{\theta'}$ but we want to use samples from π_θ

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) - b \right) \right]$$

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} \left(\sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) - b \right) \right]$$

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\prod_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left(\sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) - b \right) \right]$$

This can become very small or very large, for larger T

Off-policy version of policy gradient?

Importance sampling

Say we want to update our latest policy $\pi_{\theta'}$ but we want to use samples from π_θ

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\prod_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left(\sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) - b \right) \right]$$

This can become very small or very large, for larger T

What if we consider the expectation *over timesteps* instead of trajectories?

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}{\pi_\theta(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

Much less likely to explode/vanish …but, hard to measure $\frac{\pi_{\theta'}(\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{s}_{i,t})}$

Common final form

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

often approximated as 1

Off-policy policy gradient

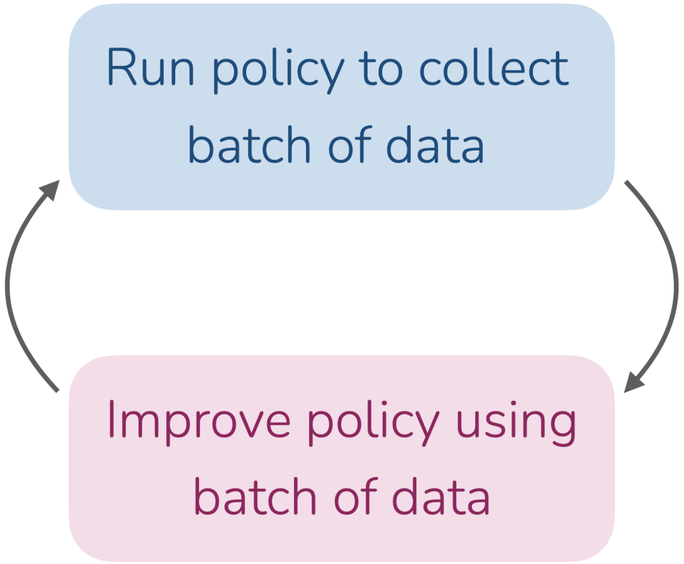
Common final form

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

Full algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
2. compute $\nabla_\theta J(\theta)$ using $\{\tau^i\}$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Can take multiple gradient steps on the same batch



What if our policy changes **a lot** before sampling new data?

Data no longer reflects states that policy will visit. Gradient estimate less accurate.

Off-policy policy gradient

Common final form

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

What if our policy changes **a lot** before sampling new data?

Data no longer reflects states that policy will visit. Gradient estimate less accurate.

Can we constrain the policy to not stray too far during gradient updates?

Common choice: $\mathbb{E}_{\mathbf{s} \sim \pi_\theta} \left[D_{KL}(\pi_\theta(\cdot \mid \mathbf{s}) \| \pi_{\theta'}(\cdot \mid \mathbf{s})) \right] \leq \delta$

Review

Online RL via policy gradients

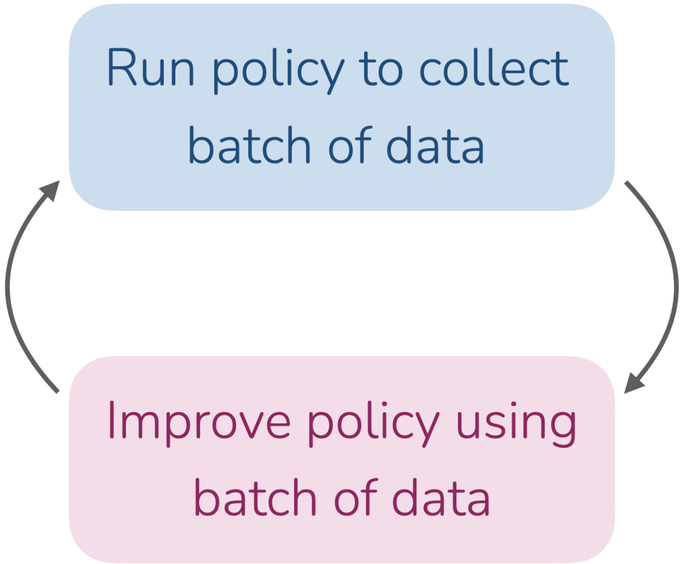
- On-policy algorithm, differentiating the RL objective
- Baselines, causality for reducing gradient variance
- collect batch of data, improve policy by applying gradient

Derived off-policy policy gradient

- importance sampling
- KL constraint on policy
- collect batch of data, apply multiple gradient updates

Intuition

- Do more high reward stuff, less low reward stuff
- Gradient still very noisy, best with large batch sizes and dense rewards



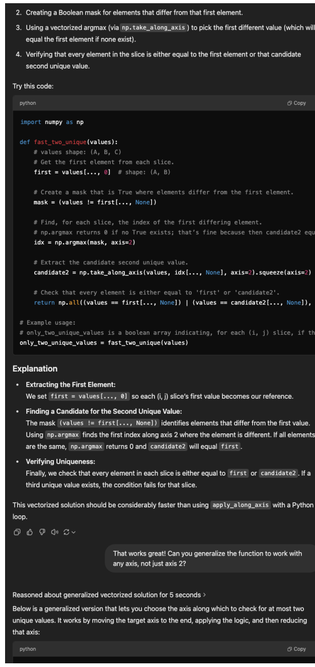
The plan for today

the basis for:

Policy gradients: our first online RL algorithm

https://www.youtube.com/watch?v=xG7WkPU8tgs

1. On-policy policy gradient
   a. Derivation and intuition of policy gradients
   b. Full algorithm
   c. How to make it better - causality and baselines
2. Off-policy policy gradients
   a. Importance sampling
   b. KL constraints


2. Creating a Boolean mask for elements that differ from that first element.
3. Using a vectorized argmax (via `np.take_along_axis`) to pick the first different value (which will equal the first element if none exist).
4. Verifying that every element in the slice is either equal to the first element or that candidate second unique value.

Try this code:

```python
import numpy as np

def fast_two_unique(values):
    # values shape: (A, B, C)
    # Get the first element from each slice.
    first = values[..., 0]  # shape: (A, B)

    # Create a mask that is True where elements differ from the first element.
    mask = (values != first[..., None])

    # Find, for each slice, the index of the first differing element.
    # np.argmax returns 0 if no True exists; that's fine because then candidate2 equ
    idx = np.argmax(mask, axis=2)

    # Extract the candidate second unique value.
    candidate2 = np.take_along_axis(values, idx[..., None], axis=2).squeeze(axis=2)

    # Check that every element is either equal to 'first' or 'candidate2'.
    return np.all((values == first[..., None]) | (values == candidate2[..., None]),

# Example usage:
# only_two_unique_values is a boolean array indicating, for each (i, j) slice, if th
only_two_unique_values = fast_two_unique(values)
```

Explanation

- **Extracting the First Element:**
  We set `first = values[..., 0]` so each (i, j) slice's first value becomes our reference.
- **Finding a Candidate for the Second Unique Value:**
  The mask `(values != first[..., None])` identifies elements that differ from the first value. Using `np.argmax` finds the first index along axis 2 where the element is different. If all elements are the same, `np.argmax` returns 0 and `candidate2` will equal `first`.
- **Verifying Uniqueness:**
  Finally, we check that every element in each slice is either equal to `first` or `candidate2`. If a third unique value exists, the condition fails for that slice.

This vectorized solution should be considerably faster than using `apply_along_axis` with a Python loop.

That works great! Can you generalize the function to work with any axis, not just axis 2?

Reasoned about generalized vectorized solution for 5 seconds

Below is a generalized version that lets you choose the axis along which to check for at most two unique values. It works by moving the target axis to the end, applying the logic, and then reducing that axis:

```python
```

Source: Unitree

Key learning goals:

- Key intuition behind policy gradients
- How to implement, when to use policy gradients

Next time

Actor critic methods

—> build closely on policy gradients!

—> basis for popular algorithms like PPO

Actor Critic Methods

CS 224R

Recap of Last Time

Online reinforcement learning with policy gradients

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

samples from policy

policy log likelihood

reward to go

baseline

Do more of the above average stuff, less of the below average stuff.

Recap of Last Time

attribute of online RL algorithms

Vanilla policy gradients is fully **on-policy**.

—> only one gradient step per batch of data

Definitions.

on-policy: update uses only data from current policy

off-policy: update can reuse data from other, past policies

Importance weights for *off-policy* policy gradient:

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

Note: Not a silver bullet. Only suitable when policies are very similar.

The plan for today

Actor critic methods

1. Improving policy gradients
2. How to estimate the value of a policy
    a. Sample & directly supervise
    b. Use your own estimate
    c. Something in-between?
3. Off-policy actor-critic
    a. Importance weights & constraining step size
    b. Full off-policy version with replay buffers

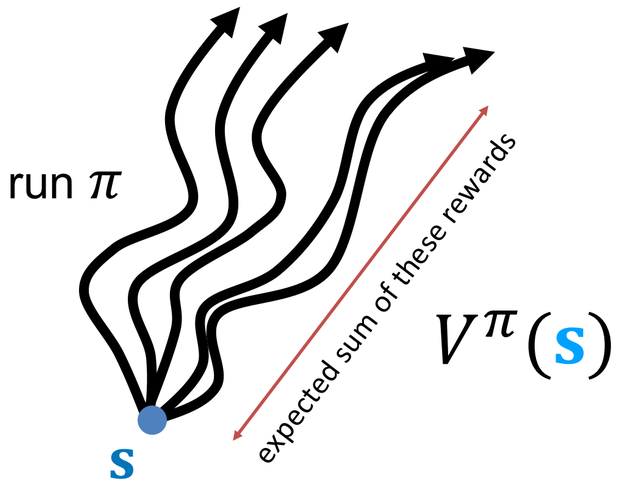
Key learning goals:

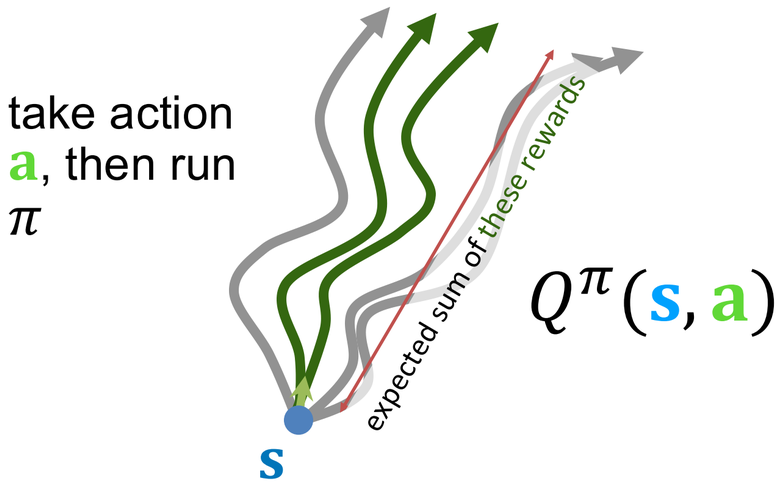
- How to estimate how good a state and action is for a policy
- How to use those estimates to form a more efficient RL algorithm

Revisiting Some Useful Objects

value function $V^{\pi}(\mathbf{s})$ - future expected rewards starting at $\mathbf{s}$ and following π

Q-*function* $Q^{\pi}(\mathbf{s}, \mathbf{a})$ - future expected rewards starting at $\mathbf{s}$, taking $\mathbf{a}$, then following π



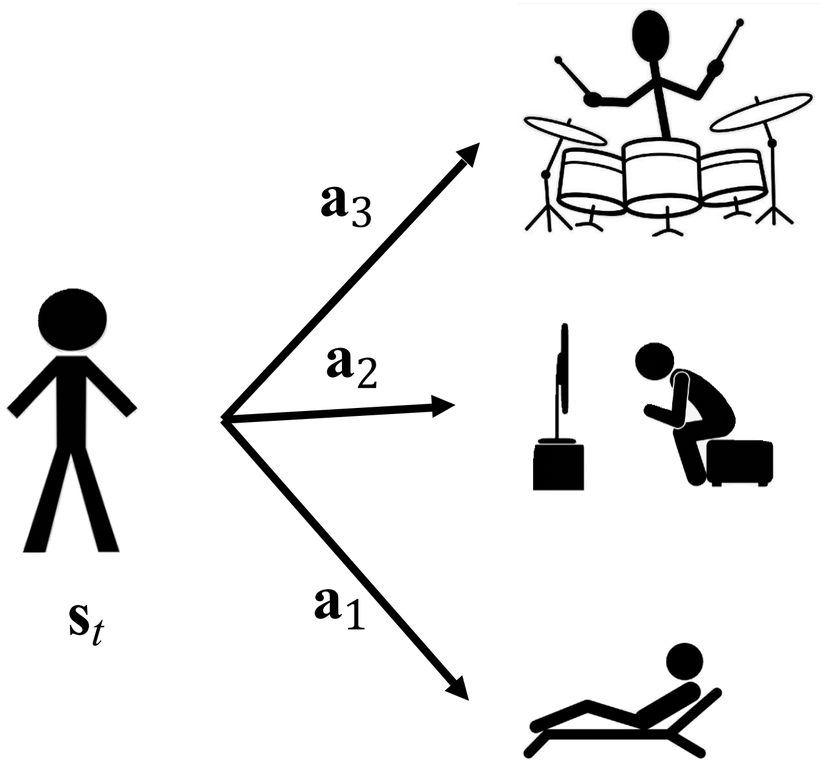


Useful relation: $V^{\pi}(\mathbf{s}) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[Q^{\pi}(\mathbf{s}, \mathbf{a})]$

advantage $A^{\pi}(\mathbf{s}, \mathbf{a})$ - how much better it is to take $\mathbf{a}$ than to follow policy π at state $\mathbf{s}$

$$A^{\pi}(\mathbf{s}, \mathbf{a}) = Q^{\pi}(\mathbf{s}, \mathbf{a}) - V^{\pi}(\mathbf{s})$$

Let's look at an example



Reward = 1 if I can play it in a month, 0 otherwise

Question:

For each action, what are these?

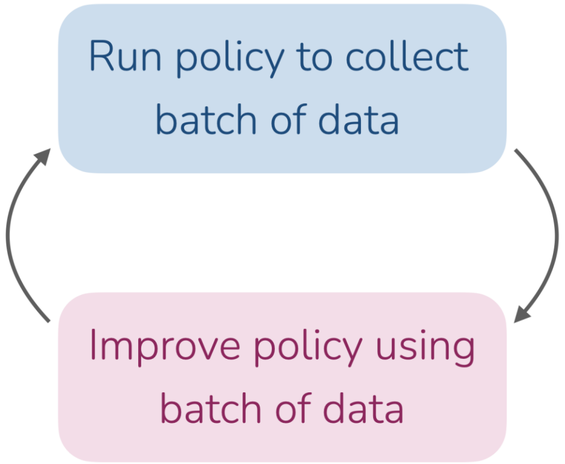
Value function: $V^{\pi}(\mathbf{s}_t) = ?$

Q function: $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = ?$

Advantage function: $A^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = ?$

Current policy: $\pi(\mathbf{a}_1 | \mathbf{s}) = 1 \forall \mathbf{s}$

What is dissatisfying about policy gradients?



$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right) - b\right)$$

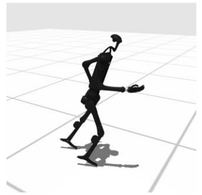
samples from policy

policy log likelihood

reward to go

baseline

https://www.youtube.com/watch?v=e2aMSzMYUh0



τ^4: one small step forward then falls backwards

τ^2: folds only the sleeves

τ^3: flattens the jacket but does not fold it

τ^4: folds the jacket

-> pushes down likelihood of step forward

-> with sparse rewards, don't utilize τ^2 or τ^3

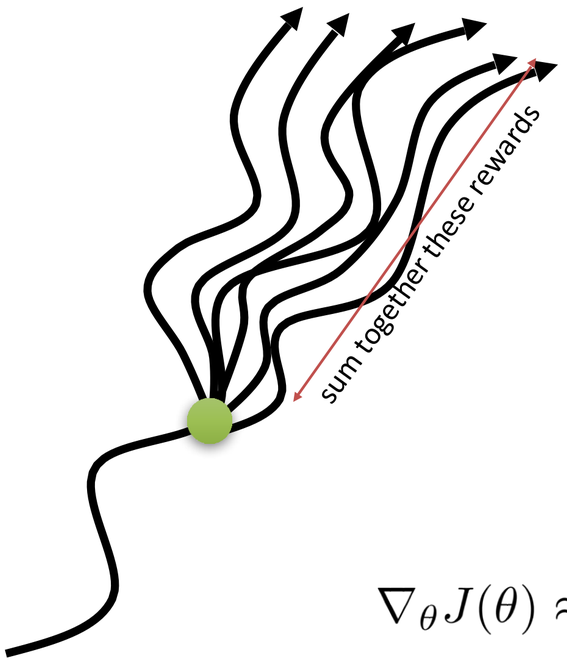
Policy gradients doesn't make efficient use of data! 💡 Can we learn what is good & bad?

Improving policy gradients

Estimating reward to go

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

"reward to go"



estimate of future rewards if we take $\mathbf{a}_{i,t}$ in state $\mathbf{s}_{i,t}$

$\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$

Can we get a better estimate?

$$\sum_{t'=t}^{T} E_{\pi_\theta} \left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t \right] = Q(\mathbf{s}_t, \mathbf{a}_t)$$

true expected rewards to go

This would be way better!

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \, Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

Improving policy gradients

What about baselines?

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})(Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) - \cancel{b})$$

$V^\pi(\mathbf{s}_t)$

$$b = \text{average reward} = \frac{1}{N}\sum_i Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$$

$$V(\mathbf{s}_t) = E_{\mathbf{a}_t\sim\pi_\theta(\cdot|\mathbf{s}_t)}[Q(\mathbf{s}_t,\mathbf{a}_t)]$$

Recall: $A^\pi(\mathbf{s}_t,\mathbf{a}_t) = Q^\pi(\mathbf{s}_t,\mathbf{a}_t) - V^\pi(\mathbf{s}_t)$

With baseline:

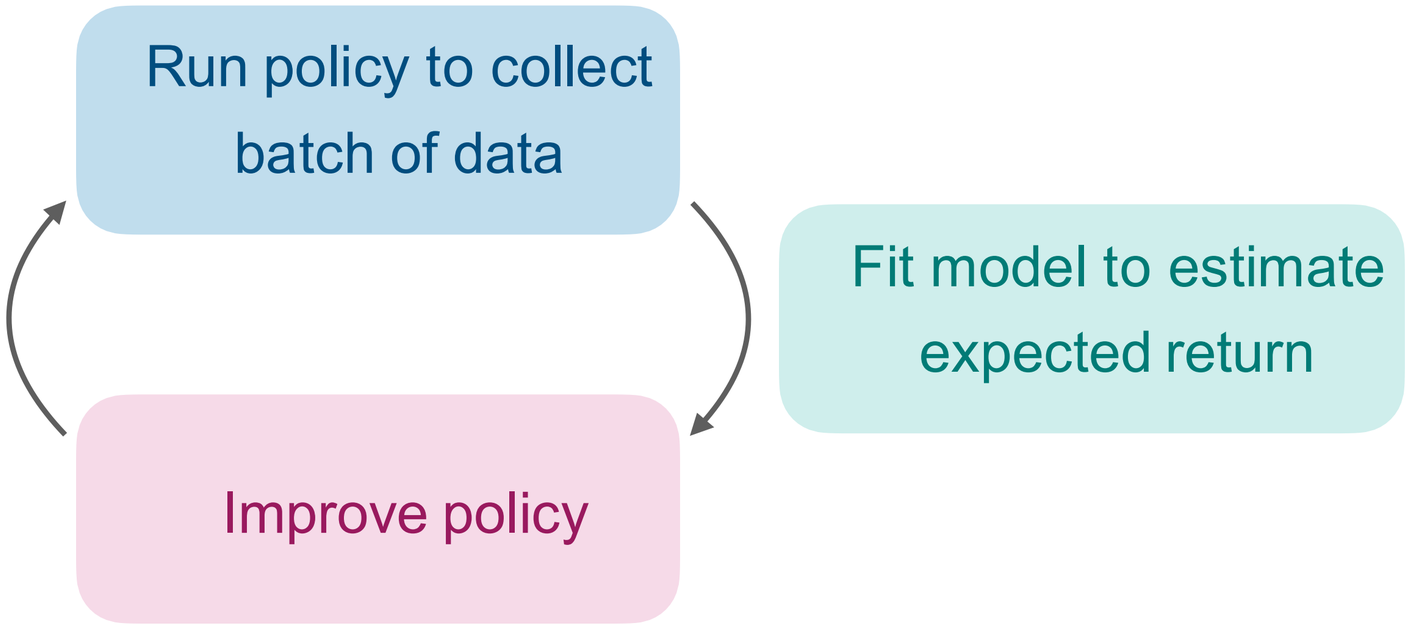
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})A(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$$

Better estimates of lead to less noisy gradients!

Online RL Outline

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

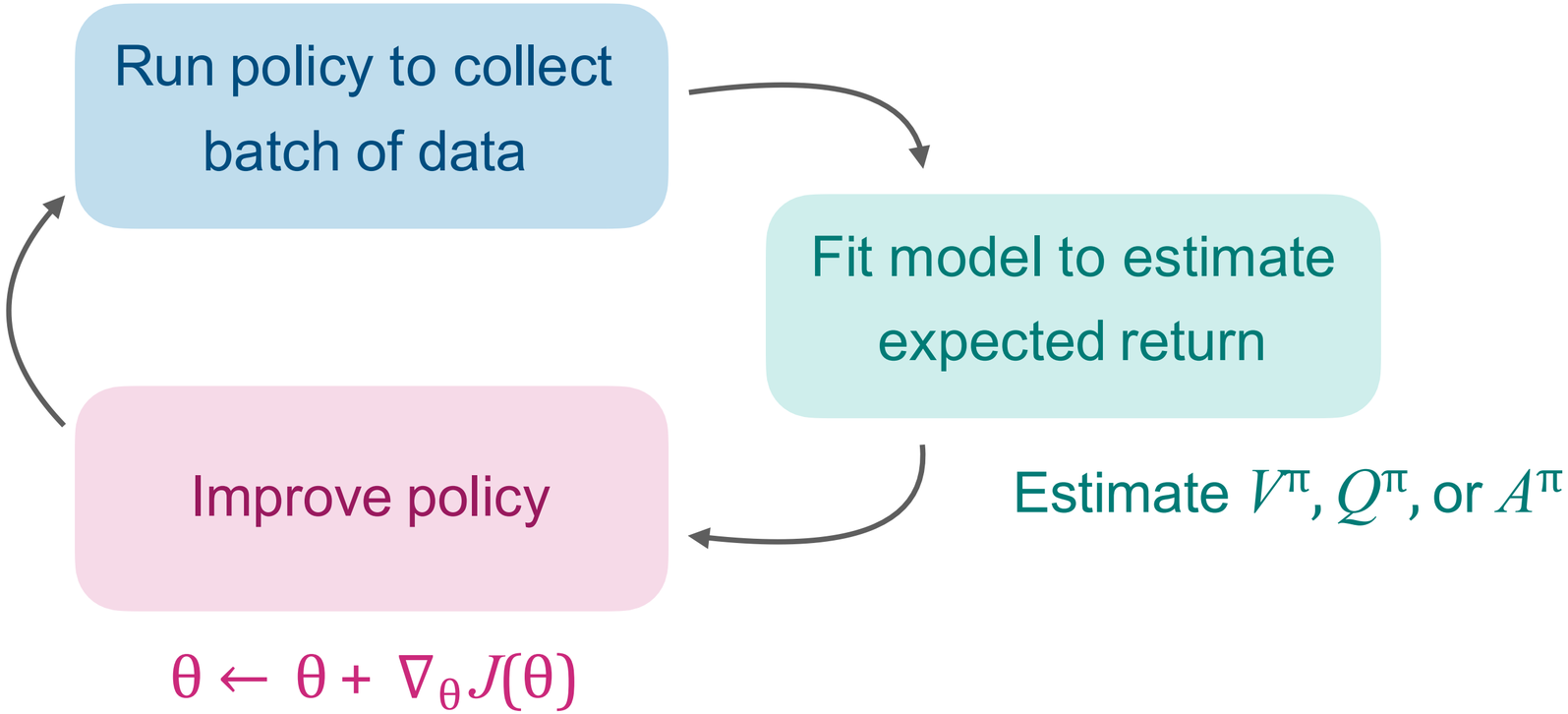
First: Initialize the policy (randomly, with imitation learning, with heuristics)



Online RL Outline

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

First: Initialize the policy (randomly, with imitation learning, with heuristics)



The plan for today

Actor critic methods

1. Improving policy gradients
2. **How to estimate the value of a policy**
   a. Sample & directly supervise
   b. Use your own estimate
   c. Something in-between?
3. Off-policy actor-critic
   a. Importance weights & constraining step size
   b. Full off-policy version with replay buffers

Key learning goals:

- How to estimate how good a state and action is for a policy
- How to use those estimates to form a better RL algorithm

Estimating expected return

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})A(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$$

Should we fit V^π, Q^π, or A^π?

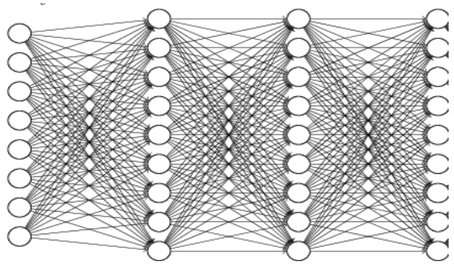
$$A^\pi(\mathbf{s}_t,\mathbf{a}_t) = Q^\pi(\mathbf{s}_t,\mathbf{a}_t) - V^\pi(\mathbf{s}_t)$$

$$\begin{aligned} Q^\pi(\mathbf{s}_t,\mathbf{a}_t) &= \textstyle\sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t\right] \\ &= r(\mathbf{s}_t,\mathbf{a}_t) + \textstyle\sum_{t'=t+1}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t\right] \\ &= r(\mathbf{s}_t,\mathbf{a}_t) + E_{\mathbf{s}_{t+1}\sim p(\cdot|\mathbf{s}_t,\mathbf{a}_t)}[V^\pi(\mathbf{s}_{t+1})] \\ &\approx r(\mathbf{s}_t,\mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) \end{aligned}$$

(use the sampled $\mathbf{s}_{t+1}$)

$$A^\pi(\mathbf{s}_t,\mathbf{a}_t) \approx r(\mathbf{s}_t,\mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)$$

Let's just fit V^π!

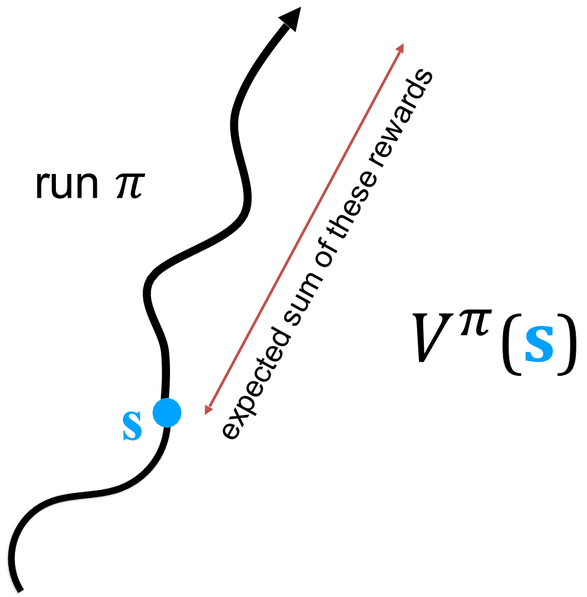


Estimating V^{π}

Version 1: Monte Carlo estimation

$$V^{\pi}(\mathbf{s}_t) \approx \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

<- original single sample estimate



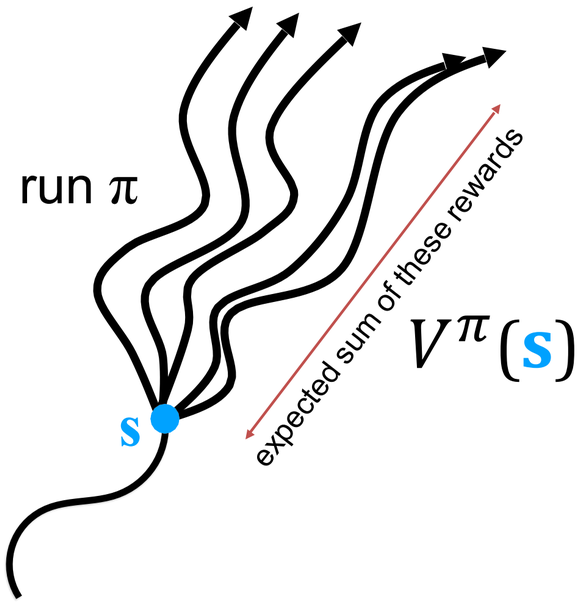
Estimating V^π

Version 1: Monte Carlo estimation

$$V^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$ <- original single sample estimate

$$V^\pi(\mathbf{s}_t) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$ <- multi-sample estimate

(but can't reset the world)



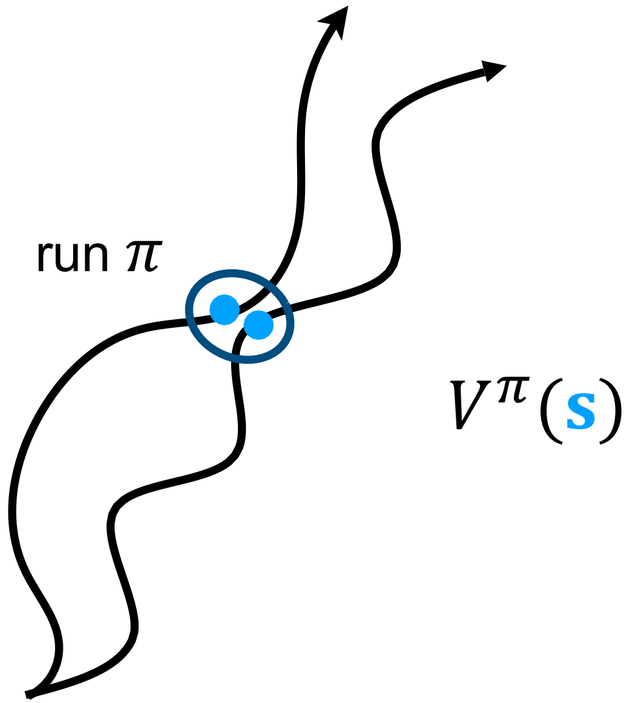
Estimating V^π

Version 1: Monte Carlo estimation

$$V^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$ <- original single sample estimate

$$V^\pi(\mathbf{s}_t) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$ <- multi-sample estimate

(but can't reset the world)

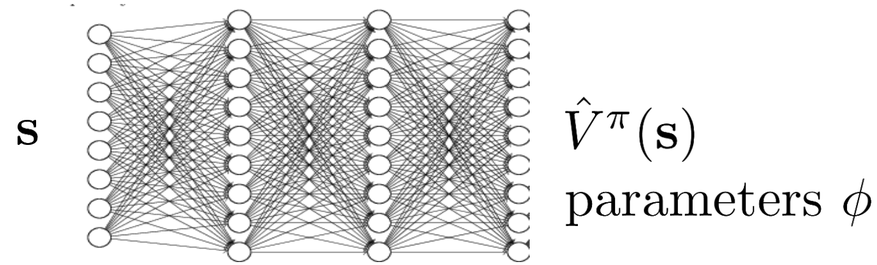


Step 1: Aggregate dataset of single sample estimates:

$$\left\{ \left(\mathbf{s}_{i,t}, \underbrace{\textstyle\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})}_{y_{i,t}} \right) \right\}$$

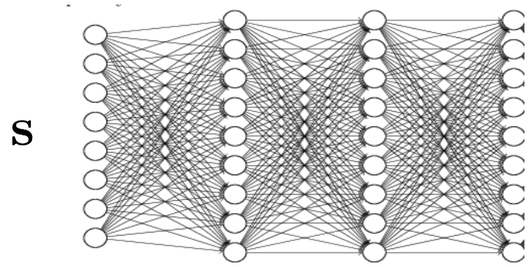
Step 2: Supervised learning to fit estimated value function

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$



Estimating V^π

Version 2: Bootstrapping



Monte Carlo target: $y_{i,t} = \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$

ideal target: $y_{i,t} = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_{i,t}\right] \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \sum_{t'=t+1}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_{i,t+1}]$

$\approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + V^\pi(\mathbf{s}_{i,t+1}) \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \underbrace{\hat{V}^\pi_\phi(\mathbf{s}_{i,t+1})}$

directly use previous fitted value function!

Step 1: Aggregate dataset of "bootstrapped" estimates:

training data: $\left\{\left(\mathbf{s}_{i,t}, \underbrace{r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}^\pi_\phi(\mathbf{s}_{i,t+1})}_{y_{i,t}}\right)\right\}$ *<- update labels every gradient update!*

Step 2: Supervised learning to fit estimated value function

$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i \left\|\hat{V}^\pi_\phi(\mathbf{s}_i) - y_i\right\|^2$$

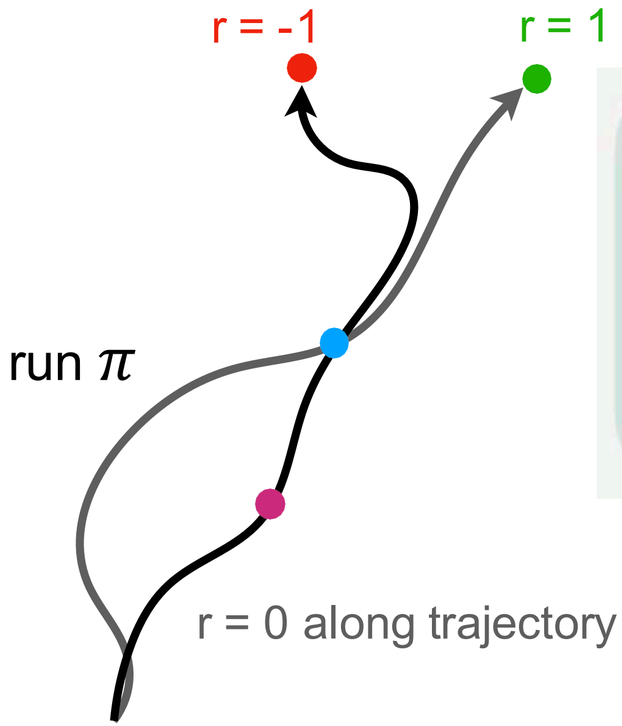
Also referred to as a form of **temporal difference learning**

Estimating V^π :Monte Carlo vs. Bootstrap

Monte Carlo $y_{i,t} = \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$ *supervise with roll-out's summed rewards*

Bootstrapped $y_{i,t} = r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$ *supervise using reward and current value estimate*

Let's look at an example:



Think-pair-share:

$\hat{V}^\pi_{MC}(\mathbf{s}) \approx ?$ $\hat{V}^\pi_{MC}(\mathbf{s}) \approx ?$

$\hat{V}^\pi_{TD}(\mathbf{s}) \approx ?$ $\hat{V}^\pi_{TD}(\mathbf{s}) \approx ?$

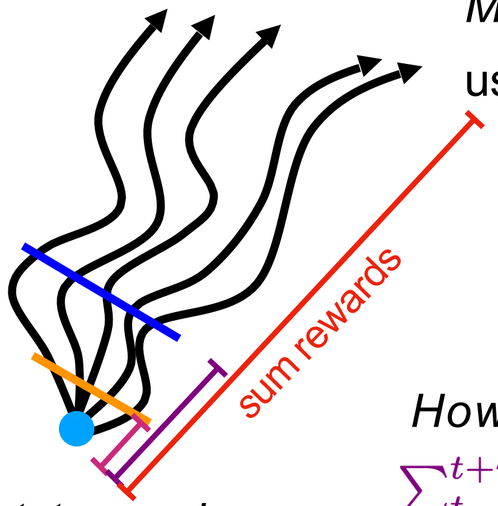
Is there middle ground?
Can we balance bias and variance?

Estimating V^π :Monte Carlo vs. Bootstrap

Monte Carlo $y_{i,t} = \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$ *supervise with roll-out's summed rewards*

Bootstrapped $y_{i,t} = r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$ *supervise using reward and current value estimate*

N-step returns $y_{i,t} = \sum_{t'=t}^{t+n-1} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+n})$



Monte Carlo:

using $\sum_t^T r$

Bootstrapped:

using $r + V$

How about

$\sum_t^{t+n-1} r + V_{t+n}$

-> less variance than MC

-> lower bias than 1-step bootstrap

$n > 1$, $n < T$ *often works the best in practice!*

Aside: discount factors

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t+1})$$

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}^{\pi}_{\phi}(\mathbf{s}_i) - y_i \right\|^2$$

what if T (episode length) is ∞?

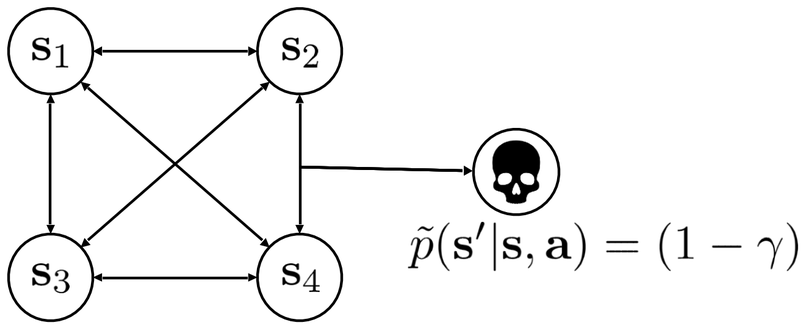
$\hat{V}^{\pi}_{\phi}$ can get infinitely large in many cases

simple trick: better to get rewards sooner than later

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t+1})$$

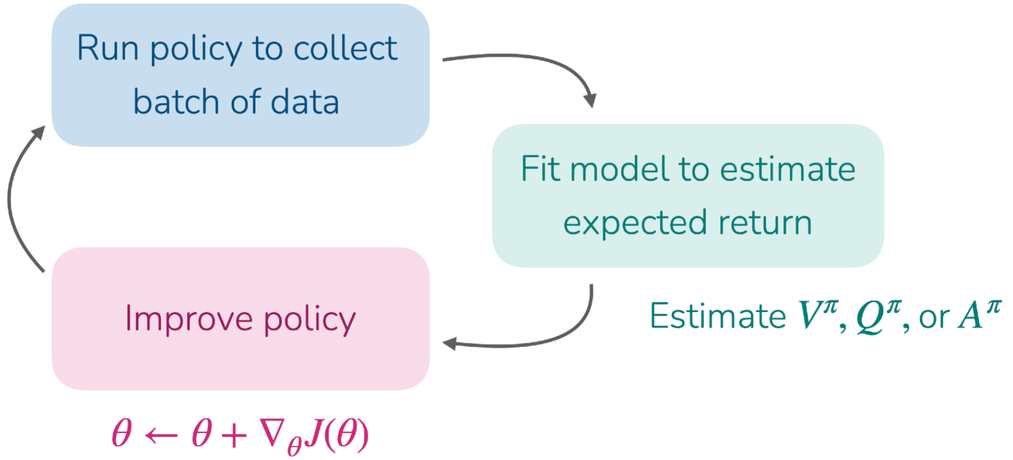
discount factor $\gamma \in [0, 1]$ (0.99 works well)

γ changes the MDP:



Slide adapted from Sergey Levine

A Full Algorithm Walkthrough



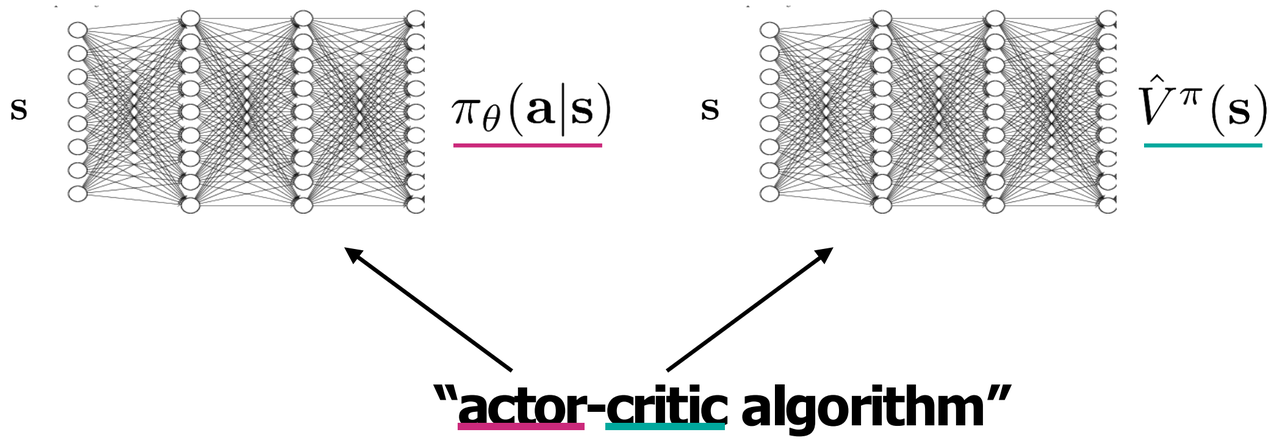
1. Sample batch of data $\{(\mathbf{s}_{1,i}, \mathbf{a}_{1,i}, \ldots, \mathbf{s}_{T,i}, \mathbf{a}_{T,i})\}$ from π_θ

2. Fit $\hat{V}_\phi^{\pi_\theta}$ to summed rewards in data

3 Evaluate $\hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) = r(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) + \gamma \hat{V}_\phi^{\pi_\theta}(\mathbf{s}_{t+1,i}) - \hat{V}_\phi^{\pi_\theta}(\mathbf{s}_{t,i}) \forall t, i$

4. Evaluate $\nabla_\theta J(\theta) \approx \sum_{t,i} \nabla_\theta \log \pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$

5. Update $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



$$y_{i,t} = \sum_{t'=t}^{t+n-1} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) + \gamma^n \hat{V}_\phi^\pi(\mathbf{s}_{i,t+n})$$

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

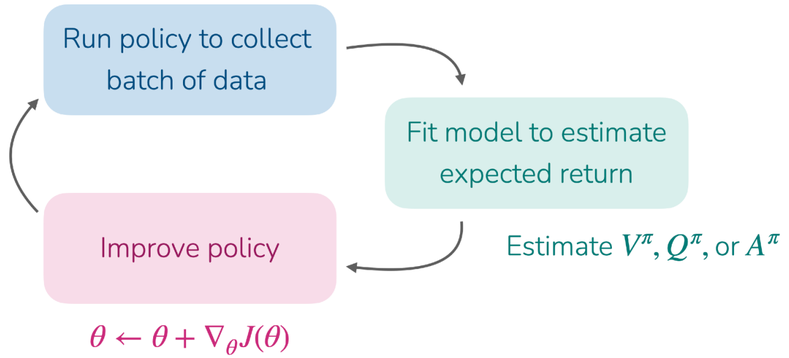
Review so far

Algorithms

Policy gradients: observe what is good vs. bad, then do more of good stuff

Actor-critic: learn to estimate what is good vs. bad, then do more of the good stuff

i.e. get a better policy gradient by using neural network to estimate value function



How to estimate value: "policy evaluation"

Supervised learning directly on observed sum of future rewards

Supervised learning on current reward + value estimate of next state

Hybrid: Supervised learning on sum of next rewards + value of state after that

What about off-policy versions?

Off-Policy Actor Critic Methods

CS 224R

Recap: Policy Gradients

Online reinforcement learning with policy gradients

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

samples from policy — policy log likelihood — reward to go — baseline

Do more of the above average stuff, less of the below average stuff.

Importance weights for *off-policy* policy gradient:

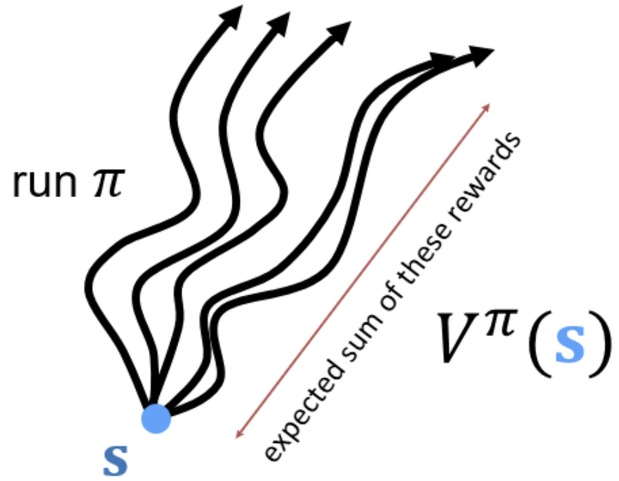
$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

Recap: Some Useful Objects

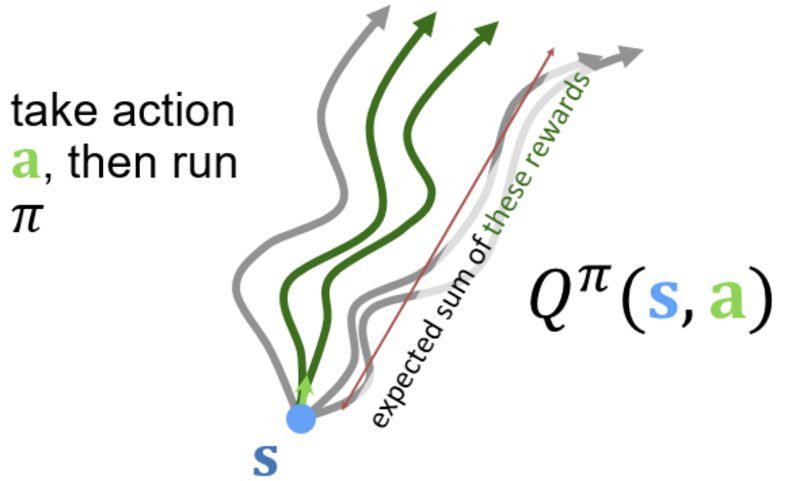
value function $V^{\pi}(\mathbf{s})$ - future expected rewards starting at $\mathbf{s}$ and following π

Q-function $Q^{\pi}(\mathbf{s}, \mathbf{a})$ - future expected rewards starting at $\mathbf{s}$, taking $\mathbf{a}$, then following π

$$V^{\pi}(\mathbf{s}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t\right]$$



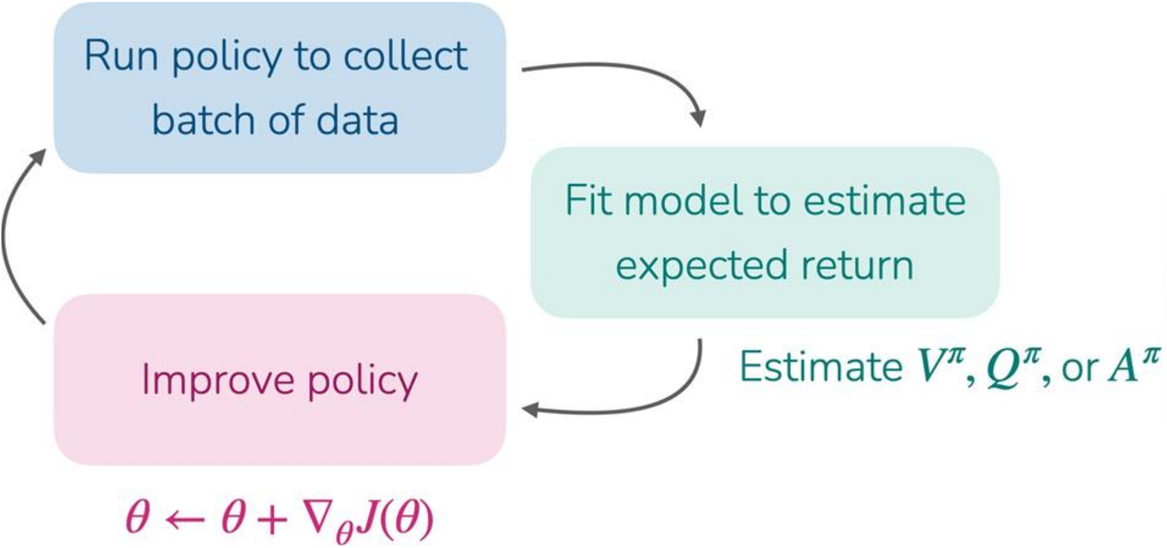
$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]$$



advantage $A^{\pi}(\mathbf{s}, \mathbf{a})$ - how much better it is to take $\mathbf{a}$ than to follow policy π at state $\mathbf{s}$

$$A^{\pi}(\mathbf{s}, \mathbf{a}) = Q^{\pi}(\mathbf{s}, \mathbf{a}) - V^{\pi}(\mathbf{s})$$

Recap: Actor-Critic Methods



Online reinforcement learning with actor-critic

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

Estimate what is good and bad, then do more of the good stuff.

To estimate value, fit $\hat{V}^\pi$:

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

where $y_{i,t} = \sum_{t'=t}^{t+n-1} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) + \gamma^n \hat{V}_\phi^\pi(\mathbf{s}_{i,t+n})$

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)$$

The plan for today

Off-policy actor critic methods

1. Taking multiple gradient steps
    a. Importance weights
    b. Constraining step size with KL penalty or clipping
    c. Practical PPO algorithm
b. Even more off-policy algorithm
    a. Fitting Q-value functions with data from other policies
    b. Practical SAC algorithm

Key learning goals:

- All of the key concepts for practical algorithms like PPO and SAC

Off-Policy Actor-Critic Methods

Version 1: Multiple Gradient Steps

1. Sample batch of data $\{(\mathbf{s}_{1,i}, \mathbf{a}_{1,i}, \ldots, \mathbf{s}_{T,i}, \mathbf{a}_{T,i})\}$ from π_θ
2. Fit $\hat{V}_\phi^{\pi_\theta}$ to summed rewards in data
3. Evaluate $\hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) = r(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) + \gamma \hat{V}_\phi^{\pi_\theta}(\mathbf{s}_{t+1,i}) - \hat{V}_\phi^{\pi_\theta}(\mathbf{s}_{t,i}) \forall t, i$
4. Evaluate $\nabla_\theta J(\theta) \approx \sum_{t,i} \nabla_\theta \log \pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$ <- use importance weights here
5. Update $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Off-Policy Actor-Critic Methods

Version 1: Multiple Gradient Steps

1. Sample batch of data $\{(\mathbf{s}_{1,i}, \mathbf{a}_{1,i}, \ldots, \mathbf{s}_{T,i}, \mathbf{a}_{T,i})\}$ from π_θ
2. Fit $\hat{V}_\phi^{\pi_\theta}$ to summed rewards in data
3. Evaluate $\hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) = r(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) + \gamma \hat{V}_\phi^{\pi_\theta}(\mathbf{s}_{t+1,i}) - \hat{V}_\phi^{\pi_\theta}(\mathbf{s}_{t,i}) \forall t, i$
4. Evaluate $\nabla_{\theta'} J(\theta') \approx \sum_{t,i} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$ <- use importance weights here
5. Update $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

What can go wrong?

Off-Policy Actor-Critic Methods

Version 1: Multiple Gradient

1. Sample batch of data $\{(\mathbf{s}_{1,i}, \mathbf{a}_{1,i}, \ldots, \mathbf{s}_{T,i}, \mathbf{a}_{T,i})\}$ from π_θ
2. Fit $\hat{V}_\phi^{\pi_\theta}$ to summed rewards in data
3. Evaluate $A^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) = r(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) + \gamma \hat{V}_\phi^{\pi_\theta}(\mathbf{s}_{t+1,i}) - \hat{V}_\phi^{\pi_\theta}(\mathbf{s}_{t,i}) \forall t, i$
4. Evaluate $\nabla_{\theta'} J(\theta') \approx \sum_{t,i} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$ <- use importance weights here
5. Update $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Let's look at surrogate objective:

$$\tilde{J}(\theta') \approx \sum_{t,i} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$

Advantages based on old policy.

Policy will increase probability on actions with high advantages.

a convenient identity

$$p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) = \nabla_\theta p_\theta(\tau)$$

What can go wrong if you take a lot of gradient steps?

Policy is incentivized to differ significantly from old policy, can easily overfit.

Off-Policy Actor-Critic Methods

Version 1: Multiple Gradient Steps

Let's look at surrogate objective:

$$\tilde{J}(\theta') \approx \sum_{t,i} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$

Advantages based on old policy.

Policy will increase probability on actions with high advantages.

a convenient identity

$$p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) = \nabla_\theta p_\theta(\tau)$$

What can go wrong if you take a lot of gradient steps?

Policy is incentivized to differ significantly from old policy, can easily overfit.

💡 Idea 1: Use KL constraint on policy.

Very common. Will see in LLM preference optimization

$$\mathbb{E}_{\mathbf{s}\sim\pi_\theta}\left[D_{KL}(\pi_{\theta'}(\cdot \mid \mathbf{s}) \| \pi_\theta(\cdot \mid \mathbf{s}))\right] \leq \delta$$

💡 Idea 2: Can we bound the importance weights?

Doesn't directly constrain policy, but removes incentives

-> A key idea behind proximal policy optimization (PPO)

Proximal Policy Optimization (PPO)

Off-policy actor-critic with a few tricks.

Surrogate objective:

$$\tilde{J}(\theta') \approx \sum_{t,i} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$

Trick #1: Clip the importance weights:

$$\tilde{J}(\theta') \approx \sum_{t,i} \text{clip}\left(\frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}, 1-\epsilon, 1+\epsilon\right) \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$

Policy no longer incentivized to deviate significantly

Trick #2: Take minimum w.r.t. original objective: (in rare event where clipping makes objective better)

$$\tilde{J}(\theta') \approx \sum_{t,i} \min\left(\frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}), \text{clip}\left(\frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}, 1-\epsilon, 1+\epsilon\right) \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})\right)$$

This is the final PPO surrogate objective!

How do we estimate the advantage?

Proximal Policy Optimization (PPO)

Off-policy actor-critic with a few tricks.

Surrogate objective:

$$\tilde{J}(\theta') \approx \sum_{t,i} \min\left(\frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}\hat{A}^{\pi_\theta}(\mathbf{s}_{t,i},\mathbf{a}_{t,i}), \text{clip}\left(\frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}, 1-\epsilon, 1+\epsilon\right)\hat{A}^{\pi_\theta}(\mathbf{s}_{t,i},\mathbf{a}_{t,i})\right)$$

Trick #3: "Generalized advantage estimation"

(GAE) Fit π with Monte Carlo or bootstrapping.

Then, use varying horizon to estimate advantage:

$$\hat{A}^{\pi}_{n}(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^{t+n}\gamma^{t'-t}r(\mathbf{s}_{t'},\mathbf{a}_{t'}) - \hat{V}^{\pi}_{\phi}(\mathbf{s}_t) + \gamma^{n}\hat{V}^{\pi}_{\phi}(\mathbf{s}_{t+n})$$

$$\hat{A}^{\pi}_{\text{GAE}}(\mathbf{s}_t,\mathbf{a}_t) = \sum_{n=1}^{\infty} w_n \hat{A}^{\pi}_{n}(\mathbf{s}_t,\mathbf{a}_t)$$

How to choose weights? cut earlier for less variance

$w_n \propto \lambda^{n-1}$ for $n = 1,\ldots, N$, where $N \boxtimes T$

Proximal Policy Optimization (PPO)

1. Sample batch of data $\{(\mathbf{s}_{1,i}, \mathbf{a}_{1,i}, \ldots, \mathbf{s}_{T,i}, \mathbf{a}_{T,i})\}$ from π_θ
2. Fit $\hat{V}_\phi^{\pi_\theta}$ to summed rewards in data
3. Evaluate $\hat{A}^\pi_{\text{GAE}}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{\infty} w_n \left(\sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}_\phi^\pi(\mathbf{s}_t) + \gamma^n \hat{V}_\phi^\pi(\mathbf{s}_{t+n}) \right)$
4. Update policy with M gradient steps on surrogate objective:

$$\tilde{J}(\theta') \approx \sum_{t,i} \min \left(\frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}), \text{clip} \left(\frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}, 1-\epsilon, 1+\epsilon \right) \hat{A}^{\pi_\theta}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) \right)$$

Some example hyperparameters:

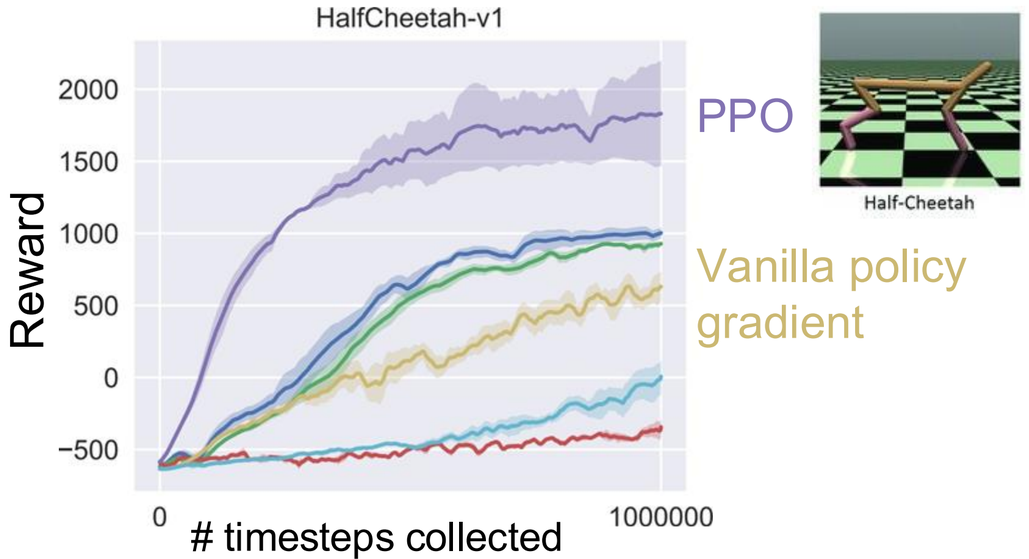
~2000 timesteps in batch of data

~10 epochs when updating policy

(=~300 gradient steps with batch size 64)

clipping range =0.2

~500 iterations —> 1M total timesteps of experience



Off-Policy Actor-Critic Methods

So far:
- use one batch of policy data for one gradient step (fully on-policy)
- use one batch of policy data for multiple gradient steps (starting to be off-policy)

Can we be even more off-policy?

Can we reuse data from previous batches, i.e. all of the past trial-and-error data?

Two key ideas:
- maintain a buffer of all past data *“replay buffer”*
- adjust equations to remove on-policy assumptions

Off-Policy Actor-Critic Methods

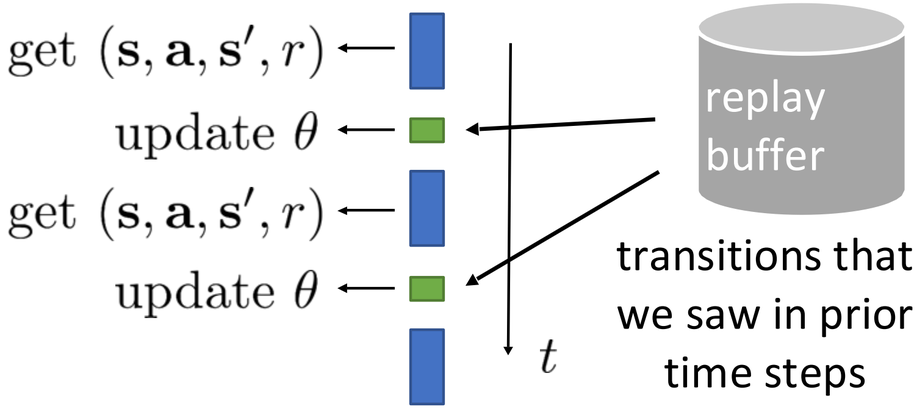
Version 2: Replay buffers

actor-critic algorithm:

1. collect experience $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$] Add this to replay buffer
2. fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Do this on minibatch sampled from all previous data

off-policy actor-critic



Slide adapted from Sergey Levine

Off-Policy Actor-Critic Methods

Version 2: Replay buffers

1. collect experience $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (& add to replay buffer)
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{V}^\pi_\phi$ using targets $y_i = r_i + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i)$ for each $\mathbf{s}_i$
4. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
5. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
6. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

not the right target value

not the action π_θ would have taken!

$$\mathcal{L}(\phi) = \frac{1}{N} \sum_i \left\| \hat{V}^\pi_\phi(\mathbf{s}_i) - y_i \right\|^2$$

mini batch size

Question: When we fit V^π on all data from replay buffer, what policy π is it learning a value function for?

The algorithm is currently broken 😢

Slide adapted from Sergey Levine

Off-Policy Actor-Critic Methods

How to we fit a value function for π_θ using replay buffer of data from past policies?

💡 What if we fit $Q(\mathbf{s}, \mathbf{a})$ instead of $V(\mathbf{s})$?

depends on past policy's actions

The datapoints we have: $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ + future reward

-> Don't want to use this.

action from past policy

-> If we fit $Q(\mathbf{s}, \mathbf{a})$, we pass the action as input
(okay if is a bit different from what policy would have done)

Recall: *Definition of Q-values*

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]$$

"total reward we get if we take $\mathbf{a}_t$ in $\mathbf{s}_t$...
... and then follow the policy π"

For *any* $(\mathbf{s}, \mathbf{a})$:

$$Q^{\pi_\theta}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\cdot|\mathbf{s},\mathbf{a}), \bar{\mathbf{a}}' \sim \pi_\theta(\cdot|\mathbf{s}')}\left[Q^{\pi_\theta}(\mathbf{s}', \bar{\mathbf{a}}')\right]$$

Off-Policy Actor-Critic Methods

How to we fit a value function for π_θ using replay buffer of data from past policies?

💡What if we fit $Q(\mathbf{s}, \mathbf{a})$ instead of $V(\mathbf{s})$?

The datapoints we have: $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$+ future reward

For *any* $(\mathbf{s}, \mathbf{a})$: $Q^{\pi_\theta}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\cdot|\mathbf{s},\mathbf{a}), \bar{\mathbf{a}}' \sim \pi_\theta(\cdot|\mathbf{s}')} \left[Q^{\pi_\theta}(\mathbf{s}', \bar{\mathbf{a}}') \right]$

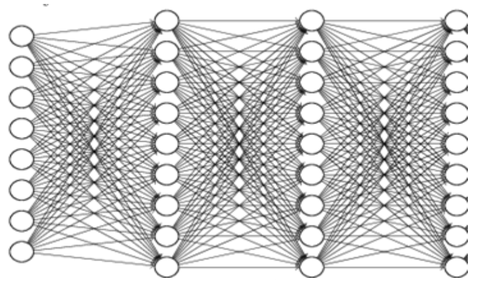
Approach:

1. Sample $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)$ from the buffer
2. Sample $\bar{\mathbf{a}}'_i \sim \pi_\theta(\cdot|\mathbf{s}'_i)$ from *current* policy

$$Q^{\pi_\theta}(\mathbf{s}_i, \mathbf{a}_i) \approx \underbrace{r(\mathbf{s}_i, \mathbf{a}_i) + \gamma Q^{\pi_\theta}(\mathbf{s}'_i, \bar{\mathbf{a}}'_i)}_{y_i}$$

Note: To be accurate, targets y_i need sufficient action coverage

s, **a**



$\hat{Q}^\pi_\phi(\mathbf{s}, \mathbf{a})$
parameters ϕ

$$\mathcal{L}(\phi) = \frac{1}{N} \sum_i \left\| \hat{Q}^\pi_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i \right\|^2$$

Off-Policy Actor-Critic Methods

Version 2: Fixing the value function

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{V}^\pi_\phi$ using targets $y_i = r_i + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i)$ for each $\mathbf{s}_i$
4. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
5. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
6. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

not the right target value

3. update $\hat{Q}^\pi_\phi$ using targets $y_i = r_i + \gamma \hat{Q}^\pi_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$

not from replay buffer $\mathcal{R}$!

$\mathbf{a}'_i \sim \pi_\theta(\mathbf{a}'_i|\mathbf{s}'_i)$

Slide adapted from Sergey Levine

Off-Policy Actor-Critic Methods

Version 2: Fixing the value function

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}^\pi_\phi$ using targets $y_i = r_i + \gamma \hat{Q}^\pi_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
4. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
5. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
6. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

First: convenient to use Q^π instead of A^π. (i.e. no average reward baseline)
higher variance, but okay b/c we're now using a lot more data (all data in buffer)

Second: current policy's actions likely better than past policy's actions

sample $\mathbf{a}^\pi_i \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$ $\quad$ $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^\pi_i|\mathbf{s}_i) \hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}^\pi_i)$

not from replay buffer $\mathcal{R}$!

Slide adapted from Sergey Levine

Off-Policy Actor-Critic Methods

Version 2: Anything else?

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}^\pi_\phi$ using targets $y_i = r_i + \gamma \hat{Q}^\pi_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
4. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^\pi_i|\mathbf{s}_i) \hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}^\pi_i)$ where $\mathbf{a}^\pi_i \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Any remaining problems?

$\mathbf{s}_i$ didn't come from $p_\theta(\mathbf{s})$

nothing we can do here, just accept it

intuition: we want optimal policy on $p_\theta(\mathbf{s})$
but we get optimal policy on a *broader* distribution

Slide adapted from Sergey Levine

Off-Policy Actor-Critic Methods

Version 2: Some implementation details

online actor-critic algorithm:

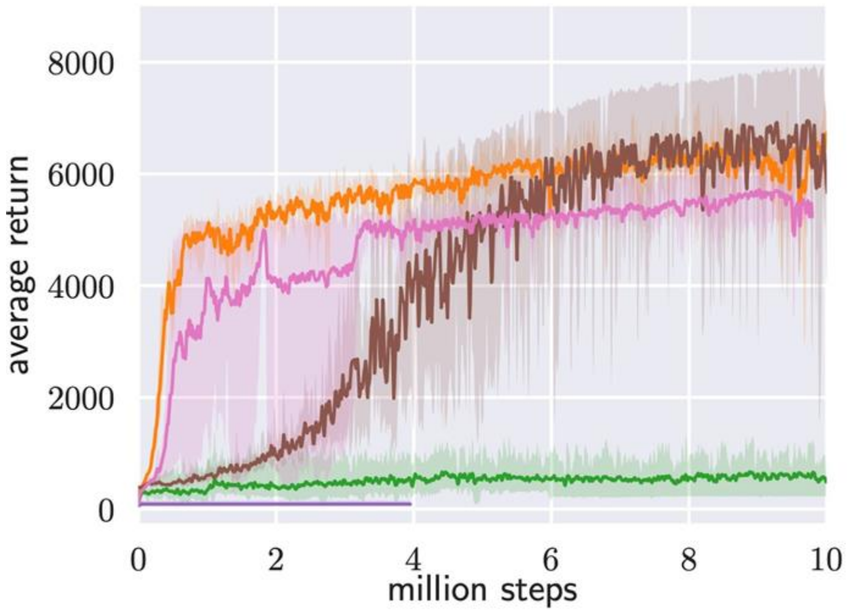
1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}^\pi_\phi$ using targets $y_i = r_i + \gamma \hat{Q}^\pi_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$ for each $\mathbf{s}_i, \mathbf{a}_i$
4. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^\pi_i|\mathbf{s}_i) \hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}^\pi_i)$ where $\mathbf{a}^\pi_i \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

also fancier ways to fit Q-functions (more on this in next two lectures)

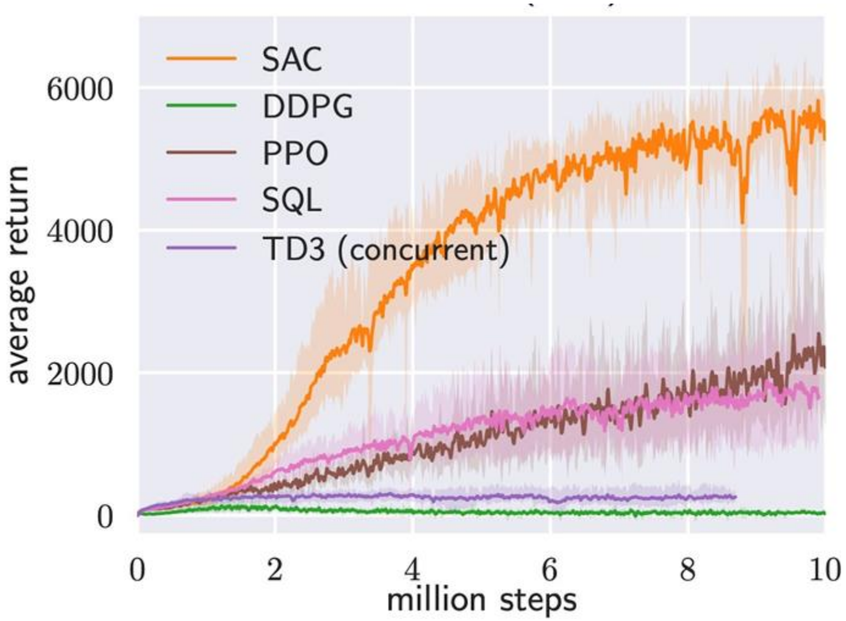
can also use **reparameterization trick** to better estimate the gradient (for Gaussian policy)

Example practical algorithm:
Haarnoja, Zhou, Abbeel, Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. 2018.

More off-policy vs. less off-policy actor critic?



(e) Humanoid-v1



(f) Humanoid (rllab)

<- more off-policy
(i.e. with replay buffer)

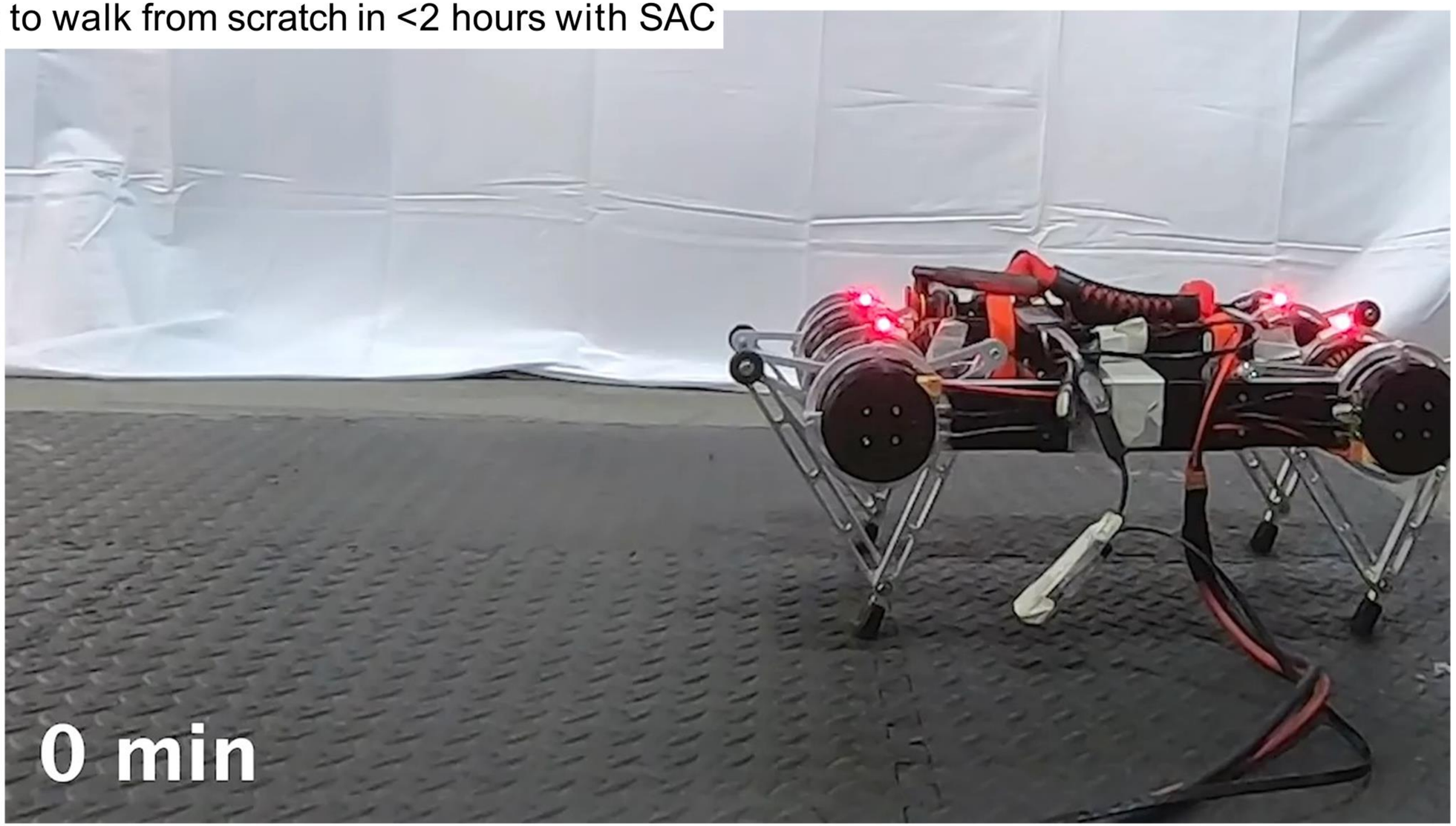
<- less off-policy
(i.e. no replay buffer)

+ Off-policy with replay buffer (e.g. soft actor-critic) can be far more data efficient

- They can also generally be a lot harder to tune hyperparameters, less stable (than e.g. PPO)

Fully Off-Policy = Can be efficient enough for real-world RL

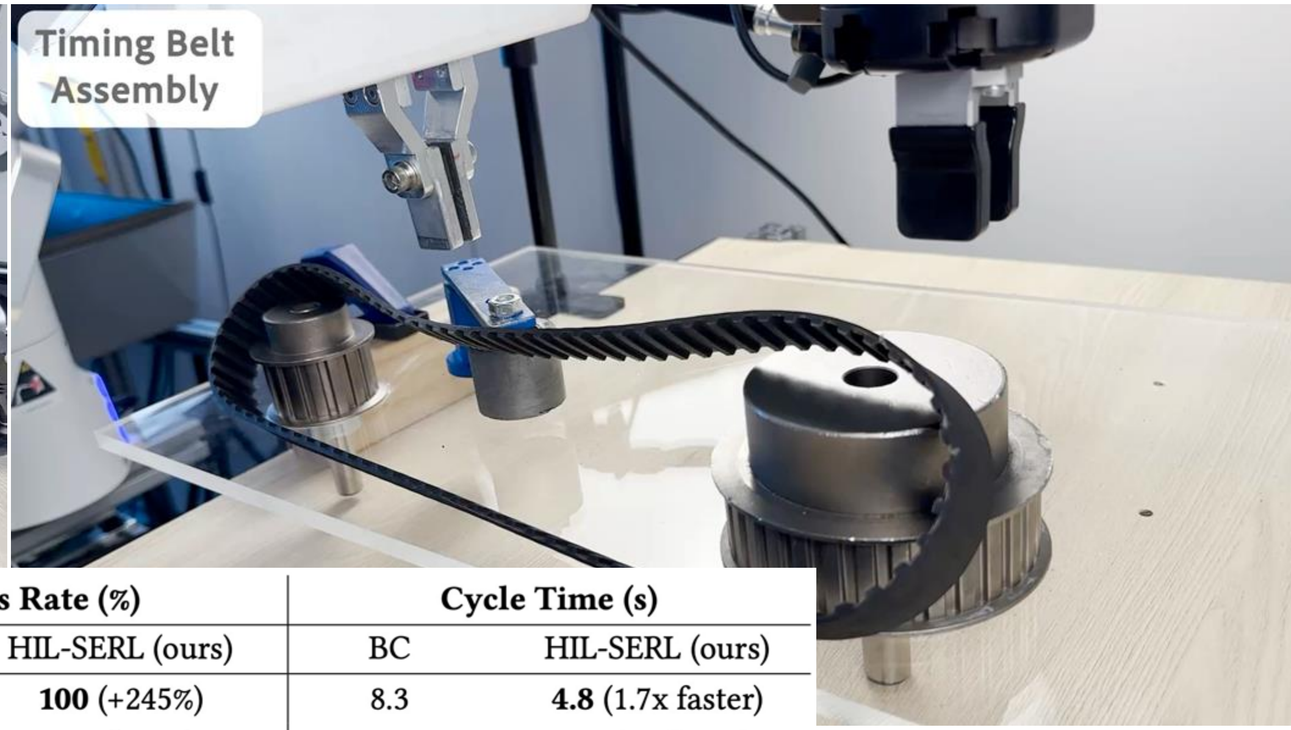
Learning to walk from scratch in <2 hours with SAC



Haarnoja, Zhou, Hartikainen, Tucker, Ha, Tan, Kumar, Zhu, Gupta, Abbeel, Levine. Soft Actor-Critic Algorithms and Applications. 2018.

Fully Off-Policy = Can be efficient enough for real-world RL

Learning precise assembly with algorithm based on SAC (seeded with demonstrations)



Task	Training Time (h)	Success Rate (%)		Cycle Time (s)	
		BC	HIL-SERL (ours)	BC	HIL-SERL (ours)
RAM Insertion	1.5	29	**100** (+245%)	8.3	**4.8** (1.7x faster)
SSD Assembly	1	79	**100** (+27%)	6.7	**3.3** (2x faster)
IKEA - Side Panel 1	2	77	**100** (+30%)	6.5	**2.7** (2.4x faster)
IKEA - Side Panel 2	1.75	79	**100** (+27%)	5.0	**2.4** (2.1x faster)
IKEA - Top Panel	1	35	**100** (+186%)	8.9	**2.4** (3.7x faster)
IKEA - Whole Assembly	–	1/10	**10/10** (+900%)	–	–
Car Dashboard Assembly	2	41	**100** (+144%)	20.3	**8.8** (2.3x faster)
Timing Belt Assembly	6	2	**100** (+4900%)	9.1	**7.2** (1.3x faster)

<- better, faster than imitation learning

Luo, Xu, Wu, Levine. HIL-SERL: Precise and Dexterous Robotic Manipulation via Human-in-the-Loop Reinforcement Learning. 2024.

PPO = Common algorithm of choice for stable, less efficient learning

Learning to walk in simulation with PPO -> transfer to real world (with careful simulation choices)

https://www.youtube.com/watch?v=xf_UXK0OTIk

Tan, Zhang, Coumans, Iscen, Bai, Hafner, Bohez, Vanhoucke. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. 2018.

PPO = Common algorithm of choice for stable, less efficient learning

Learning to solve Rubik's cube in simulation -> transfer to real hand

(though, transfer from simulation generally much harder, less successful for robot manipulation)

https://www.youtube.com/watch?v=jm-ihc7CASY

OpenAI. Solving Rubik's Cube with a Robot Hand. 2019.

What about RL for language models?

PPO is a common choice!

Will start to discuss them when talking about reward modeling next week.

Q-Learning

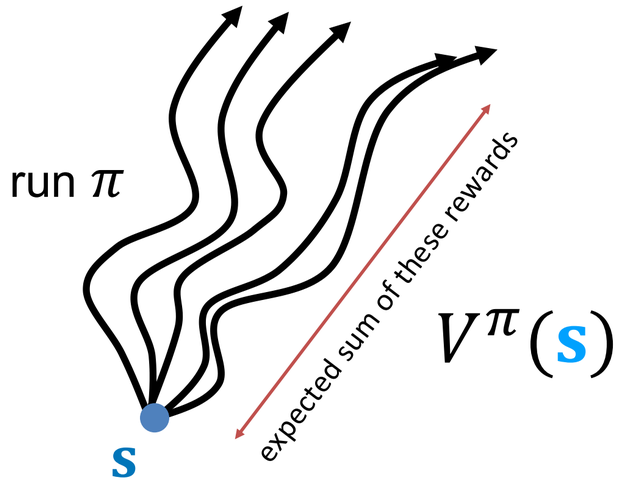
CS 224R

Recap: Some Useful Objects

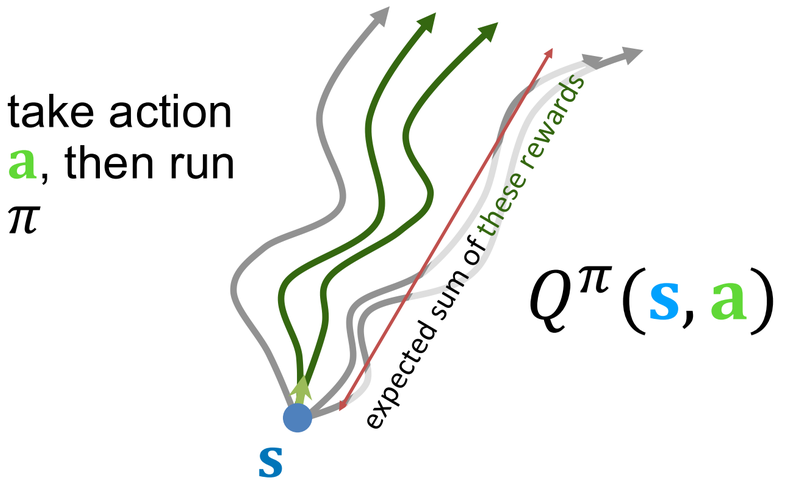
value function $V^{\pi}(\mathbf{s})$ - future expected rewards starting at $\mathbf{s}$ and following π

Q-*function* $Q^{\pi}(\mathbf{s}, \mathbf{a})$ - future expected rewards starting at $\mathbf{s}$, taking $\mathbf{a}$, then following π

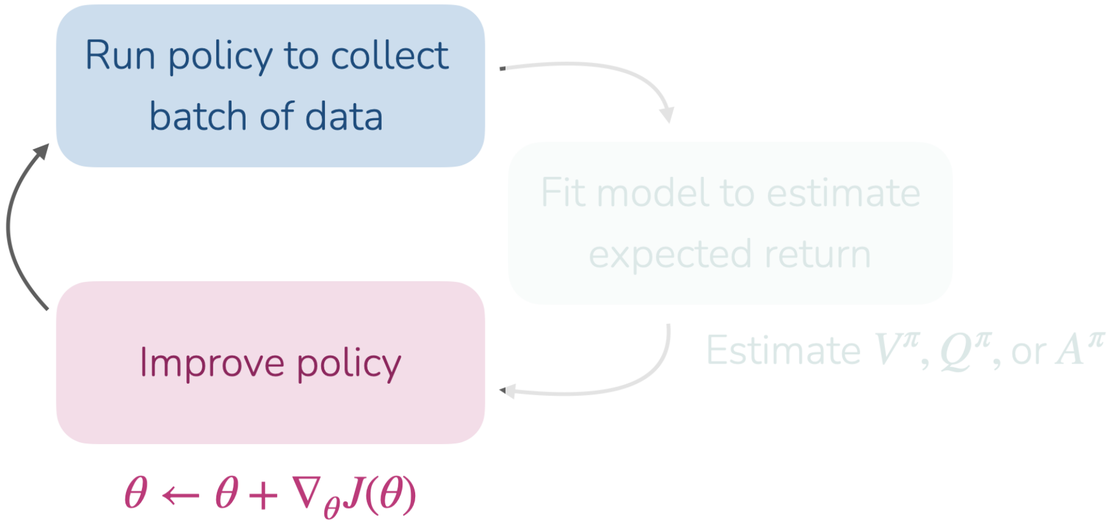
$$V^{\pi}(\mathbf{s}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t\right]$$



$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t\right]$$



Recap: Methods



Online RL with policy gradients

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

samples from policy — policy log likelihood — reward to go — baseline

Do more of the above average stuff, less of the below average stuff.

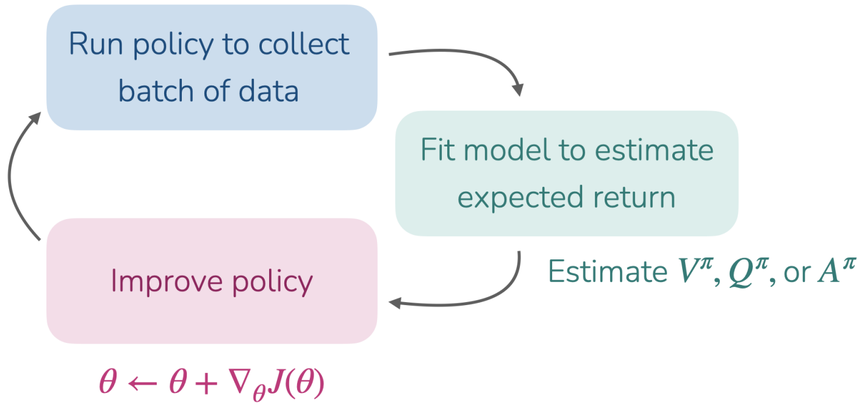
Online RL with actor-critic

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

Estimate what is good and bad, then do more of the good stuff.

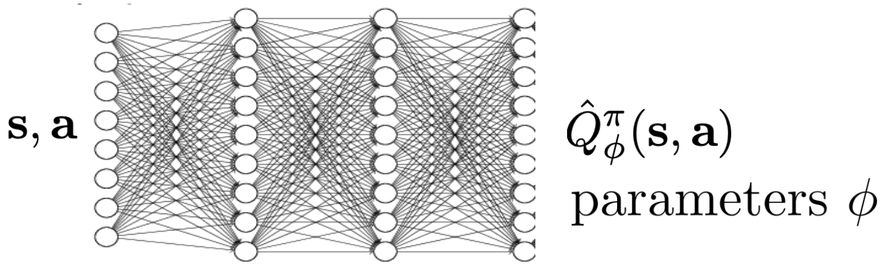
Recap: Off-Policy Policy Evaluation

Online RL with actor-critic



Estimate what is good and bad, then do more of the good stuff.

For off-policy version: can we estimate Q-function using past policy data?



For *any* $(\mathbf{s}, \mathbf{a})$:

$$Q^{\pi_\theta}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\cdot|\mathbf{s},\mathbf{a}), \bar{\mathbf{a}}' \sim \pi_\theta(\cdot|\mathbf{s}')} \left[Q^{\pi_\theta}(\mathbf{s}', \bar{\mathbf{a}}') \right]$$

1. Sample $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)$ from the buffer
2. Sample $\bar{\mathbf{a}}'_i \sim \pi_\theta(\cdot|\mathbf{s}'_i)$ from *current* policy

$$Q^{\pi_\theta}(\mathbf{s}_i, \mathbf{a}_i) \approx \underbrace{r(\mathbf{s}_i, \mathbf{a}_i) + \gamma Q^{\pi_\theta}(\mathbf{s}'_i, \bar{\mathbf{a}}'_i)}_{y_i}$$

$$\mathcal{L}(\phi) = \frac{1}{N} \sum_i \left\| \hat{Q}^\pi_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i \right\|^2$$

Note: To be accurate, targets y_i need sufficient action coverage

The plan for today

Value-based RL methods

1. Q-learning RL method
    a. Policy iteration
    b. Bellman optimality equation
    c. How to collect data for Q-learning methods
2. Q-learning in practice
    d. Target networks
    e. Double DQN
    f. N-step returns

Key learning goals:

- How Q-functions relate to policies
- How to do RL without learning an explicit policy
- How to stabilize Q-learning in practice

A thought exercise

Recall: *Definition of Q-values*

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_\pi\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]$$

"total reward we get if we take $\mathbf{a}_t$ in $\mathbf{s}_t$...
... and then follow the policy π"

For some policy π, say you have an accurate estimate $\hat{Q}^\pi(\mathbf{s}, \mathbf{a})$ for all $\mathbf{s}, \mathbf{a}$

Define a new policy $\pi_{\text{new}}(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}} \hat{Q}^\pi_\phi(\mathbf{s}_t, \mathbf{a}) \\ 0 & \text{otherwise} \end{cases}$

2D navigation example

Initial state distribution $p(\mathbf{s}_1)$

Reward is 1 at ★, 0 elsewhere

Current policy π goes right in all states

takes 1 timestep, 2 timesteps to travel these distances

Questions: Is the new policy better, worse, or the same as the original policy?

Is it the optimal policy? Why or why not?

Can we omit policy gradient completely?

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$: expected reward from taking $\mathbf{a}_t$ and subsequently following π

$\arg\max_{\mathbf{a}_t} Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$: best action from $\mathbf{s}_t$, if we then follow π afterwards

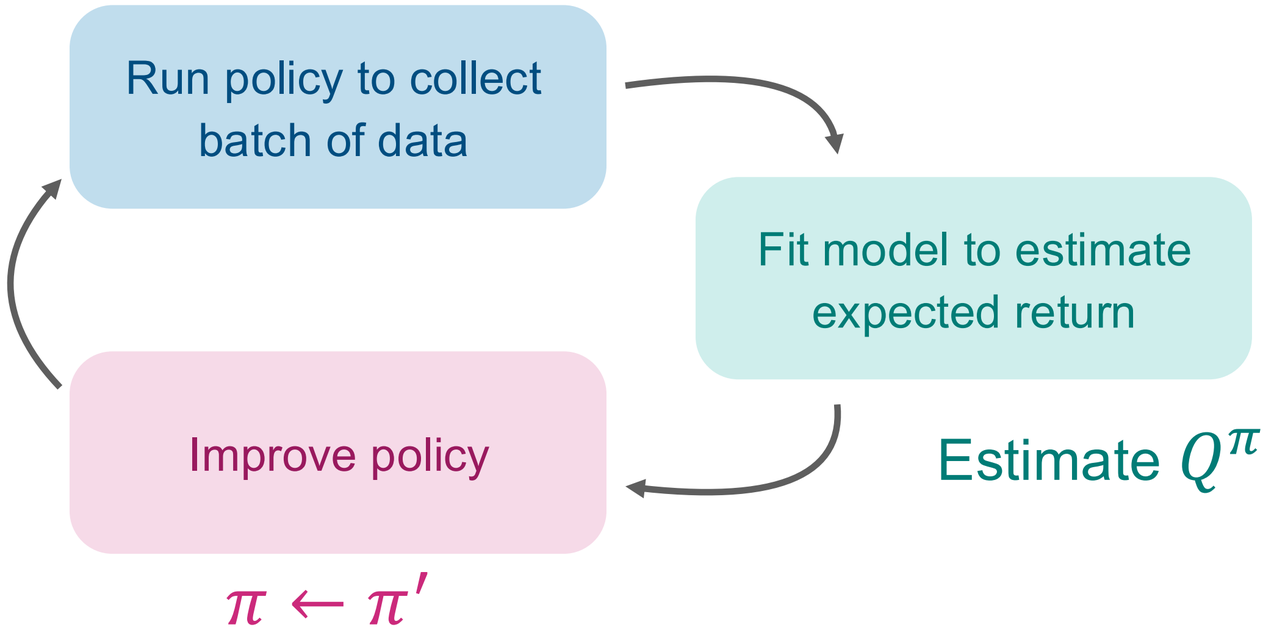
forget policies, let's just do this!

at *least* as good as any $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$

regardless of what $\pi(\mathbf{a}_t|\mathbf{s}_t)$ is!

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}} Q^\pi(\mathbf{s}_t, \mathbf{a}) \\ 0 \text{ otherwise} \end{cases}$$

as good as π
(probably better)



Slide adapted from Sergey Levine

From actor-critic to critic only

~~Off-policy actor critic with replay buffer~~ Q-learning

1. take action $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}^\pi_\phi$ using targets $y_i = r_i + \gamma \hat{Q}^\pi_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$ where $\mathbf{a}'_i \sim \pi(\cdot|\mathbf{s}'_i)$
4. ~~$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^\pi_i|\mathbf{s}_i) \hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}^\pi_i)$ where $\mathbf{a}^\pi_i \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$~~
5. ~~$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$~~
4. define new policy $\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}} \hat{Q}_\phi(\mathbf{s}_t, \mathbf{a}) \\ 0 & \text{otherwise} \end{cases}$

From actor-critic to critic only

~~Off-policy actor critic with replay buffer~~

1. take action $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}^\pi_\phi$ using targets $y_i = r_i + \gamma \hat{Q}^\pi_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$ where $\mathbf{a}'_i \sim \pi(\cdot|\mathbf{s}'_i)$ <- policy *evaluation*
4. ~~$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^\pi_i|\mathbf{s}_i) \hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}^\pi_i)$ where $\mathbf{a}^\pi_i \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$~~ (**Note**: can do multiple gradient steps here)
5. ~~$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$~~
4. define new policy $\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}} \hat{Q}_\phi(\mathbf{s}_t, \mathbf{a}) \\ 0 & \text{otherwise} \end{cases}$ <- policy *improvement*

"Policy iteration"

Can we improve the policy in the Q-function update?

From actor-critic to critic only

~~Off-policy actor critic with replay buffer~~ Q-learning

1. take action $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}^{\pi}_{\phi}$ using targets ~~$y_i = r_i + \gamma \hat{Q}^{\pi}_{\phi}(\mathbf{s}'_i, \mathbf{a}'_i)$ where $\mathbf{a}'_i \sim \pi(\cdot|\mathbf{s}'_i)$~~ $y_i = r_i + \gamma \max_{\mathbf{a}'} \hat{Q}_{\phi}(\mathbf{s}'_i, \mathbf{a}')$
4. ~~$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^{\pi}_i|\mathbf{s}_i) \hat{Q}^{\pi}(\mathbf{s}_i, \mathbf{a}^{\pi}_i)$ where $\mathbf{a}^{\pi}_i \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$~~
5. ~~$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$~~
4. define new policy $\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}} \hat{Q}_{\phi}(\mathbf{s}_t, \mathbf{a}) \\ 0 & \text{otherwise} \end{cases}$

Q-values for new policy!

From actor-critic to critic only

Q-learning

1. take action $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}_\phi$ using targets $y_i = r_i + \gamma \max_{\mathbf{a}'} \hat{Q}_\phi(\mathbf{s}'_i, \mathbf{a}')$
4. define new policy $\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}} \hat{Q}_\phi(\mathbf{s}_t, \mathbf{a}) \\ 0 & \text{otherwise} \end{cases}$

Terminology

Bellman equation

Bellman optimality equation

Why does make sense?

Recall: $Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\cdot|\mathbf{s},\mathbf{a}), \bar{\mathbf{a}}' \sim \pi(\cdot|\mathbf{s}')} [Q^\pi(\mathbf{s}', \bar{\mathbf{a}}')]$ for *all* $(\mathbf{s}, \mathbf{a})$

-> This holds for any policy π (including the optimal policy π^*)

If π^* is the optimal policy, we also get:

$$Q^{\pi^*}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\cdot|\mathbf{s},\mathbf{a})} \left[\max_{\bar{\mathbf{a}}'} Q^{\pi^*}(\mathbf{s}', \bar{\mathbf{a}}') \right] \text{ for } \textit{all } (\mathbf{s}, \mathbf{a})$$

the action π^* would take in state $\mathbf{s}'$

When we optimize step 3, we are trying to make this equation hold!

From actor-critic to critic only

Q-learning

1. take action $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}_\phi$ using targets $y_i = r_i + \gamma \max_{\mathbf{a}'} \hat{Q}_\phi(\mathbf{s}'_i, \mathbf{a}')$
4. define new policy $\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}} \hat{Q}_\phi(\mathbf{s}_t, \mathbf{a}) \\ 0 & \text{otherwise} \end{cases}$

Note: Q-learning is off-policy

Why does make sense?

If π^* is the optimal policy, we also get:

$$Q^{\pi^*}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\cdot|\mathbf{s},\mathbf{a})} \left[\max_{\bar{\mathbf{a}}'} Q^{\pi^*}(\mathbf{s}', \bar{\mathbf{a}}') \right] \quad \text{for } \textit{all } (\mathbf{s}, \mathbf{a})$$

When we optimize step 3, we are trying to make this equation hold!

Will this algorithm converge to optimal Q^{π^*}?

Yes, if you maintain a table of Q-values for every state and action. More generally, no. 😢

Can construct scenarios where it diverges, even with linear Q. *But*, it can be made to work well

What kind of data should we collect?

Q-learning

from some exploration policy

1. take action $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
3. update $\hat{Q}_\phi$ using targets $y_i = r_i + \gamma \max_{\mathbf{a}'} \hat{Q}_\phi(\mathbf{s}'_i, \mathbf{a}')$
4. define new policy $\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}} \hat{Q}_\phi(\mathbf{s}_t, \mathbf{a}) \\ 0 & \text{otherwise} \end{cases}$

Note: Q-learning is off-policy

want coverage for many actions $\mathbf{a}$

What are good choices for data collection?

1. With probability ϵ, take uniformly random action

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 - \epsilon \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ \epsilon/(|\mathcal{A}| - 1) \text{ otherwise} \end{cases}$$

"epsilon-greedy"

Often start with larger ϵ, decrease over training.

2. Take actions with probability proportional to their Q-value.

$$\pi(\mathbf{a}_t|\mathbf{s}_t) \propto \exp(Q_\phi(\mathbf{s}_t, \mathbf{a}_t))$$

"Boltzmann exploration"

Putting it together

full Q-learning with replay buffer:

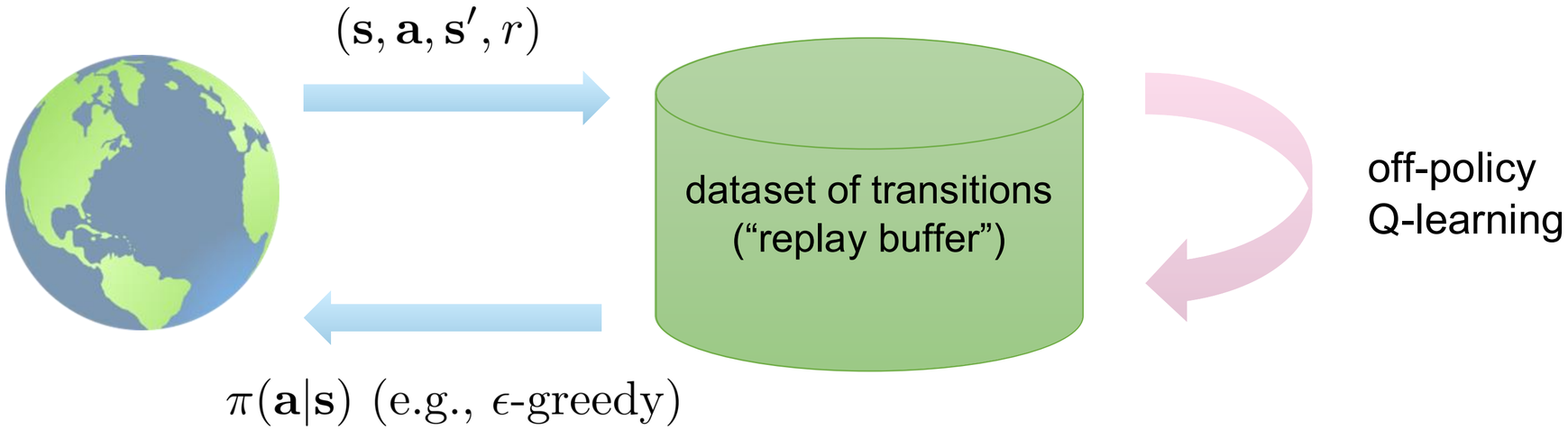
1. collect data $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{R}$

$K\times$

2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{R}$
3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}')])$

K = 1 is common, though larger K more efficient

result: Q_ϕ final policy $\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}_t, \mathbf{a}) \\ 0 & \text{otherwise} \end{cases}$



Slide adapted from Sergey Levine

The plan for today

Value-based RL methods

1. Q-learning RL method
    a. Policy iteration
    b. Bellman optimality equation
    c. How to collect data for Q-learning methods
2. **Q-learning in practice**
    d. Target networks
    e. Double DQN
    f. N-step returns

How to make Q-learning stable?

full Q-learning with replay buffer:

1. collect data $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{R}$

$K\times$

2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{R}$
3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}')])$

this is a moving target

can lead to unstable optimization

Can we change the target Q-values more slowly?

💡Simple idea: freeze parameters used for the target Q-values, update periodically

Slide adapted from Sergey Levine

How to make Q-learning stable?

Q-learning with replay buffer and target network:

0. save target network parameters: $\phi' \leftarrow \phi$

$N\times$

1. collect data $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{R}$

$K\times$

2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{R}$

3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}')])$

Inner loop is doing supervised learning

frozen target network

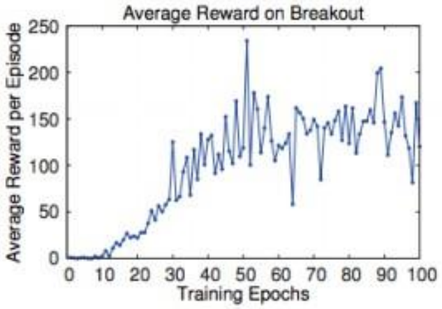
labels don't change in inner loop

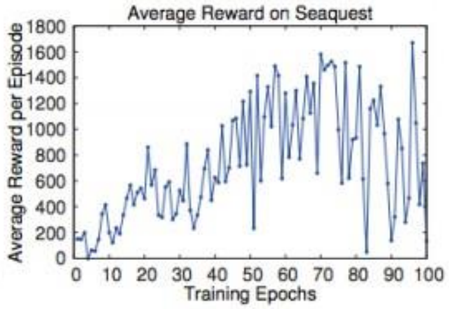
Corresponds to DQN algorithm ("deep Q network")

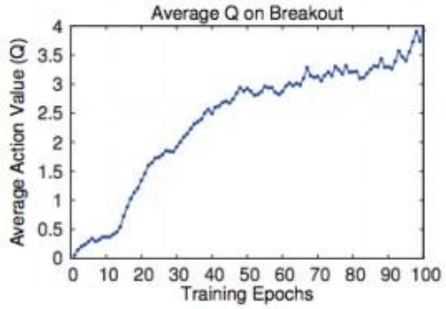
Mnih et al. Playing Atari with Deep Reinforcement Learning. 2013.

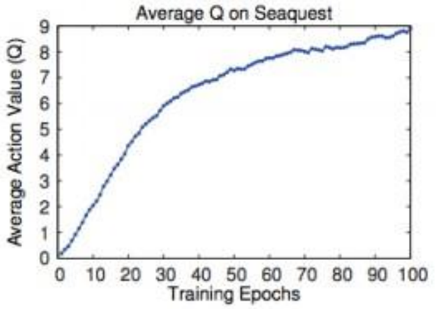
Along with an actor-critic method, you'll implement this algorithm in HW 2!

Are the Q-values accurate?

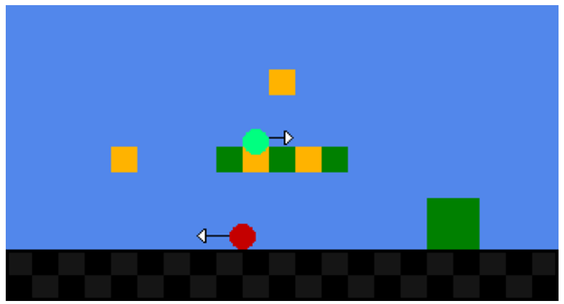






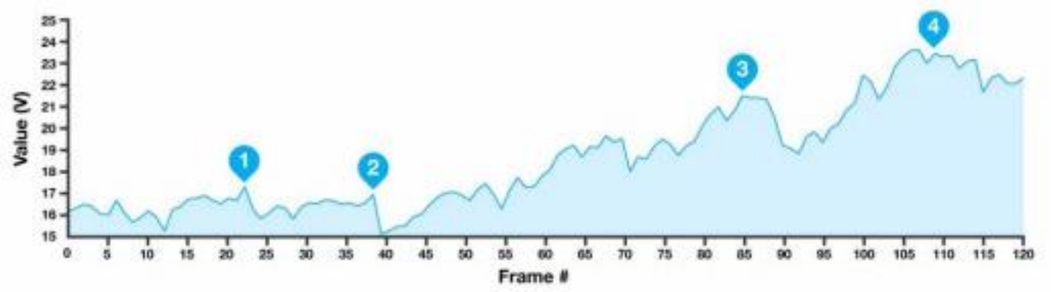


As predicted Q increases, so does the return



ATARI®

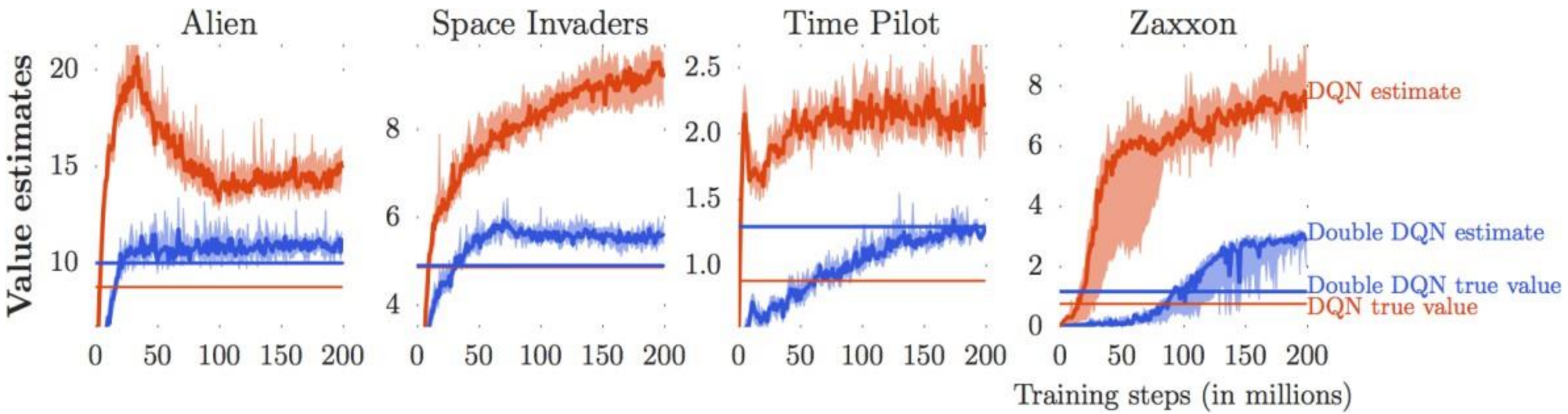
Game of Pong
https://www.youtube.com/watch?v=jiFCgAcM2tU



Atari Logo Image: Atari, Inc., Public domain, via Wikimedia Commons

Slide adapted from Sergey Levine

Are the Q-values accurate?



Slide adapted from Sergey Levine

Overestimation in Q-learning

target value $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$

this last term is the problem

$Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ is not perfect – it looks "noisy"

hence $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ *overestimates* the next value!

note that $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = \underline{Q_{\phi'}}(\mathbf{s}', \underline{\arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')})$

value *also* comes from $Q_{\phi'}$ action selected according to $Q_{\phi'}$

Slide adapted from Sergey Levine

Double Q-learning

note that $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = \underline{Q_{\phi'}}(\mathbf{s}', \underline{\arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')})$

value *also* comes from $Q_{\phi'}$ action selected according to $Q_{\phi'}$

if the noise in these is decorrelated, the problem goes away!

idea: don't use the same network to choose the action and evaluate value!

"double" Q-learning: use two networks:

$$Q_{\phi_A}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_B}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_A}(\mathbf{s}', \mathbf{a}'))$$

$$Q_{\phi_B}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_A}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_B}(\mathbf{s}', \mathbf{a}'))$$

if the two Q's are noisy in *different* ways, there is no problem

Slide adapted from Sergey Levine

Double Q-learning in practice

where to get two Q-functions?

just use the current and target networks!

standard Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))$

double Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi}(\mathbf{s}', \mathbf{a}'))$

just use current network (not target network) to evaluate action

still use target network to evaluate value!

Slide adapted from Sergey Levine

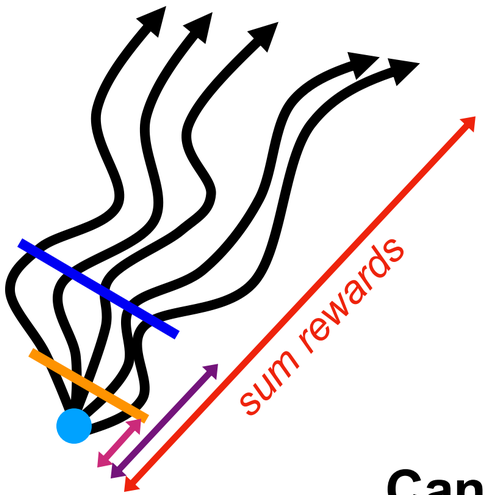
N-step returns?

Q-learning target: $y_{j,t} = r_{j,t} + \gamma \max_{\mathbf{a}_{j,t+1}} Q_{\phi'}(\mathbf{s}_{j,t+1}, \mathbf{a}_{j,t+1})$

these are the only values that matter if $Q_{\phi'}$ is bad!

these values are important if $Q_{\phi'}$ is good

Recall: fitting value functions with rewards + bootstrapped V



Monte Carlo:

using $\sum_t^T r$

How about

$\sum_t^{t+n-1} r + V_{t+n}$

-> less variance than MC

-> lower bias than 1-step bootstrap

Bootstrapped:
using $r + V$

Can we construct these multi-step targets for Q-learning?

$$y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})$$

N-step returns

Q-learning target: $y_{j,t} = r_{j,t} + \gamma \max_{\mathbf{a}_{j,t+1}} Q_{\phi'}(\mathbf{s}_{j,t+1}, \mathbf{a}_{j,t+1})$

these are the only values that matter if $Q_{\phi'}$ is bad!

these values are important if $Q_{\phi'}$ is good

Can we construct these multi-step targets for Q-learning?

N-step target: $y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})$

these are the rewards for policy that collected the data

+ (far) less biased target values when Q-values are inaccurate

+ typically faster learning, especially early on

- only actually correct when learning on-policy

(not an issue when N=1!)

Ways to fix? Most commonly: ignore the problem & still use N > 1

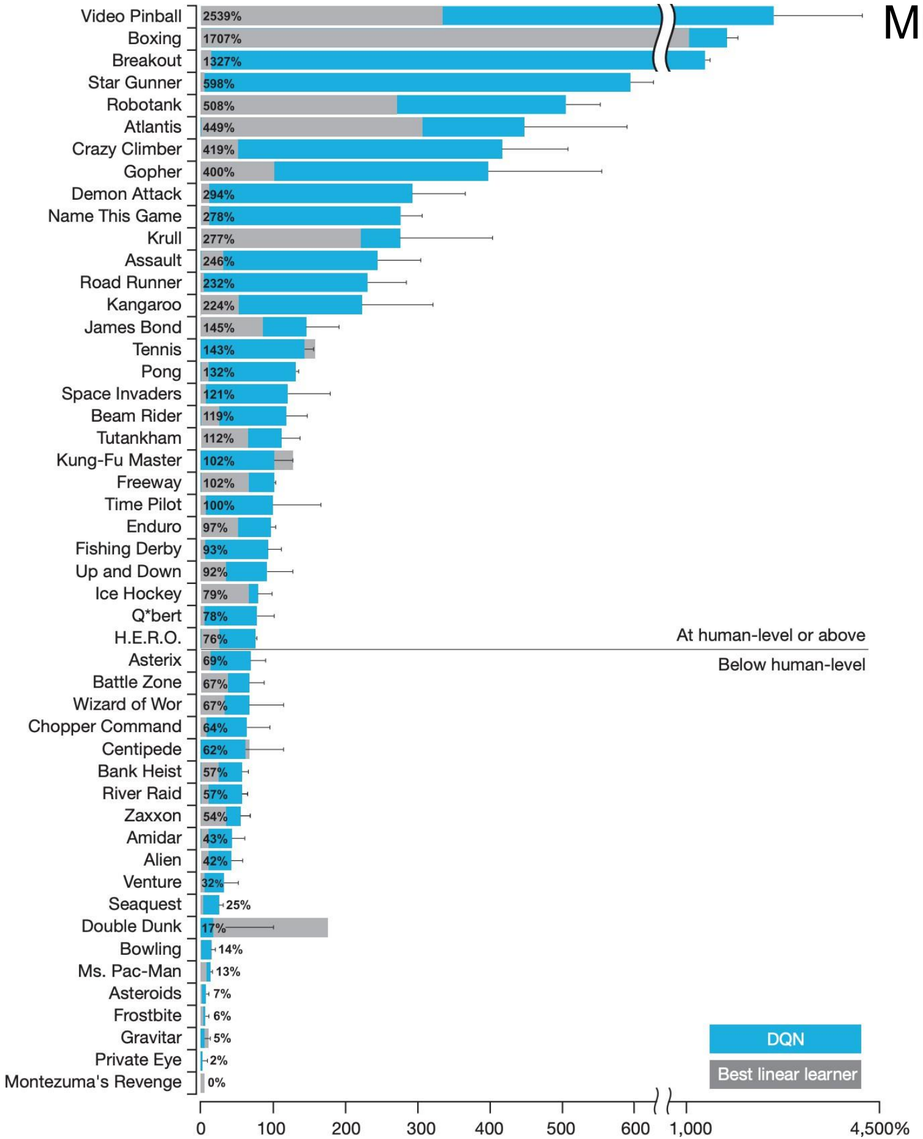
Can also:

- dynamically choose N to only use data that follows current policy (if data mostly on-policy, action space is small)
- use importance sampling

Q-learning in practice?

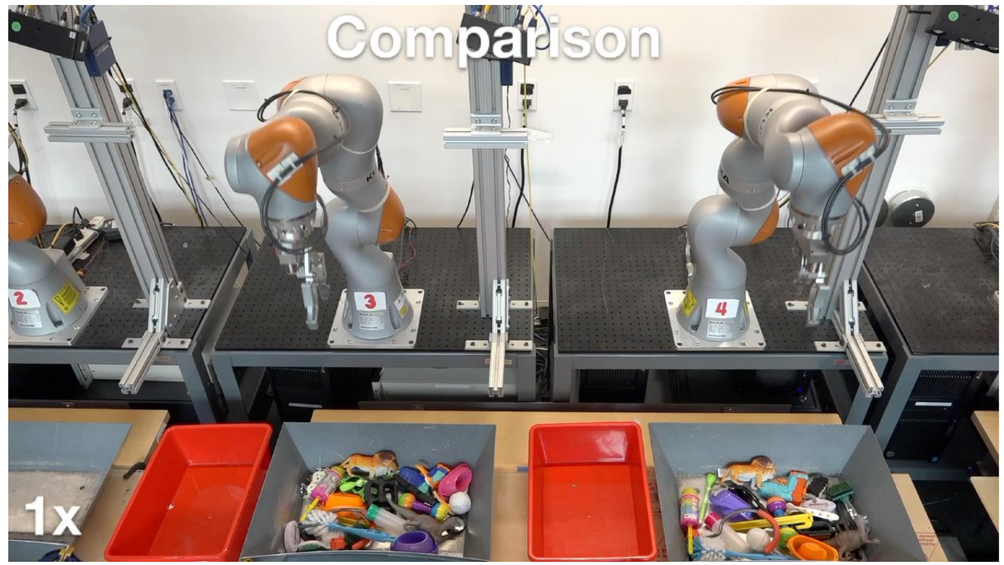
Q-learning learns to outperform people on most Atari games

Mnih et al. 2015



Q-learning trains robot grasping system

Kalashnikov et al. 2018



Method	Dataset	Test
QT-Opt (ours)	580k off-policy + 28k on-policy	**96%**
Levine et al. [27]	900k grasps from Levine et al. [27]	78%
QT-Opt (ours)	580k off-policy grasps only	87%
Levine et al. [27]	400k grasps from our dataset	67%

When to use one online RL algorithm vs. another?

Chelsea's advice

PPO & variants	When you care about stability, ease-of-use
	When you don't care about data efficiency
DQN & variants	When you have discrete actions or low-dimensional continuous actions
SAC & variants	When you care most about data efficiency
	When you are okay with tuning hyperparameters, less stability

The plan for today

Value-based RL methods

1. Q-learning RL method
    a. Policy iteration
    b. Bellman optimality equation
    c. How to collect data for Q-learning methods
2. Q-learning in practice
    d. Target networks
    e. Double DQN
    f. N-step returns

Key learning goals:

- How Q-functions relate to policies
- How to do RL without learning an explicit policy
- How to stabilize Q-learning in practice

Next time

Done with online RL methods!

CS 224R Tutorial

Review of Q-Learning

Anikait Singh

Outline of Tutorial

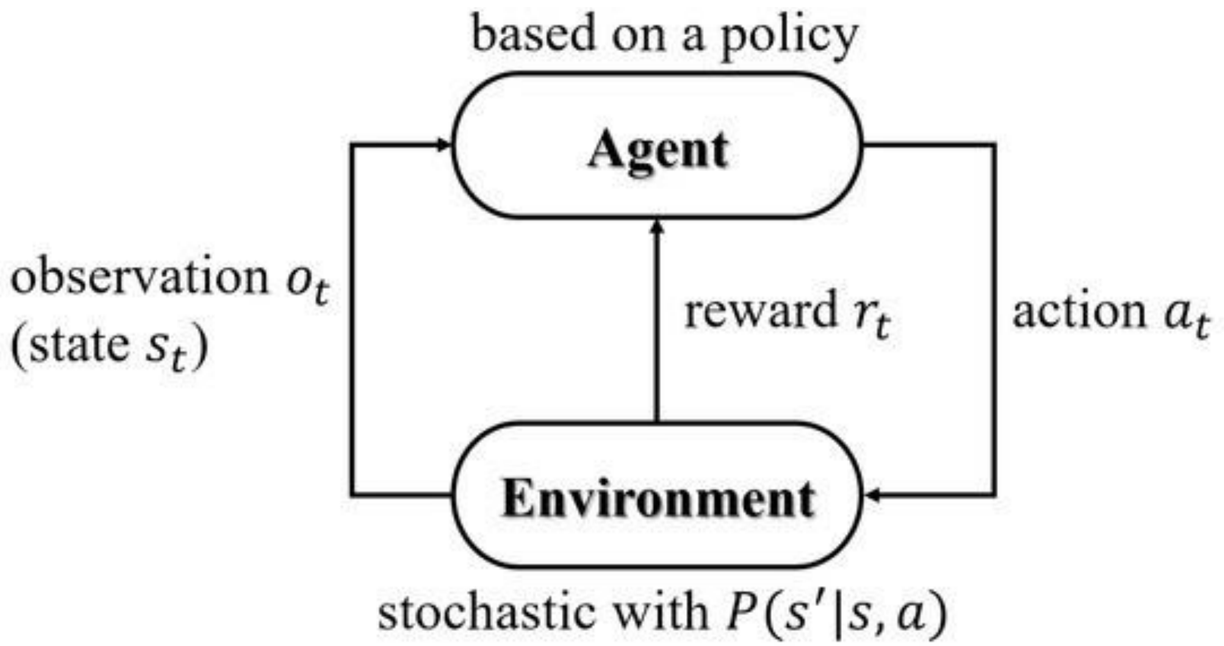
- Review of Markov Decision Processes (MDP)
- Exact MDPs
  - Fitted Q Iteration
- Parametric Q-Learning
  - Bias Variance Tradeoff (TD vs MC)
- Practical Details
  - Replay Buffer, Overestimation, TD Gradients
- Q-Learning/Actor Critic Algorithm Walkthrough

Many thanks to Pieter Abbeel, David Silver, Aviral Kumar, and Justin Fu!

MDP Formulation

An MDP is defined by:

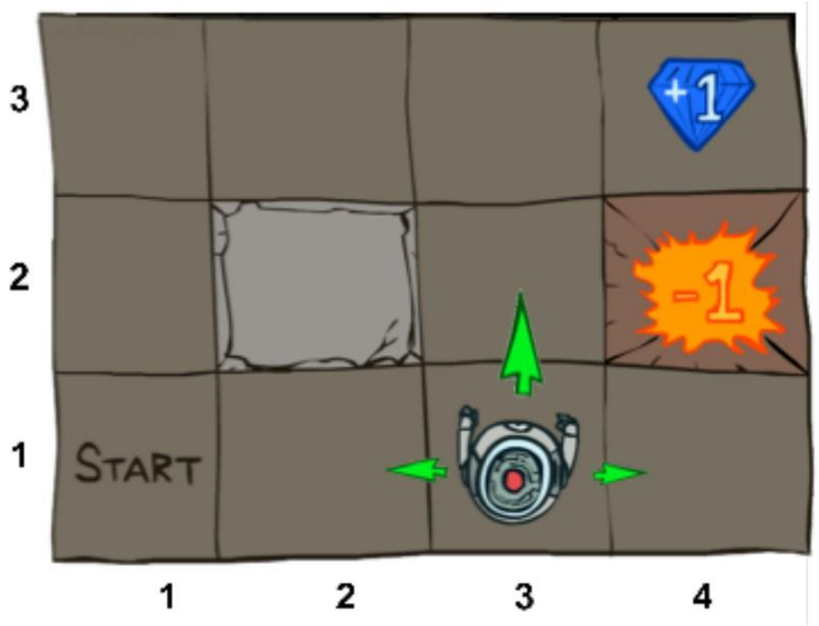
- Set of states s_t
- Set of actions a_t
- Transition function $p(s_{t+1} \mid s_t, a_t)$
- Reward Function $r(s_t, a_t)$
- Start state $_1$
- Discount Factor
- Horizon



MDP Formulation

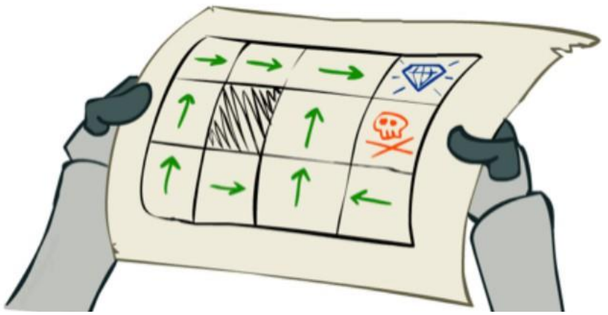
An MDP is defined by:

- Set of states s_t
- Set of actions a_t
- Transition function $p(s_{t+1} \mid s_t, a_t)$
- Reward Function $r(s_t, a_t)$
- Start state $_1$
- Discount Factor
- Horizon



Goal: $\max_{\pi} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^t r(s_t, a_t) \mid \pi\right]$

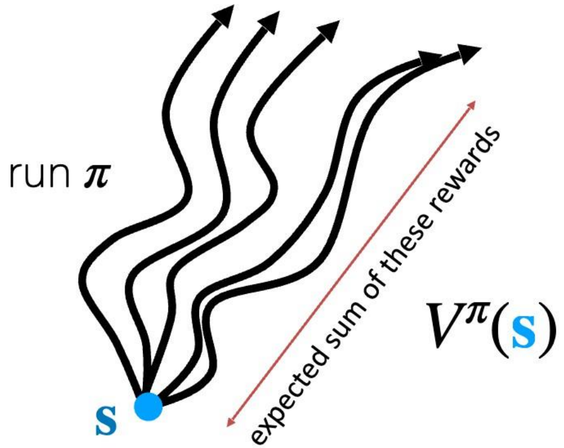
π:

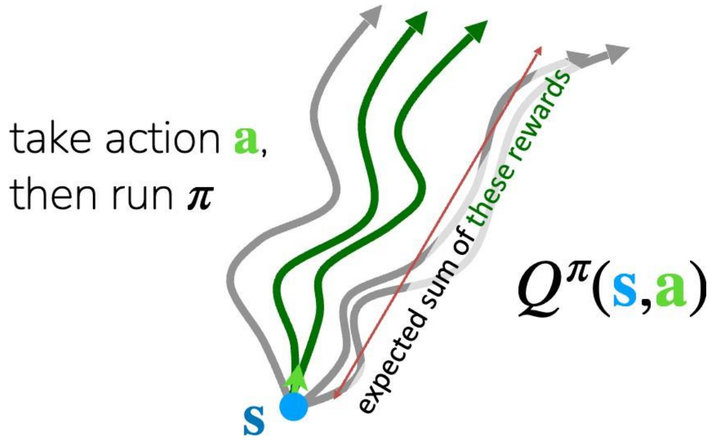


Revisiting Some Useful Objects

value function $V^{\pi}(\mathbf{s})$ - future expected rewards starting at $\mathbf{s}$ and following π

Q-function $Q^{\pi}(\mathbf{s}, \mathbf{a})$ - future expected rewards starting at $\mathbf{s}$, taking $\mathbf{a}$, then following π





Useful relation: $V^{\pi}(\mathbf{s}) = \mathbb{E}_{\mathbf{a}\sim\pi(\cdot|\mathbf{s})}\left[Q^{\pi}(\mathbf{s}, \mathbf{a})\right]$

advantage $A^{\pi}(\mathbf{s}, \mathbf{a})$ - how much better it is to take $\mathbf{a}$ than to follow policy π at state $\mathbf{s}$

$$A^{\pi}(\mathbf{s}, \mathbf{a}) = Q^{\pi}(\mathbf{s}, \mathbf{a}) - V^{\pi}(\mathbf{s})$$

Tabular Learning - Fitted Q Iteration

Q-Function $Q^{\pi}(\mathbf{s}, \mathbf{a})$ - future expected rewards starting at state $\mathbf{s}$, taking action $\mathbf{a}$, then following π

$$Q^*(s,a) = \sum_{s_{t+1}} p(s_{t+1} \mid s_t, a_t) \left(r(s_t, a_t) + \gamma \max_{a_t} Q^*(s_{t+1}, a_{t+1}) \right)$$

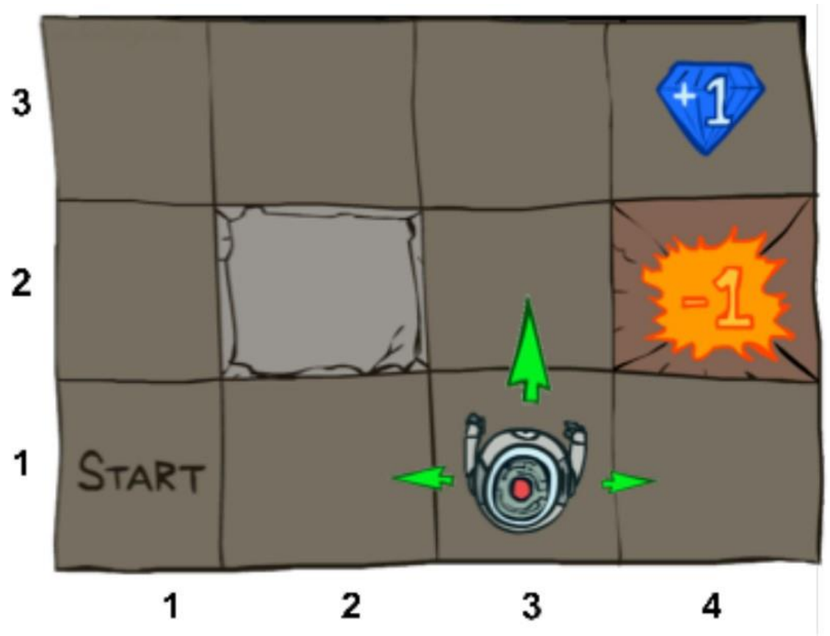
Q-Value Iteration

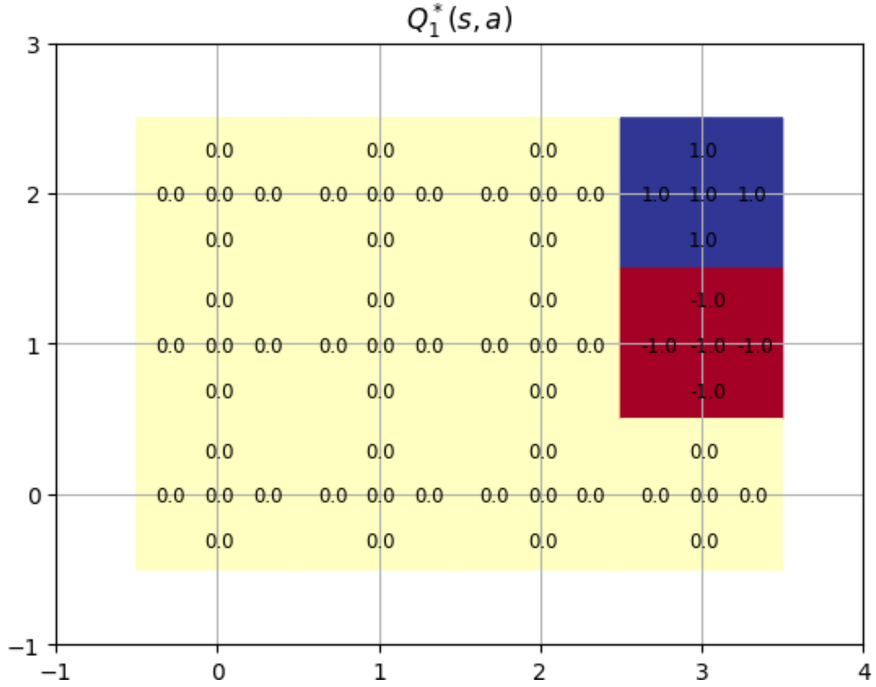
$$Q^*_{k+1}(s,a) = \sum_{s_{t+1}} p(s_{t+1} \mid s_t, a_t) \left(r(s_t, a_t) + \gamma \max_{a_t} Q^*_k(s_{t+1}, a_{t+1}) \right)$$

Fitted Q Iteration with the MDP

Discount = 0.9

$$Q^*_{k+1}(s,a) = \sum_{s_{t+1}} p(s_{t+1} \mid s_t, a_t)\Big(r(s_t, a_t) + \gamma \max_{a_t} Q^*_k(s_{t+1}, a_{t+1})$$

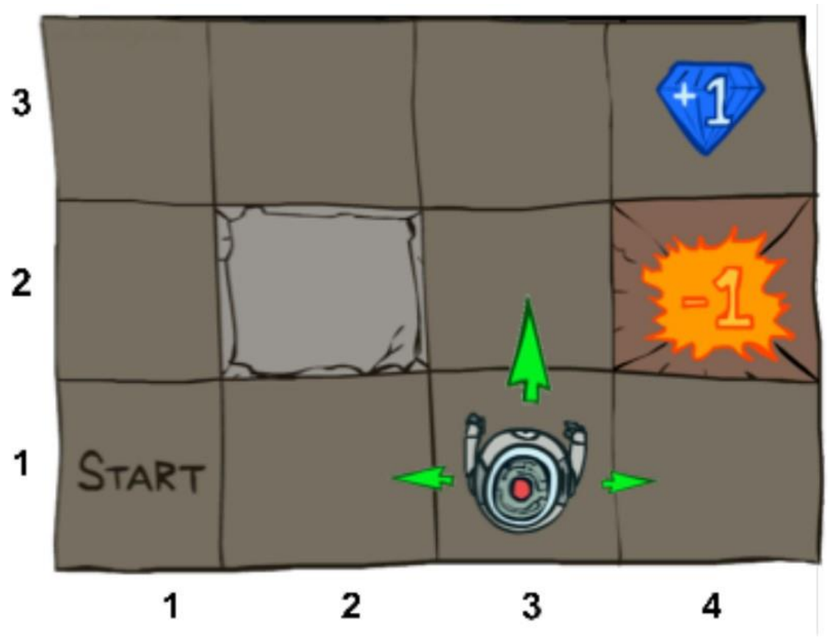


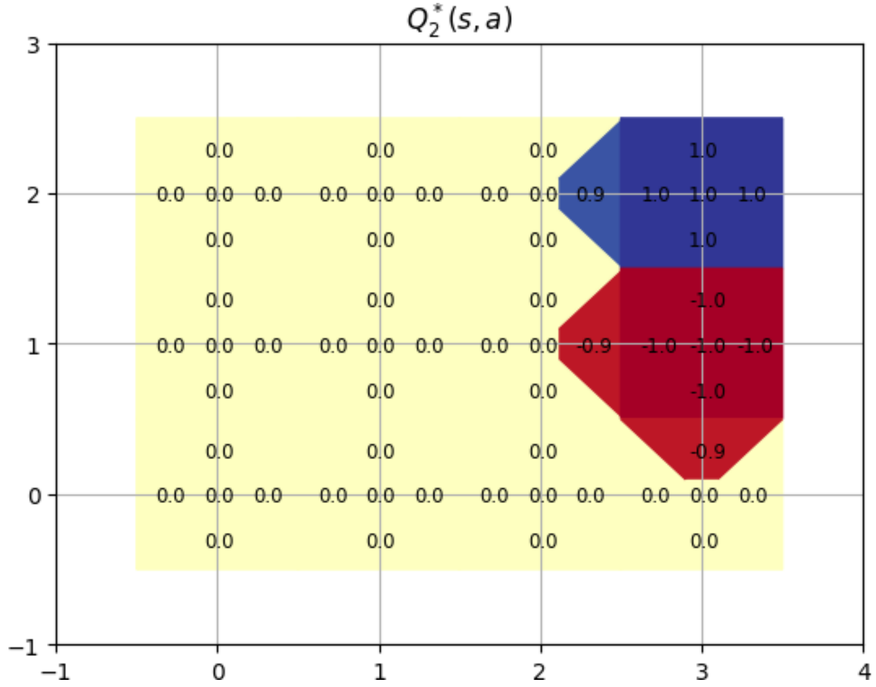


Fitted Q Iteration with the MDP

Discount = 0.9

$$Q^*_{k+1}(s,a) = \sum_{s_{t+1}} p(s_{t+1} \mid s_t, a_t)\Big(r(s_t, a_t) + \gamma \max_{a_t} Q^*_k(s_{t+1}, a_{t+1})$$

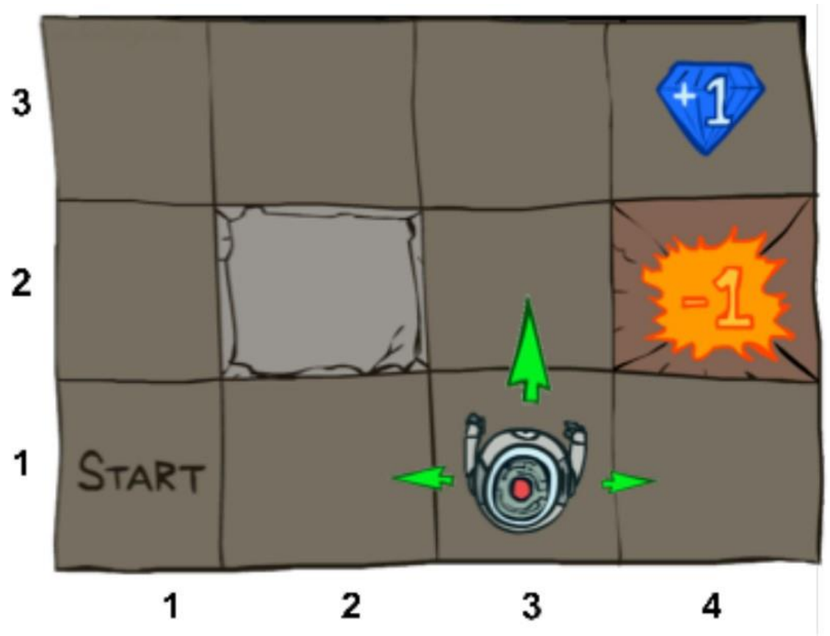


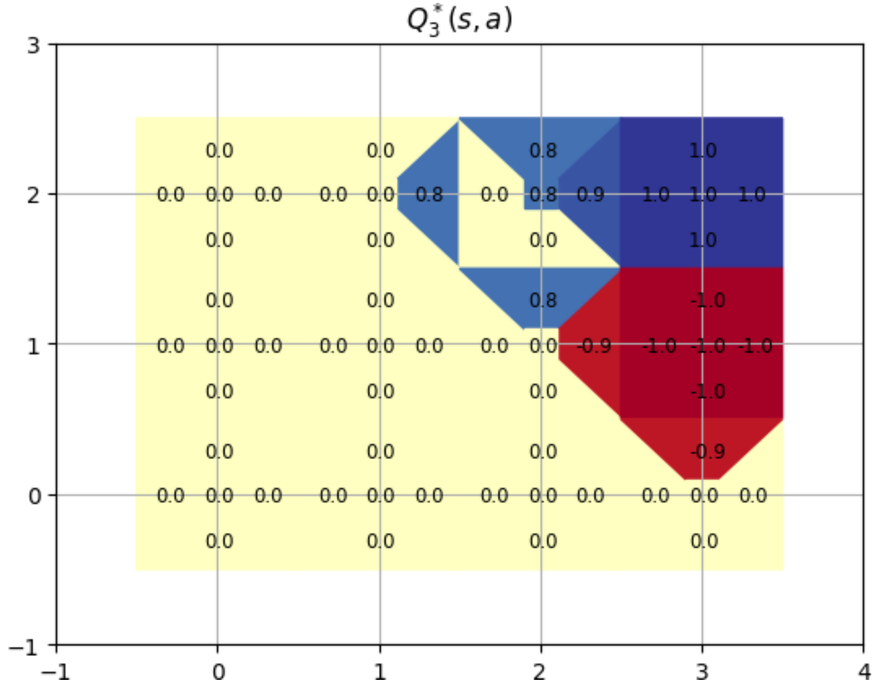


Fitted Q Iteration with the MDP

Discount = 0.9

$$Q^*_{k+1}(s,a) = \sum_{s_{t+1}} p(s_{t+1} \mid s_t, a_t)\Big(r(s_t, a_t) + \gamma \max_{a_t} Q^*_k(s_{t+1}, a_{t+1})\Big)$$

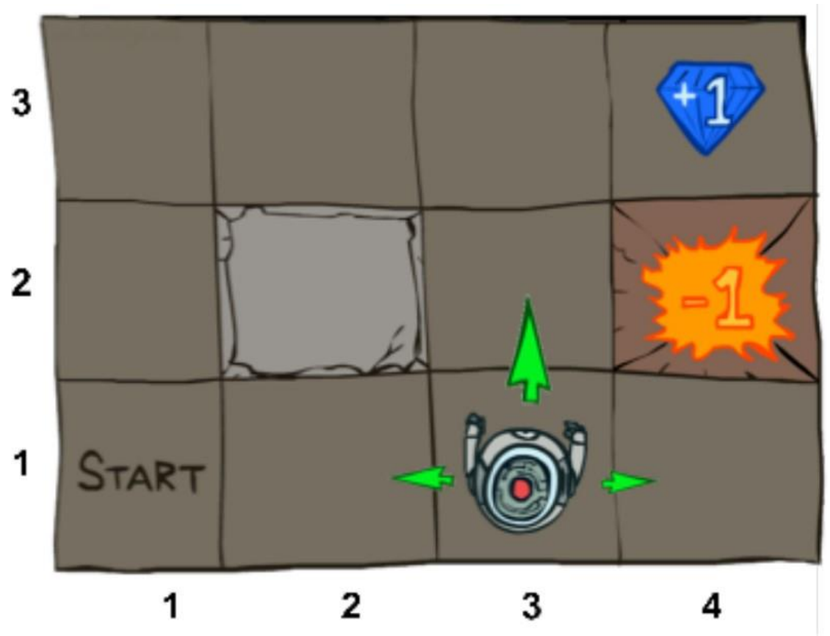


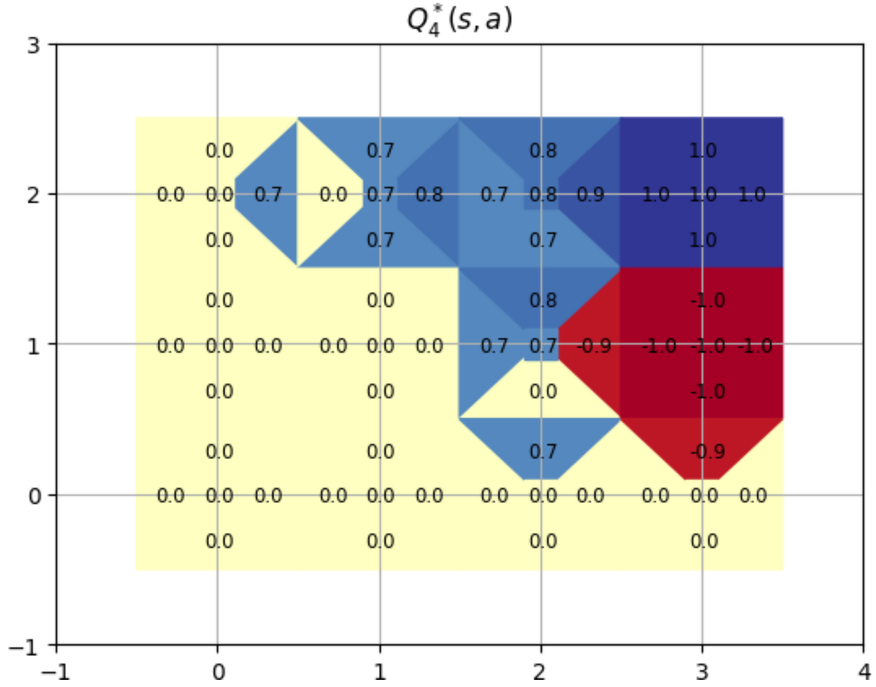


Fitted Q Iteration with the MDP

Discount = 0.9

$$Q^*_{k+1}(s,a) = \sum_{s_{t+1}} p(s_{t+1} \mid s_t, a_t)\Big(r(s_t, a_t) + \gamma \max_{a_t} Q^*_k(s_{t+1}, a_{t+1})\Big)$$

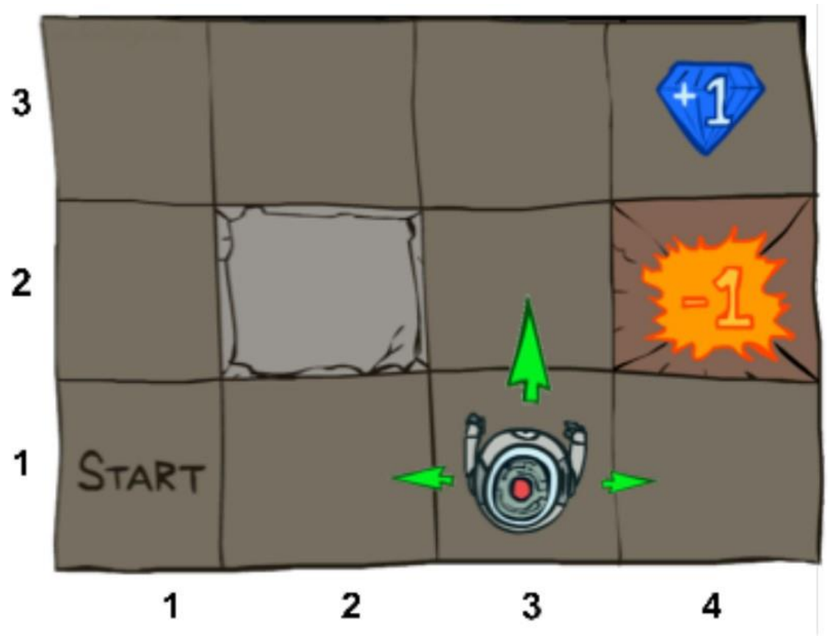


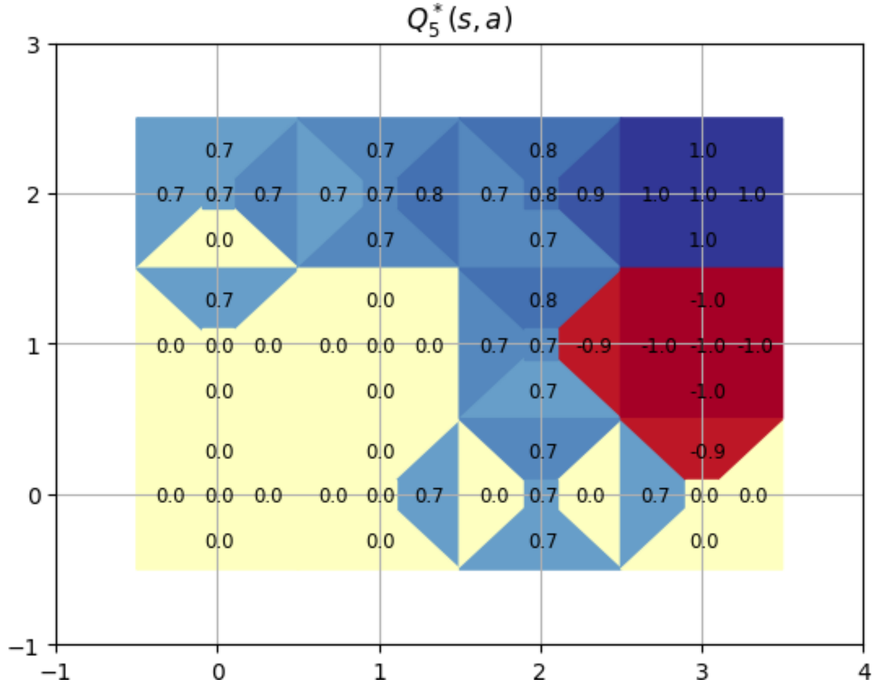


Fitted Q Iteration with the MDP

Discount = 0.9

$$Q^*_{k+1}(s,a) = \sum_{s_{t+1}} p(s_{t+1} \mid s_t, a_t)\Big(r(s_t, a_t) + \gamma \max_{a_t} Q^*_k(s_{t+1}, a_{t+1})\Big)$$

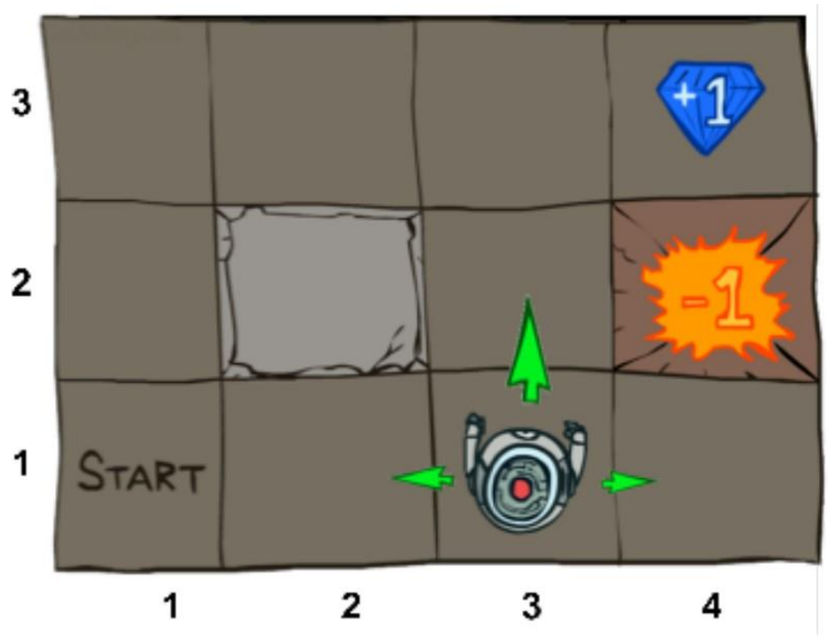


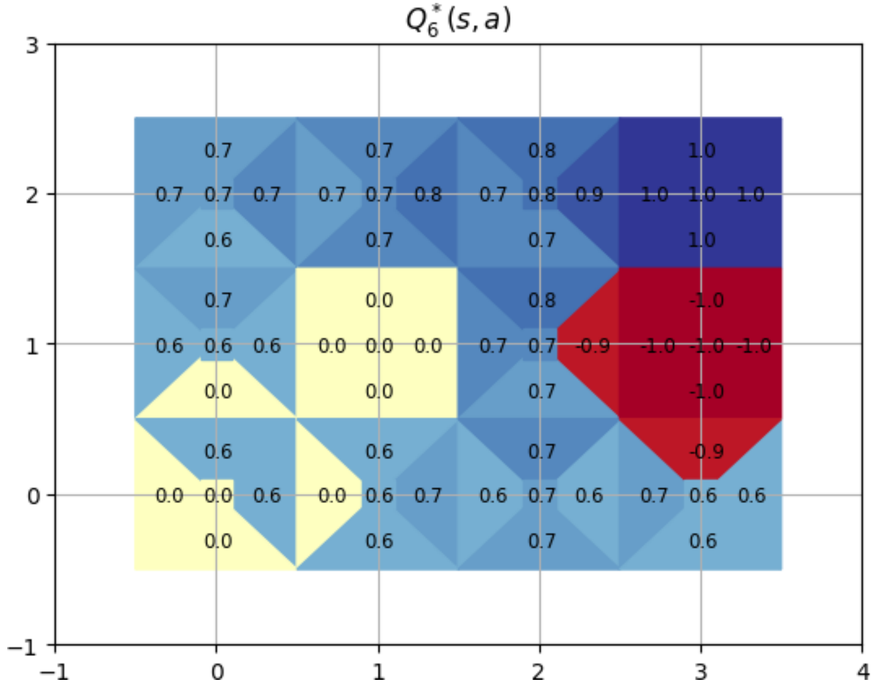


Fitted Q Iteration with the MDP

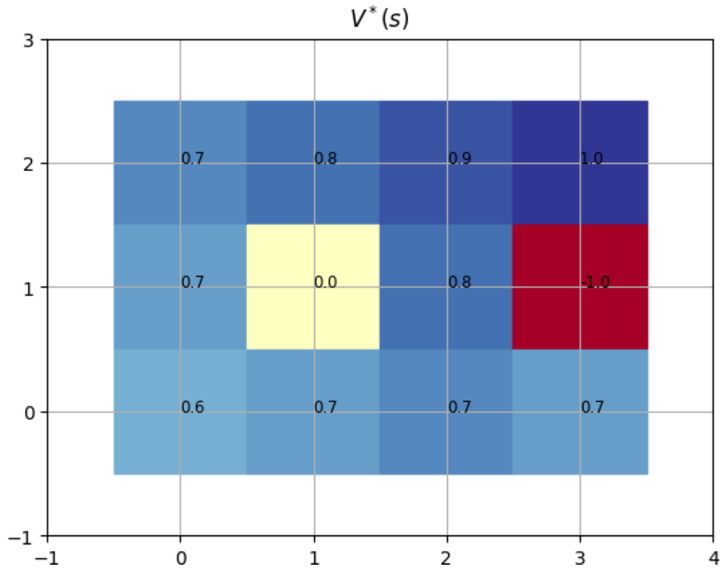
Discount = 0.9

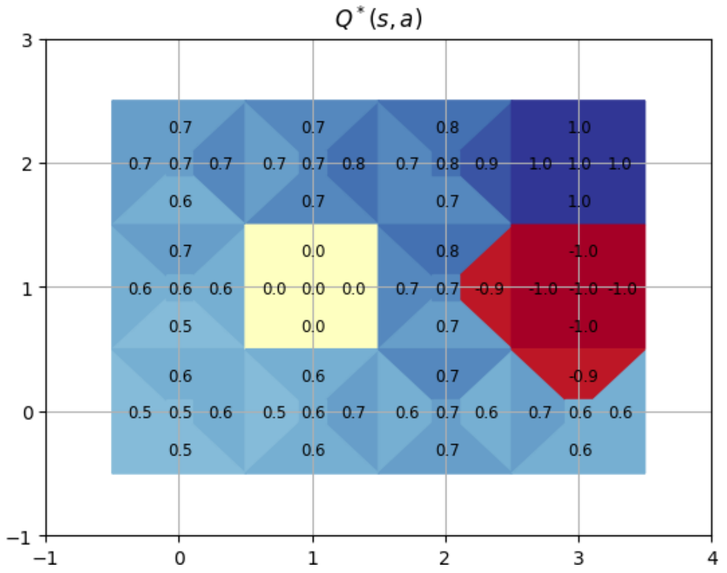
$$Q^*_{k+1}(s,a) = \sum_{s_{t+1}} p(s_{t+1} \mid s_t, a_t)\Big(r(s_t, a_t) + \gamma \max_{a_t} Q^*_k(s_{t+1}, a_{t+1})\Big)$$

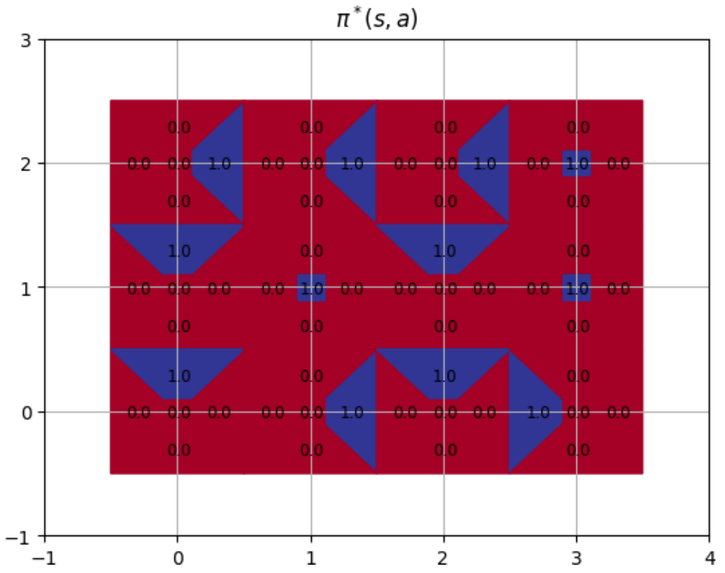




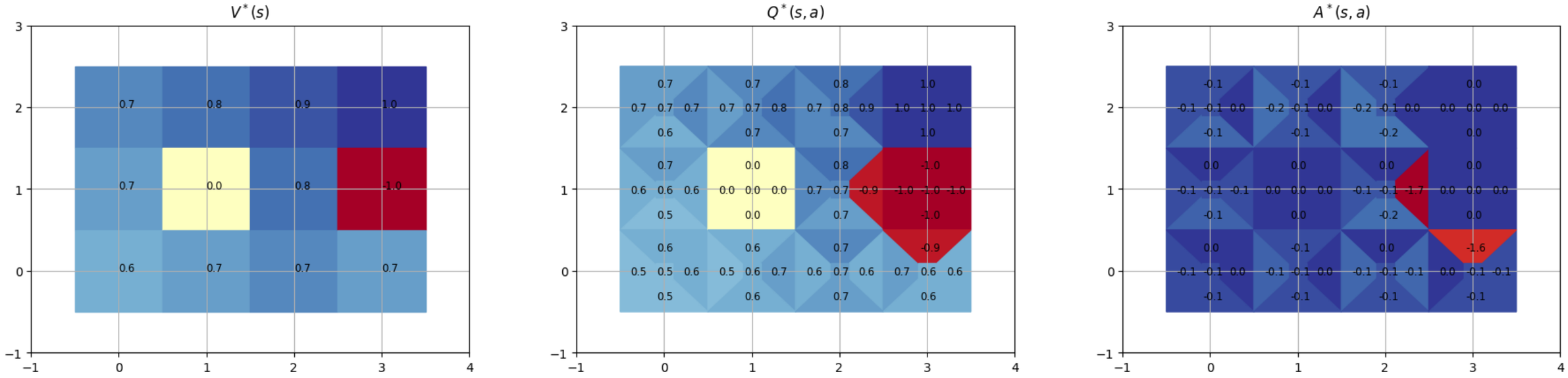
Optimal Value, Q-Value, Policy







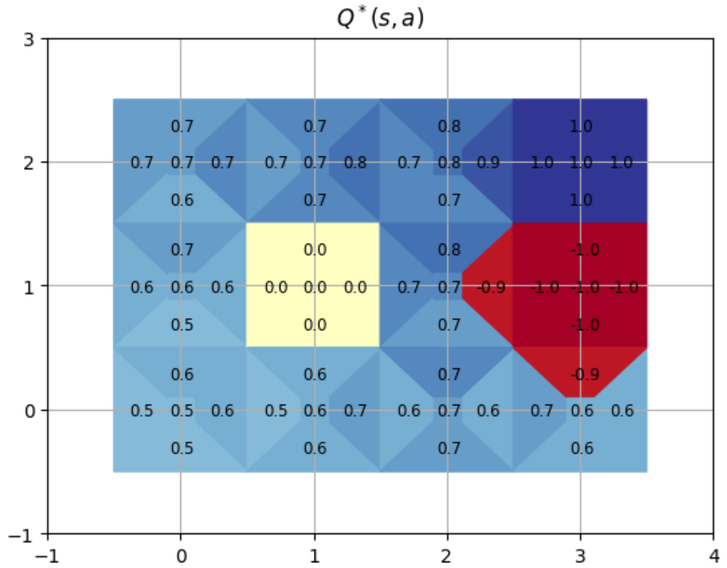
Advantage (Optimal Policy)

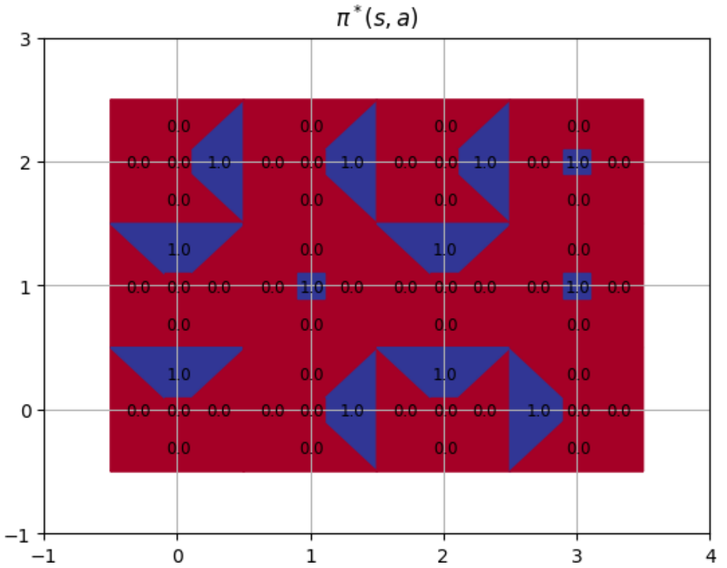


$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)\text{: how much better } \mathbf{a}_t \text{ is}$$

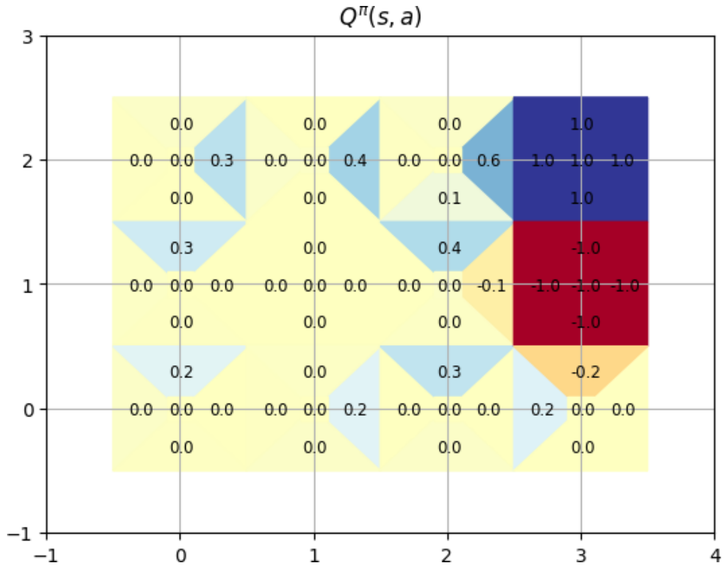
Q^* vs Q^π

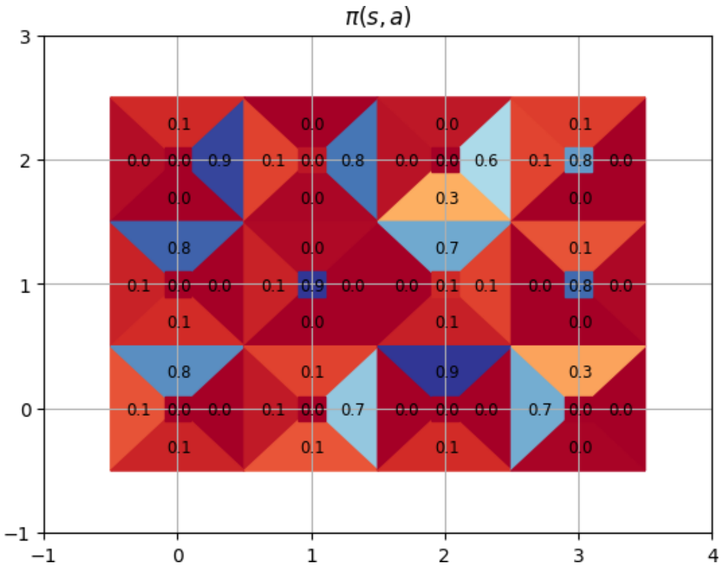
Optimal Policy
(Deterministic for MDP)





Suboptimal Policy





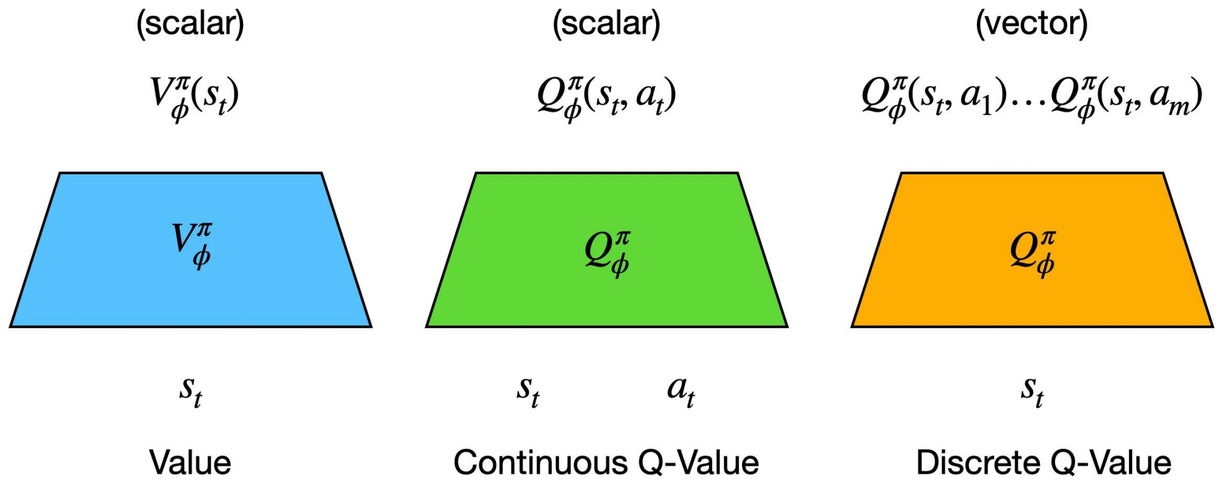
To play more with the Tabular Environment, checkout the colab

Pause for Questions.

Limitations of Tabular Environment

Moving towards parametric value functions

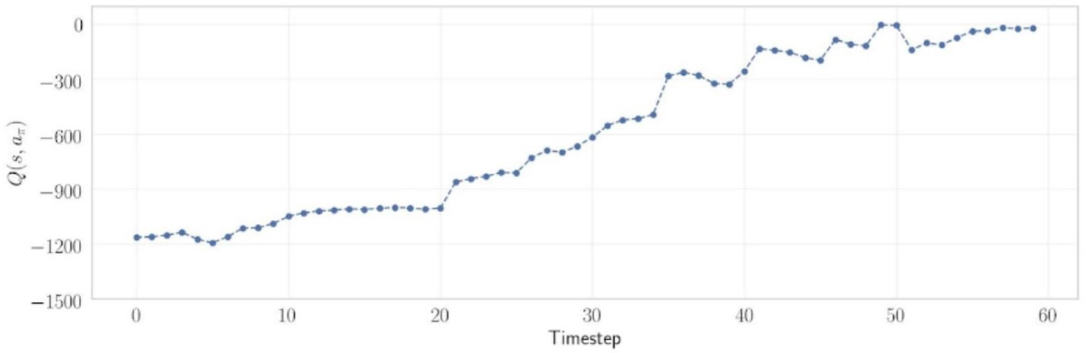
- A tabular value function intractable as state/action space grows!
- Learning a parametric value function can also allow for generalization
  - Related states have similar q-values (smoothness)
- Can use regression to learn value functions with a neural network



Discrete vs Continuous Q-values

- Different formulations for continuous and discrete action spaces:
  - In **continuous (a)**, typically condition on an action and output a scalar, where you sample actions from your policy for an expectation calculation
  - In **discrete (b)**, typically learn a value for each bin (output) and can enumerate over all actions for an expectation

a



ATARI Game of Pong

b

Atari Logo Image: Atari, Inc., Public domain, via Wikimedia Commons

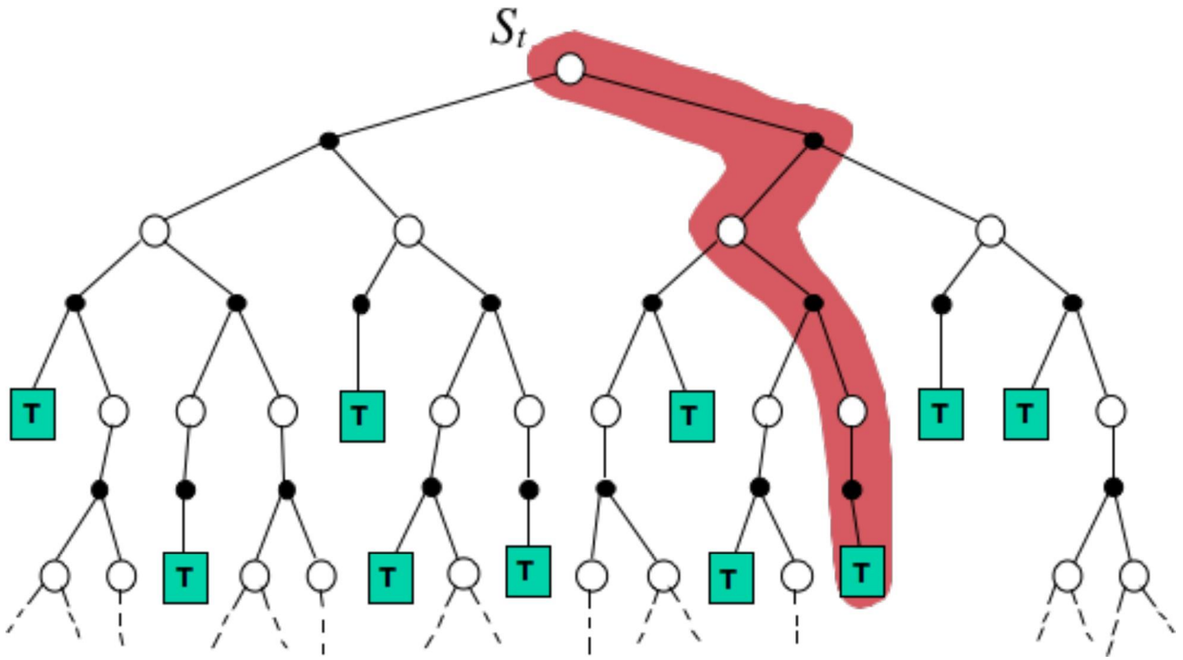
Q-Learning as a Regression Problem

Whiteboard

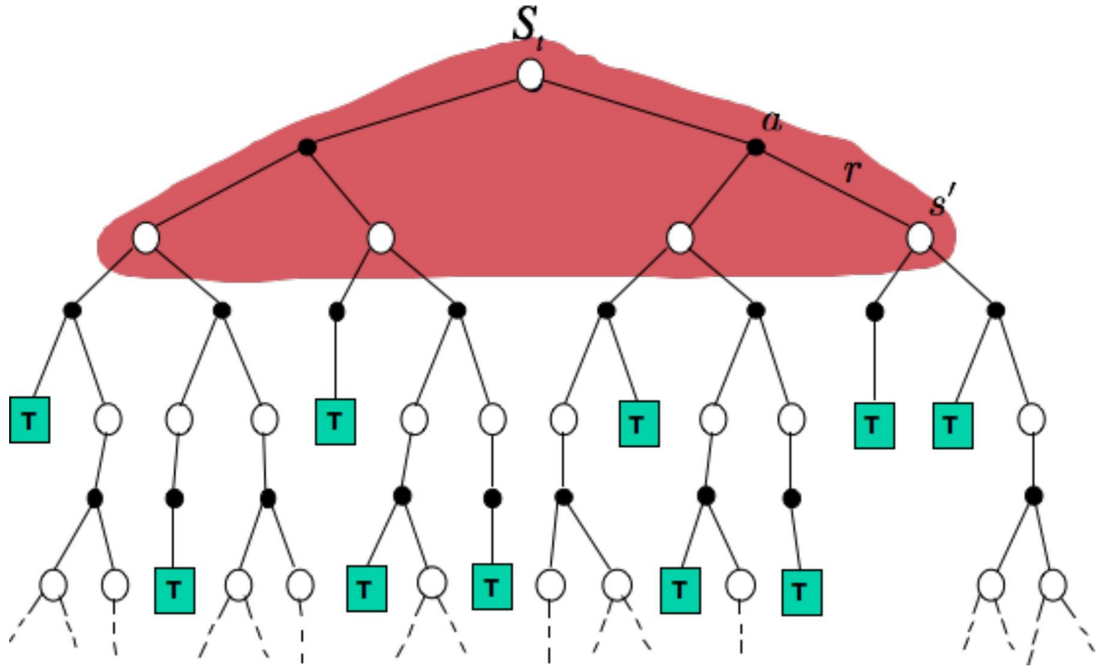
$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left[\mathcal{R}_t \mid S_t = s\right] && \text{(Definition)} \\
&= E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k \, r(s_{t+k}, a_{t+k}) \;\middle|\; S_t = s\right] && \text{(Monte–Carlo Rollout, MC)} \\
&= E_{\pi}\left[r(s_t, a_t) + \gamma \sum_{k=0}^{\infty} \gamma^k \, r(s_{t+k+1}, a_{t+k+1}) \;\middle|\; S_t = s\right] && \text{(Expansion)} \\
&= E_{\pi}\left[r(s_t, a_t) + \gamma \, V^{\pi}(s_{t+1}) \mid S_t = s\right] && \text{(Dynamic Programming, TD)} \\
&= E_{\pi}\left[r(s_t, a_t) + \gamma \, r(s_{t+1}, a_{t+1}) + \cdots + \gamma^{k+1} \, V^{\pi}(s_{t+k+1}) \;\middle|\; S_t = s\right] && \text{(N-Step)}
\end{aligned}
$$

Bias vs Variance Tradeoff!

MC vs TD Visualization



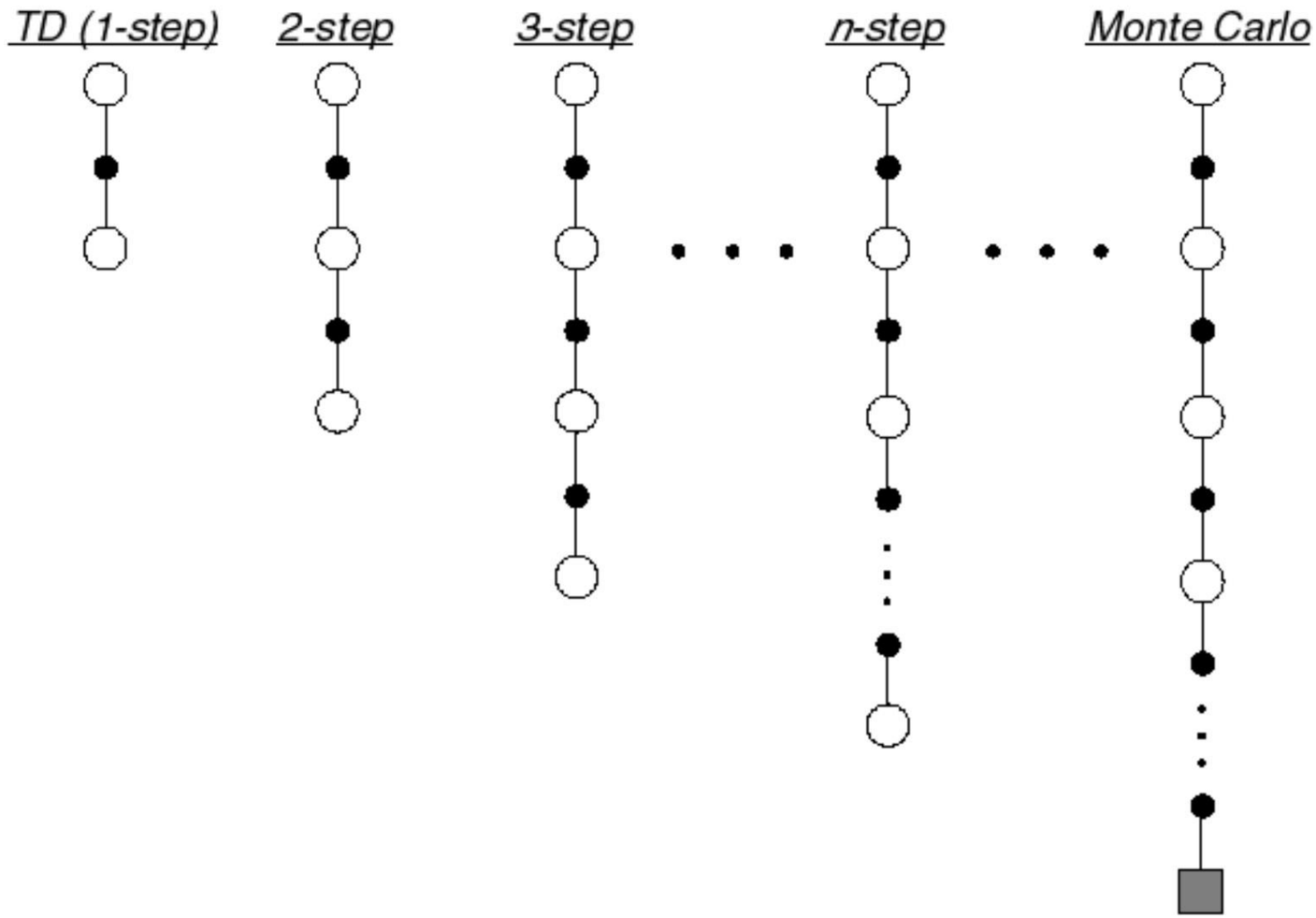
Monte-Carlo Rollouts
(Regression)



TD Backup
(Dynamic Programming)

N-Step Returns

Let TD target look *n* steps into the future



Bias vs Variance Tradeoff

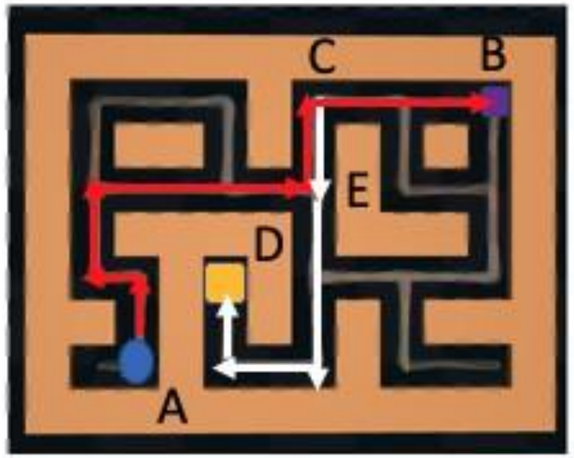
TD, MC, N-Step Return

Method	Bootstrap Depth	Bias	Variance
TD	1 Steps	High	Low
N-Step	N Steps	Medium	Medium
Monte Carlo	Full Episode	Low	High

Why Dynamic Programming?

Aside on Stitching

Our maze navigation task

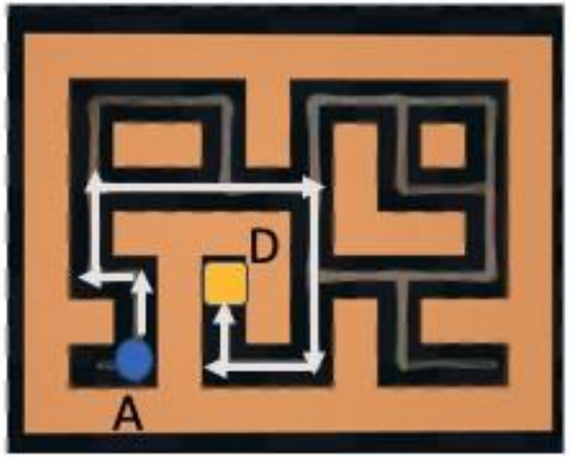


Start (A) Goal (D)

trajectory from A → B

trajectory from C → D

"Stitching" enables finding optimal behavior



Trajectory found by stitching segments of trajectories

Pause for Questions.

Practical Implementation Detail

Semi-Gradient and Target Network

Going back to our formulation:

$$Q(s,a) \leftarrow r(s,a) + \gamma \max_{\hat{a}} Q(s', \hat{a})$$

Semi-Gradient

$$Q_\theta(s,a) \leftarrow r(s,a) + \gamma \max_{\hat{a}} \text{stopgrad}\left(Q_\theta(s', \hat{a})\right)$$

Target Network

$$Q_\theta(s,a) \leftarrow r(s,a) + \gamma \max_{\hat{a}} \text{stopgrad}\left(Q_{\text{target}}(s', \hat{a})\right)$$

Update Types

$w' \leftarrow w$ when $\text{mod}(n, N) = 0$

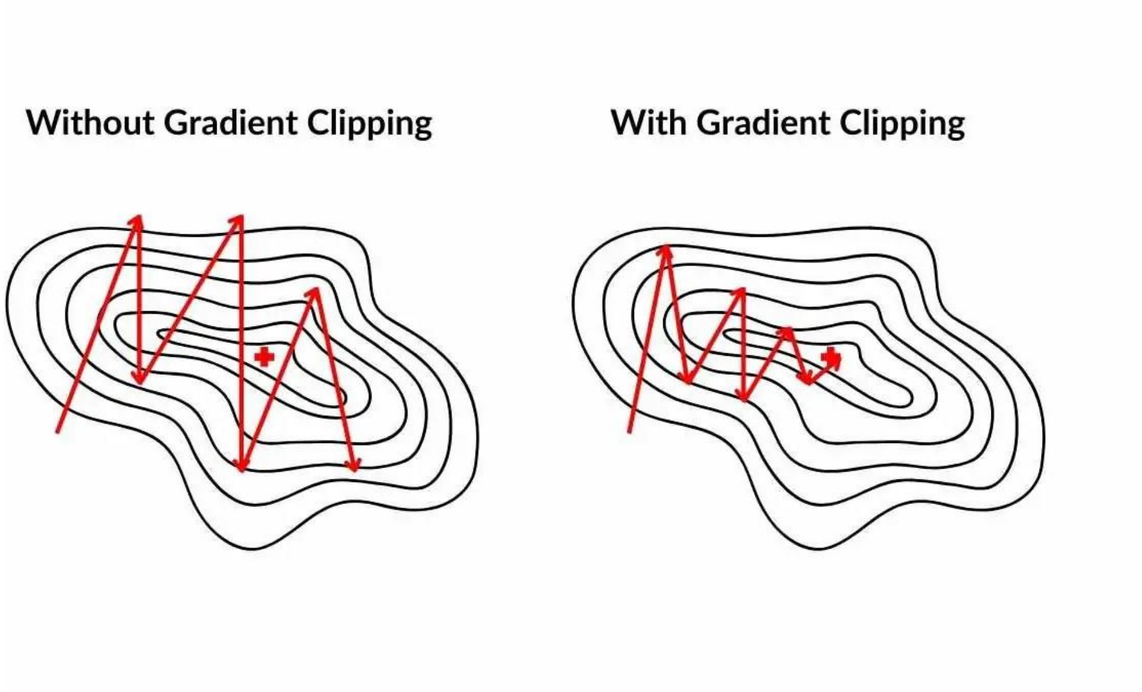
Hard Target Update

$w' \leftarrow \tau \cdot w + (1 - \tau) \cdot w'$

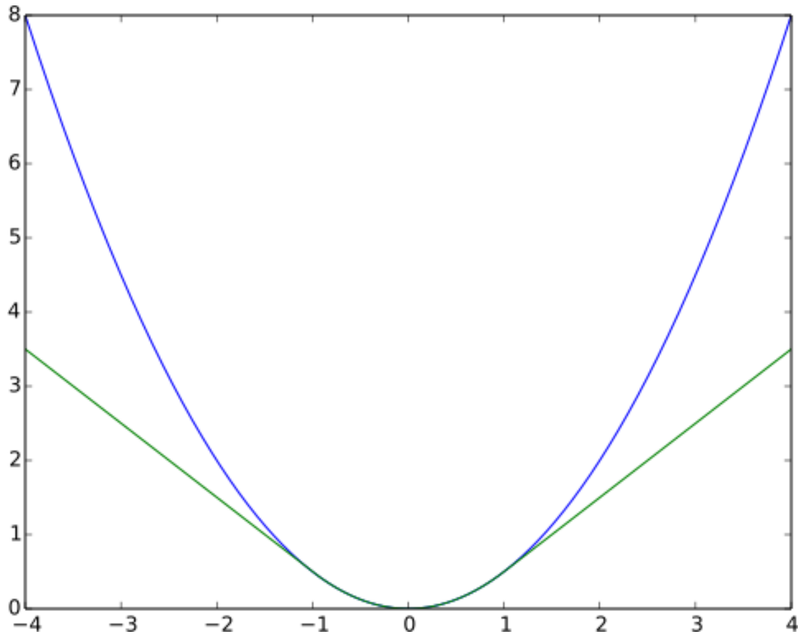
Soft Target Update (Polyak)

Practical Implementation Details

Dealing with TD Gradients



Gradient Clipping

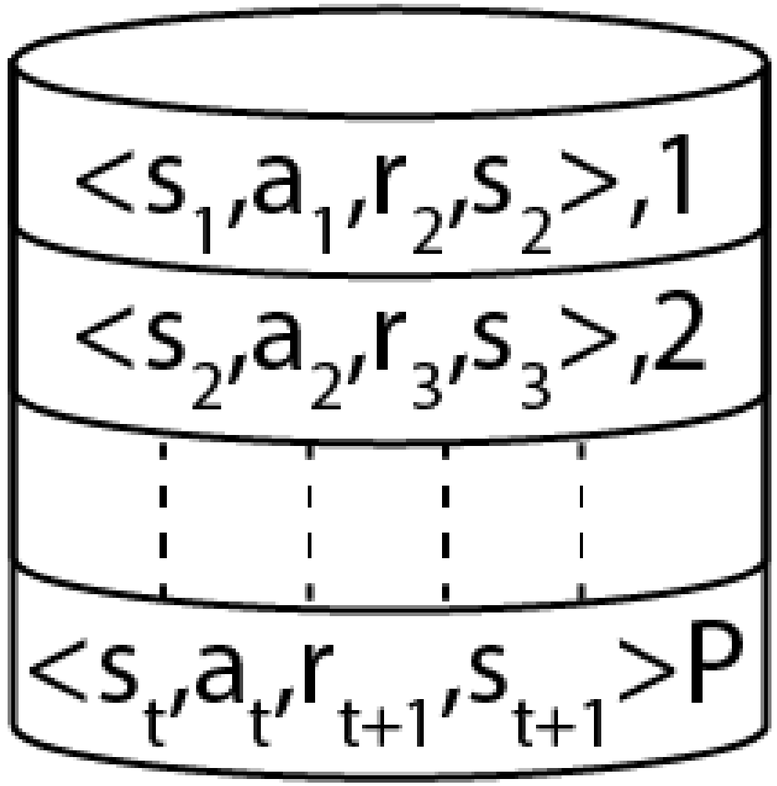


Huber Loss

Practical Implementation Details

Replay Buffers

- Replay Buffers are **memory structures** that store past experiences encountered by the agent, allowing the **experiences to be reused** instead of discarded after a single update
- They **break temporal correlations** between consecutive training samples and prevents recency bias
- **Improves sample efficiency** in Q-learning algorithms, allowing an agent to reuse its experience effectively



Q-Learning Overestimation

Q-Learning is typically formulated as follows:

$$Q(s,a) \leftarrow r(s,a) + \gamma \max_{\hat{a}} Q(s',\hat{a})$$

When parameterized by a function approximator like a NN, there is some inherent noise in the q-value estimates due to modeling errors

$$Q^{\text{approx}}(s',\hat{a}) = Q^{\text{target}}(s',\hat{a}) + Y^{a}_{s'}$$

Due to the noise on the RHS ($Y^{\hat{a}}_{s'}$), there results in error on the LHS, expressed as:

$$Z_s := \gamma \left(\max_{\hat{a}} Q^{\text{approx}}(s',\hat{a}) - \max_{\hat{a}} Q^{\text{target}}(s',\hat{a}) \right)$$

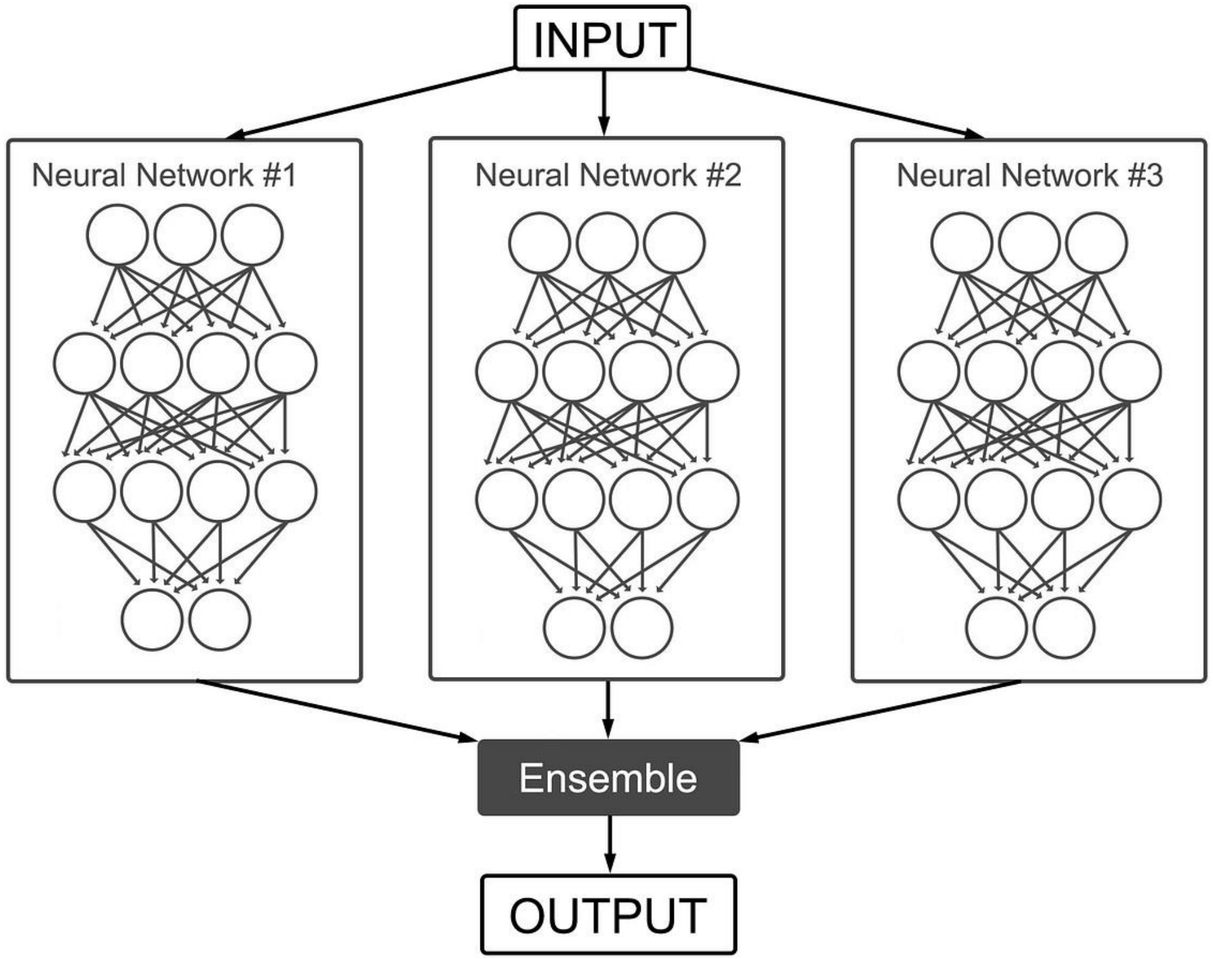
Even with zero mean ($Y^{\hat{a}}_{s'}$), Z_s may have positive mean:

$$\mathbb{E}[Y^{\hat{a}}_{s'}] = 0 \forall \hat{a} \implies \mathbb{E}[Z_s] > 0 \quad \text{(often)}$$

Practical Implementation Detail

Critic Ensembling

- One approach to tackle overestimation is to measure **epistemic uncertainty** with an ensemble of networks
- Since different model initializations may have **different modeling errors**, this can lead to ensembling with a **min reduction** to be effective
- Has been even shown to be effective in **offline RL** where distribution shift is a larger concern (Sac-N and EDAC)



Q-Learning vs Double Q-Learning

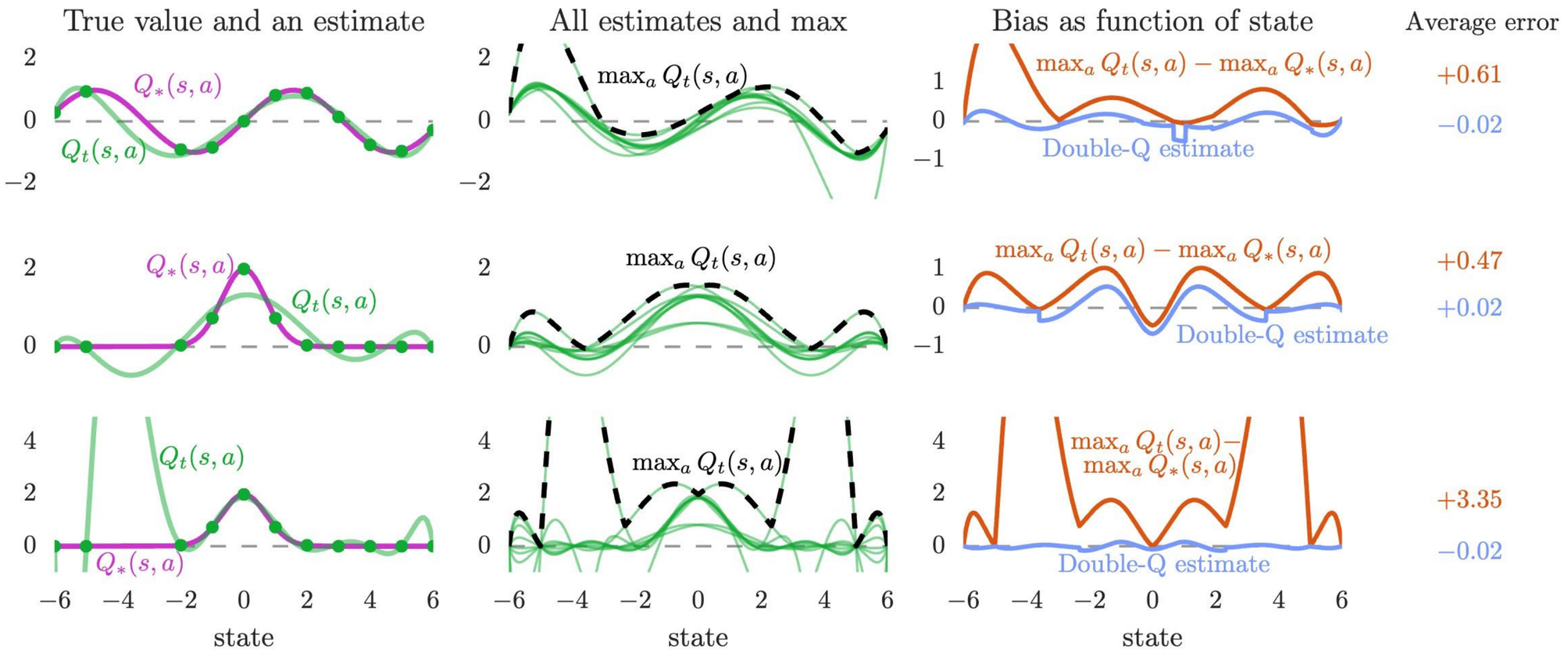


Figure 2: Illustration of overestimations during learning. In each state (x-axis), there are 10 actions. The **left column** shows the true values $V_*(s)$ (purple line). All true action values are defined by $Q_*(s,a) = V_*(s)$. The green line shows estimated values $Q(s,a)$ for one action as a function of state, fitted to the true value at several sampled states (green dots). The **middle column** plots show all the estimated values (green), and the maximum of these values (dashed black). The maximum is higher than the true value (purple, left plot) almost everywhere. The **right column** plots shows the difference in orange. The blue line in the right plots is the estimate used by Double Q-learning with a second set of samples for each state. The blue line is much closer to zero, indicating less bias. The three **rows** correspond to different true functions (left, purple) or capacities of the fitted function (left, green). (Details in the text)

Understanding HW Algorithms

Q-Learning : DQN

Whiteboard

Algorithm 1: DQN

Input: Replay buffer $\mathcal{D}$, Q-network Q_θ, target network $Q_{\theta'}$ with $\theta' \leftarrow \theta$, discount factor γ, batch size N, learning rate α, target update frequency C, exploration schedule ϵ

for *each interaction step* $t = 1, 2, \ldots$ **do**

Observe state s_t

Choose action a_t using ϵ-greedy:

$$a_t = \begin{cases} \text{random action}, & \text{with probability } \epsilon, \\ \arg\max_a Q_\theta(s_t, a), & \text{otherwise} \end{cases}$$

Execute a_t, observe (r_t, s_{t+1}), and store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$

`// Sample a mini-batch`

Sample $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^{N} \sim \mathcal{D}$

`// Compute targets`

for $i = 1, \ldots, N$ **do**

$y_i \leftarrow r_i + \gamma \max_{a'} Q_{\theta'}(s_{i+1}, a')$

`// Q-network update`

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{N} \sum_{i=1}^{N} \left[Q_\theta(s_i, a_i) - y_i \right]^2$$

`// Periodic target update`

if $t \bmod C = 0$ **then**

$\theta' \leftarrow \theta$

Actor Critic: SAC

Whiteboard

Algorithm 1: Soft Actor-Critic (no entropy term)

Input: Replay buffer $\mathcal{D}$, critic nets $Q_{\theta_1}, Q_{\theta_2}$, actor net π_ϕ, target critics $\theta'_i \leftarrow \theta_i$, discount γ, smoothing τ, batch size N

for *each interaction step* $t = 1, 2, \ldots$ **do**

- Observe state s_t
- Sample action $a_t \sim \pi_\phi(\cdot \mid s_t)$
- Execute a_t, observe (r_t, s_{t+1}) and store in $\mathcal{D}$
- `// Sample a mini-batch`
- Sample $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^{N} \sim \mathcal{D}$
- `// Compute targets (no entropy term)`
- **for** $i = 1, \ldots, N$ **do**
  - $a'_{i+1} \leftarrow \pi_\phi(s_{i+1})$;
  - $y_i \leftarrow r_i \; + \; \gamma \, \min_{j=1,2} Q_{\theta'_j}\left(s_{i+1}, a'_{i+1}\right)$
- `// Critic update`
- **for** $j = 1, 2$ **do**
  - $\theta_j \leftarrow \theta_j \; - \; \lambda_Q \, \nabla_{\theta_j} \frac{1}{N} \sum_i \left[Q_{\theta_j}(s_i, a_i) - y_i \right]^2$
- `// Actor update (maximize Q only)`
- $\phi \leftarrow \phi \; + \; \lambda_\pi \, \nabla_\phi \frac{1}{N} \sum_i Q_{\theta_1}\left(s_i, \pi_\phi(s_i)\right)$
- `// Target networks soft-update`
- **for** $j = 1, 2$ **do**
  - $\theta'_j \leftarrow \tau \, \theta_j + (1 - \tau) \, \theta'_j$