Foundations of RL
Lecture 9: Modern RL Algorithms

Enrique Mallada

Goals:

- Give an overview of Deep RL methods
- Deep Policy Gradient Methods

Sources

- **Courses from the Experts:**
  1. Sergey Levine, Deep Reinforcement Learning, Decision Making, and Control (2023)
  2. David Silver, Reinforcement Learning (2015)
  3. **Jared Markowitz, Applied Physics Lab Lectures (2023)**

- **Textbook:**

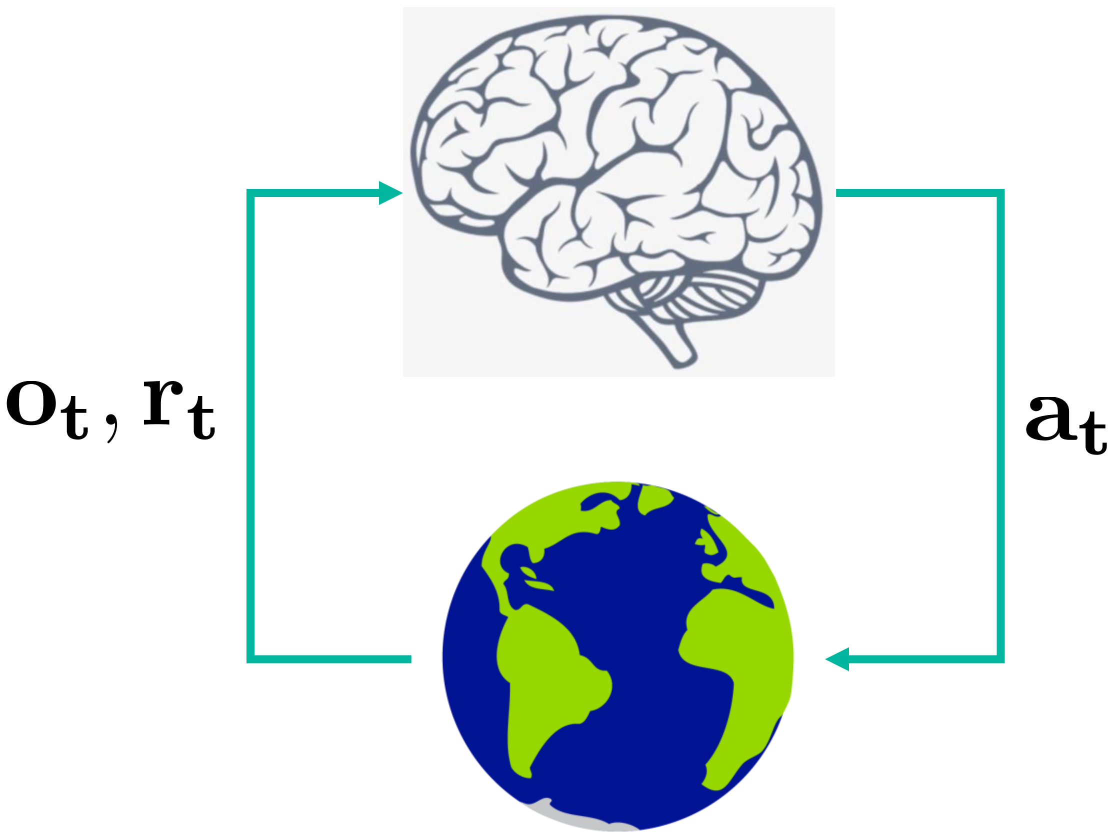
  Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. Available online at http://www.incompleteideas.net/book/the-book-2nd.html

- **OpenAI: Spinning Up in Deep RL**

- **Research papers** will be cited throughout

➢ **We will heavily leverage Levine's course, particularly for theory.**

What is Reinforcement Learning?



At each timestep t, the agent:

- Receives observation o_t
- Receives scalar reward r_t
- Performs action a_t

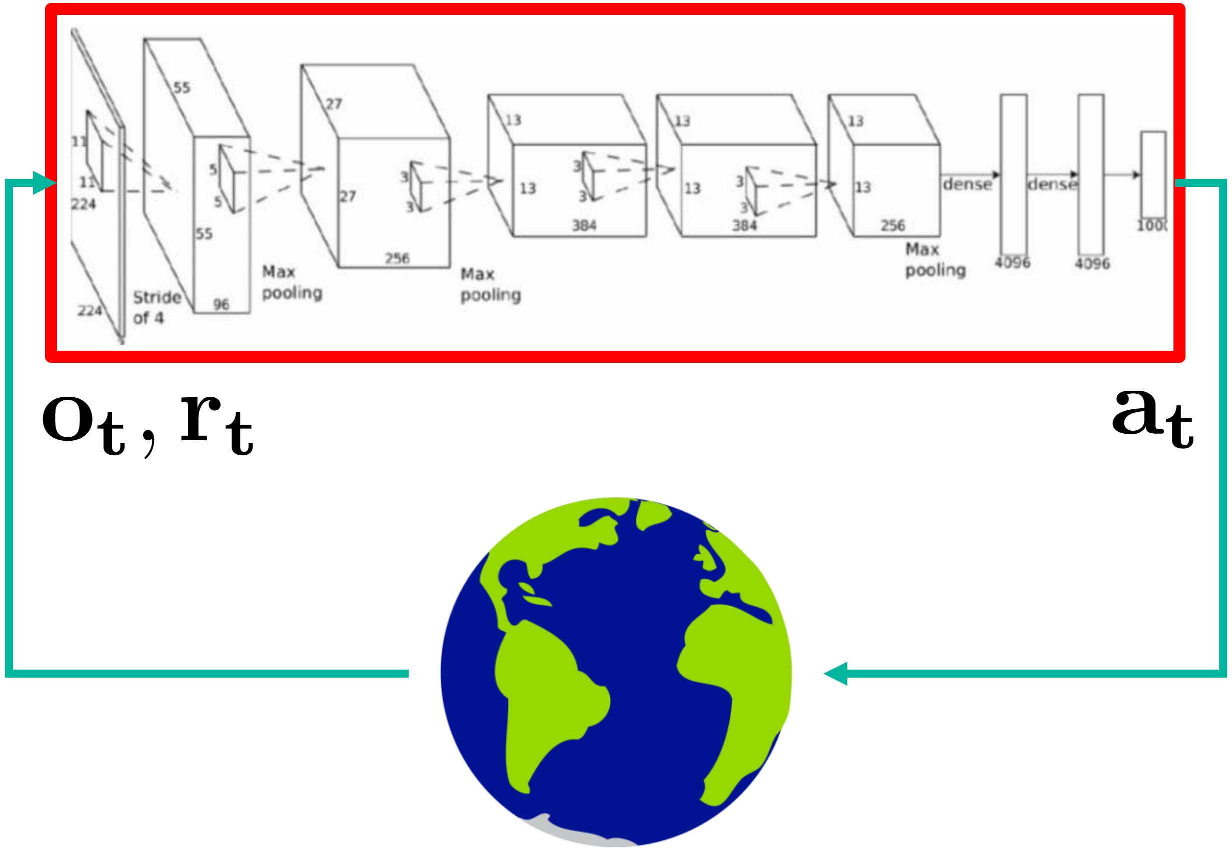
The environment:

- Receives action a_t
- Provides observation o_t
- Emits scalar reward r_t

Goal: Select the actions that maximize future expected rewards

What is *Deep* Reinforcement Learning?

$$\pi_\theta(\mathbf{a_t}|\mathbf{o_t})$$



- Deep neural network performs *function approximation* for the agent.
- Greatly increases the versatility and scalability of RL

Evolution of DRL Mirrors Evolution of Computer Vision

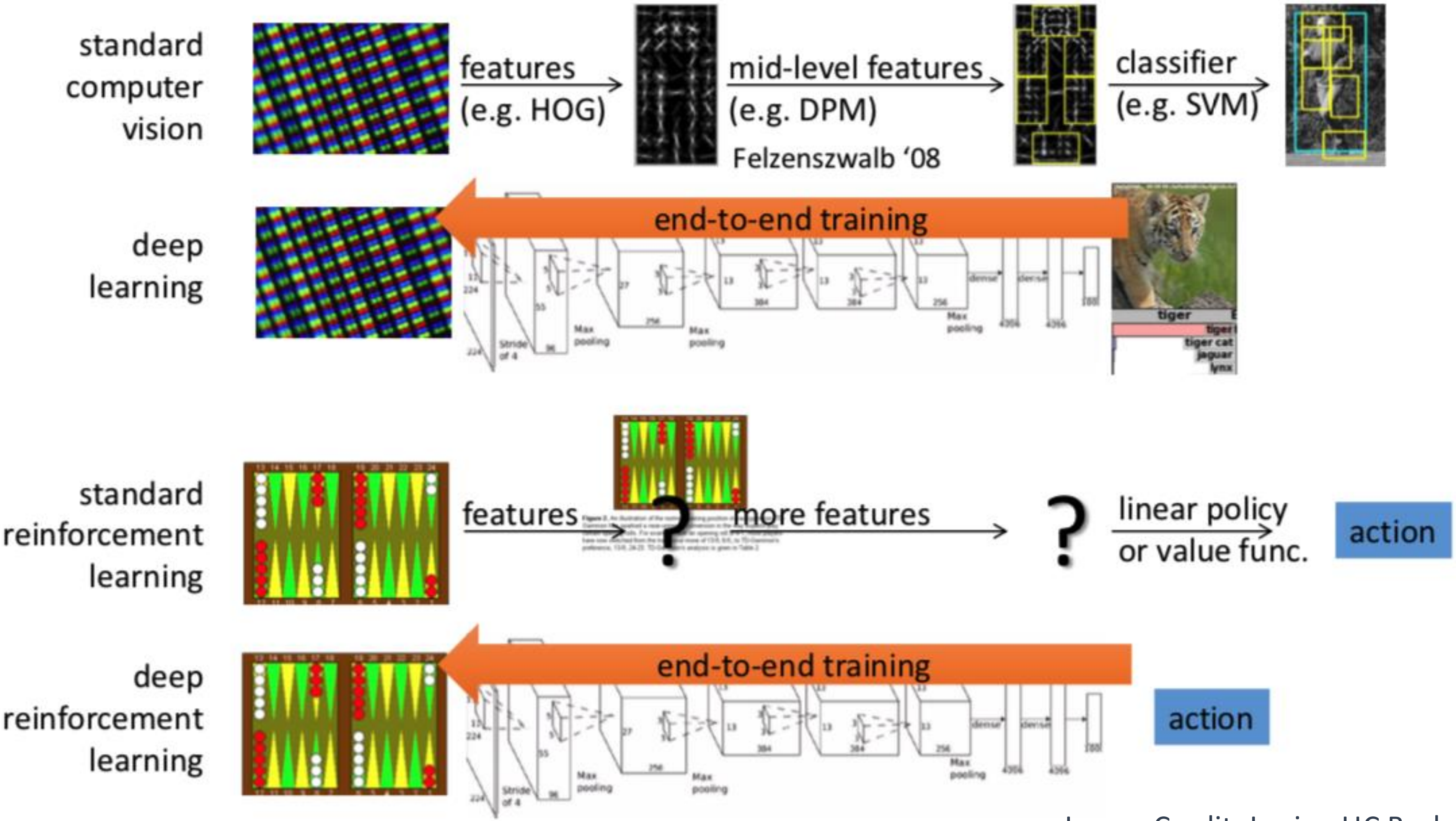
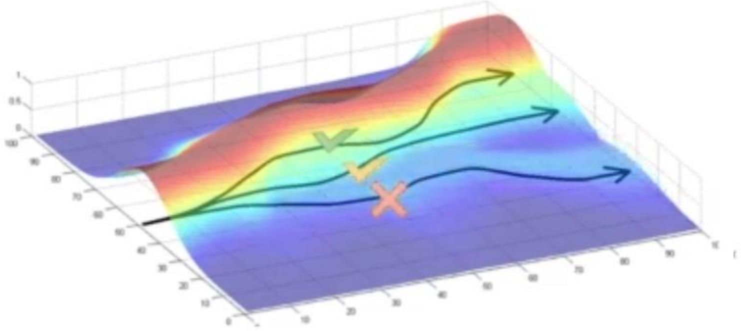
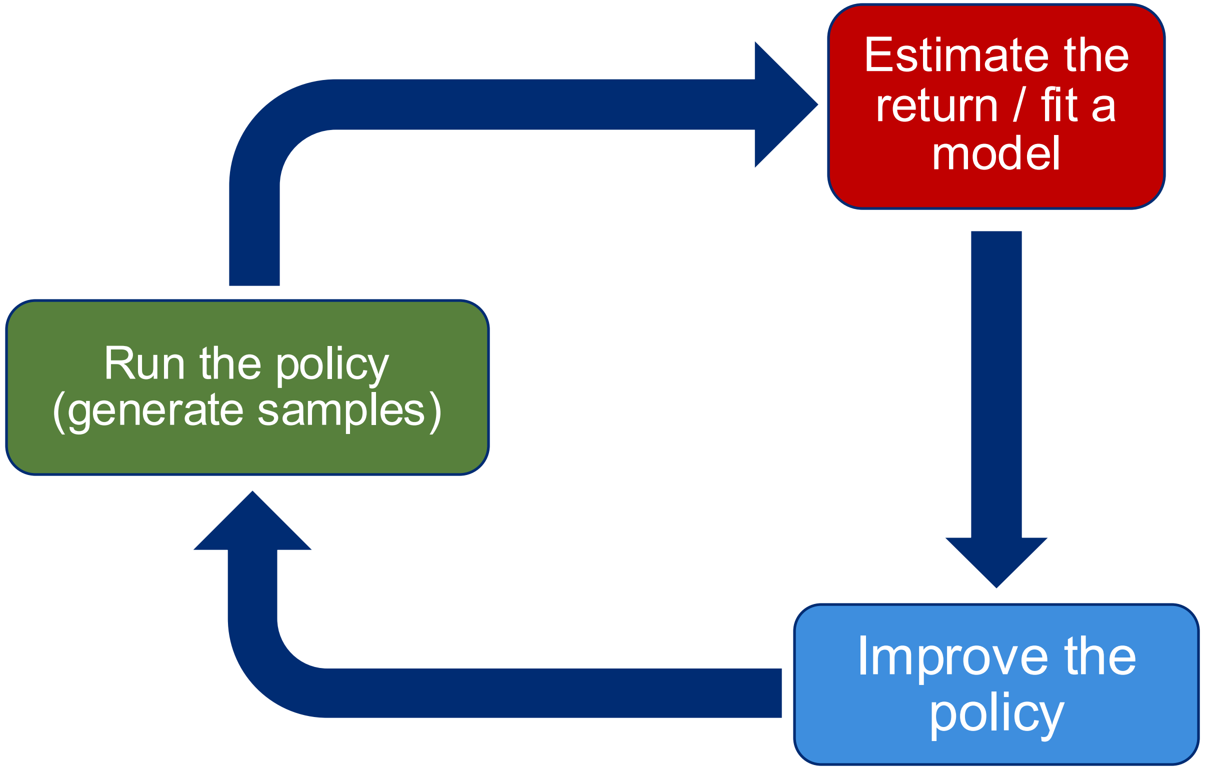


Image Credit: Levine UC Berkeley Course

Anatomy of an RL Agent





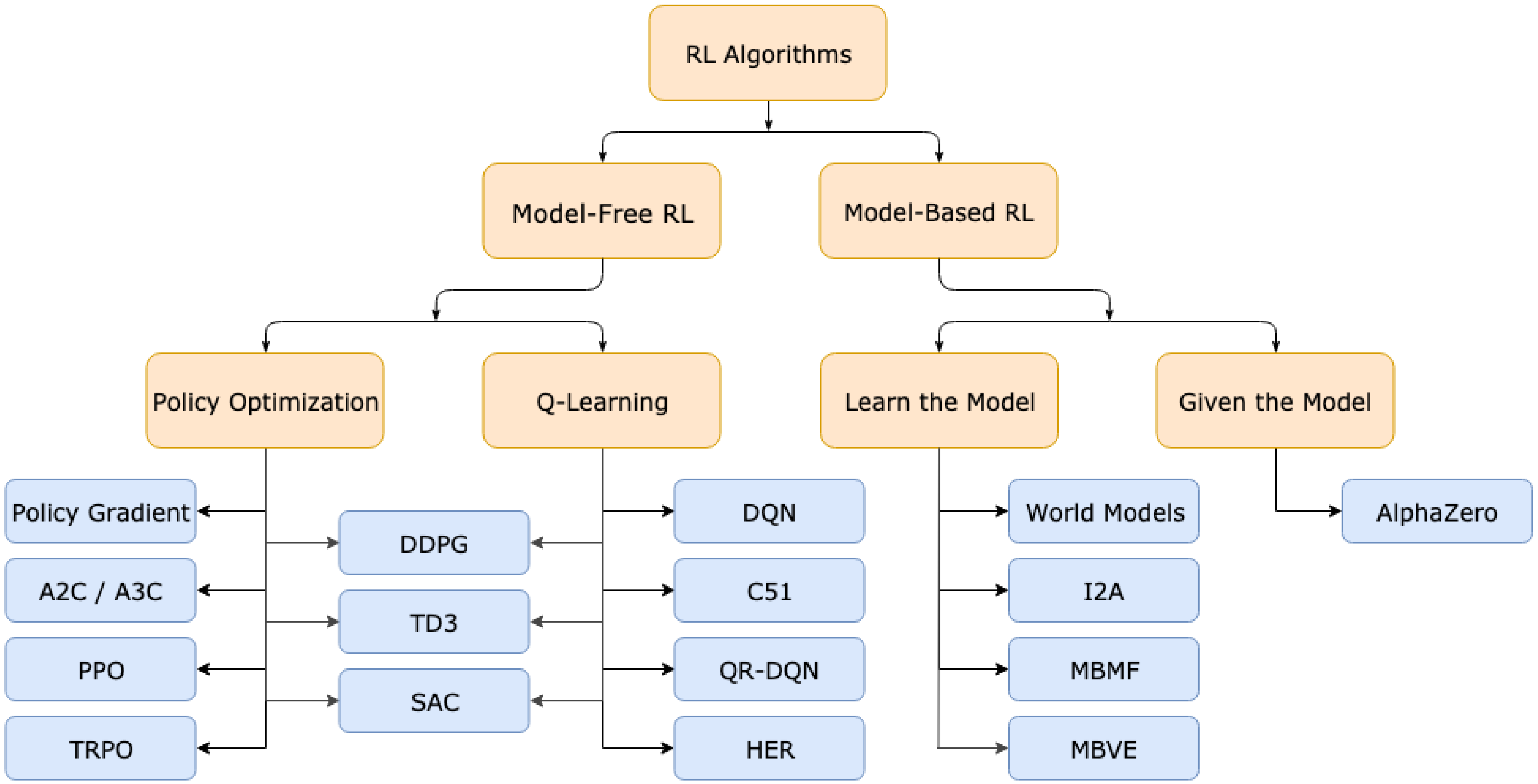
Slide sequence credit: Levine UC Berkeley Course

The four main types of DRL algorithms

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

1. **Policy gradients**: directly differentiate the objective; perform gradient ascent on parameterized policy
2. **Value-based**: no explicit policy representation; instead estimate Q or value function of the optimal policy
3. **Actor-critic**: estimate Q or value function of current policy; use it to improve policy
4. **Model-based**: learn parameterized representation of state transition model, use it either for planning or to improve a policy (e.g., via synthetic experiences)

Taxonomy of DRL Approaches



https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Extensions

- There are numerous extensions to the basic RL formulation that may be used to address operational needs:

- **Transferability**:
  - **Transfer learning**: fine-tuning
  - **Multi-task RL**: learn multiple tasks at once
  - **Meta-learning**: learn to learn
- **Unknown reward function**:
  - **Inverse RL**: learn reward function
- **Learning from static data**:
  - **Offline RL**: learn without a simulator
- **Safety Concerns**:
  - **Constrained RL**: constraint on objective (e.g. Lagrangian formulation)
  - **Distributional RL**: carry distribution through value-based methods

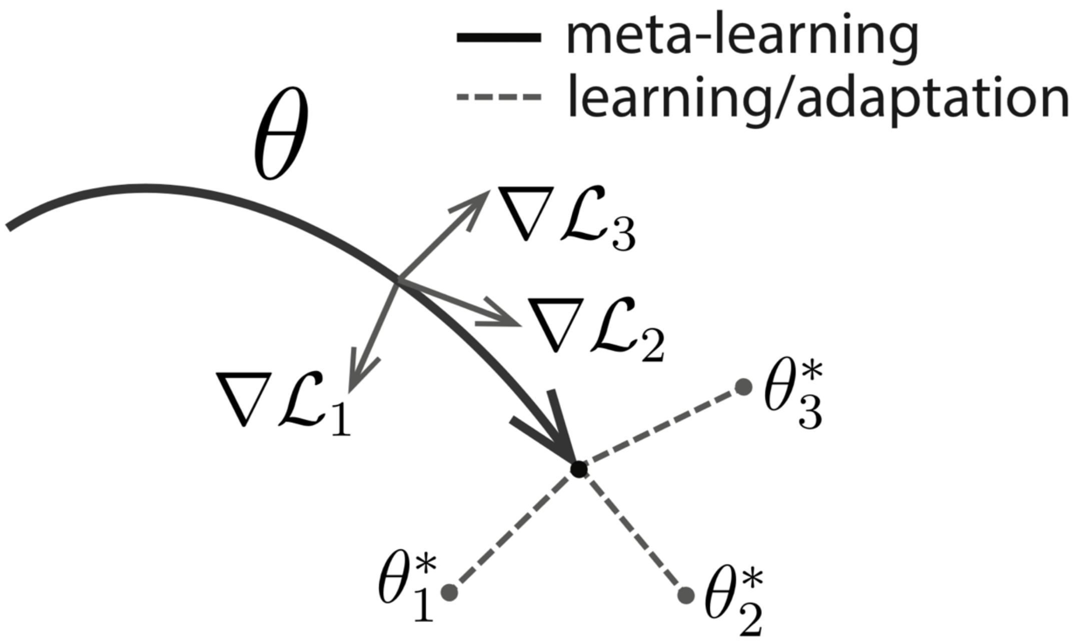


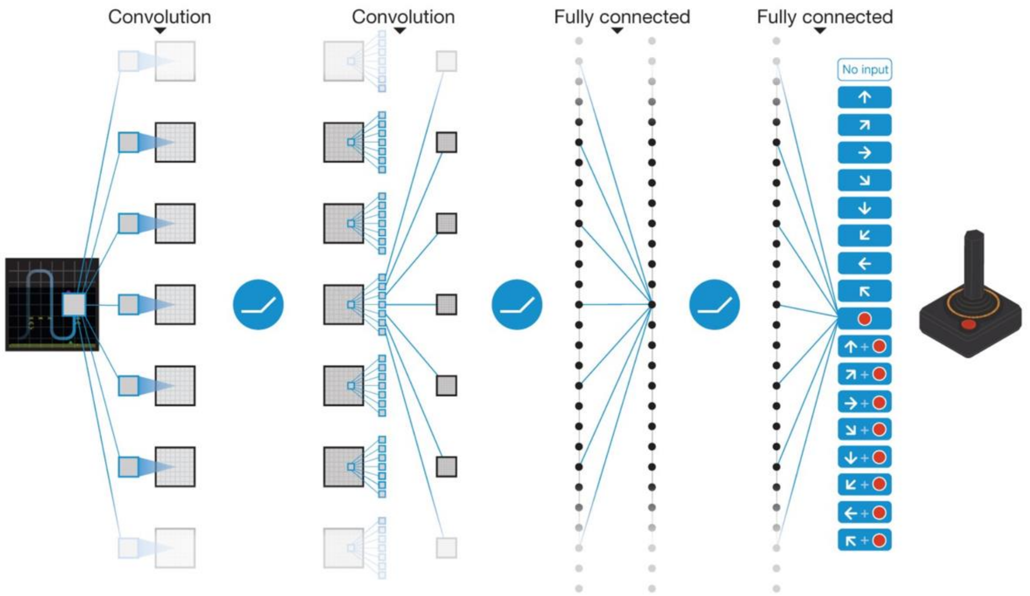
Image credit: C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *arXiv:1703.03400v3* (2017).

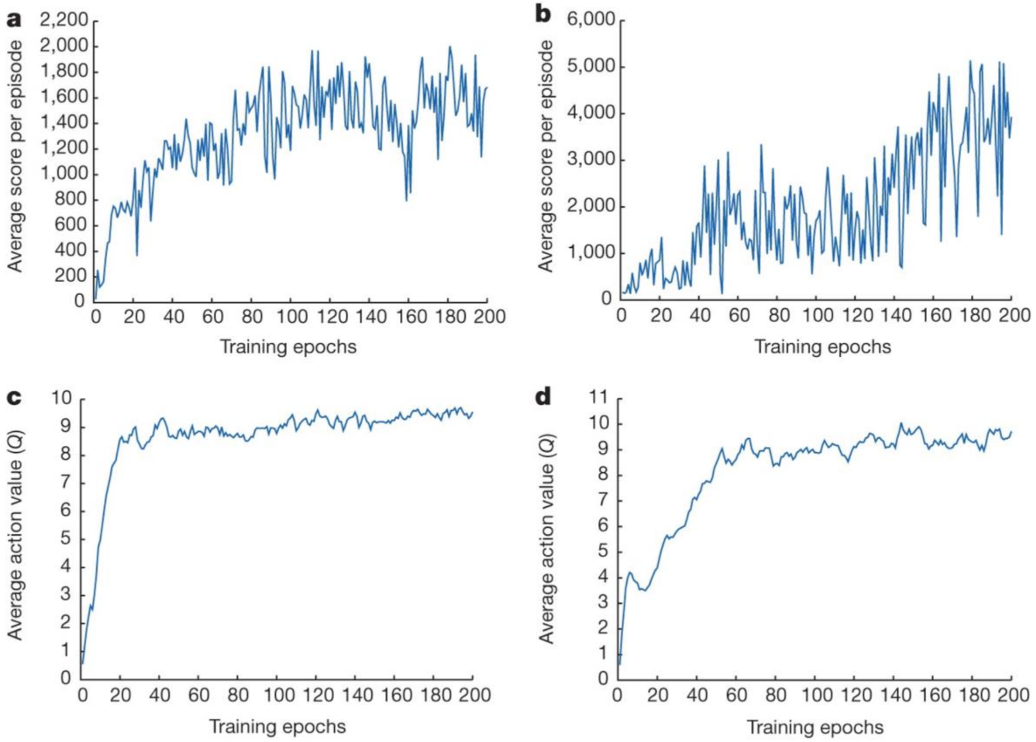
Canonical Examples of DRL

- Deep Q-learning reaches human level on Atari suite
- AlphaZero masters Chess, Shogi, and Go
- AlphaStar bests human professionals in StarCraft
- OpenAI programs robotic hand to solve Rubik’s Cube
- RL for tuning Large Language Models (RL from Human Feedback)

Deep Q Learning for Atari

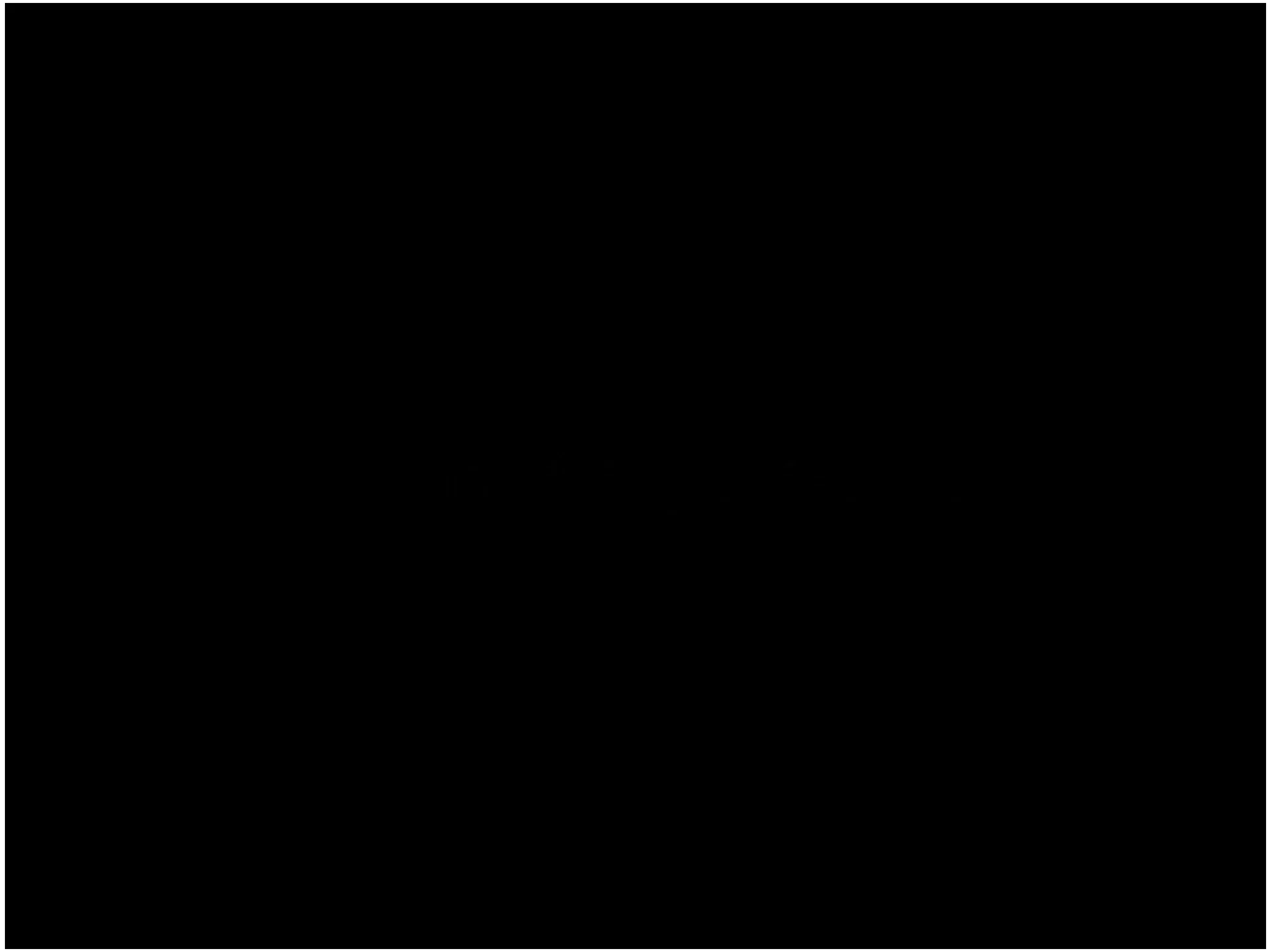
V. Mnih et al. Human-level control through deep reinforcement learning. *Nature* **518**, p. 529–533 (2015).





The level of professional humans is met on 49 of 57 Atari games with the same network architecture, learning strategy, and hyperparameters.

DQN Learning Breakout



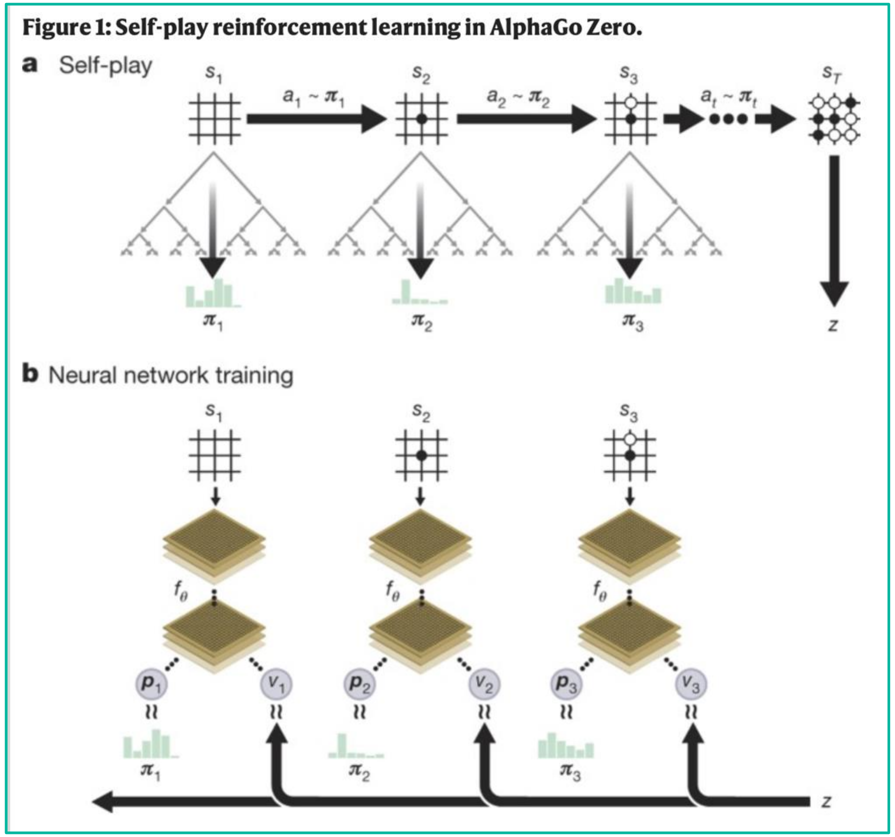
AlphaGo Zero, AlphaZero: Policy-Guided Monte Carlo Tree Search

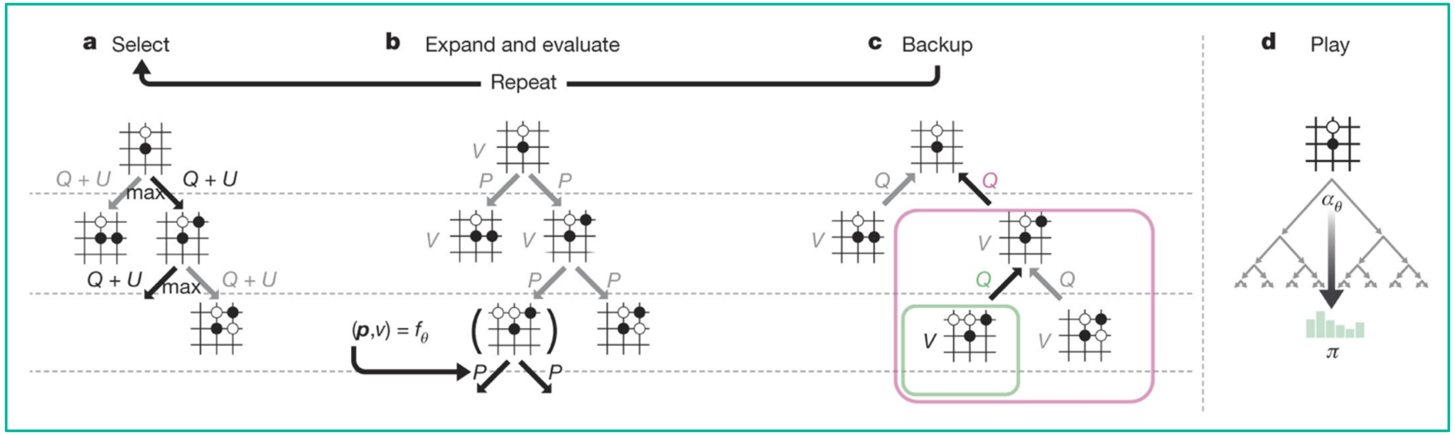
"Plays like a human on fire"

$$(\mathbf{p}, \mathbf{v}) = f_\theta(\mathbf{s})$$

$$l = (z - v)^2 - \pi^{\mathbf{T}} \log \mathbf{p} + \mathbf{c}||\theta||^{\mathbf{2}}$$

$$a_t = \operatorname{argmax}_a \left(Q(\mathbf{s}, \mathbf{a}) + C(\mathbf{s}) P(\mathbf{s}, a) \frac{\sqrt{N(\mathbf{s})}}{1 + N(\mathbf{s}, a)} \right)$$





- D. Silver et al. Mastering the game of Go without human knowledge. *Nature* **550**, p 354–359 (2017).
- D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* Vol. 362 Issue 6419, p 1140-1144 (2019).

AlphaStar

A “Grand Challenge” for AI Research

O. Vinyals et al. Grandmaster level in Starcraft II using multi-agent reinforcement learning. *Nature*, **575**, 350-354 (2019).

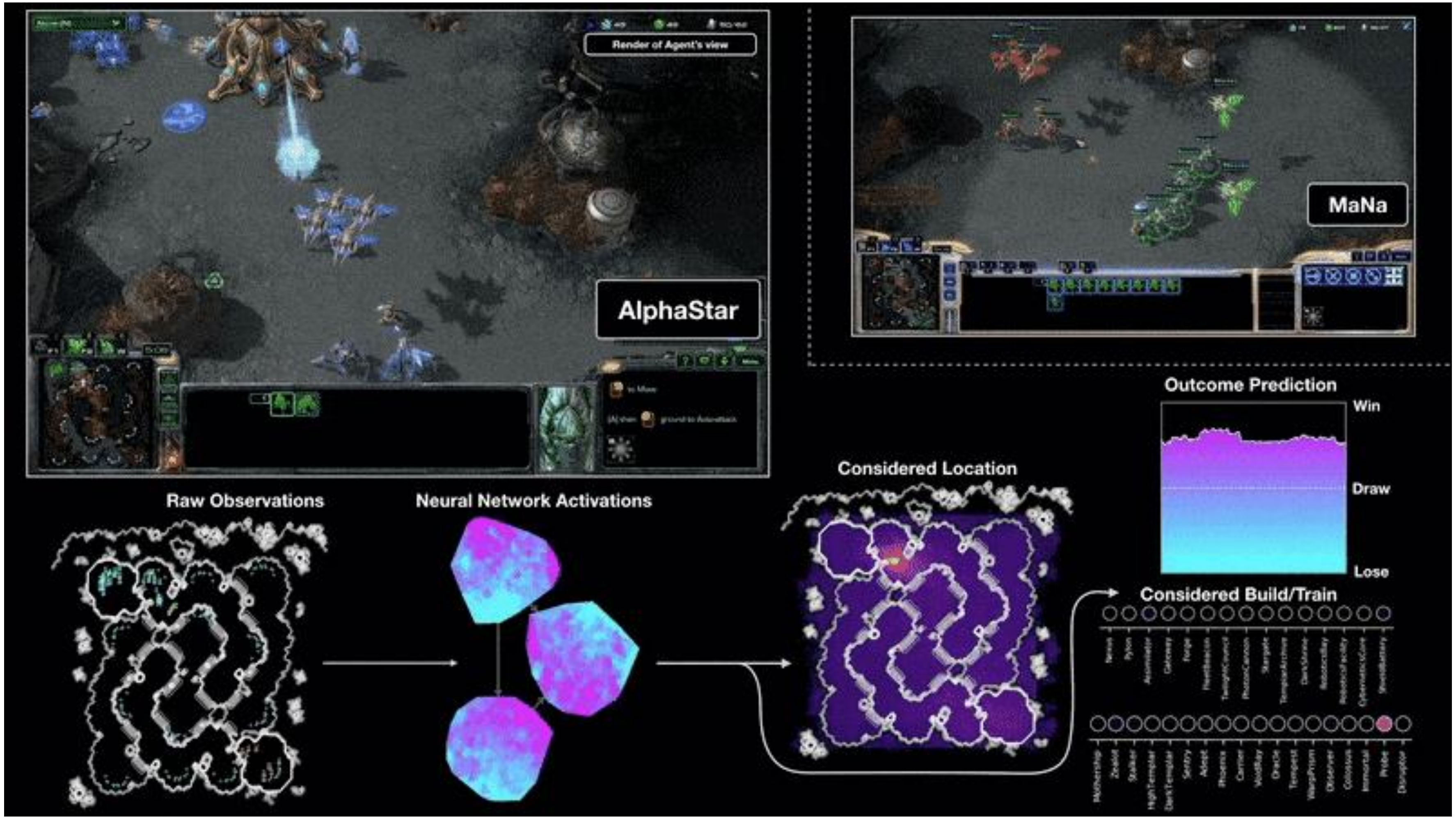
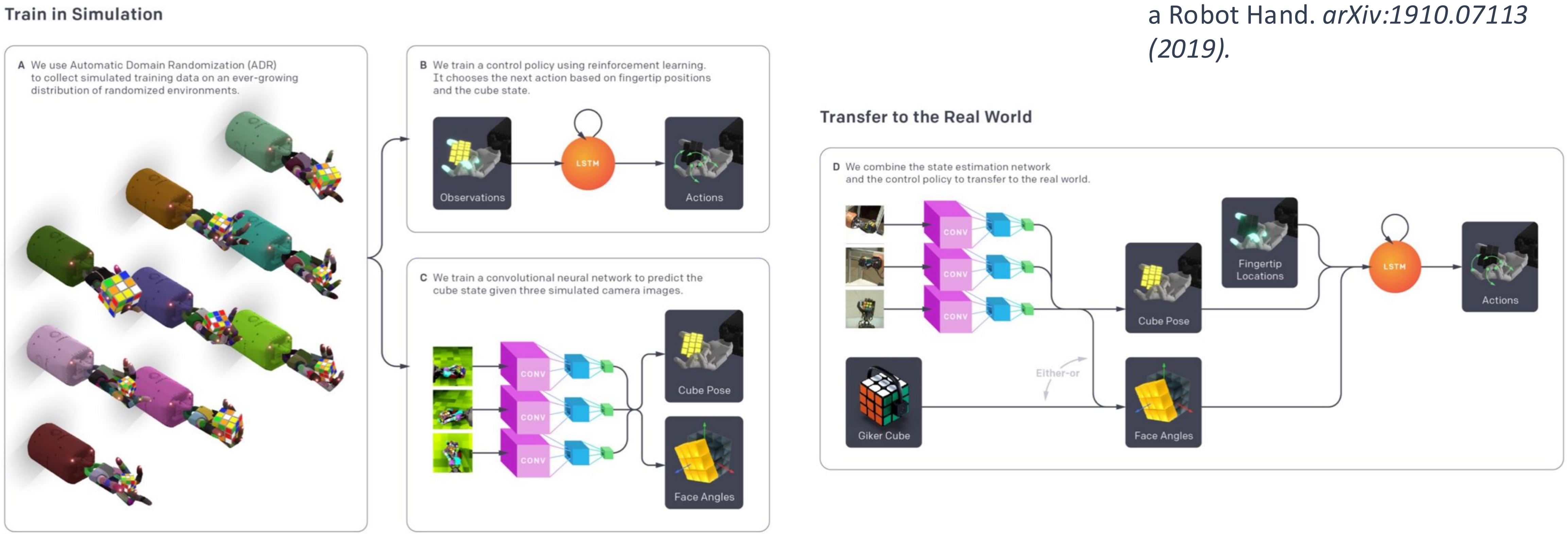


Image credit : [DeepMind Blog]

OpenAI: Solving a Rubik's Cube with a Robotic Hand

Akkaya et al. Solving Rubik's Cube with a Robot Hand. *arXiv:1910.07113 (2019).*



In many applications, transferring from simulation to reality is the dominant challenge.

Credit: Hugging Face

RL for Fine-Tuning Large Language Models

Reinforcement Learning from Human Feedback (RLHF)

LE

Can you explain to me how RL with human feedback works for language models like I am five?

Sure! So, imagine you have a robot named Rufus who wants to learn how to talk like a human. Rufus has a language model that helps him understand words and sentences.

First, Rufus will say something using his language model. For example, he might say "I am a robot."

Then, a human will listen to what Rufus said and give him feedback on whether it sounded like a natural sentence a human would say. The human might say, "That's not quite right, Rufus. Humans don't usually say 'I am a robot.' They might say 'I'm a robot' or 'I am a machine.'"

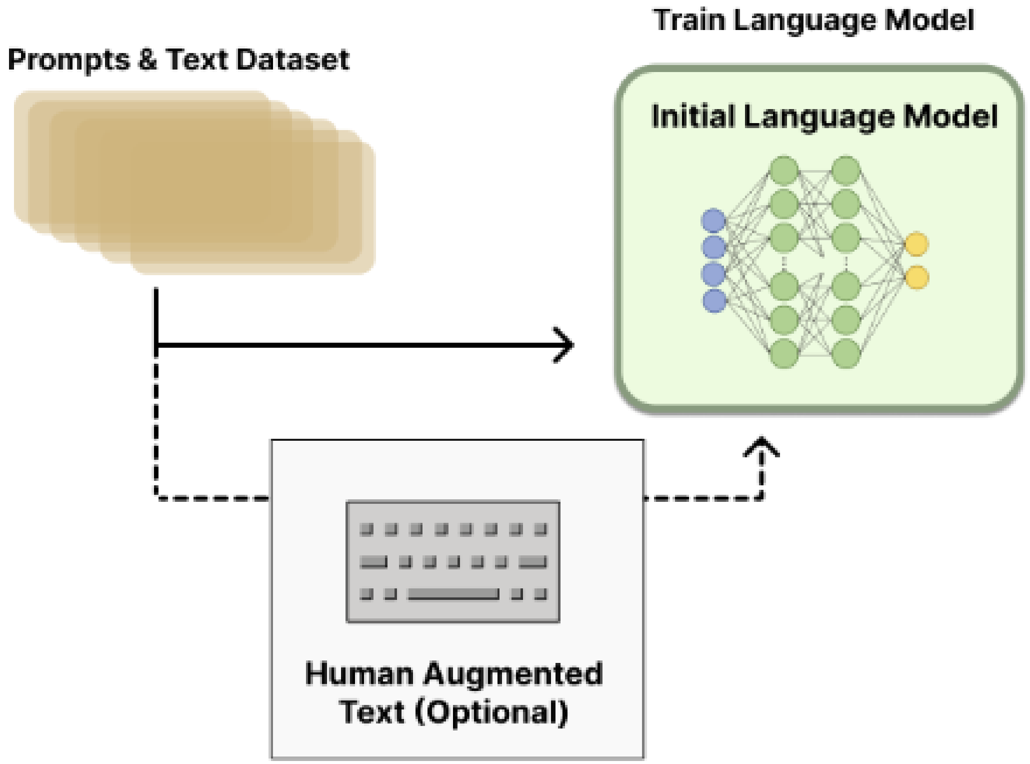
Rufus will take this feedback and use it to update his language model. He will try to say the sentence again, using the new information he received from the human. This time, he might say "I'm a robot."

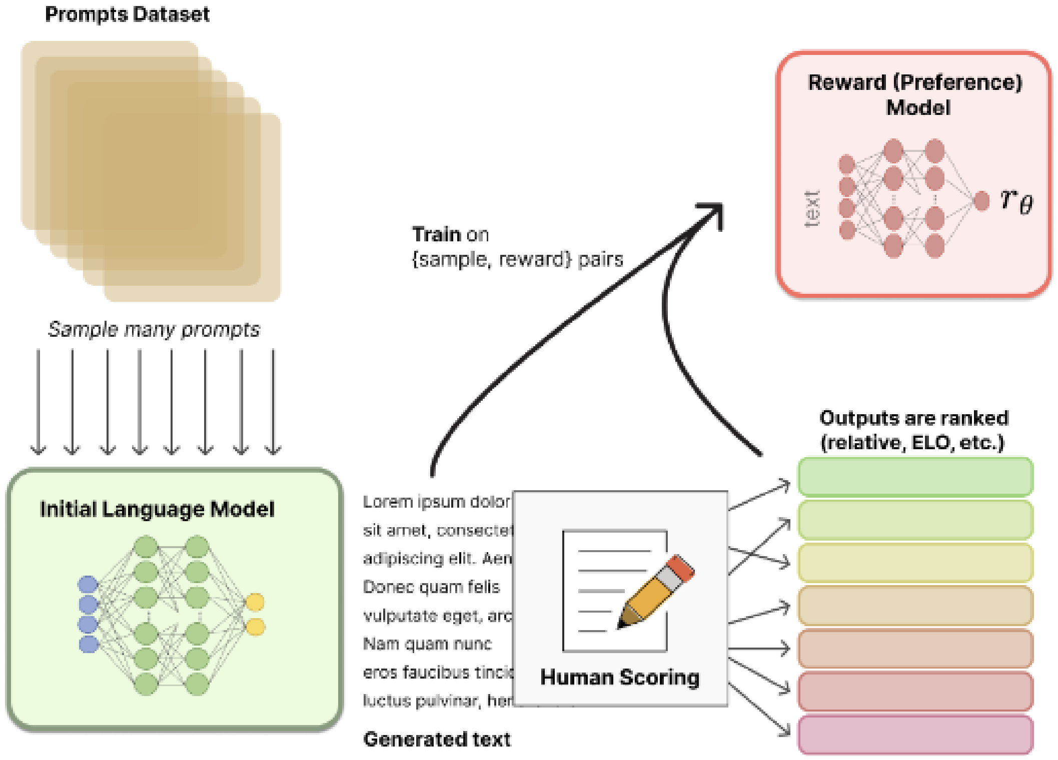
The human will listen again and give Rufus more feedback. This process will continue until Rufus can say sentences that sound natural to a human.

Over time, Rufus will learn how to talk like a human thanks to the feedback he receives from humans. This is how language models can be improved using RL with human feedback.

RL for Fine-Tuning Large Language Models

Reinforcement Learning from Human Feedback (RLHF)

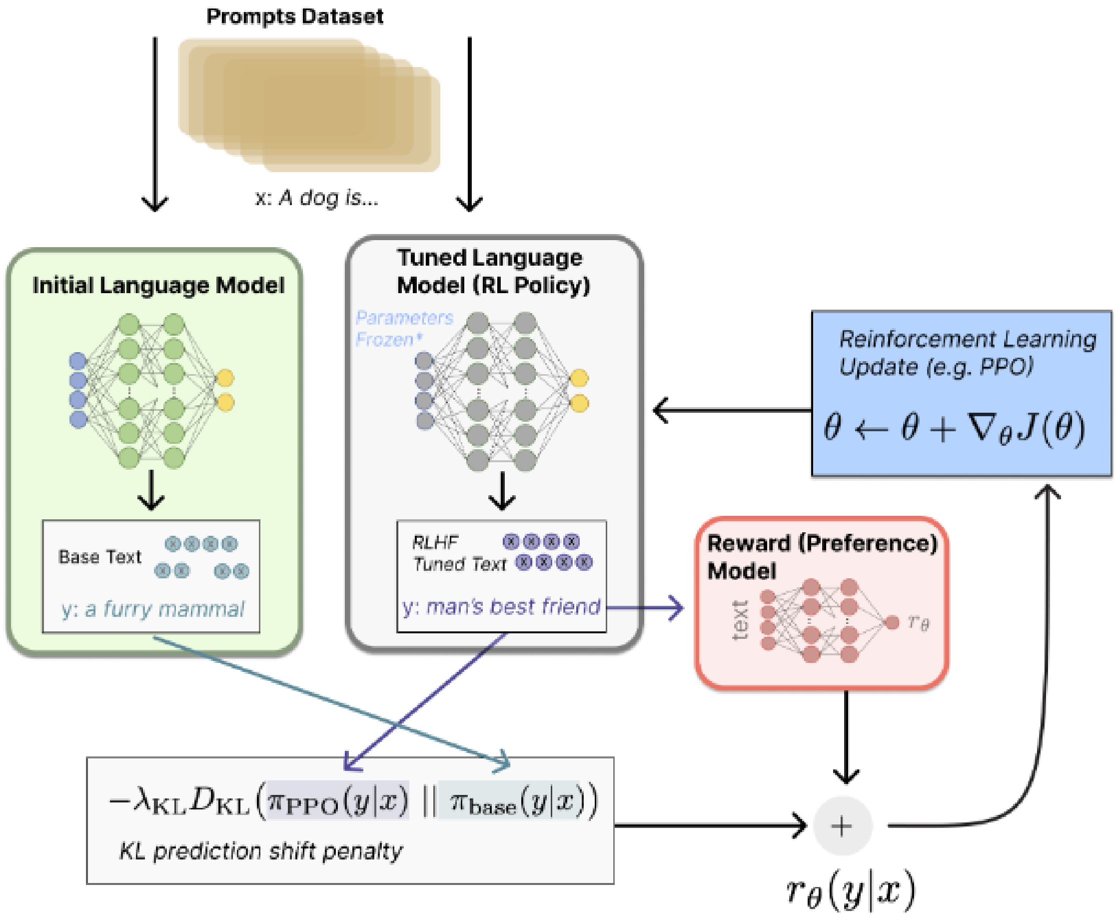




Credit: Hugging Face

RL for Fine-Tuning Large Language Models

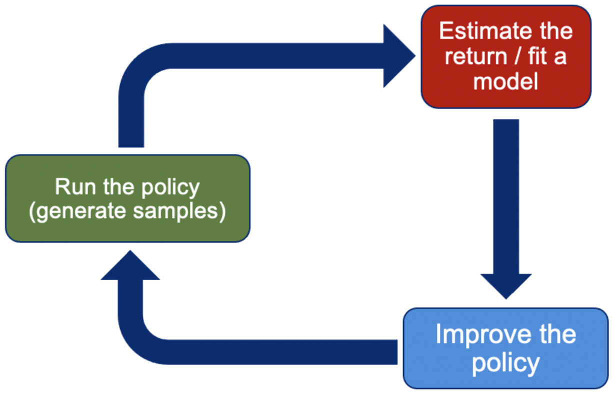
Reinforcement Learning from Human Feedback (RLHF)



Implementation Tips 1

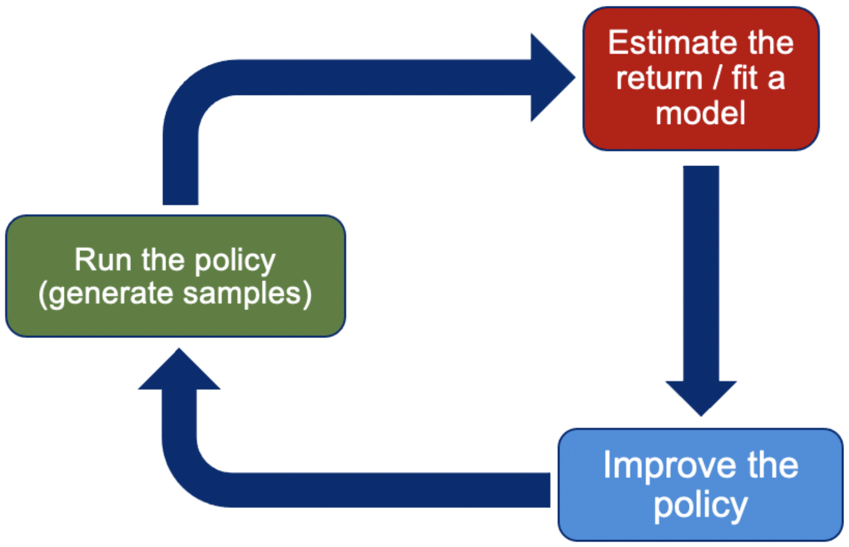
DRL Implementation

- Requirements
- Basic components
- A few words about parallelization
- Open-source code



What is required upfront?

- **Sufficient compute**
  - Typically want a lot (10^1 – 10^3) of parallel cores
  - GPUs useful for larger network architectures
  - Permission, ability to run deep learning software
  - Will need to iterate- no one trains their best agent on the first try
- **Fast simulation** that appropriately captures system dynamics
  - Level of realism required depends on application
  - Speed required depends on compute; usually need faster than real-time
  - Need a clear picture of how the agent will interact with the rest of the system
- **Time** to iterate, test
  - Less of a well-established recipe than other types of ML
  - Lots of factors impact agent performance
  - Challenging to bridge sim-to-real gap



IDEs / Editors

- Obviously feel free to use whatever you’re comfortable with.
- It has a particularly accessible debugger, which I have found useful.

Required Software Components

- **Environment**
  - Usually follows OpenAI Gym API
  - Contains simulation of the world the agent is interacting with
- **Agent**
  - Contains decision-making code, including learning scheme
- **Network/Model**
  - Optimized and called by the agent for decision making
- **Utilities ("Other")**
  - Whatever helper functions are needed

A few words about parallelization

- DRL can be computationally expensive; parallelization can help.
- While GPUs can be useful, many times you can get away with CPU workers
  - The networks are not typically as large as in supervised learning
- There are numerous ways to parallelize DRL code
  - MPI, multiprocessing, pickle passing, Dask, Ray, etc.
- These approaches exhibit tradeoffs in terms of flexibility, transparency, robustness, scalability, and applicability to different compute resources.
- MPI is probably the best first thing to try, as it scores highly in all categories.

➢ **The best option depends on the particular project.**

Open-Source Code

- As mentioned before, it can be useful to write your own code.
- However, in many cases you can save time by using at least some open source components. If you're careful, this can be the best route.
- Good open source options include
  - OpenAI Spinning Up (https://github.com/openai/spinningup)
  - Stable-baselines3 (https://github.com/DLR-RM/stable-baselines3/)
  - RLLib (https://docs.ray.io/en/latest/rllib/index.html)
  - Unity ml-agents (https://github.com/Unity-Technologies/ml-agents)
- While the problem sets in this course will require you to work with the class codebase (to which you will contribute!), you are free to use whatever you want on your project.

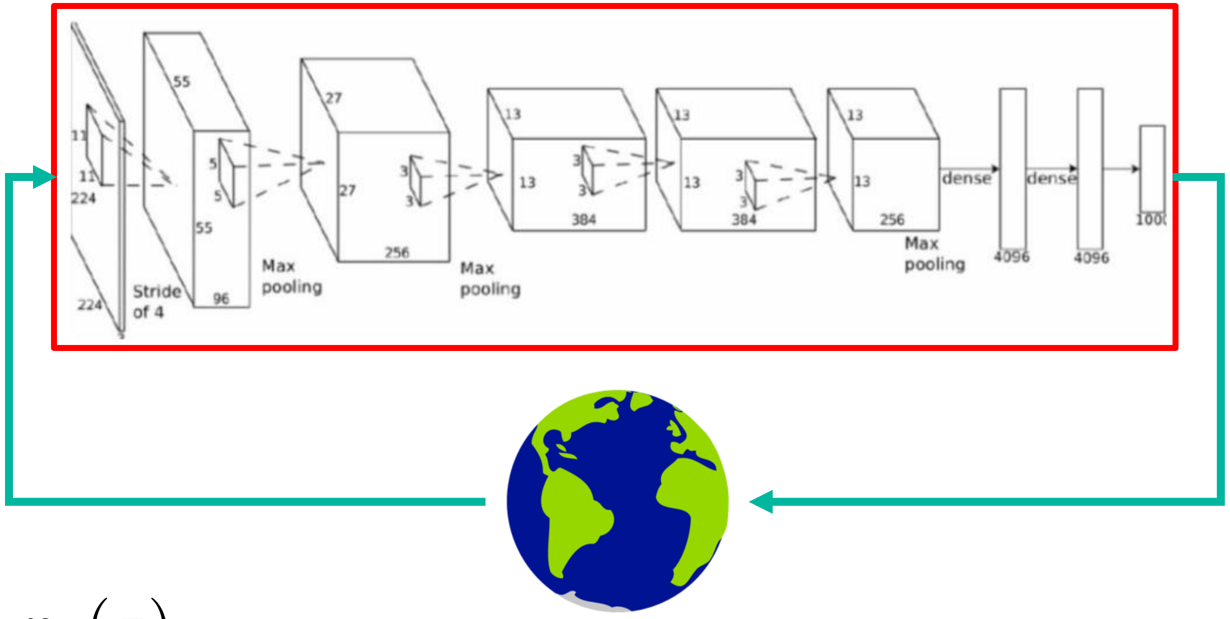
➢ **The best option depends on the particular project.**

Learning a Policy

- The RL Objective
- Finite vs. Infinite Horizon

Learning a Policy

$$\pi_\theta(\mathbf{a_t}|\mathbf{s_t})$$



$\mathbf{s_t}, \mathbf{r_t}$

$\mathbf{a_t}$

Fully observable (for now)

$$\overbrace{p_\theta(\mathbf{s_1}, \mathbf{a_1}, \ldots, \mathbf{s_T}, \mathbf{a_T})}^{p_\theta(\tau)} = p(\mathbf{s_1}) \prod_{t=1}^{T} \pi_\theta(\mathbf{a_t}|\mathbf{s_t}) \, p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})$$

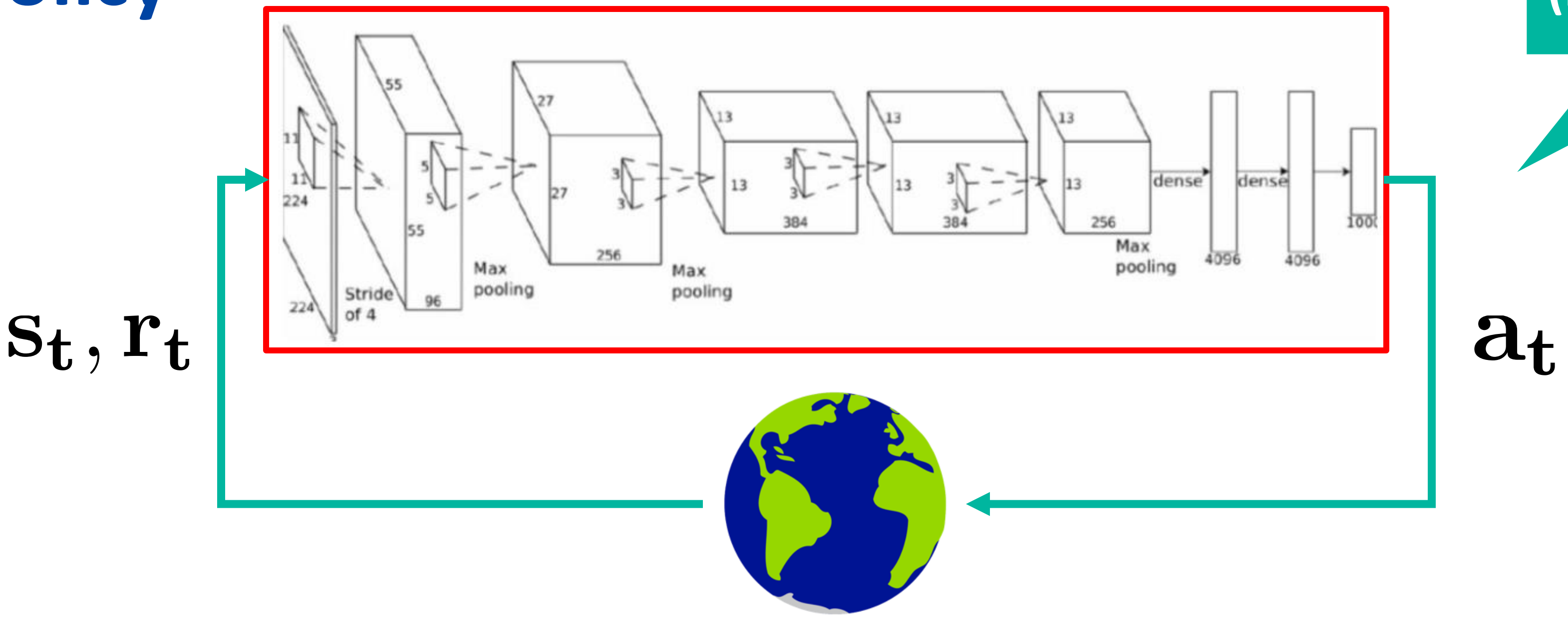
$$\text{Goal: find } \theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s_t}, \mathbf{a_t}) \right]$$

Undiscounted (for now)

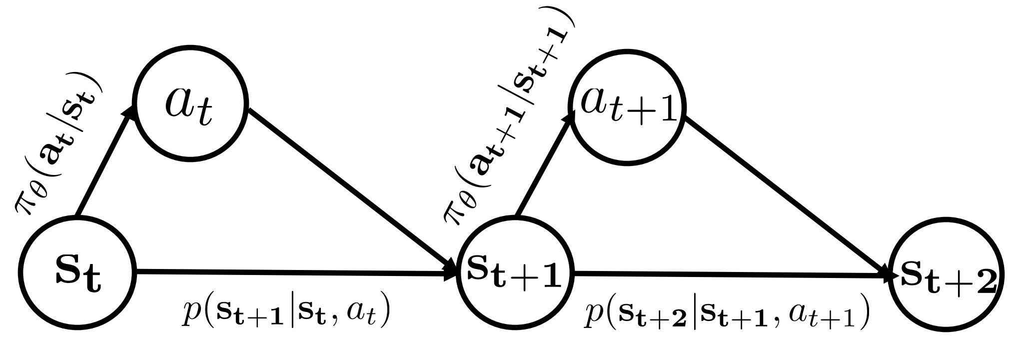
Learning a Policy

Fully observable (for now)

$$\pi_\theta(\mathbf{a_t}|\mathbf{s_t})$$



$$p_\theta(\mathbf{s_1}, \mathbf{a_1}, \ldots, \mathbf{s_T}, \mathbf{a_T}) = p(\mathbf{s_1}) \prod_{t=1}^{T} \pi_\theta(\mathbf{a_t}|\mathbf{s_t})\, p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})$$

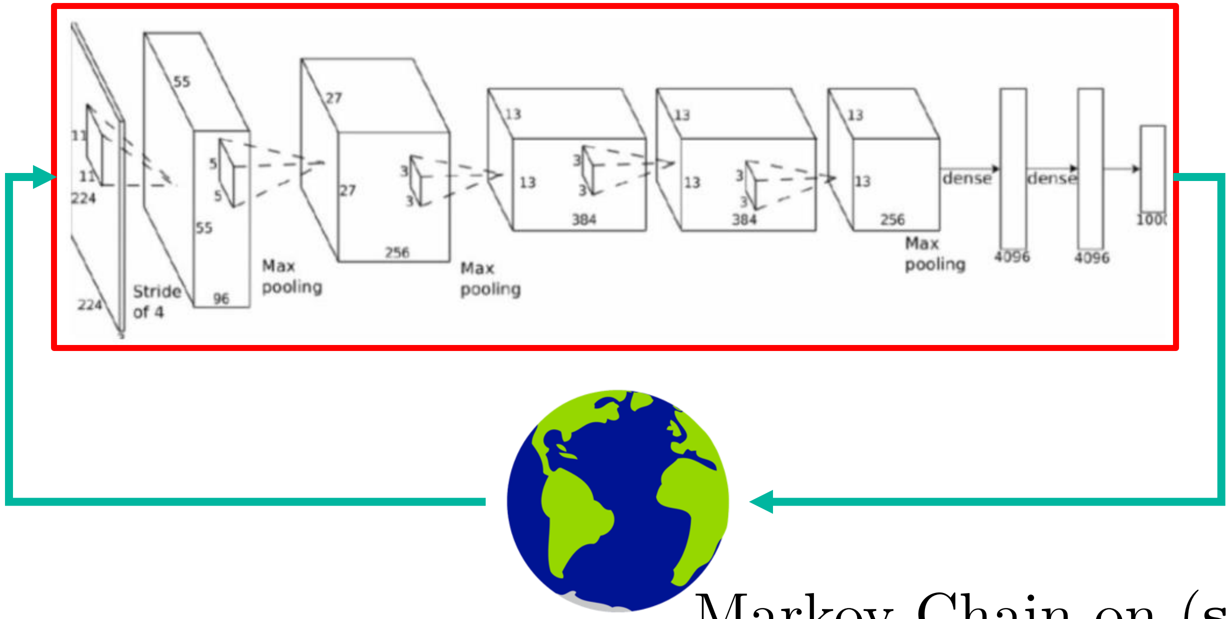


Learning a Policy

Fully observable (for now)

$$\pi_\theta(\mathbf{a_t}|\mathbf{s_t})$$

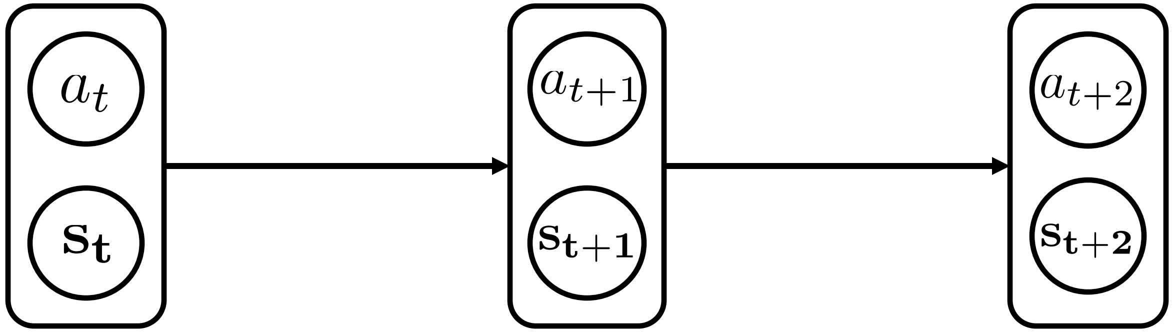
$\mathbf{s_t}, \mathbf{r_t}$ $\mathbf{a_t}$



Markov Chain on $(\mathbf{s}, \mathbf{a})$

$$p_\theta(\mathbf{s_1}, \mathbf{a_1}, \ldots, \mathbf{s_T}, \mathbf{a_T}) = p(\mathbf{s_1}) \prod_{t=1}^{T} \underbrace{\pi_\theta(\mathbf{a_t}|\mathbf{s_t})\, p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})}$$

$$p(\mathbf{s_{t+1}}, \mathbf{a_{t+1}}|\mathbf{s_t}, \mathbf{a_t}) = p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})\pi_\theta(\mathbf{a_{t+1}}|\mathbf{s_{t+1}})$$



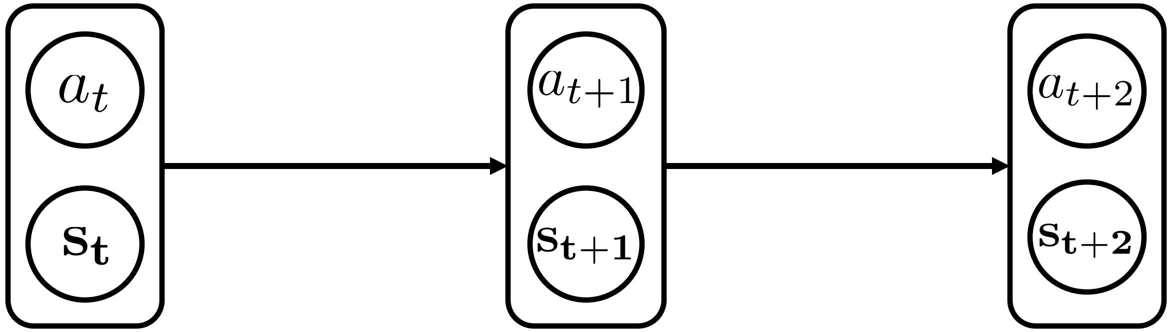
Finite Horizon Case

Undiscounted (for now)

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s_t},\mathbf{a_t})\right]$$

$$= \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s_t},\mathbf{a_t})\sim p_\theta(\mathbf{s_t},\mathbf{a_t})}\left[r(\mathbf{s_t},\mathbf{a_t})\right] \longrightarrow p_\theta(\mathbf{s_t},\mathbf{a_t}) \text{ is the state-action marginal}$$

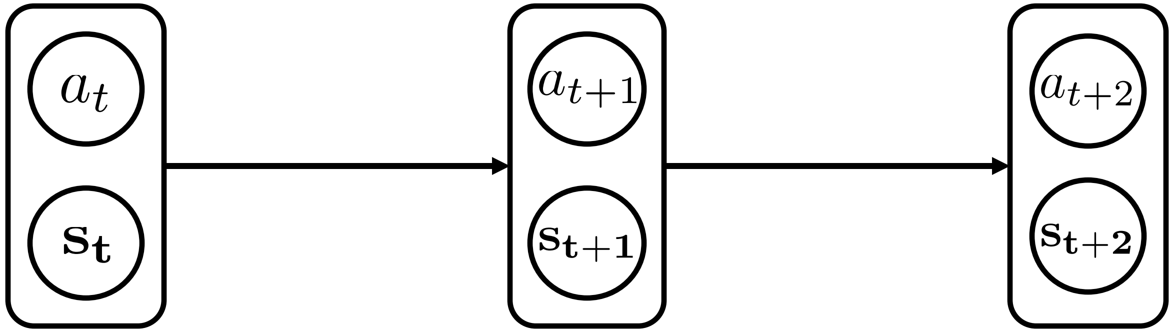
$$p(\mathbf{s_{t+1}},\mathbf{a_{t+1}}|\mathbf{s_t},\mathbf{a_t}) = p(\mathbf{s_{t+1}}|\mathbf{s_t},\mathbf{a_t})\pi_\theta(\mathbf{a_{t+1}}|\mathbf{s_{t+1}})$$



Infinite Horizon Case (Stationary Distribution)

$$\theta^\star = \arg\max_\theta \frac{1}{T}\sum_{t=1}^{T} E_{(\mathbf{s_t},\mathbf{a_t})\sim p_\theta(\mathbf{s_t},\mathbf{a_t})}[r(\mathbf{s_t},\mathbf{a_t})]$$

What if T $\longrightarrow \infty$?

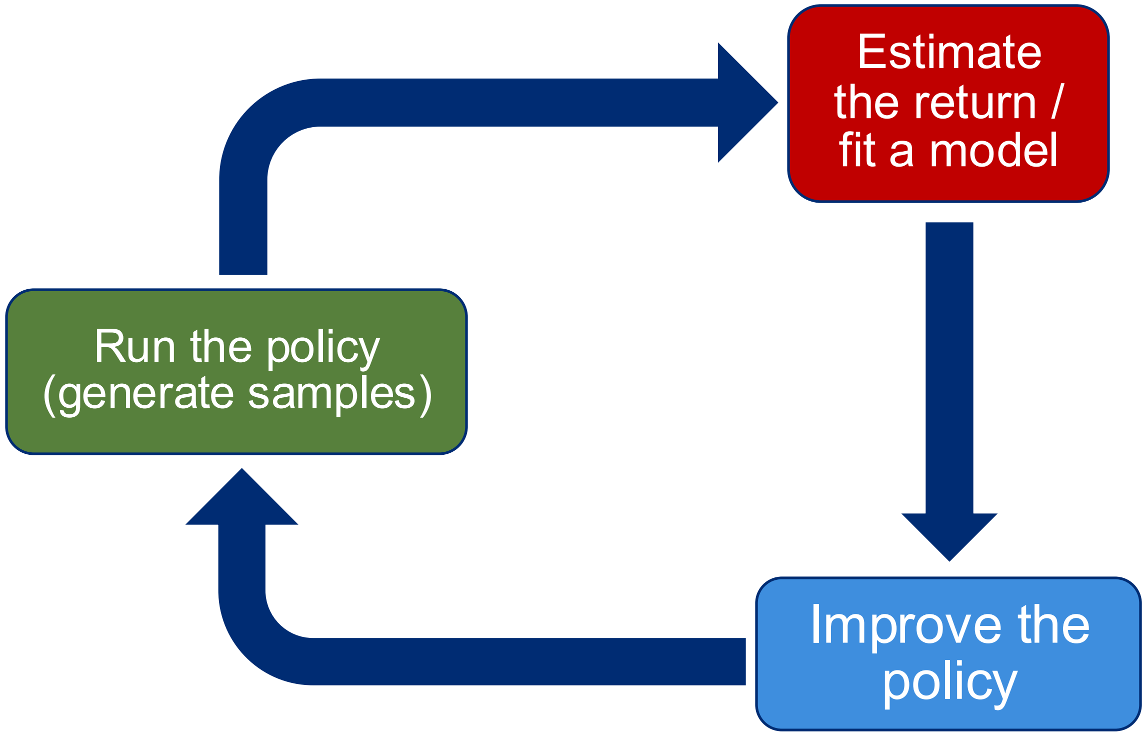


$$\begin{pmatrix}\mathbf{s_{t+1}}\\ \mathbf{a_{t+1}}\end{pmatrix} = \mathcal{T}\begin{pmatrix}\mathbf{s_t}\\ \mathbf{a_t}\end{pmatrix} \longrightarrow \begin{pmatrix}\mathbf{s_{t+k}}\\ \mathbf{a_{t+k}}\end{pmatrix} = \mathcal{T}^k\begin{pmatrix}\mathbf{s_t}\\ \mathbf{a_t}\end{pmatrix}$$

If ergodic, as $T \to \infty$:

$p_\theta(\mathbf{s_t},\mathbf{a_t})$ converges to a *stationary* distribution μ

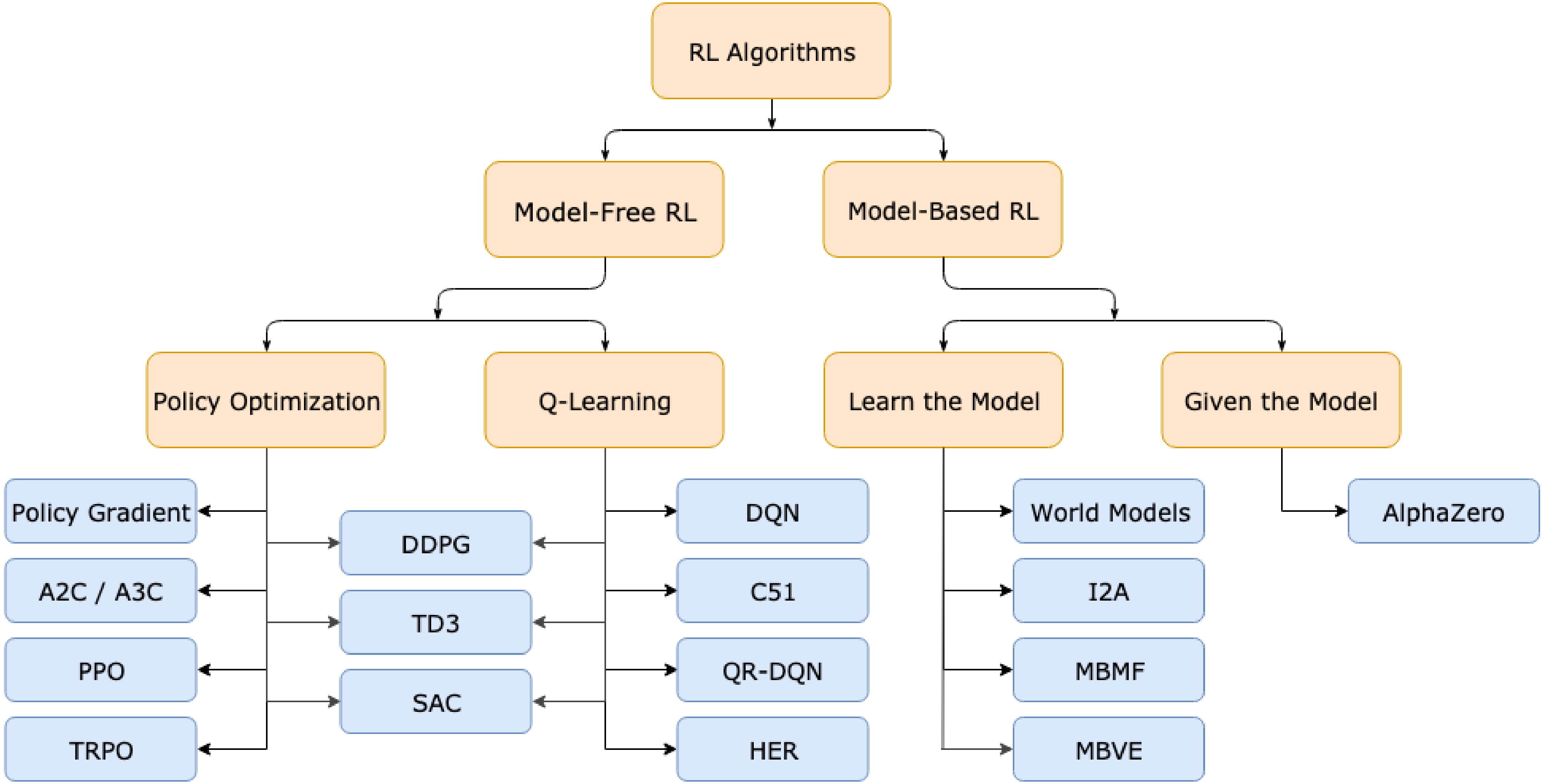
$\mu = \mathcal{T}\mu \longrightarrow (\mathcal{T} - \mathbf{I})\mu = 0 \longrightarrow \mu$ is an eigenvector of $\mathcal{T}$ with eigenvalue 1.



Structure of a DRL Algorithm

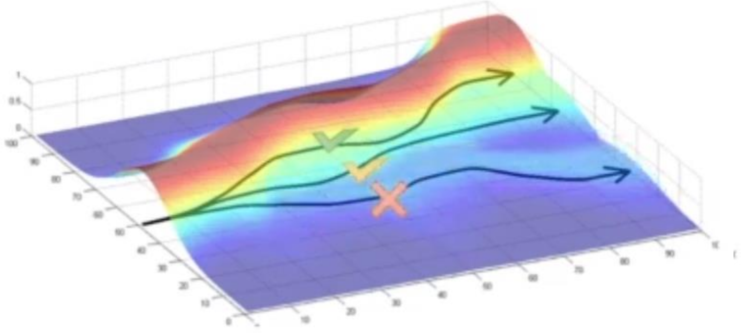
- Sample Generation
- Return Estimation / Model Fitting
- Policy Improvement

Algorithms for Learning Policies

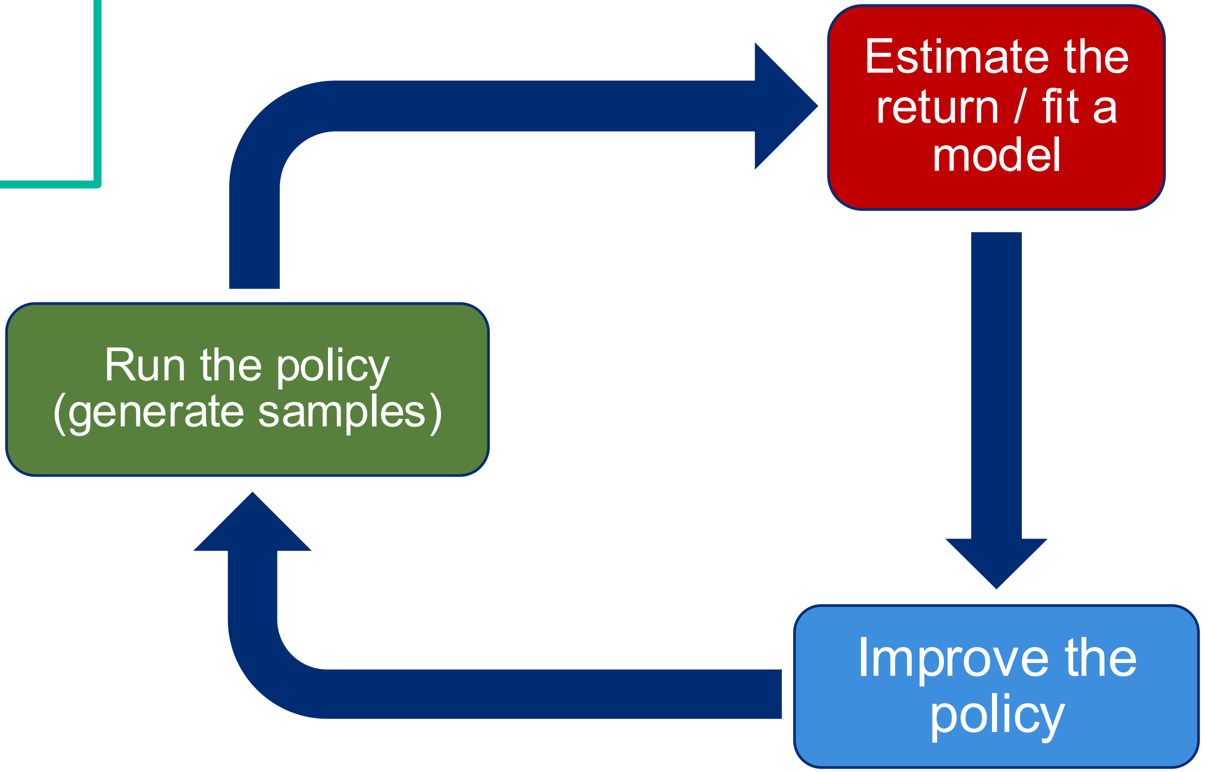


https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

General RL Learning Cycle



Key consideration: which part(s) are expensive?



$$J(\theta) = E_\pi\left[\sum_t r_t\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r_t^i$$

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

Slide sequence credit: Levine UC Berkeley Course

Expectations and Recursion

$$E_{\mathcal{T}\sim p_\theta(\mathcal{T})}\left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

$$E_{\mathbf{s_1}\sim p(\mathbf{s_1})}$$

$$Q(\mathbf{s}_1, \mathbf{a}_1) := r(\mathbf{s}_1, \mathbf{a}_1) + \underbrace{E_{\mathbf{s}_2\sim p(\mathbf{s}_2|\mathbf{s}_1,\mathbf{a}_1)}\left[E_{\mathbf{a}_2\sim\pi(\mathbf{a}_2|\mathbf{s}_2)}\left[r(\mathbf{s}_2, \mathbf{a}_2) + \ldots|\mathbf{s}_2\right]|\mathbf{s}_1, \mathbf{a}_1\right]}$$

$$E_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right] = E_{\mathbf{s}_1\sim p(\mathbf{s}_1)}\left[E_{\mathbf{a}_1\sim\pi(\mathbf{a}_1|\mathbf{s}_1)}\left[Q(\mathbf{s}_1, \mathbf{a}_1)|\mathbf{s}_1\right]\right]$$

$\longrightarrow$ We can improve $\pi_\theta(\mathbf{a}_1, \mathbf{s}_1)$ if we know $Q(\mathbf{s}_1, \mathbf{a}_1)$.

Definitions: Q and V

Undiscounted (for now)

Q-function:

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t\right] : \text{ total expected future reward from taking } \mathbf{a}_t \text{ in } \mathbf{s}_t$$

Value function:

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

$$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)}\left[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)\right]$$

$E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)}\left[V^\pi(\mathbf{s}_1)\right]$ is the RL objective!

Using Q-functions and Value Functions

Method 1 (**Q-learning**): Improve policy by taking action with highest Q value and continuously refining Q estimate.

Method 2 (**Actor-Critic**): Continuously update policy to increase the probability of taking good actions, where "good" is defined as $Q^\pi(\mathbf{s}, \mathbf{a}) > V^\pi(\mathbf{s})$

➢ **We will come back to these!**

Types of RL Algorithms

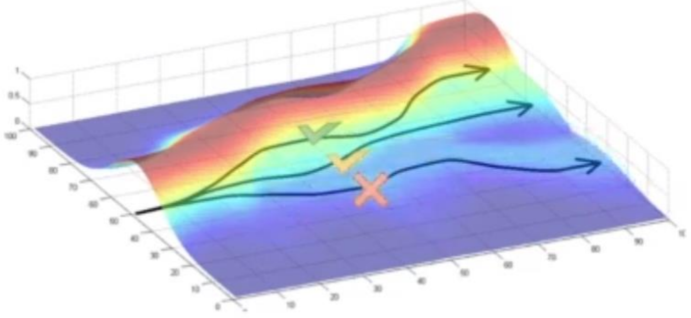
- Considerations for when to use what

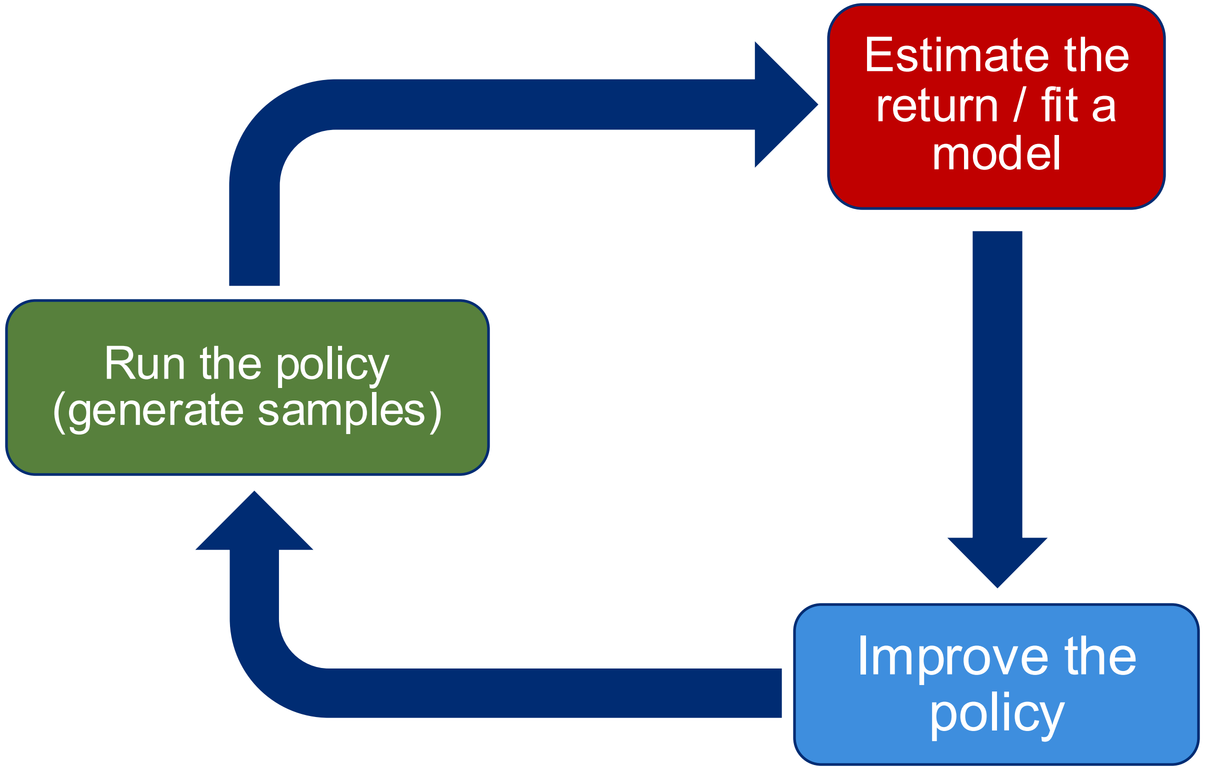
The Four Main Types of RL Algorithms

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

1. **Policy gradients**: directly differentiate the objective; perform gradient ascent on parameterized policy
2. **Value-based**: no explicit policy representation; instead estimate Q or value function of the optimal policy
3. **Actor-critic**: estimate Q or value function of current policy; use it to improve policy
4. **Model-based**: learn parameterized representation of state transition model, use it either for planning or to improve a policy (e.g. via synthetic experiences)

Policy Gradient





$$J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r_t\right]$$

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

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

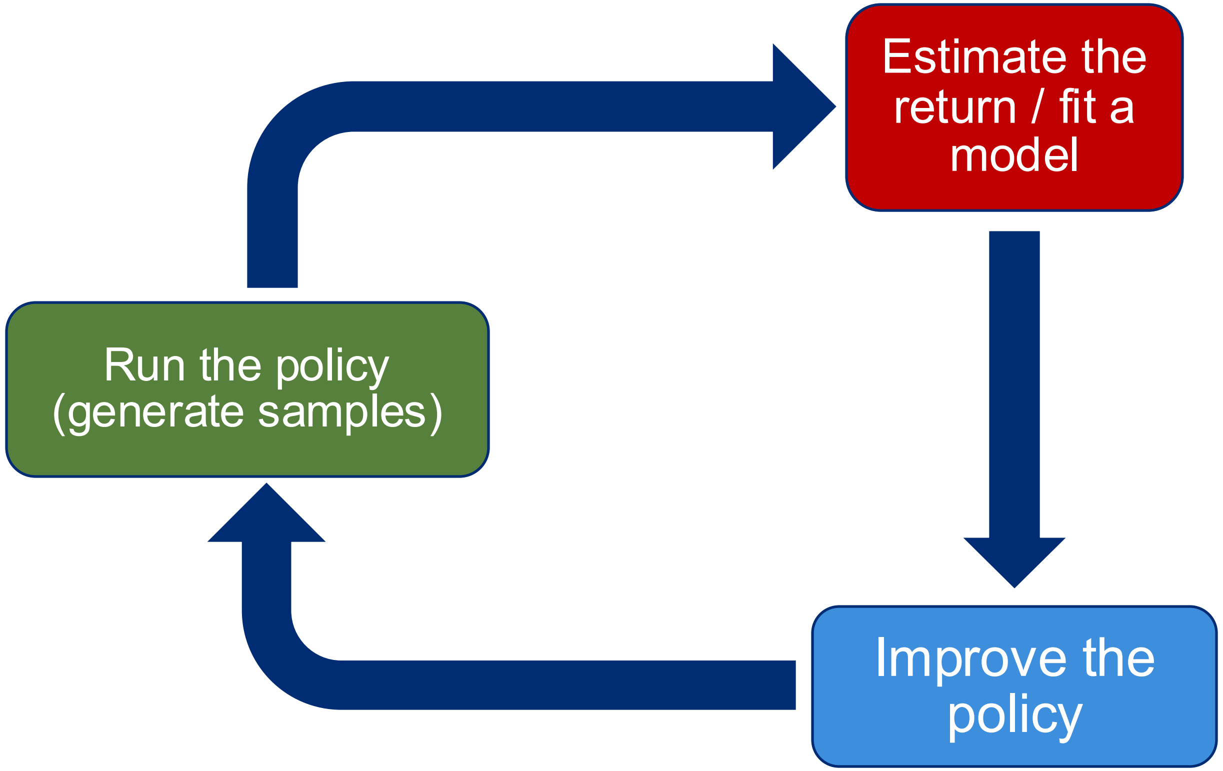
$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

- **Yields unbiased but high-variance estimates of gradients**
- **Variance reduction measures required for practical application**

Value-Based Algorithms

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t\right] \text{ : total expected future reward from taking } \mathbf{a}_t \text{ in } \mathbf{s}_t$$

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

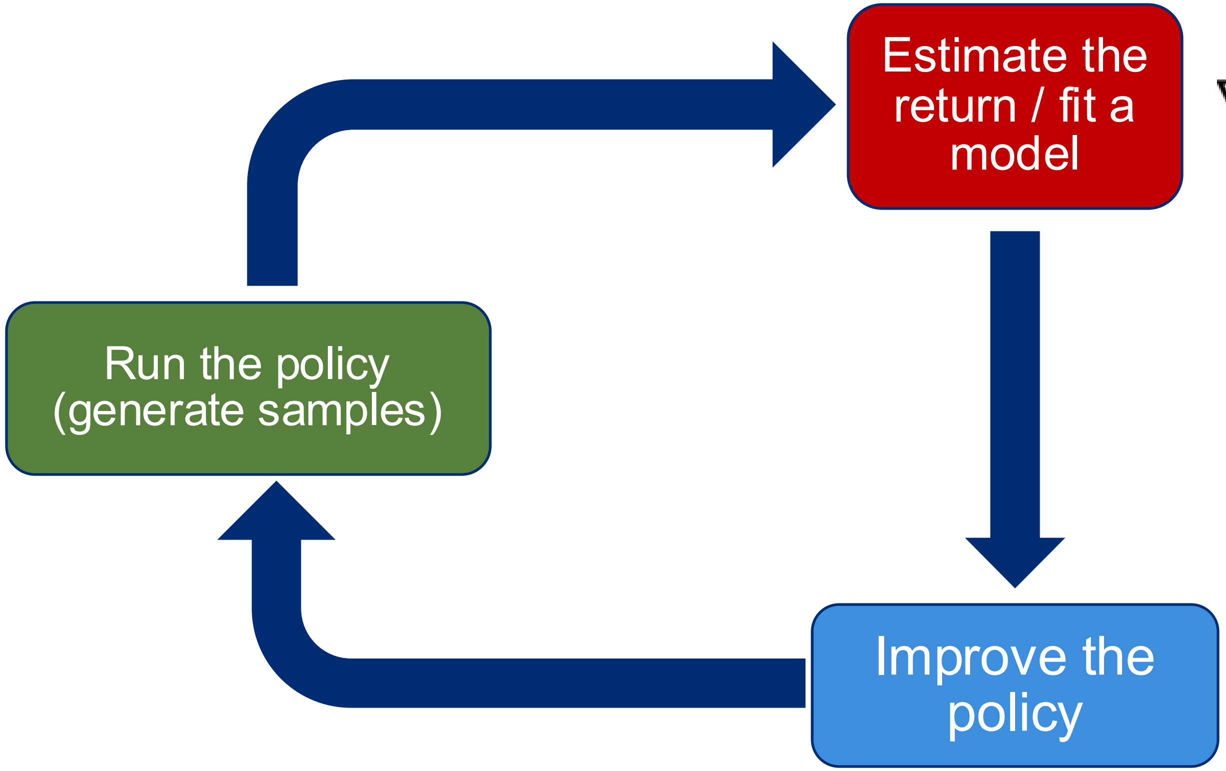


fit $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$

set $\pi(\mathbf{s}) = \text{argmax}_a Q(\mathbf{s}, \mathbf{a})$

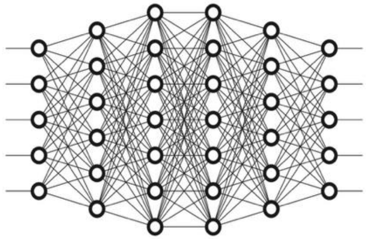
Actor-Critic: Value Functions + Policy Gradients

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



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

$\mathbf{s}_i$



$\hat{V}^\pi_\phi(\mathbf{s}_i)$

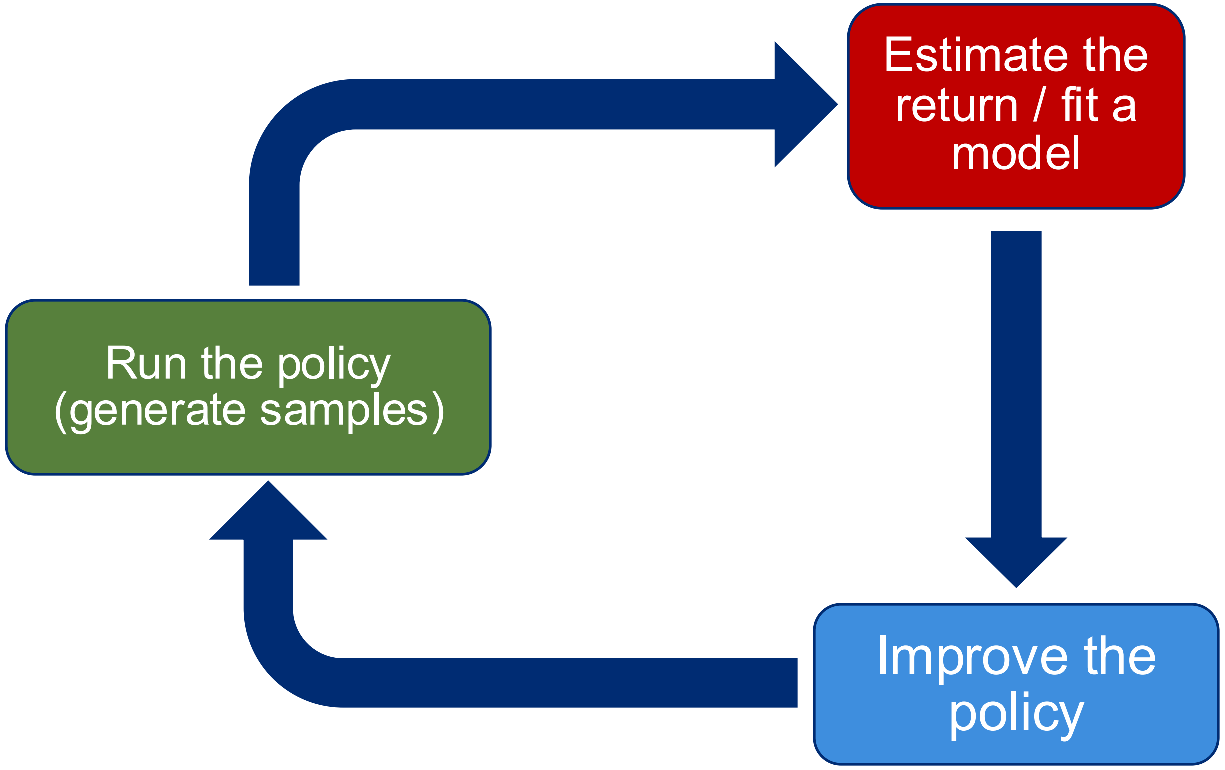
(Parameters ϕ)

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

→ Different value targets, advantage estimates may be used to modulate bias-variance tradeoff.

→ Pure actor-critic uses bootstrapped value targets; some combination with Monte Carlo usually works best.

Model-Based RL algorithms

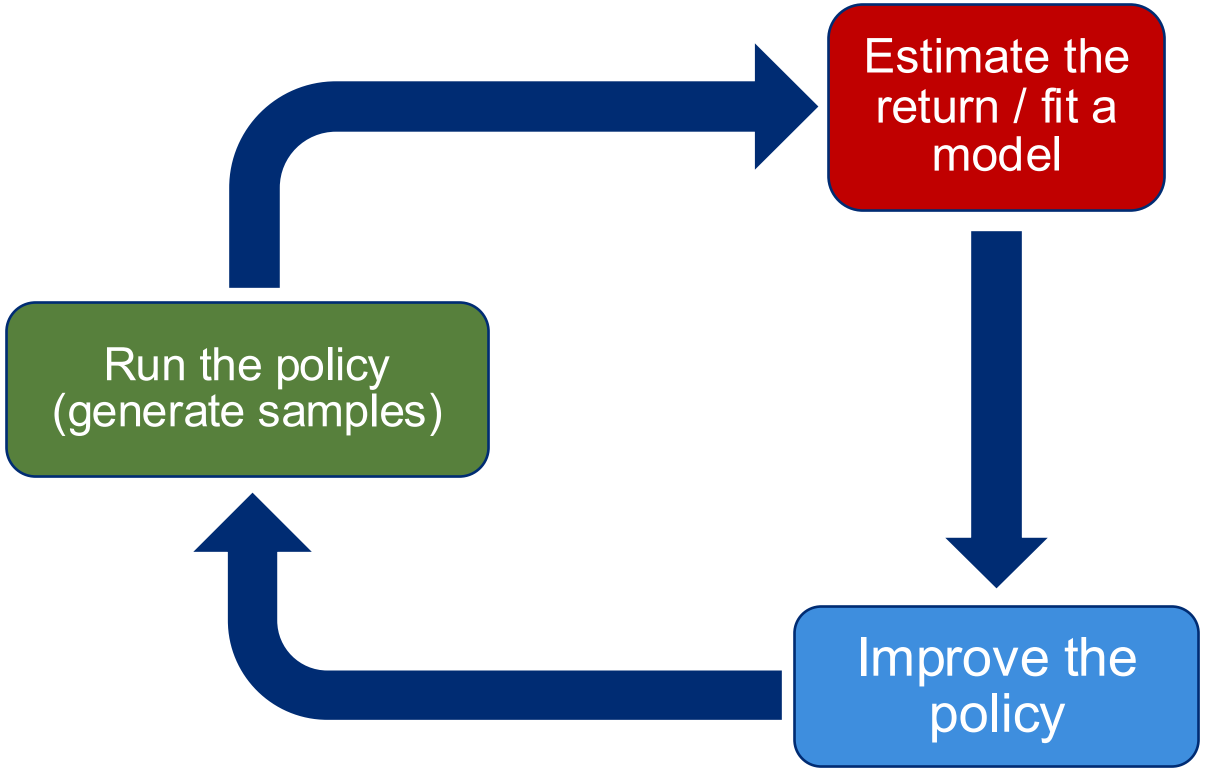


learn $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

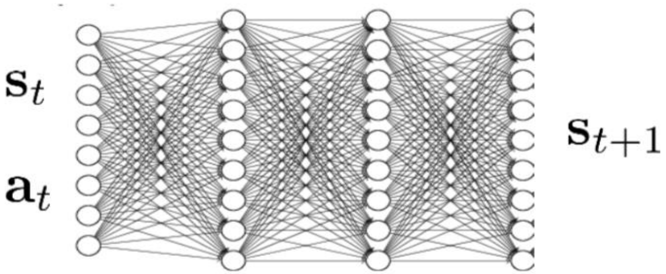
A few options…

1. **Use the model to plan**
2. **Backpropagate gradients into policy**
3. **Use the model to learn a value function**

Model-Based DRL (via Backprop)



learn f_ϕ such that $s_{t+1} \approx f_\phi(s_t, a_t)$



backprop through f_ϕ and r to train $\pi_\theta(s_t) = a_t$

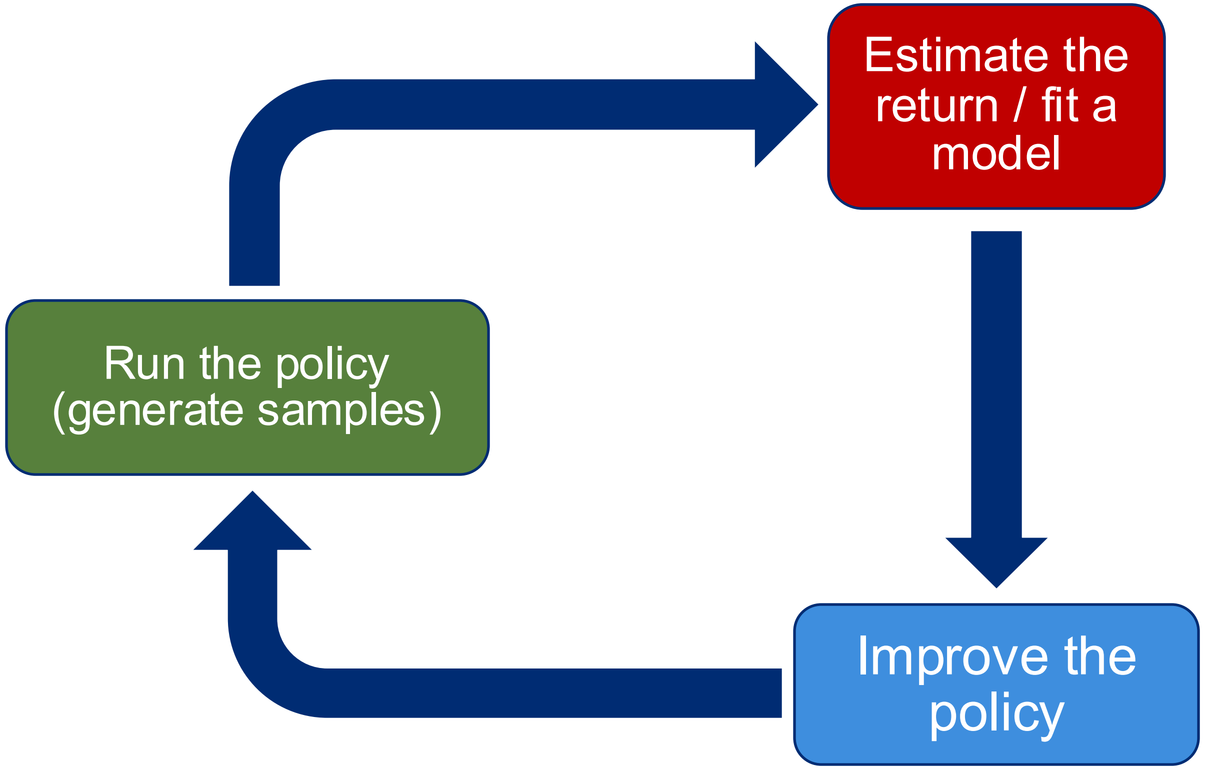
Cost Considerations

Real data:

- Expensive
- Real-time

Simulated data:

- Often cheap
- Often fast



$J(\theta) = E_\pi\left[\sum_t r_t\right] = \frac{1}{N}\sum_{i=1}^{N}\sum_t r_t^i$

→ cheap, fast

learn $s_{t+1} \approx f_\phi(s_t, a_t)$

→ expensive, slow

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

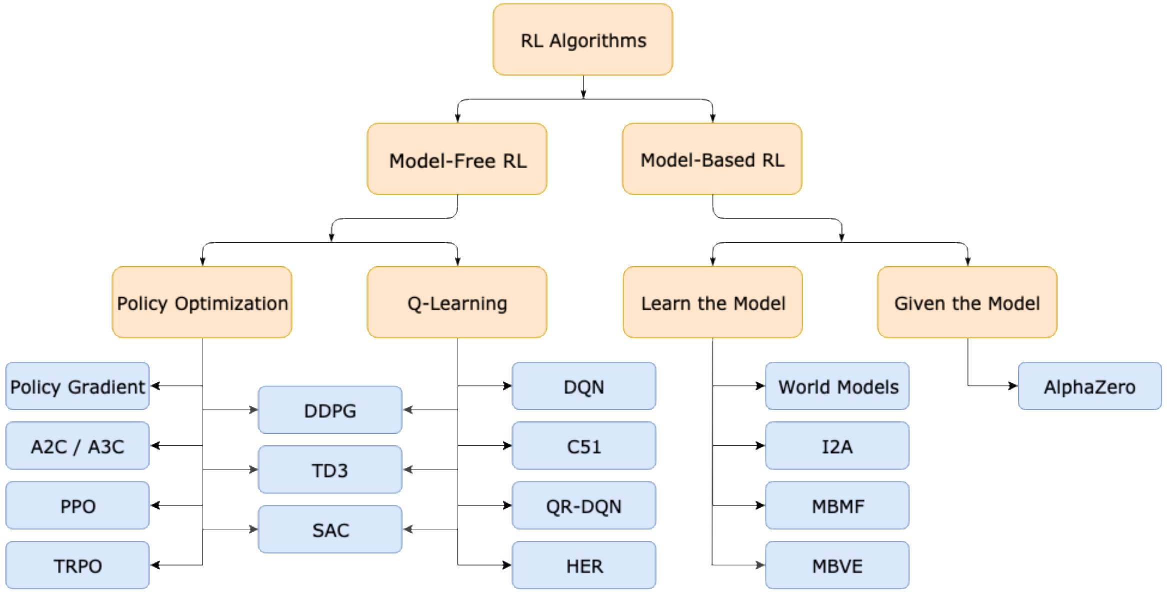
→ cheap, fast

backprop through f_ϕ and r to train $\pi_\theta(s_t) = a_t$

→ expensive, slow

Matching Algorithms with Categories

- **Policy Gradient:**
  - REINFORCE (“Vanilla” policy gradient)
  - Natural policy gradient
- **Value Function Fitting:**
  - Q-learning (DQN)
  - Temporal difference learning
  - Fitted value iteration
- **Actor-Critic:**
  - Asynchronous advantage actor-critic (A3C)
  - Soft actor-critic (SAC)
  - Trust region policy optimization (TRPO)
  - Proximal Policy Optimization (PPO)
- **Model-based RL:**
  - Dyna
  - Guided policy search

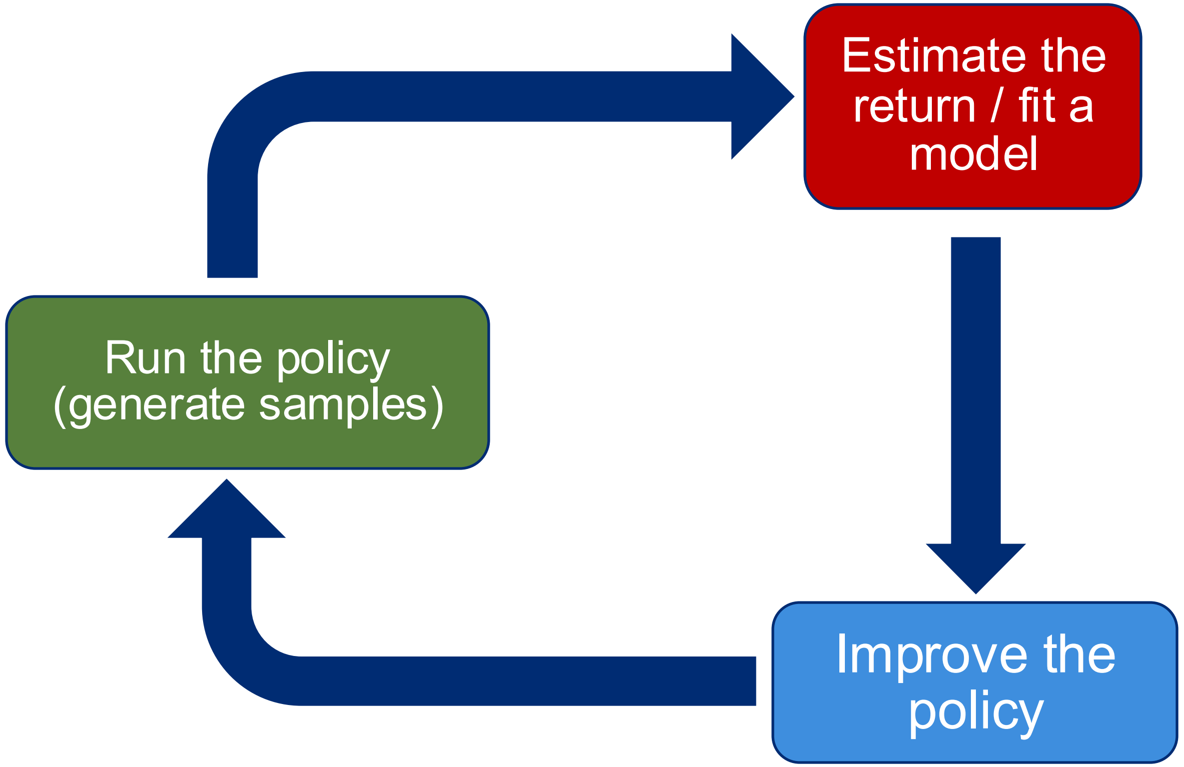


Why are there so many RL algorithms?

Can't we just use the best?

➢ **What is the "best" approach depends on the particular problem and the constraints of the user:**

1. System characteristics
   - Continuous or discrete
   - Stochastic or deterministic
   - Episodic or infinite horizon
   - Where is the complexity?
2. User constraints
   - Sample efficiency
   - Stability
   - Ease of use

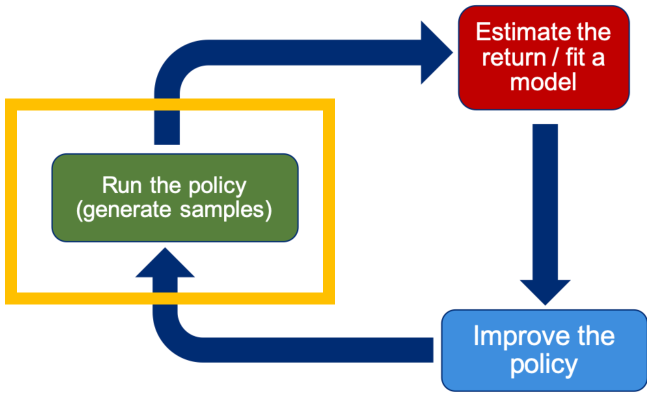


On-Policy vs. Off-Policy

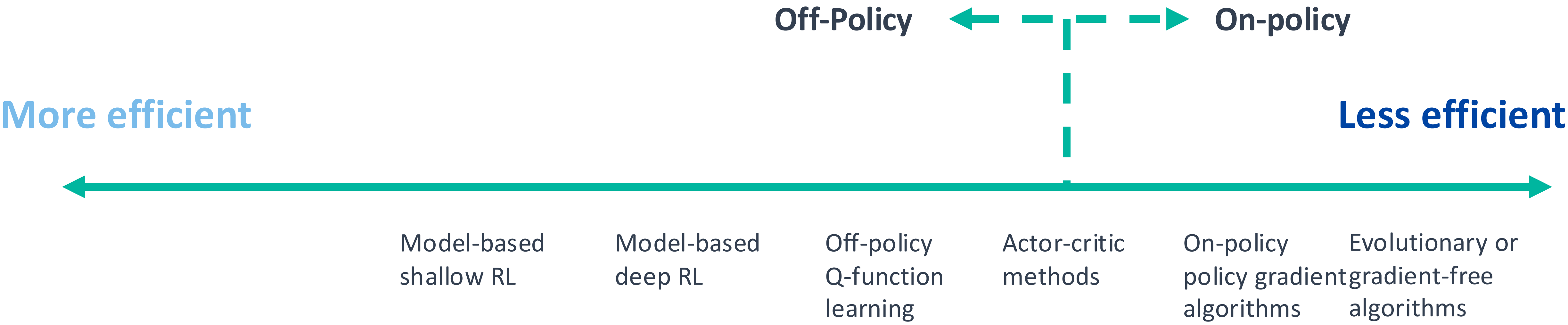
- A key characteristic of a policy is its **sample efficiency**.

- Related to this, algorithms fall into two broad categories:
  1. ***On-policy***: can only learn/improve from data collected using current policy
  2. **Off-policy**: can learn/improve using data collected from other policies

- Off-policy algorithms are generally more sample efficient.



Sample Efficiency By Method



Maximizing sample efficiency and minimizing wall time are not the same!

Adapted from Levine CS285

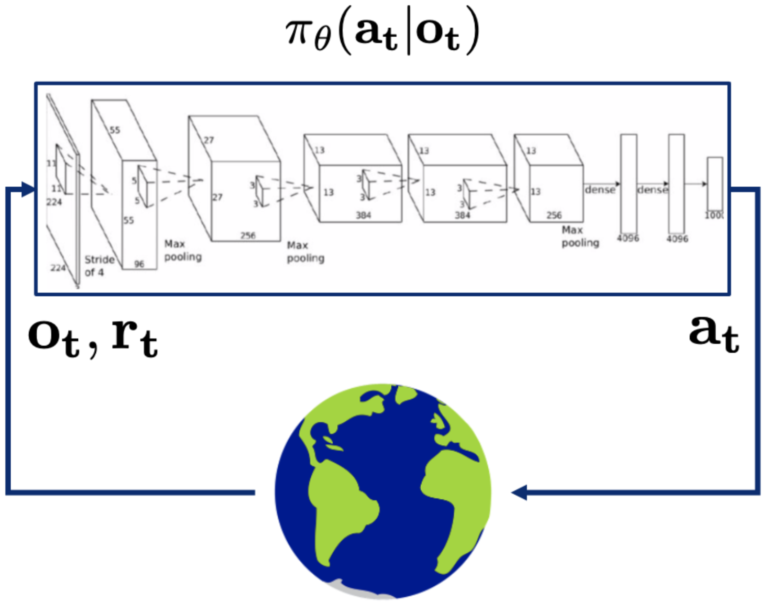
Considerations: Stability and Ease of Use

- **Does the algorithm I picked converge every time? To something useful?**

- **Q Learning / value function fitting**
  - Fixed point iteration
  - Minimizes error of fit (“Bellman Error”): not the same as reward!
  - Not guaranteed to converge in nonlinear case

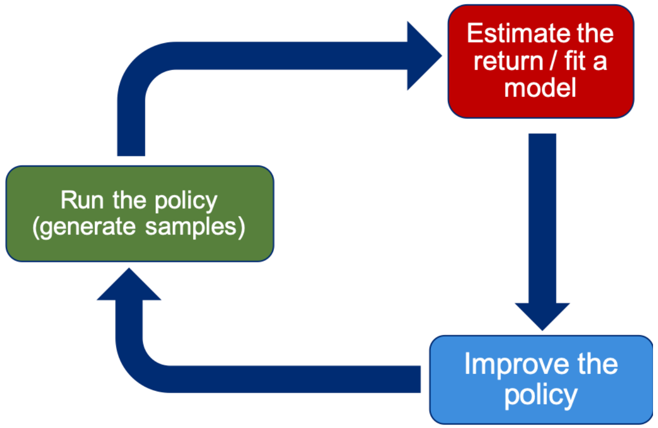
- **Model-Based**
  - Minimizes error of fit- will converge
  - Better model doesn’t necessarily imply a better policy!

- **Policy Gradient**
  - The only method that performs gradient ascent on the true objective
  - Also the least sample-efficient



Comparison: Assumptions to Consider

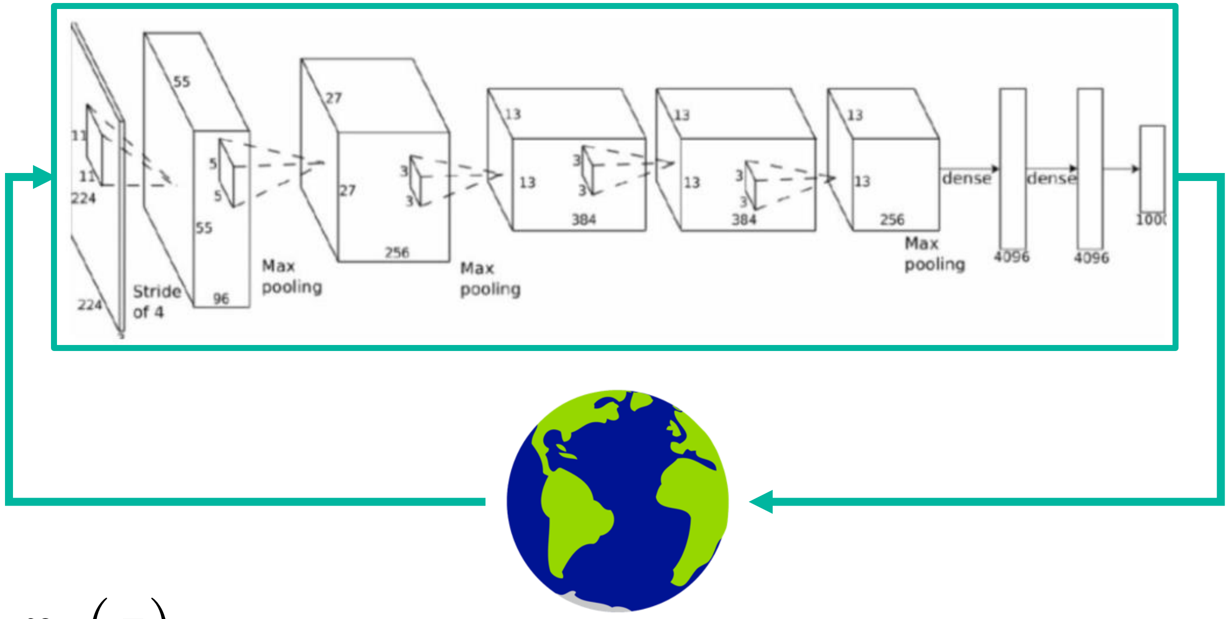
- **Common assumption #1:** full observability
  - Assumed for value function fitting, model-based methods
  - Reliance on it can be mitigated through use of recurrence
- **Common assumption #2:** episodic learning
  - Typically assumed by pure policy gradient methods, model-based RL methods
- **Common assumption #3:** continuity or smoothness
  - Assumed by continuous value function learning methods, model-based RL



Basic Policy Gradient

The Goal of DRL

$\pi_\theta(\mathbf{a_t}|\mathbf{s_t})$



$\mathbf{s_t}, \mathbf{r_t}$

$\mathbf{a_t}$

Fully observable (for now)

$$\overbrace{p_\theta(\mathbf{s_1}, \mathbf{a_1}, \ldots, \mathbf{s_T}, \mathbf{a_T})}^{p_\theta(\tau)} = p(\mathbf{s_1}) \prod_{t=1}^{T} \pi_\theta(\mathbf{a_t}|\mathbf{s_t}) \, p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})$$

$$\text{Goal: find } \theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s_t}, \mathbf{a_t}) \right]$$

Undiscounted (for now)

Finite Horizon

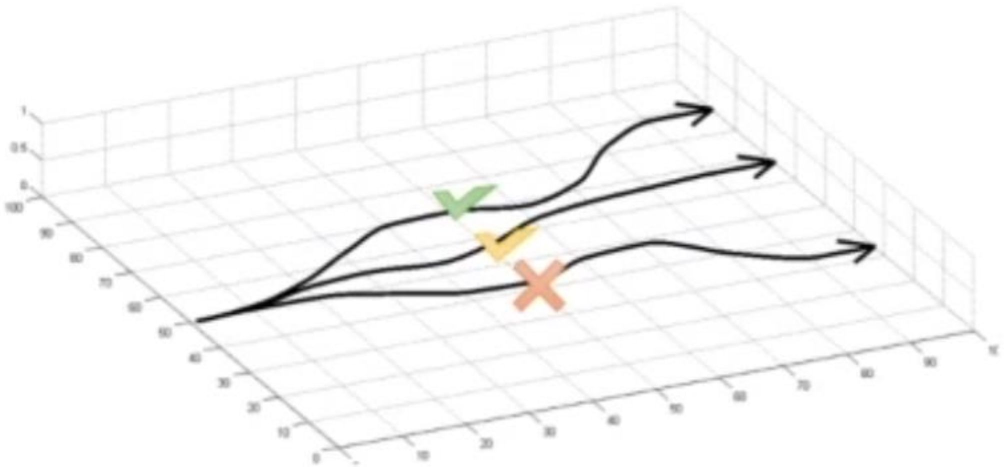
$$\theta^{\star} = \arg\max_{\theta} E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$\theta^{\star} = \arg\max_{\theta} \frac{1}{T} \sum_{t=1}^{T} E_{(\mathbf{s_t}, \mathbf{a_t}) \sim p_\theta(\mathbf{s_t}, \mathbf{a_t})} \left[r(\mathbf{s_t}, \mathbf{a_t}) \right], \; T \to \infty$$

$$\theta^{\star} = \arg\max_{\theta} \sum_{t=1}^{T} E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_\theta(\mathbf{s}_t, \mathbf{a}_t)} \left[r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Evaluating the 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 p_θ

Direct Policy Differentiation

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

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

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

$$\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau =$$

Direct Policy Differentiation

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

$$\theta^\star = \arg\max_\theta J(\theta)$$

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

Look at joint distribution over history:

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

Take log of both sides…

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

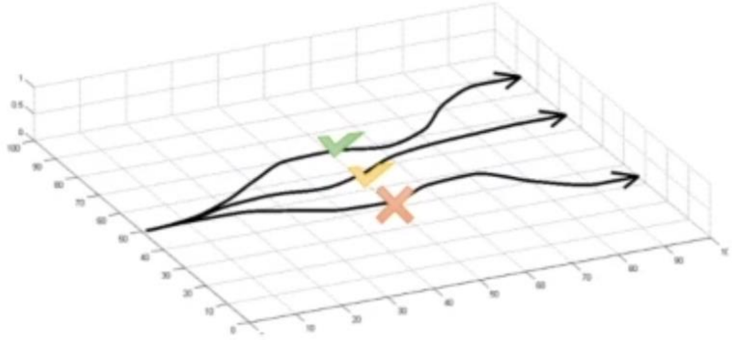
Evaluate gradient…

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

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

Evaluating the Policy Gradient via Sampling

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



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

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

Improve the policy

REINFORCE Algorithm

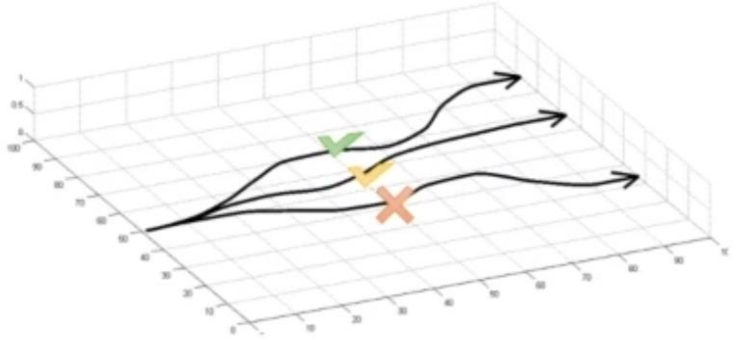
While not converged:

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. $\theta \longleftarrow \theta + \alpha \nabla_\theta J(\theta)$

Typically works poorly…

Evaluating the Policy Gradient via Sampling

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



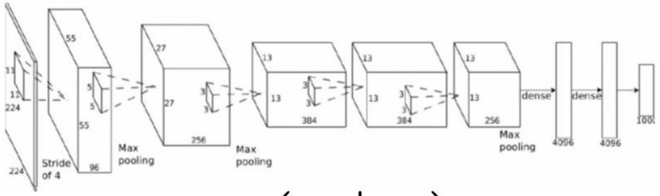
$$\nabla_\theta J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

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

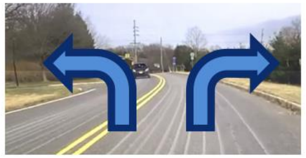
Improve the policy

What is this?

$\mathbf{o}_t$



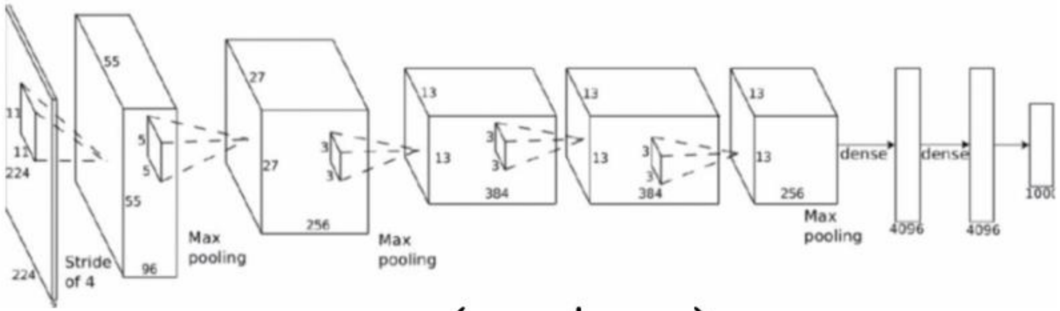
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$



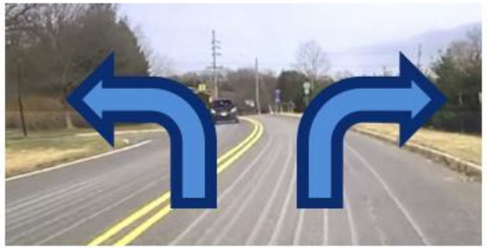
$\mathbf{a}_t$

Partial Observability

$\mathbf{o}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$



$\mathbf{a}_t$

Can we use the policy gradient in a partially-observed setting?

- Yes. Never used an assumption of policy depending on state; can just be a mapping conditioned on what we have:

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

Reward still depends on state; we just don't observe it directly.

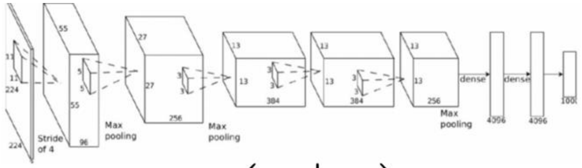
Just substitute observation for state!

Relation to Maximum Likelihood

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

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

$\mathbf{o_t}$



$\pi_\theta(\mathbf{a_t}|\mathbf{o_t})$

$\mathbf{a_t}$

Training Data

$\mathbf{o_t}$

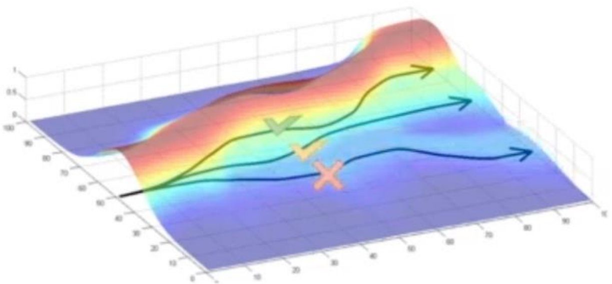
$\mathbf{a_t}$

→ Supervised Learning → $\pi_\theta(\mathbf{a_t}|\mathbf{o_t})$

Some Intuition

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

$$\text{maximum likelihood: } \nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau_i)$$



REINFORCE Algorithm

While not converged:

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. $\theta \longleftarrow \theta + \alpha \nabla_\theta J(\theta)$

- **Makes good trajectories more likely**
- **Makes bad trajectories less likely**
- **Formalization of trial and error!**

Example: Gaussian policies

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

example: $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t);\Sigma)$

$$\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\|f(\mathbf{s}_t) - \mathbf{a}_t\|_\Sigma^2 + \text{const}$$

$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\Sigma^{-1}(f(\mathbf{s}_t) - \mathbf{a}_t)\frac{df}{d\theta}$$

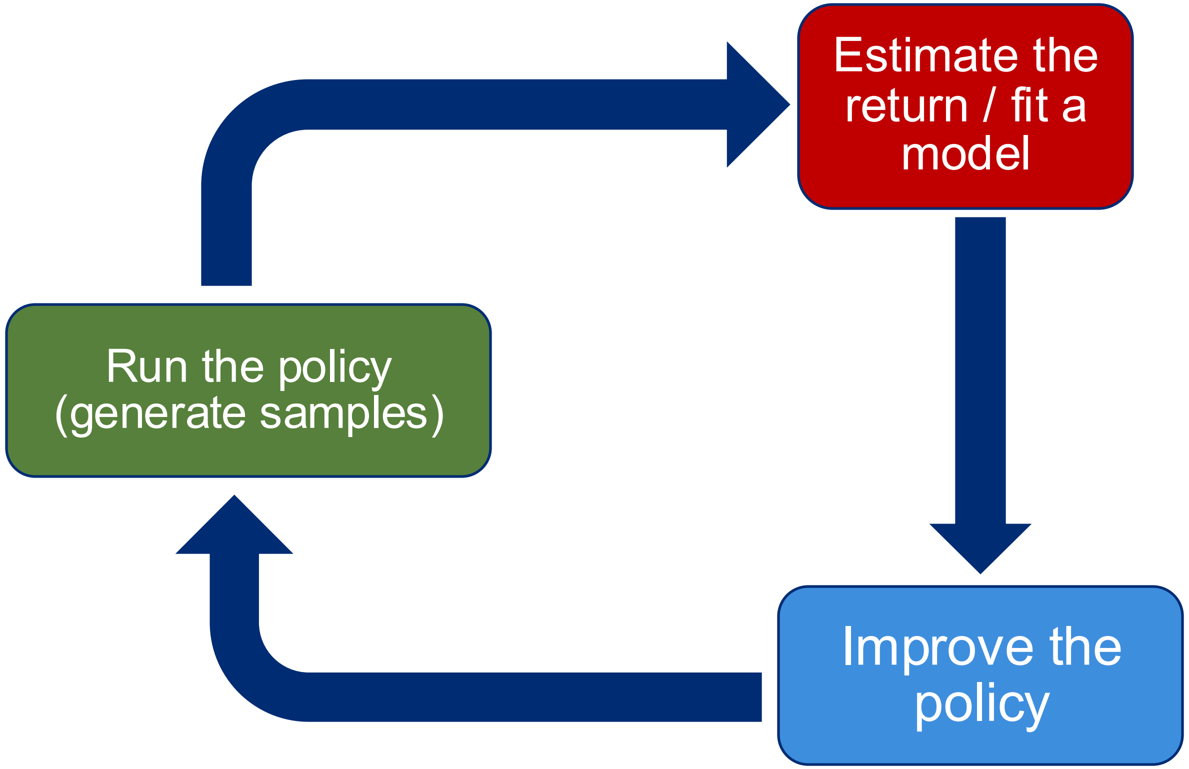
REINFORCE Algorithm

While not converged:

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. $\theta \longleftarrow \theta + \alpha\nabla_\theta J(\theta)$

Recap: Basic Policy Gradient

- Directly differentiates RL objective
- Can be evaluated via sampling
- Works under partial observability
- Can be seen as making good trajectories more likely, bad trajectories less likely

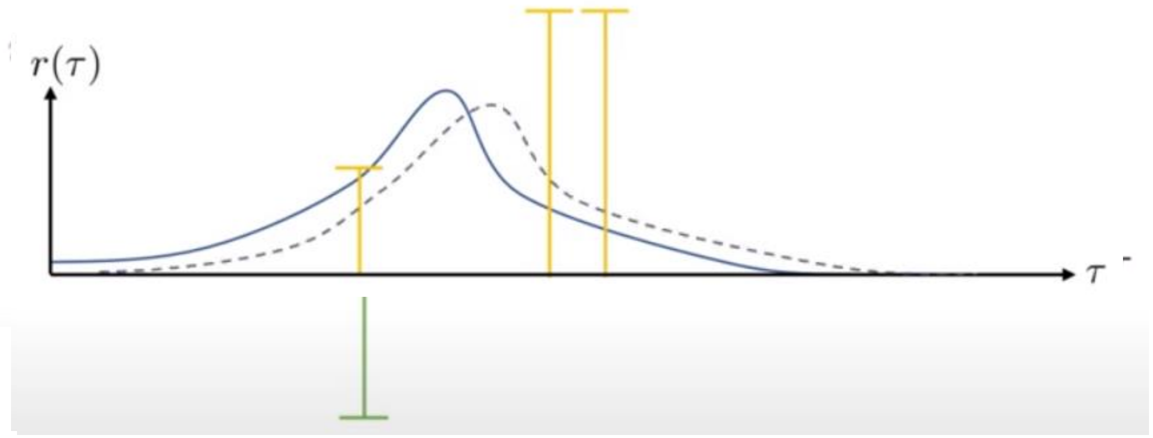


Problems with policy gradient

- High variance, sample inefficiency

Problem: Variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log p_\theta(\tau) r(\tau)$$



Thought experiment: What if we were to add a constant to make the two "good" outcomes 0 reward?

- **Problem: policy gradient has high variance!**
  - **Often leads to noisy gradient estimates**

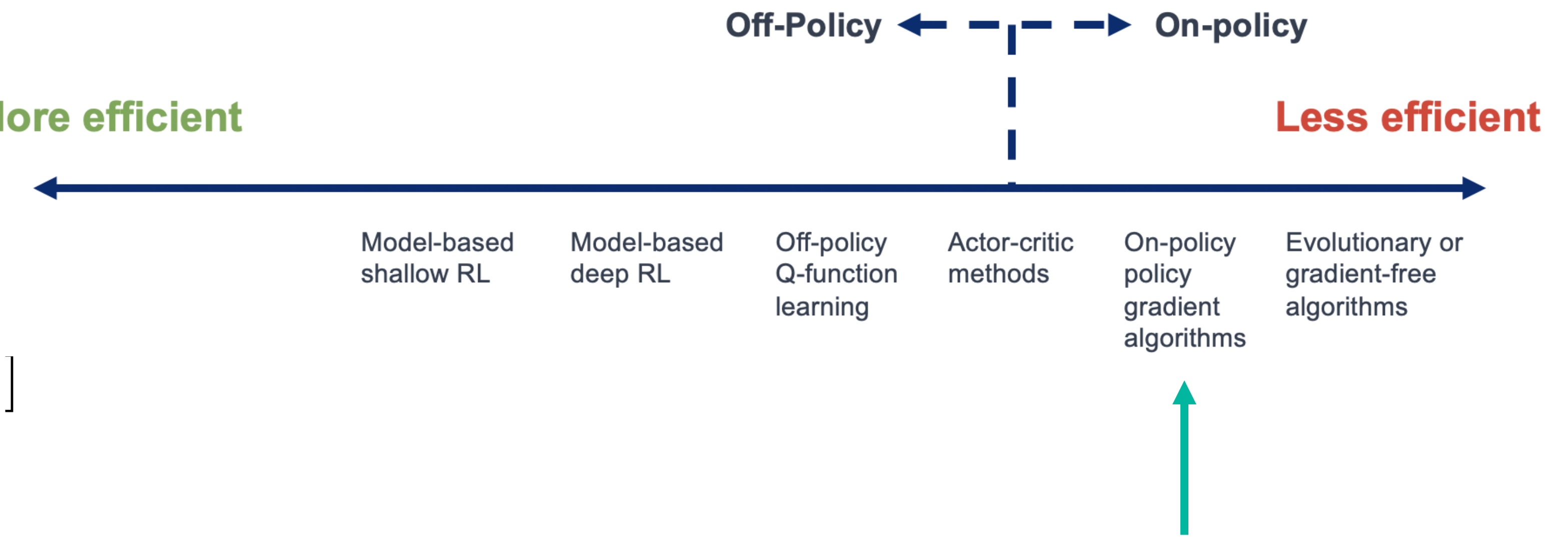
Animation credit: Levine UC Course

Problem: Data Hungry

$$\theta^\star = \arg\max_\theta J(\theta)$$

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

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



REINFORCE Algorithm

While not converged:

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. $\theta \longleftarrow \theta + \alpha \nabla_\theta J(\theta)$

- **We can't get around this sampling**
- **Because NN updates slowly, this can be highly inefficient!**

Improving Policy Gradient

- Leveraging causality, baselines, off-policy policy gradient

Reducing Variance: Causality

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

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

Causality : policy at time t' cannot affect rewards at time t when $t < t'$

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

→ $$\hat{Q}_{i,t} = \left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)$$

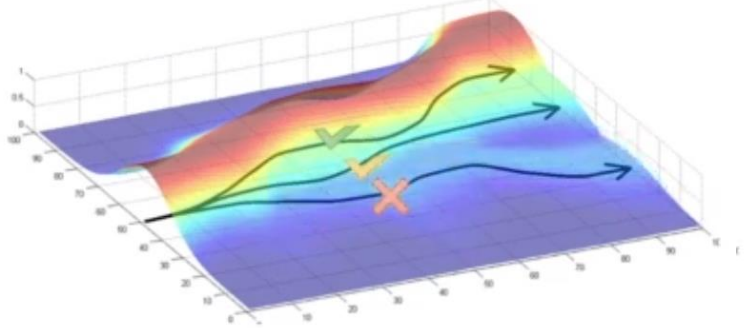
"reward to go"

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

➢ **Smaller gradients, smaller variance!**

Reducing Variance: Baselines

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log p_\theta(\tau)[r(\tau) - b)]$$



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

- **Subtracting a baseline is unbiased in expectation!**
- **It turns out that the mean reward is a good baseline to use.**
- **If you want the baseline that reduces variance the most, choose**

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]}$$

Note: varies by parameter dimension

Analyzing the variance

- We can also use a neural network to estimate a state-dependent baseline!
- This is also free of bias (more later).

Variance definition: $\text{Var}[x] = E[x^2] - E[x]^2$

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

$$\text{Var} = E_{\tau \sim p_\theta(\tau)}[(\nabla_\theta \log p_\theta(\tau)(r(\tau) - b))^2] - \underbrace{E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)(r(\tau) - b)]^2}$$

this bit is just $E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$

(baselines are unbiased in expectation)

$$\frac{d\text{Var}}{db} = \frac{d}{db} E[g(\tau)^2(r(\tau) - b)^2] = \frac{d}{db}(\cancel{E[g(\tau)^2 r(\tau)^2]} - 2E[g(\tau)^2 r(\tau) b] + b^2 E[g(\tau)^2])$$

$$= -2[E g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0$$

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]}$$

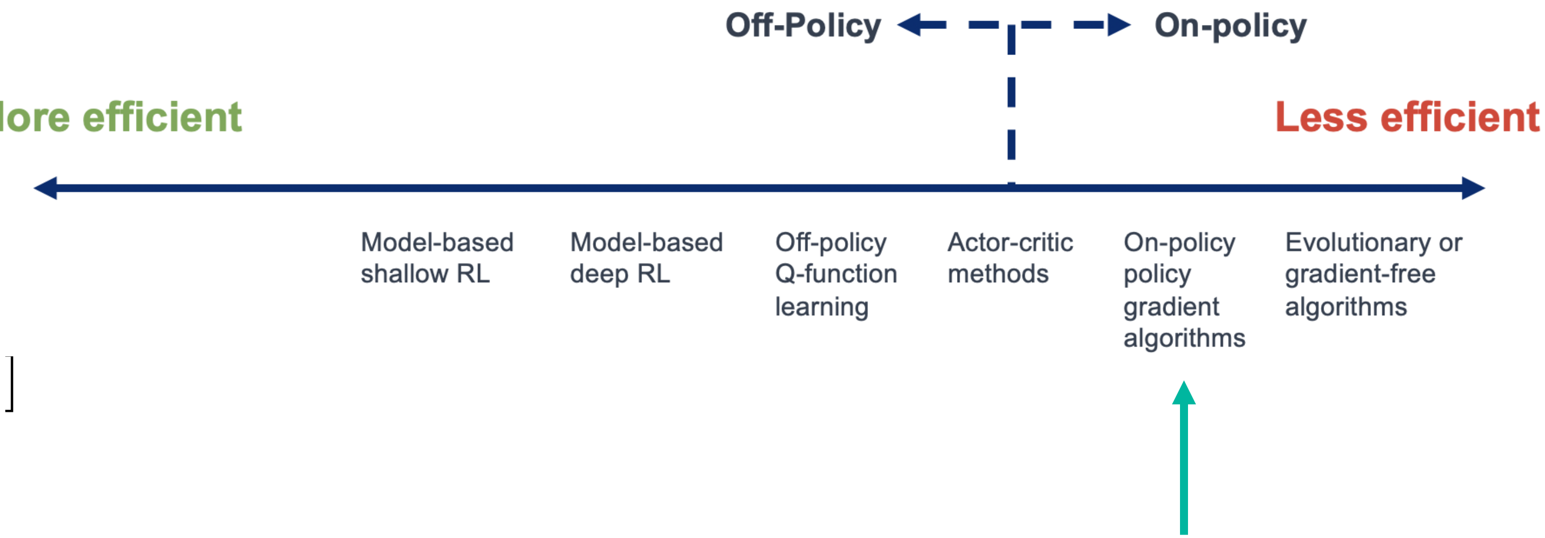
→ Expected reward weighted by gradient magnitudes.

On-Policy Learning: Data Hungry

$$\theta^\star = \arg\max_\theta J(\theta)$$

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

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



REINFORCE Algorithm

While not converged:

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. $\theta \longleftarrow \theta + \alpha \nabla_\theta J(\theta)$

- **We can't get around this sampling**
- **Because NN updates slowly, this can be highly inefficient!**

Off-policy learning via importance sampling

$$\theta^\star = \arg\max_\theta J(\theta)$$

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

What if we don't have samples from $p_\theta(\tau)$,
but instead have samples from a different $\bar{p}(\tau)$?

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

$$p_\theta(\tau) = p(\mathbf{s_1}) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})$$

$$\frac{p_\theta(\tau)}{\bar{p}(\tau)} = \frac{p(\mathbf{s_1}) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})}{p(\mathbf{s_1}) \prod_{t=1}^{T} \bar{\pi}(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})} = \frac{\prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T} \bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)}$$

Importance sampling:

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

Off-policy policy gradient

$$\theta^\star = \arg\max_\theta J(\theta)$$

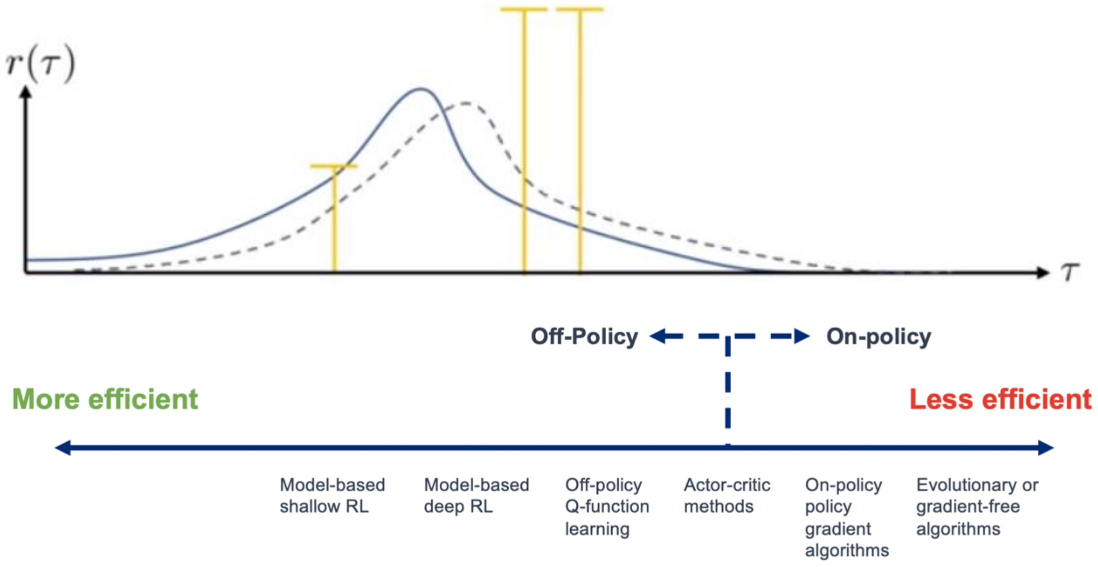
$$J(\theta) = E_{\tau\sim p_\theta(\tau)}[r(\tau)]$$

$$\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} = \frac{\prod_{t=1}^{T} \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}$$

$$\nabla_{\theta'} J(\theta') = E_{\tau\sim p_\theta(\tau)}\left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)}\nabla_{\theta'}\log p_{\theta'}(\tau) r(\tau)\right] \text{ when } \theta \neq \theta'$$

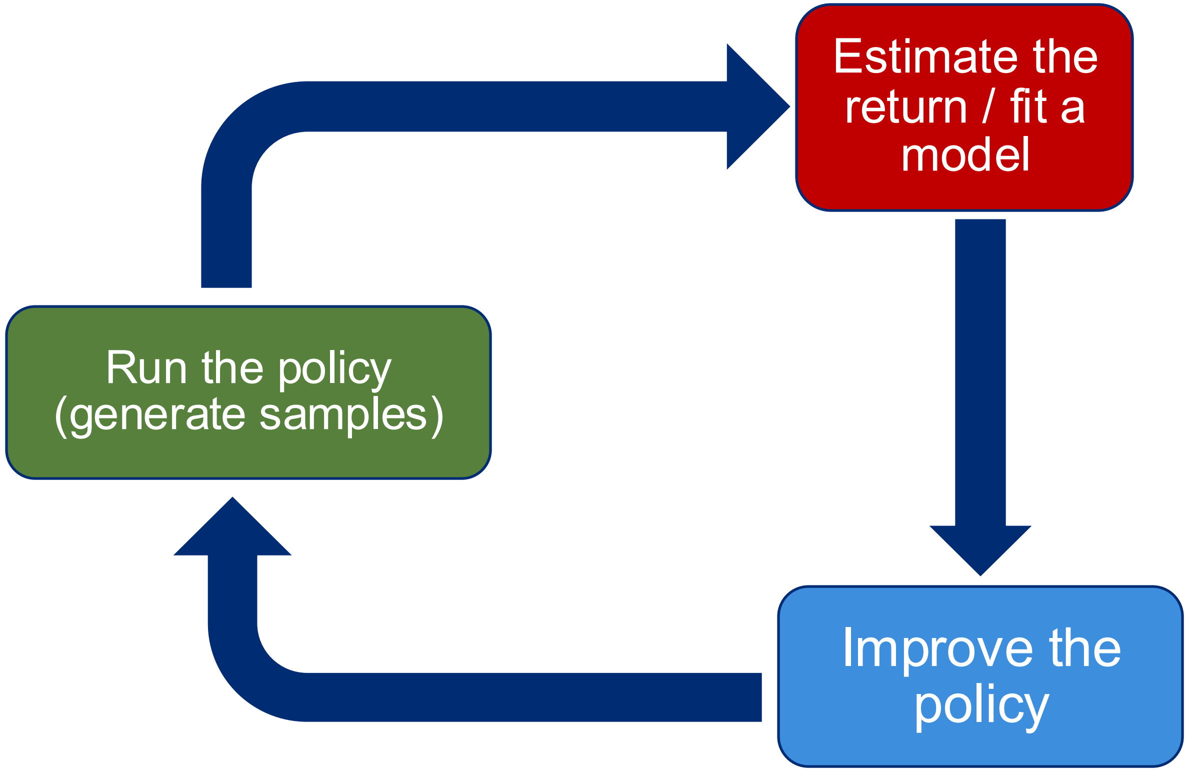
- Increases variance!
- Need further tricks to make this tenable; will re-visit.

Recap: Improving the Policy Gradient



- **Reducing variance**
  - Leveraging causality
  - Adding a baseline
- **Reducing data needs**
  - Off-policy policy gradient

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

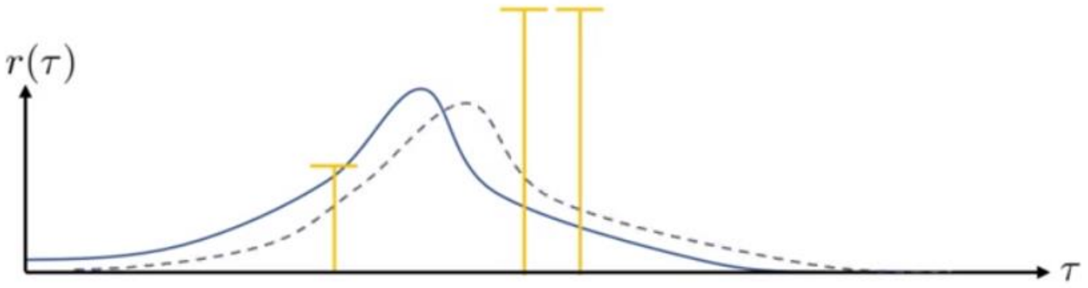


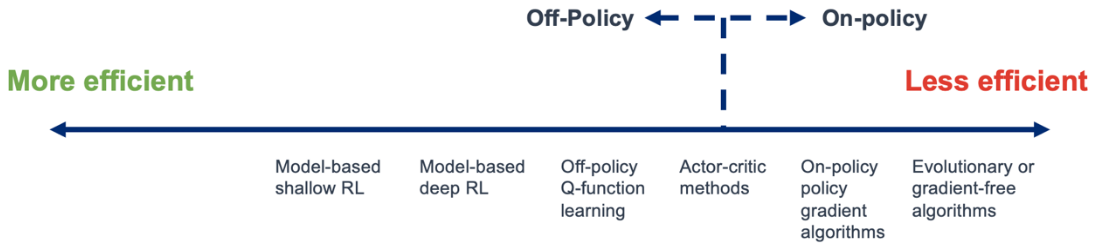
$$\theta \longleftarrow \theta + \alpha \nabla_{\theta} J(\theta)$$

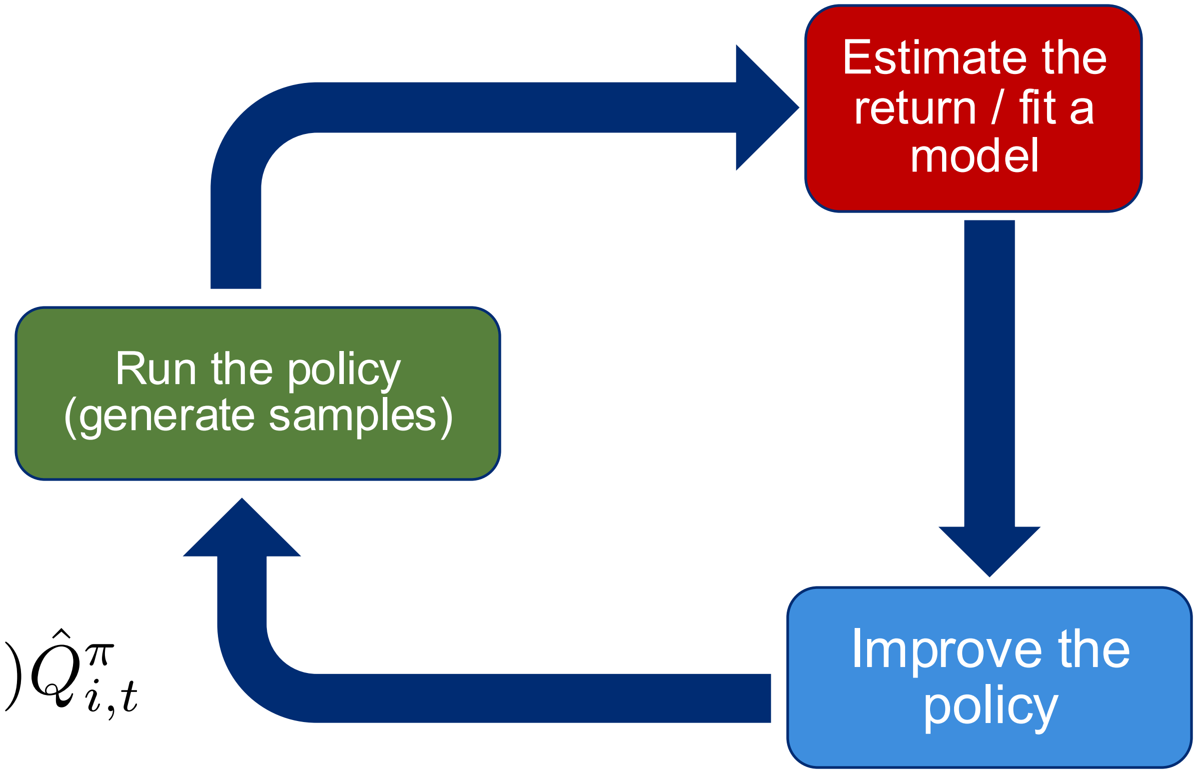
Actor-Critic Algorithms

Recap: Improving the Policy Gradient

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







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

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

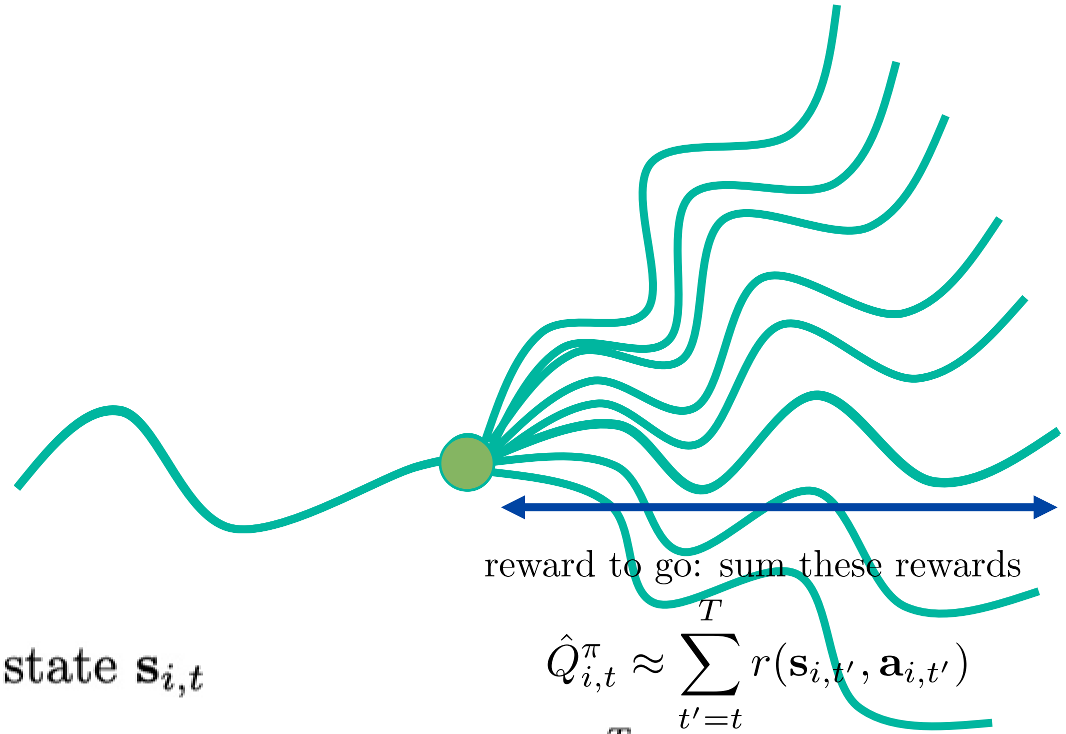
➢ **Actor-critic is about estimating *expected* reward to-go.**

(Further) improving the policy gradient

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

reward to go

$\hat{Q}_{i,t}$



$\hat{Q}_{i,t}$: estimate of expected reward if we take action $\mathbf{a}_{i,t}$ in state $\mathbf{s}_{i,t}$

Can we get a better estimate?

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

$$Q(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}'_t, \mathbf{a}'_t)|\mathbf{s}_t, \mathbf{a}_t]$$: true *expected* reward-to-go

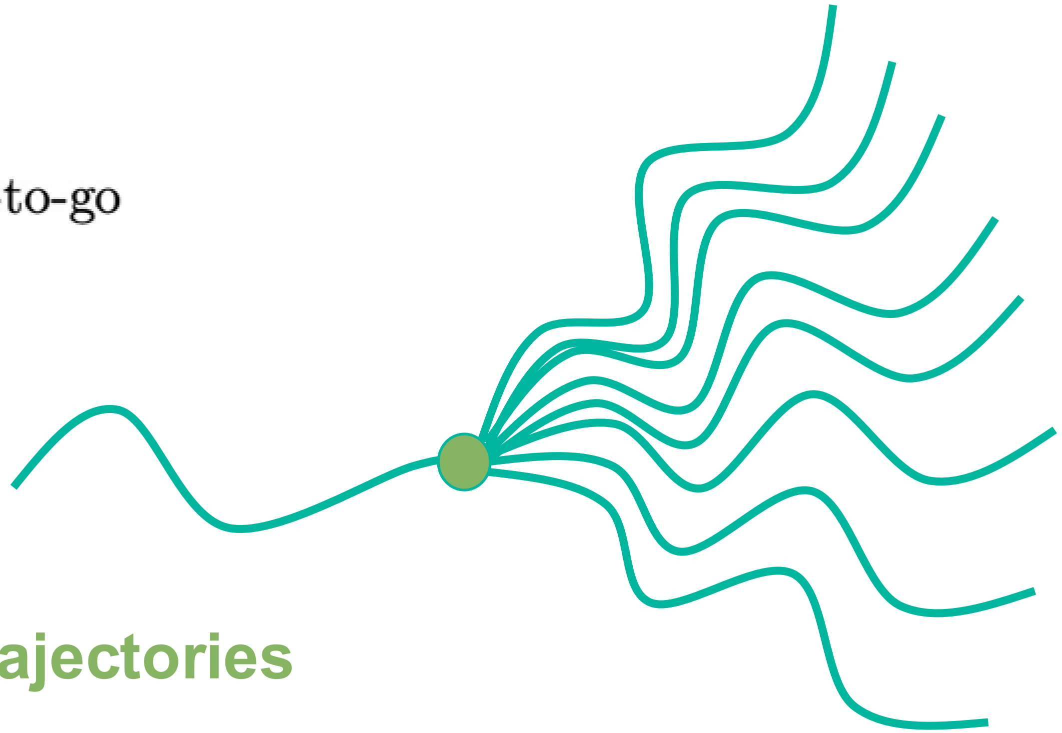
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

Lower variance!

What about the baseline?

$$Q(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t]\text{: true } \textit{expected} \text{ reward-to-go}$$

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



$$b = \text{ average reward}$$

Could average over trajectories

$$b_t = \frac{1}{N} \sum_i Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

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

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

Reduce variance more by using state-dependent baseline (gradient still unbiased)

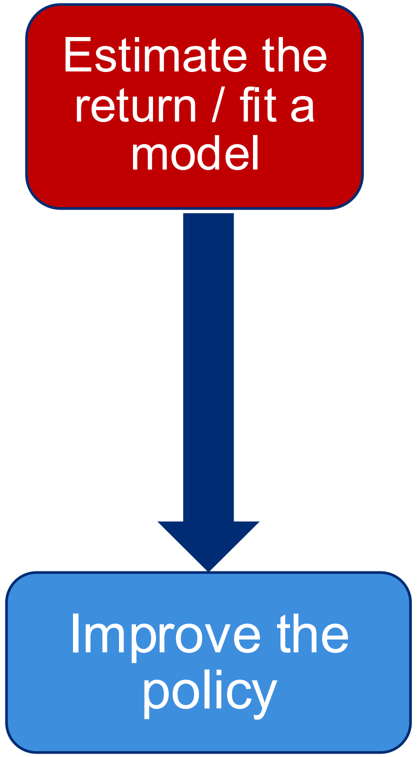
Terminology Recap

$$Q^\pi(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t]$$: quality; total expected reward from taking $\mathbf{a}_t$ in $\mathbf{s}_t$

$$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t\sim\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t,\mathbf{a}_t)]$$: value; expected total reward from $\mathbf{s}_t$

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

fit $Q^\pi, V^\pi,$ or A^π



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

Updating policy:

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

this estimate can be used to reduce variance!

Recall:

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

Unbiased but high-variance (single-sample)

Fitting a value function

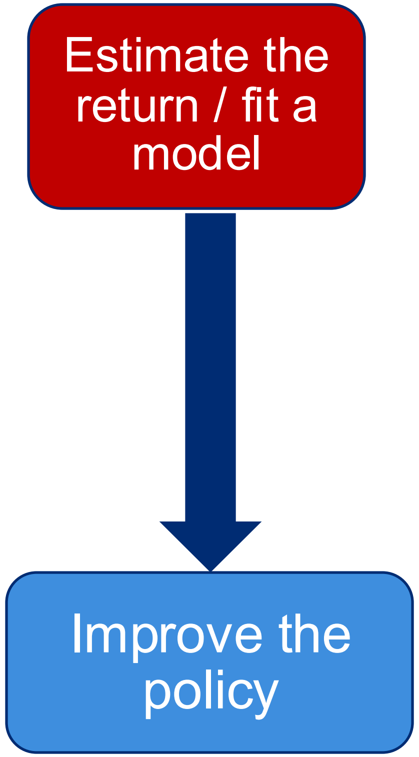
$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t]$$

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

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

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

fit Q^π, V^π, or A^π



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

What to fit?

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

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

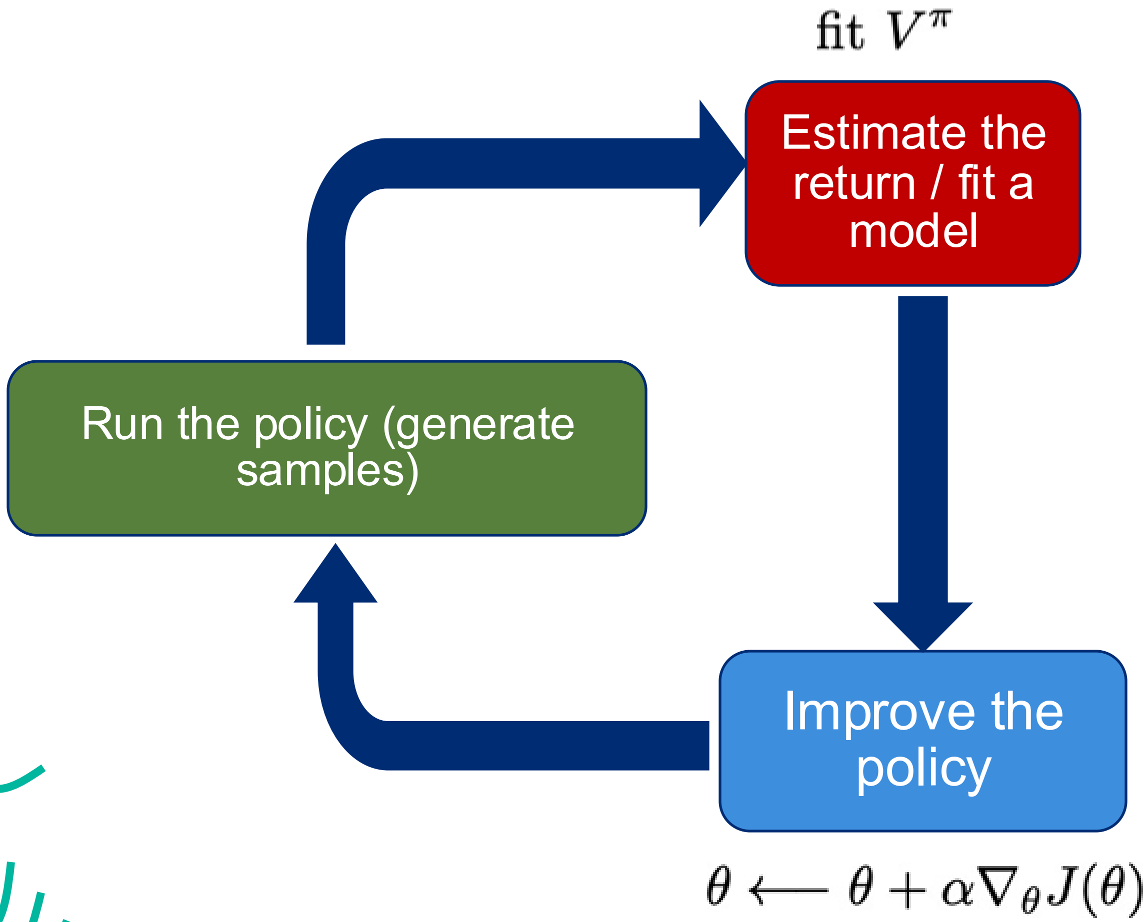
$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)$$

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

Policy evaluation

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

$$J(\theta) = E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)}[V^{\pi}(\mathbf{s}_1)]$$

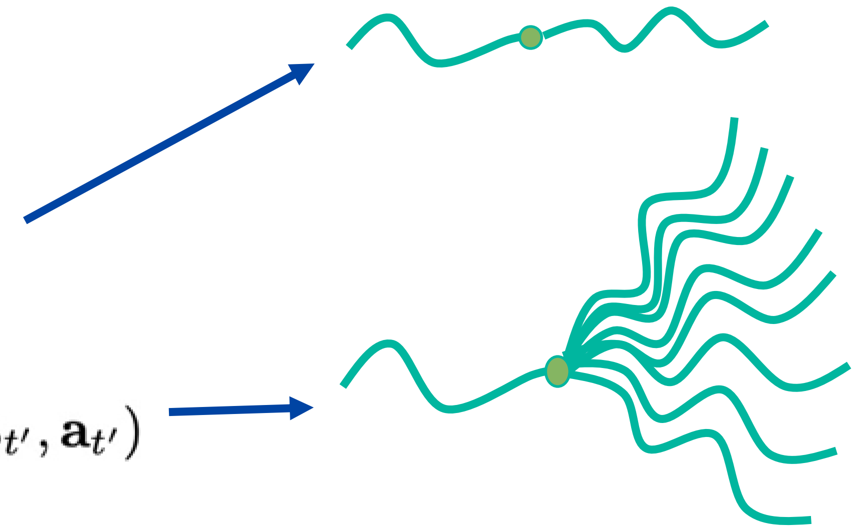


How can we evaluate a policy?

Monte Carlo!

$$V^{\pi}(\mathbf{s_t}) \approx \sum_{t'=t}^{T} r(\mathbf{s_{t'}}, \mathbf{a_{t'}})$$

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



Can't actually re-set to every state, so go with single-trajectory estimate

Monte Carlo evaluation with function approximation

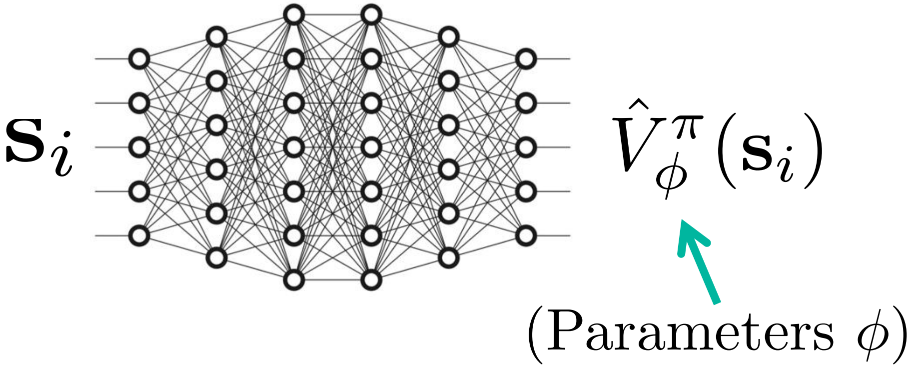
$$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

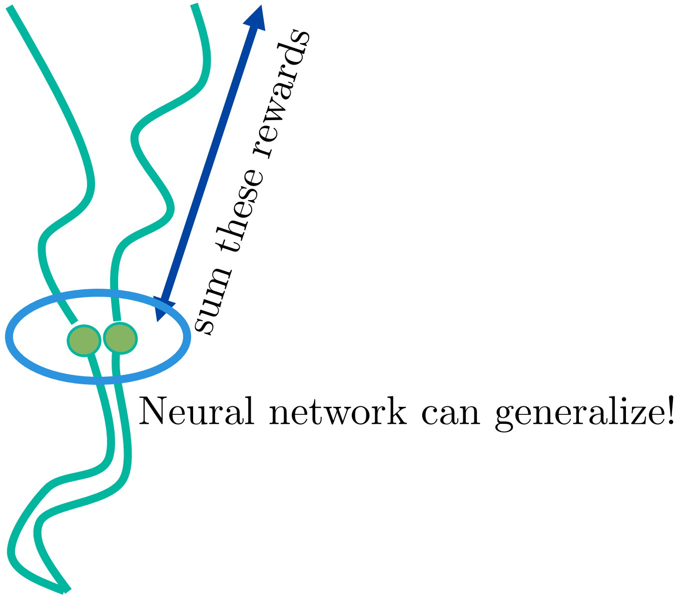
Not quite $V^\pi(\mathbf{s}_t) = \frac{1}{N} \sum_{i=1}^{T} \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$

but often good enough!

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

supervised regression: $\mathcal{L}(\phi) = \sum_i ||\hat{V}_\phi^\pi(\mathbf{s}_i) - y_i||^2$





Bootstrapping

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

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

Training data become: $\left\{\left(\mathbf{s}_{i,t}, \underbrace{r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})}_{y_{i,t}}\right)\right\}$

Supervised regression: $\mathcal{L}(\phi) = \sum_i ||\hat{V}_\phi^\pi(\mathbf{s}_i) - y_i||^2$

Often referred to as "bootstrapping"

> ➢ **While biased, often better because lower variance!**

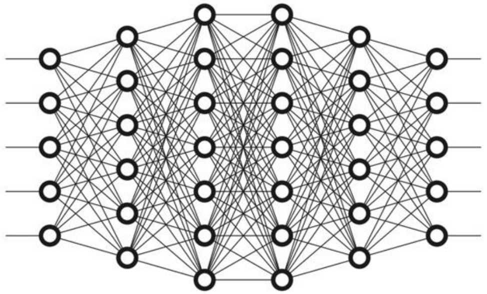
Policy Gradient to Actor-Critic

REINFORCE Algorithm

While not converged:

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. $\theta \longleftarrow \theta + \alpha \nabla_\theta J(\theta)$

$\mathbf{s}_i$



$\hat{V}_\phi^\pi(\mathbf{s}_i)$

Batch Actor-Critic Algorithm

While not converged:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$
2. fit $\hat{V}_\phi^\pi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \hat{V}_\phi^\pi(\mathbf{s}_i') - \hat{V}_\phi^\pi(\mathbf{s}_i)$
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. $\theta \longleftarrow \theta + \alpha \nabla_\theta J(\theta)$

$$y_{i,t} = \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$$ ← No bias, higher variance

or

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

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

Discount Factor

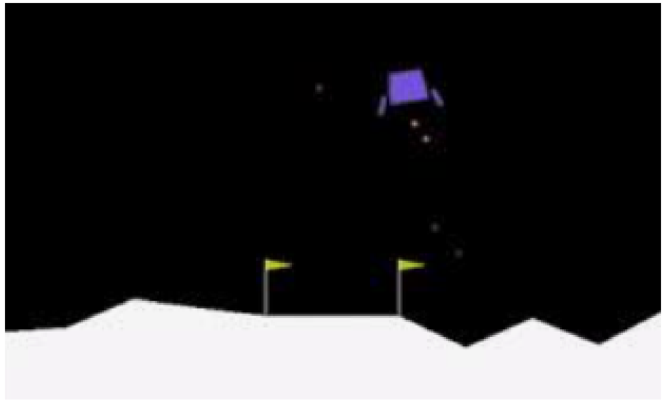
Discount Factors

What if task is infinite horizon?

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

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

$\longrightarrow \hat{V}^{\pi}_{\phi}$ can get infinitely large!



Finite Horizon

Infinite Horizon

Solution: discount factor

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t+1}); \quad \gamma \in [0, 1] \text{ (usually 0.95 - 0.99)}$$

Intuition:

1.) γ reduces variance by de-emphasizing uncertain future events
2.) Decreasing γ increases bias but reduces variance
3.) Better to get rewards sooner rather than later (might reach terminal state)
4.) What I do now impacts the near future more than the distant future

Discounts in the gradient

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t+1}); \quad \gamma \in [0,1] \text{ (usually 0.95 - 0.99)}$$

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

➢ Methods now apply to infinite horizon cases.

Monte-Carlo, no critic:

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

With critic:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \underbrace{\left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t+1}) - \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t}) \right)}_{\hat{A}^{\pi}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}$$

Actor-critic algorithm Design

- Batch or online, architecture choices
- Architecture choices

Actor-critic algorithms (with discount)

Batch Actor-Critic Algorithm:

While not converged:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$
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. $\theta \longleftarrow \theta + \alpha\nabla_\theta J(\theta)$

Online Actor-Critic algorithm:

While not converged:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi(\mathbf{s})$ using target $r + \gamma\hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \hat{V}^\pi_\phi(\mathbf{s}') - \hat{V}^\pi_\phi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s})\hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \longleftarrow \theta + \alpha\nabla_\theta J(\theta)$

- **Policy Gradient: Need to run to end of episode**
- **Actor-Critic: Can update in middle of episode**
- **Unfortunately, need more to make Online Actor-Critic actually work…**

Parallelization

Online Actor-Critic algorithm:

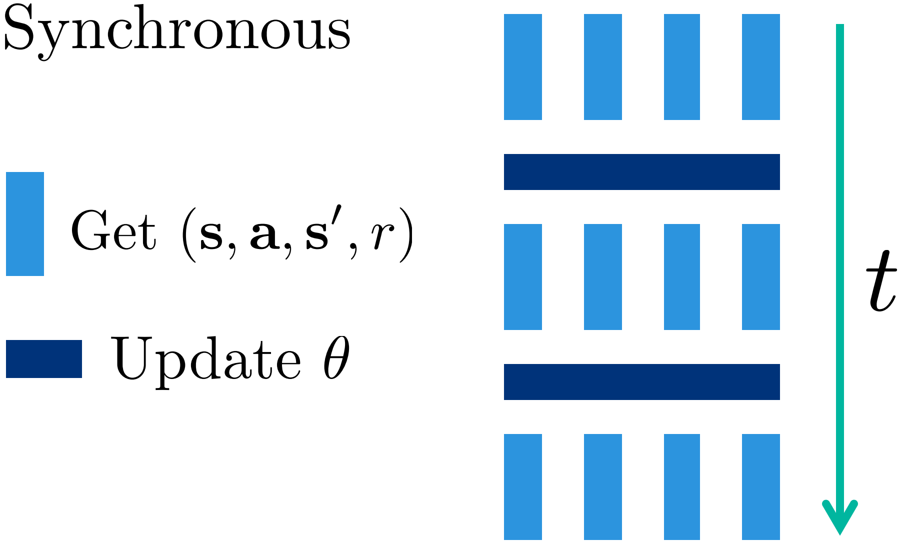
While not converged:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi(\mathbf{s})$ using target $r + \gamma \hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \hat{V}^\pi_\phi(\mathbf{s}') - \hat{V}^\pi_\phi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \longleftarrow \theta + \alpha \nabla_\theta J(\theta)$

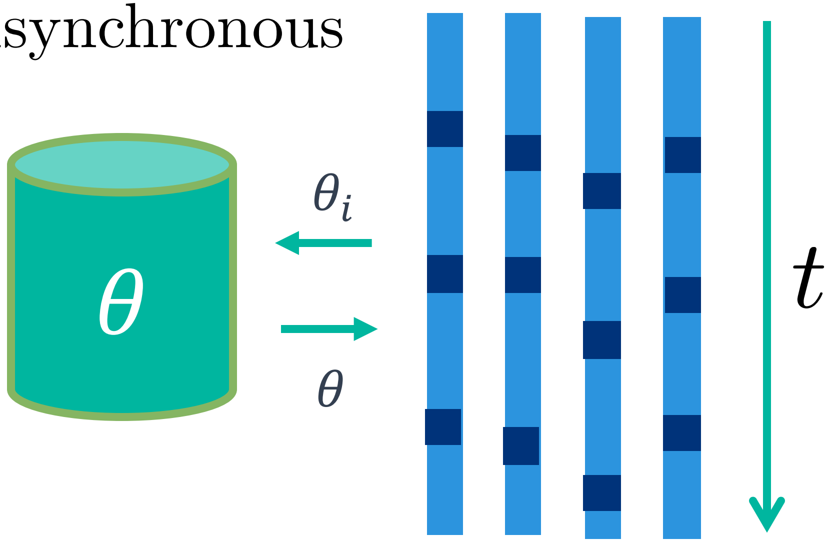
Work best with batches generated from multiple (parallel) workers

- Parallelization needed to get enough samples for batches
- Asynchronous: technically incorrect, but usually works in practice and faster.

Synchronous

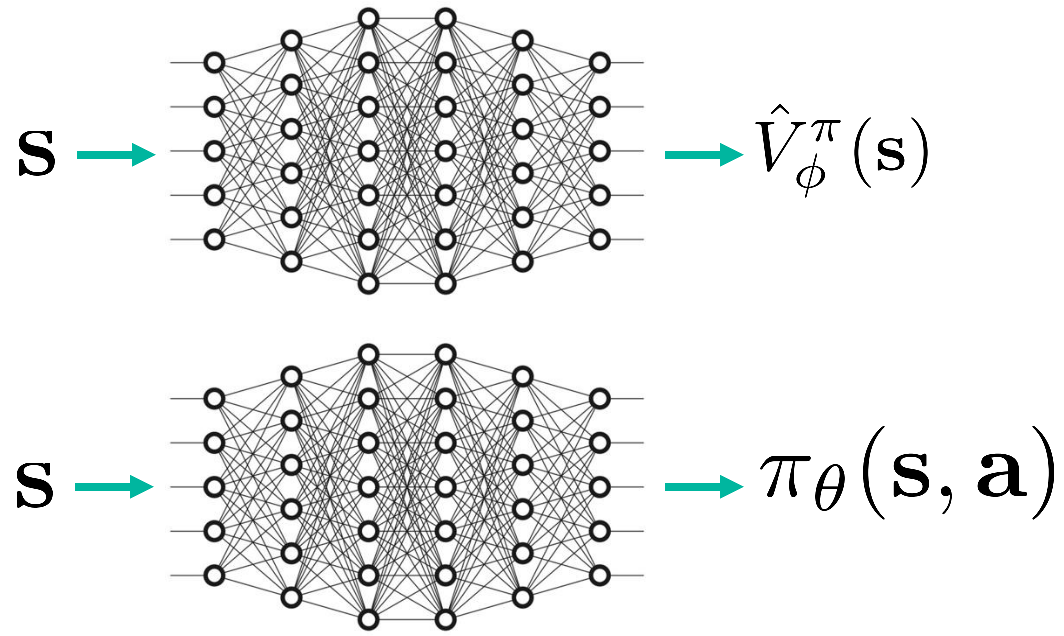


Asynchronous



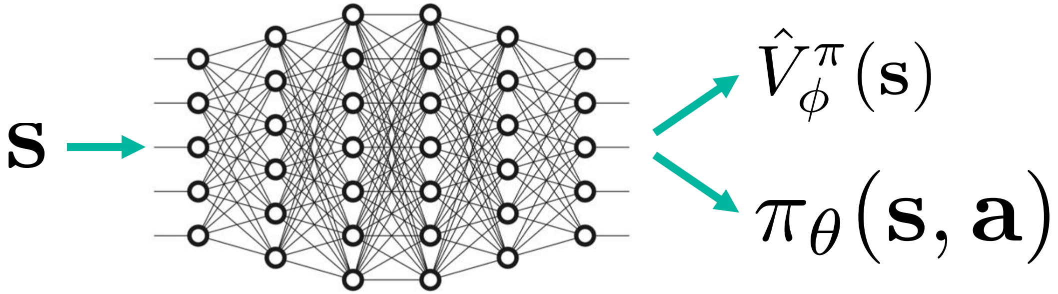
Architecture Design

Two Networks



- ➢ Simple, stable, no shared features

One Network



- ➢ Less stable, but also less features
- ➢ How to prevent interference between different objectives?

Bias-variance tradeoff

- Generalized Advantage Estimation

Critics as state-dependent baselines

Actor-critic: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(r(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1}) - \hat{V}_\phi^\pi(\mathbf{s}_{i,t})\right)$

- Lower variance (from critic)
- Biased (assuming critic is imperfect)

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

- Higher variance (single-sample estimate)
- Unbiased

Can we use $\hat{V}_\phi^\pi$ without introducing a bias?

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

- Use critic as baseline
- No bias
- Lower variance!

Eligibility traces, n-step returns

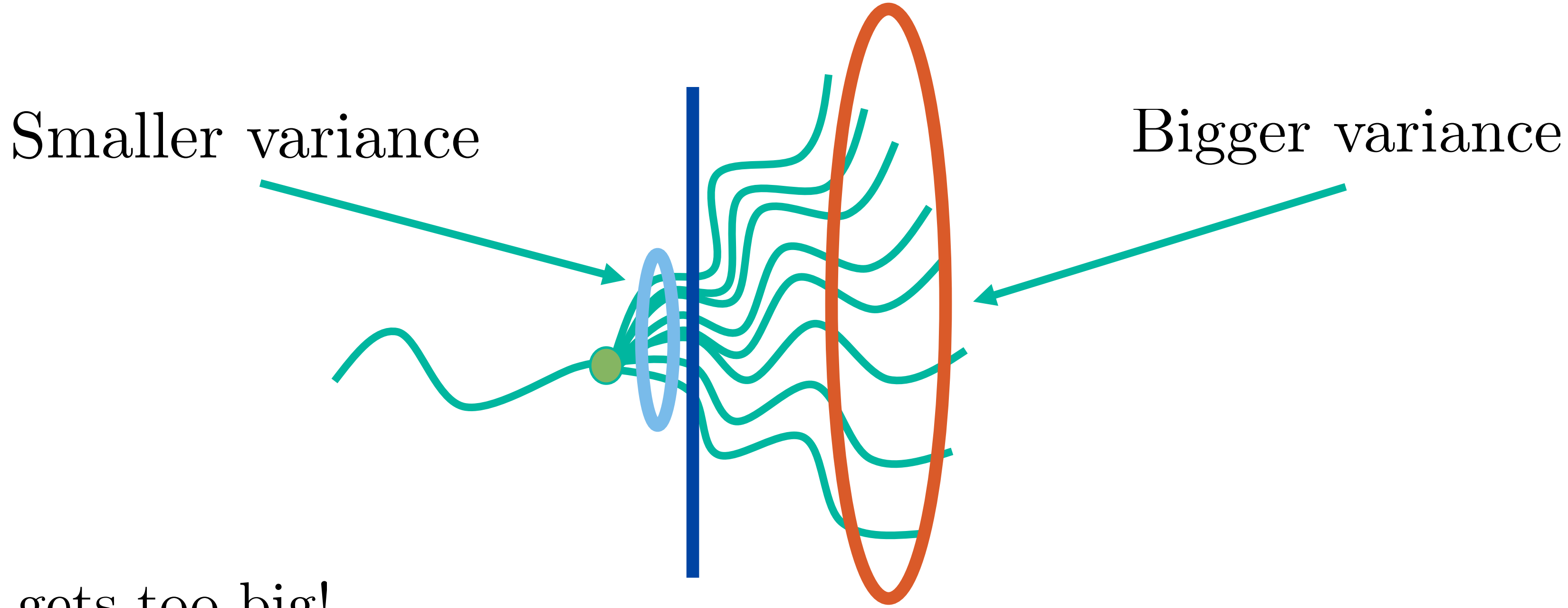
$$\hat{A}^\pi_C(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \hat{V}^\pi_\phi(\mathbf{s}_{t+1}) - \hat{V}^\pi_\phi(\mathbf{s}_t)$$

Bias, low variance

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

No bias, higher variance

Can we combine these two to control the trade-off between bias and variance?



Cut before variance gets too big!

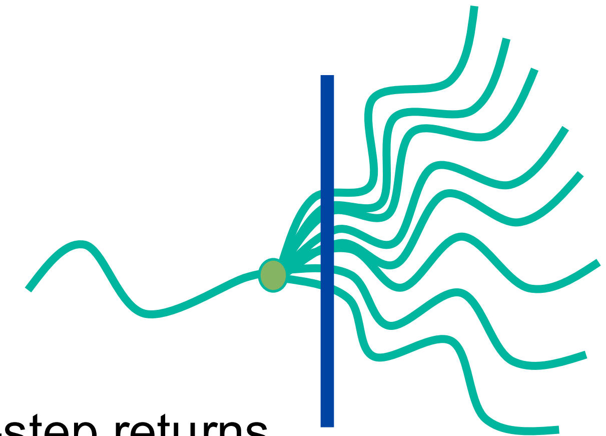
$$\hat{A}^\pi_n(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) + \gamma^n \hat{V}^\pi_\phi(\mathbf{s}_{t+n}) - \hat{V}^\pi_\phi(\mathbf{s}_t)$$

(Choosing n > 1 often works better)

Generalized Advantage Estimation

How do we know where to cut? Can we balance multiple cuts?

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



$$\hat{A}_{\mathrm{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{\infty} w_n \hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t)$$ Weighted combination of different n-step returns

How should we choose our weights?

Generally prefer reducing variance (cutting earlier) → $w_n \propto \lambda^{n-1}$

$$\hat{A}_{\mathrm{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = -\hat{V}_\phi^\pi(\mathbf{s}_t) + r(\mathbf{s}_t, \mathbf{a}_t) + \gamma(1-\lambda)\hat{V}_\phi^\pi(\mathbf{s}_{t+1}) + \lambda(r(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) + \gamma((1-\lambda)\hat{V}_\phi^\pi(\mathbf{s}_{t+2}) + \lambda r(\mathbf{s}_{t+2}, \mathbf{a}_{t+2}) + \ldots)$$

Can re-write as

$$\hat{A}_{\mathrm{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'}$$ where $$\delta_{t'} = r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{t'+1}) - \hat{V}_\phi^\pi(\mathbf{s}_{t'})$$

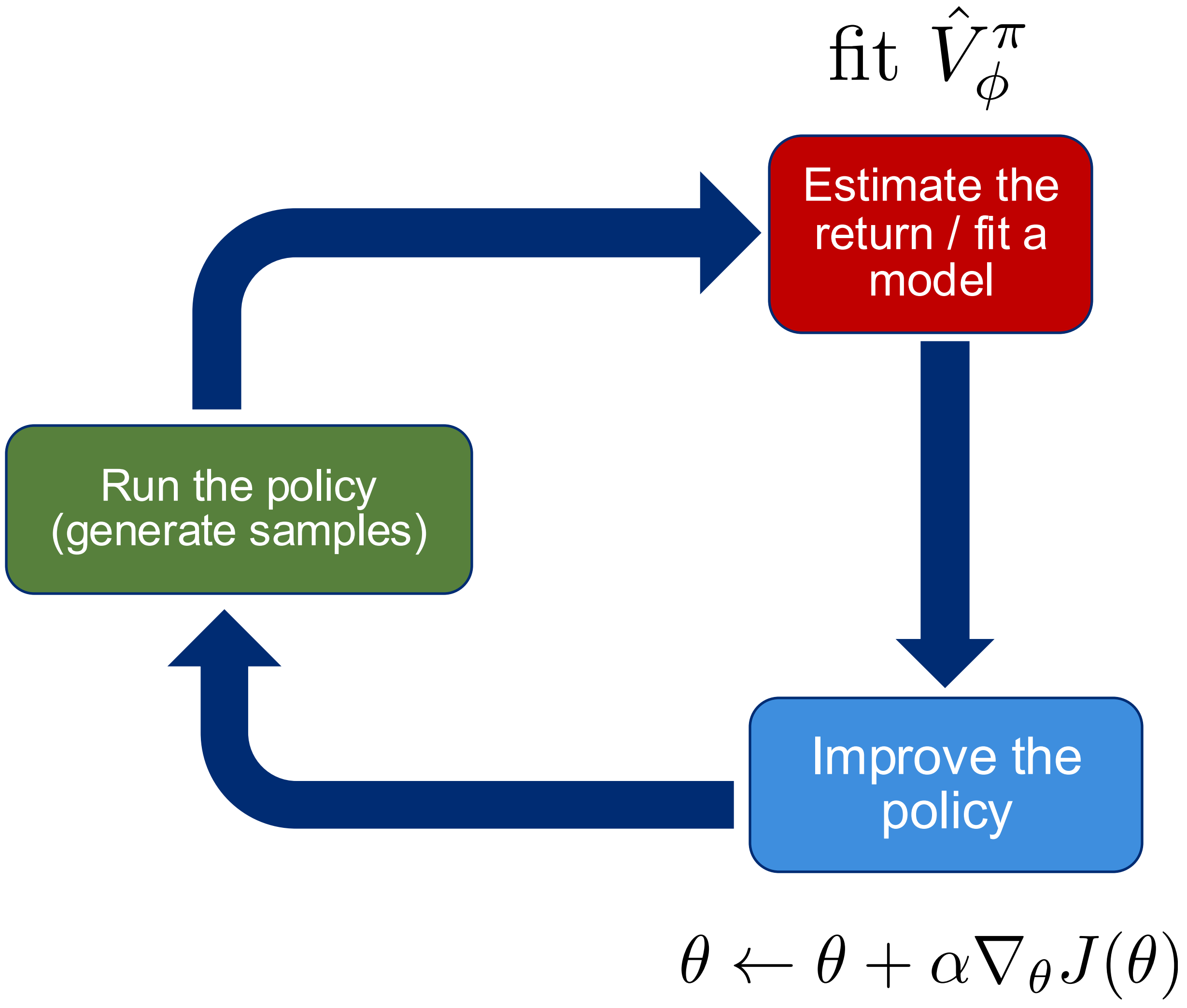
Actor-critic advantage estimate

γ, λ both contribute to bias-variance tradeoff!

J. Schulman et al. High-Dimensional Continuous Control Using Generalized Advantage Estimations. *arXiv: 1506.02438*.

Summary: Actor-Critic Learning

- **Actor-critic algorithms**
  - Actor: policy
  - Critic: value
- **Policy evaluation**
  - Via a value network
- **Discount factors**
  - Enable infinite horizon applications
  - Used for variance reduction
- **Actor-critic algorithm design**
  - Batch or Online
  - Architecture choices
- **State-dependent baselines**
  - Generalized Advantage Estimation



Advanced Policy Gradient

- VPG, TROP, PPO

Spinning Up page

Vanilla Policy Gradient

Algorithm 1 Vanilla Policy Gradient Algorithm

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

  Compute rewards-to-go $\hat{R}_t$.

  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

  Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

  Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

  or via another gradient ascent algorithm like Adam.

  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

  typically via some gradient descent algorithm.

end for

- Reduce bias through use of causality, baselining
- Can use Monte Carlo, bootstrapped, or hybrid (e.g. GAE) advantage estimates
- Monte Carlo estimates are unbiased but higher variance than boostrapped
- Actor-Critic refers to the use of at least some bootstrapping in baseline network

$\rightarrow$ On-policy

$\rightarrow$ Continuous or discrete actions

Comparing Policies

Policy gradient as **policy iteration**

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

$$\text{claim: } J(\theta') - J(\theta) = E_{\tau \sim p_{\theta'}(\tau)}\left[\sum_t \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)\right]$$

$$\begin{aligned}
J(\theta') - J(\theta) &= J(\theta') - E_{\mathbf{s}_0 \sim p(\mathbf{s}_0)}\left[V^{\pi_\theta}(\mathbf{s}_0)\right] \\
&= J(\theta') - E_{\tau \sim p_{\theta'}(\tau)}\left[V^{\pi_\theta}(\mathbf{s}_0)\right] \\
&= J(\theta') - E_{\tau \sim p_{\theta'}(\tau)}\left[\sum_{t=0}^{\infty} \gamma^t V^{\pi_\theta}(\mathbf{s}_t) - \sum_{t=1}^{\infty} \gamma^t V^{\pi_\theta}(\mathbf{s}_t)\right] \\
&= J(\theta') + E_{\tau \sim p_{\theta'}(\tau)}\left[\sum_{t=0}^{\infty} \gamma^t (\gamma V^{\pi_\theta}(\mathbf{s}_{t+1}) - V^{\pi_\theta}(\mathbf{s}_t))\right] \\
&= E_{\tau \sim p_{\theta'}(\tau)}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right] + E_{\tau \sim p_{\theta'}(\tau)}\left[\sum_{t=0}^{\infty} \gamma^t (\gamma V^{\pi_\theta}(\mathbf{s}_{t+1}) - V^{\pi_\theta}(\mathbf{s}_t))\right] \\
&= E_{\tau \sim p_{\theta'}(\tau)}\left[\sum_{t=0}^{\infty} \gamma^t (r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V^{\pi_\theta}(\mathbf{s}_{t+1}) - V^{\pi_\theta}(\mathbf{s}_t))\right] \\
&= E_{\tau \sim p_{\theta'}(\tau)}\left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)\right]
\end{aligned}$$

Comparing Policies

Policy gradient as **policy iteration**

$$J(\theta') - J(\theta) = E_{\tau \sim p_{\theta'}(\tau)}\left[\sum_t \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)\right]$$

expectation under $\pi_{\theta'}$ — advantage under π_θ

importance sampling

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

$$\begin{aligned} E_{\tau \sim p_{\theta'}(\tau)}\left[\sum_t \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)\right] &= \sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)}\left[E_{\mathbf{a}_t \sim \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}\left[\gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)\right]\right] \\ &= \sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)}\left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}\left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)\right]\right] \end{aligned}$$

is it OK to use $p_\theta(\mathbf{s}_t)$ instead?

Comparing Policies

Ignoring distribution mismatch?

$$\sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right] \overset{?}{\approx} \underbrace{\sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]}_{\bar{A}(\theta')}$$

why do we want this to be true?

$$J(\theta') - J(\theta) \approx \bar{A}(\theta') \quad \Rightarrow \quad \theta' \leftarrow \arg\max_{\theta'} \bar{A}(\theta)$$

2. Use $\hat{A}^\pi(\mathbf{s}_t, \mathbf{a}_t)$ to get *improved* policy π'

$$\theta' \leftarrow \arg\max_{\theta'} \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

such that $D_{\text{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)) \leq \epsilon$

for small enough ϵ, this is guaranteed to improve $J(\theta') - J(\theta)$

Trust Region Policy Optimization (TRPO)

J. Schulman et al. Trust Region Policy Optimization. *arXiv:1502.05477* (2015).

Spinning Up page

→ Aims for fast, monotonic improvement

→ Constrains the difference between previous and current policy across observed states via Kullback-Leibler (KL) Divergence.

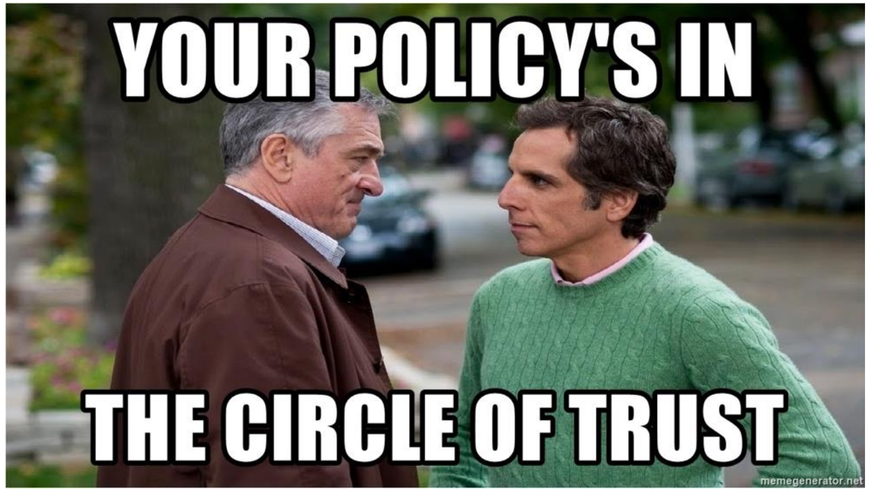
→ Optimizes constrained *surrogate* advantage:

$$\theta_{k+1} = \text{argmax}_\theta \mathcal{L}(\theta_k, \theta) \text{ s.t. } \bar{D}_{KL}(\theta||\theta_k) \leq \delta$$

$$L(\theta_k, \theta) = \mathrm{E}_{\mathbf{s},\mathbf{a}\sim p_{\theta_k}} \left[\frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_k}(\mathbf{a}|\mathbf{s})} A^{\pi_{\theta_k}}(\mathbf{s}, \mathbf{a}), \right]$$

$$\bar{D}_{KL}(\theta||\theta_k) = \mathrm{E}_{\tau\sim p_{\theta_k}} \left[D_{KL}(\pi_\theta(\cdot|\mathbf{s})||\pi_{\theta_k}(\cdot|\mathbf{s})) \right]$$

→ Approximations required to make a practical algorithm.



→ On-policy

→ Continuous or discrete actions

TRPO: Pseudocode

Algorithm 1 Trust Region Policy Optimization

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps K
for $k = 0, 1, 2, \ldots$ **do**
    Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
    Compute rewards-to-go $\hat{R}_t$.
    Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
    Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

    Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$$

    where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.
    Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$

    where $j \in \{0, 1, 2, \ldots K\}$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.
    Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

    typically via some gradient descent algorithm.
end for

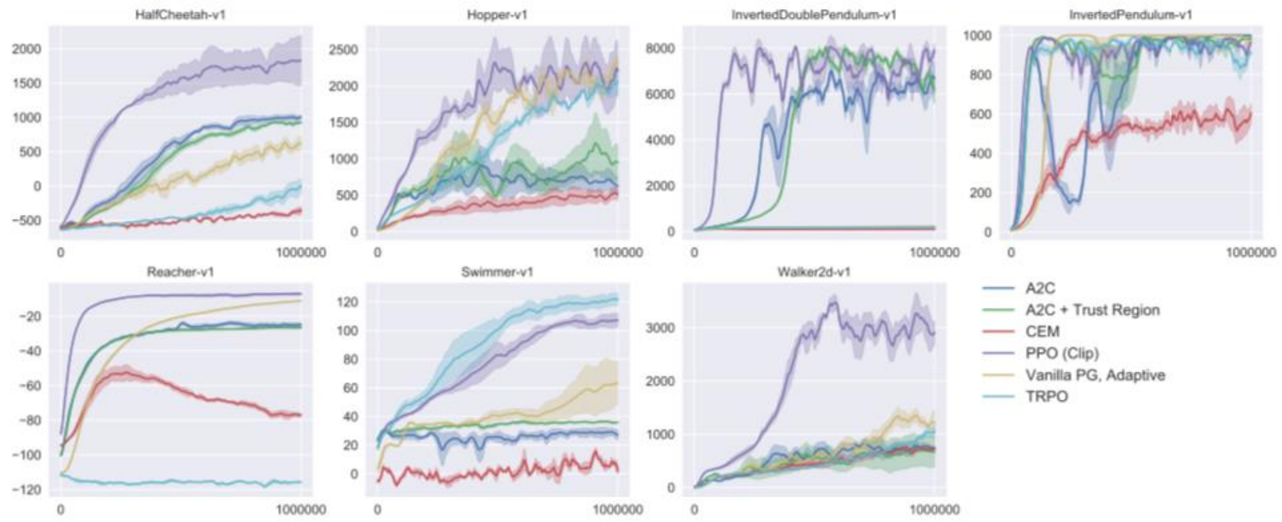
Proximal Policy Optimization (PPO)

J. Schulman et al. Proximal Policy Optimization Algorithms. *arXiv:1707.06347* (2017).
[Spinning Up page]

→ Same motivation as TRPO: fast and nearly monotonic improvement

→ Simpler than TRPO; first-order

→ Often works better than TRPO



→ Constrains update to not change policy too much, via either KL-divergence or clipping

→ Clipping more commonly used.

→ On-policy (approximately)

→ Continuous or discrete actions

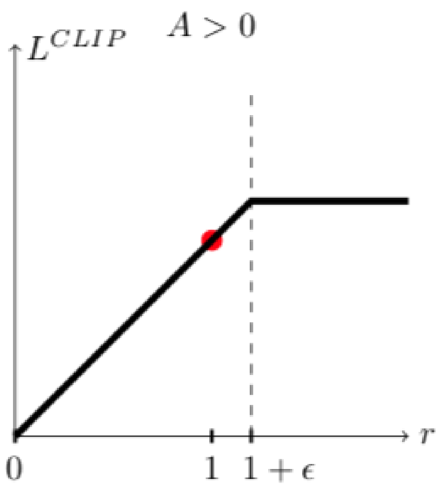
PPO Clipped Objective

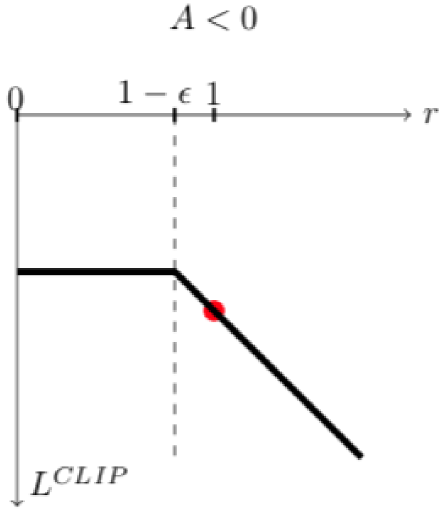
Policy:

$$L(\mathbf{s}, \mathbf{a}, \theta_k, \theta) = \min\left(\frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_k}(\mathbf{a}|\mathbf{s})} A^{\pi_{\theta_k}}(\mathbf{s}, \mathbf{a}), g(\epsilon, A^{\pi_{\theta_k}}(\mathbf{s}, \mathbf{a}))\right)$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & \text{if } A \geq 0 \\ (1-\epsilon)A & \text{if } A < 0 \end{cases}$$





Value:

$$L(\mathbf{s}) = \left(V_\phi(\mathbf{s}) - V^{\text{target}}\right)^2$$

Entropy (encourages exploration):

$$H(\mathbf{s}) = -\sum_i \pi(a_i|\mathbf{s}) \log \pi(a_i|\mathbf{s})$$

→ Replace with integrals for continuous case

- **Can use either combined or separate policy, value networks**
- **Combine loss terms in former case, separate in latter case**
- **Our PPO code does the latter**

Spinning Up page

PPO: Pseudocode for Two Flavors

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
  Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

  typically via stochastic gradient ascent with Adam.
  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

  typically via some gradient descent algorithm.
end for

Algorithm 1 PPO, Actor-Critic Style

for iteration$=1, 2, \ldots$ **do**
  for actor$=1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Clipped-Objective Policy Gradient

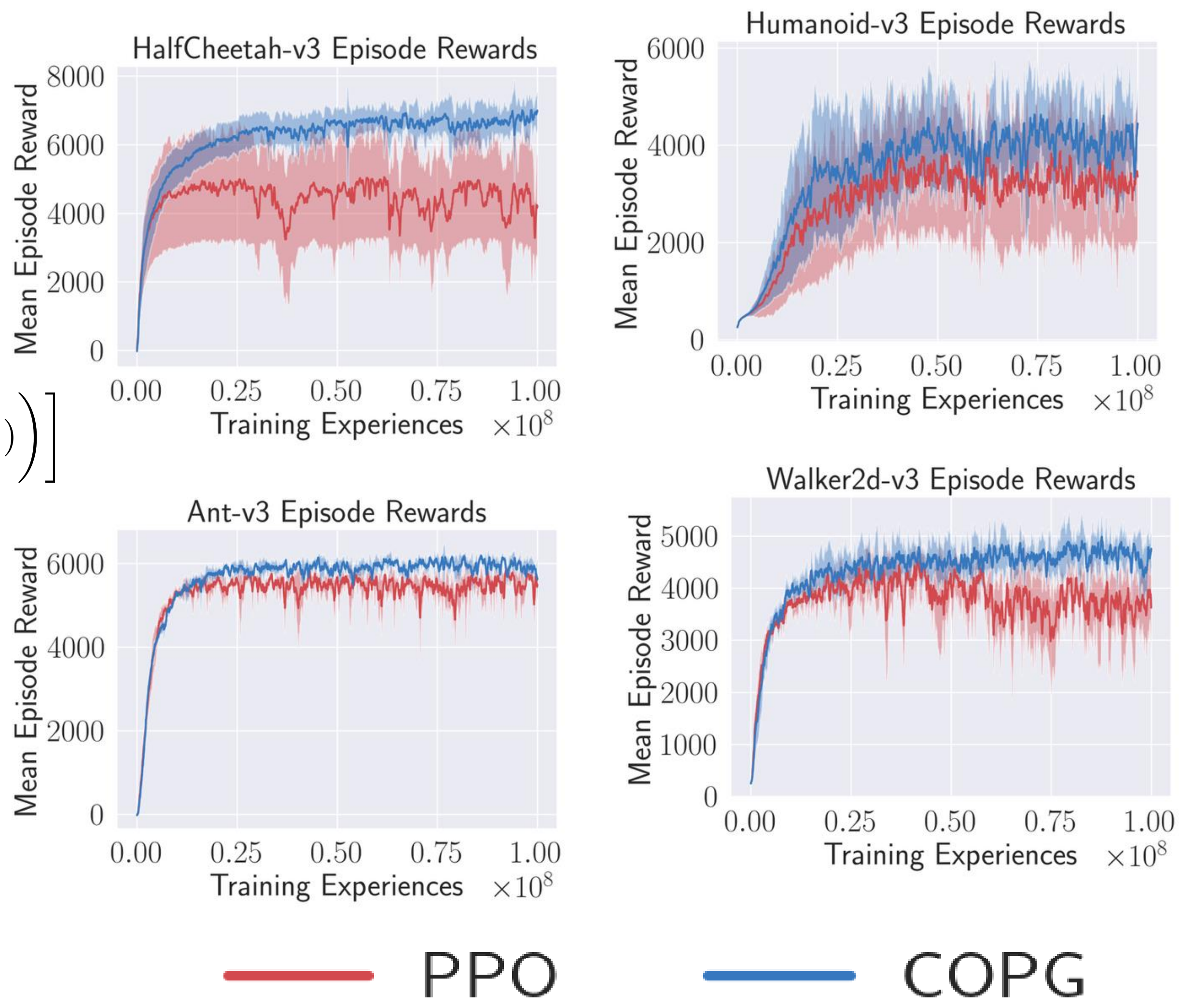
J. Markowitz, E.W. Staley. Clipped-Objective Policy Gradients for Pessimistic Policy Optimization. *arXiv:2311.05846v1*. 2023.

- We used a slightly-modified version of the PPO objective, performing similar **clipping on a *policy gradient* (rather than surrogate) objective**:

$$J_{\text{COPG}}(\theta') = \mathbb{E}_t\left[\min\left(\log \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\hat{A}^{\pi}(\mathbf{s}_t,\mathbf{a}_t), \log\left(\text{clip}\left(\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}, 1\pm\epsilon\right)\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\hat{A}^{\pi}(\mathbf{s}_t,\mathbf{a}_t)\right)\right]$$

- The result is a **slightly more pessimistic** method that consistently works better in continuous action spaces:

$$\frac{\nabla J_{\text{COPG}}}{\nabla J_{\text{PPO}}} = \begin{cases} \pi_\theta/\pi_{\theta'} & \text{if no clipping} \\ 1/(1\pm\epsilon) & \text{if clipping} \end{cases}$$



Fundamental Challenges in Policy Optimization

- Fractal Landscapes
- Mollification

Policy Optimization in Continuous Action Spaces

$$\max_{\theta} J(\theta)$$

Based on Policy Gradient:

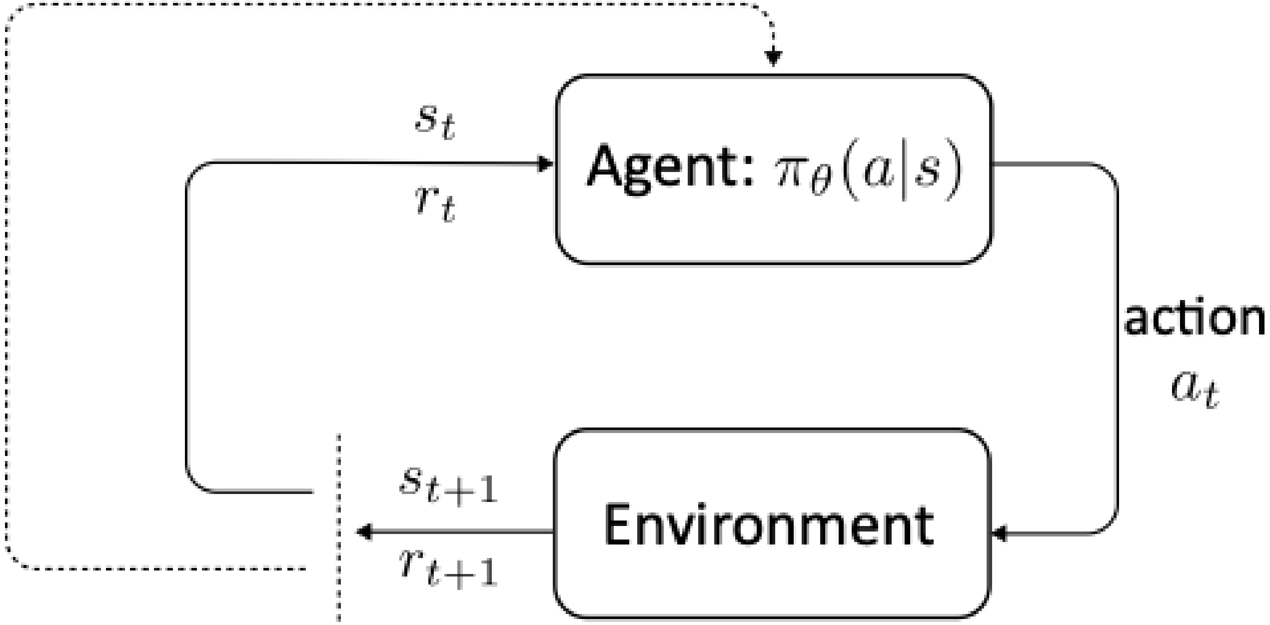
- Use experience to approximate $\nabla_\theta J(\theta) \approx \hat{g}$

$$\hat{g} := \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) \hat{R}_t^{(i)}$$

$$\hat{R}_t^{(i)} = \sum_{k=t}^{T} \gamma^{k-t} r_k^{(i)}$$

cumulative return

update: $\theta_{k+1} = \theta_k + \eta\,\hat{g}$



Many Challenges:

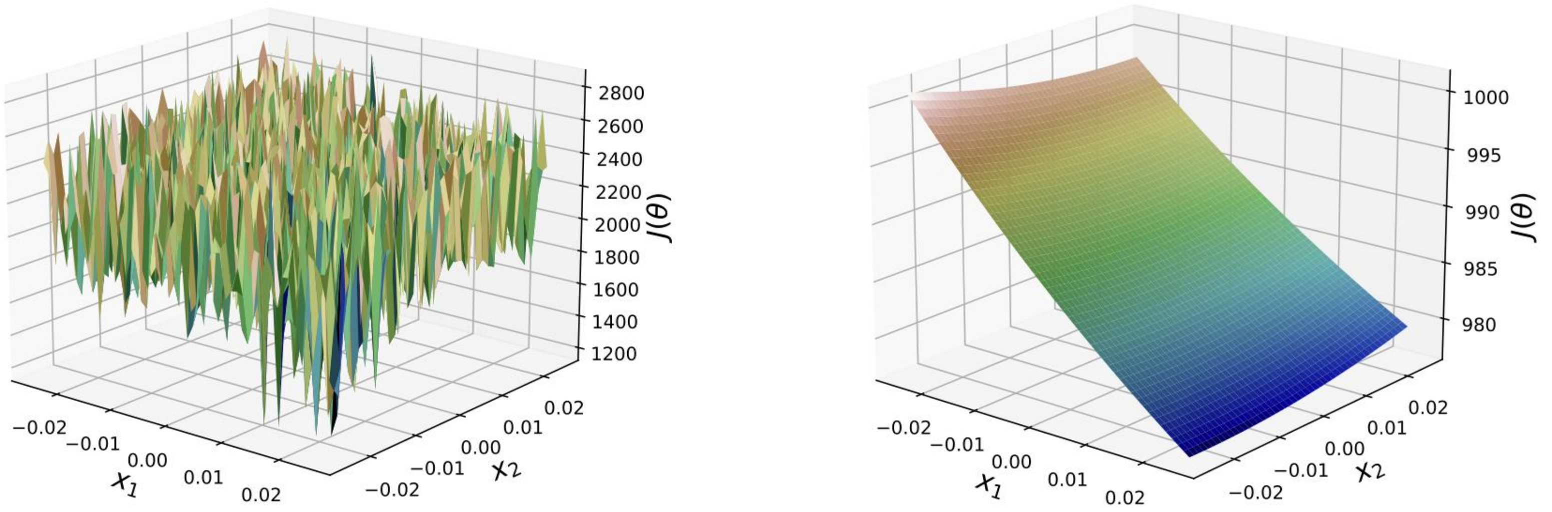
- Estimation variance
- Non-smoothness
- Fractal landscape
- Mollification
- …

Fractal Landscapes in Policy Optimization

- **Goal:** $\max_\theta \quad J(\theta) := E_{\pi_\theta, s_0 \sim \rho} \left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) \right]$

- **Approach:** $\theta_{k+1} = \theta_k + \eta \hat{\nabla}_\theta J(\theta)$

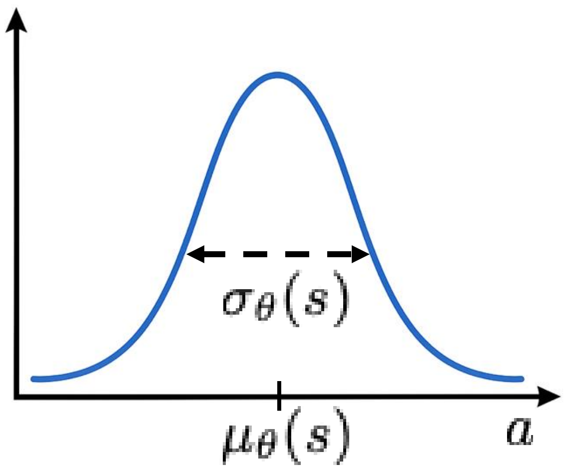
- **Challenge:** Optimization landscapes are not always "nice"

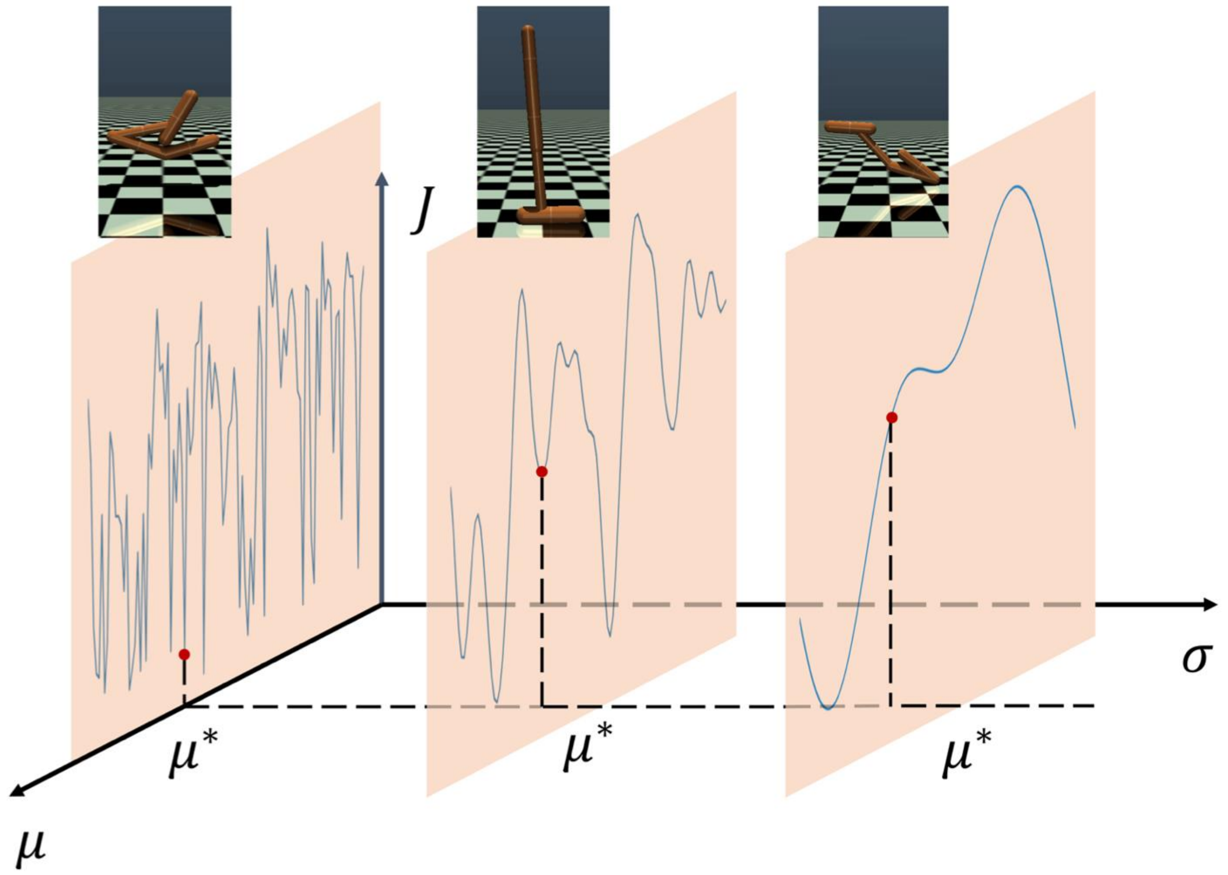


Tao Wang, Sylvia Hebert, Sicun Gao, Mollification effects of policy gradient, NeurIPS 23

Mollification of Policy Gradient

- **Goal:** $$\max_{\theta} J(\theta) := E_{s_0 \sim \rho, a_0 \sim \pi_\theta(s_0)} \left[Q^{\pi_\theta}(s_0, a_0) \right]$$
- **Policy:** $a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta^2 I)$





- **Policy Gradient**

$$\theta_{k+1} = \theta_k + \eta \hat{\nabla}_\theta J(\theta)$$

Tao Wang, Sylvia Hebert, Sicun Gao, Mollification effects of policy gradient, ICML 24

Agustin Castellano
Johns Hopkins University

Sohrab Rezaei
Johns Hopkins University

Jared Markowitz
Johns Hopkins Applied Physics Laboratory

Nonparametric policy improvement in continuous action spaces

A. Castellano, S. Rezaei, J. Markovitz, and E. Mallada, Nonparametric Policy Improvement for Continuous Action Spaces via Expert Demonstrations, 2025, submitted to Reinforcement Learning Conference.

Problem Setup

Goal: find optimal policy

$$\max_{\theta} J(\theta) := E_{s_0 \sim \rho, a_0 \sim \pi_\theta(s_0)} \left[Q^{\pi_\theta}(s_0, a_0) \right]$$

Problem Setup

Goal: find optimal **nonparametric** policy

$$\max_{\mathcal{D}} J(\pi_{\mathcal{D}}) := E_{s_0 \sim \rho, a_0 \sim \pi_{\mathcal{D}}(s_0)} \left[Q^{\pi_{\mathcal{D}}}(s_0, a_0) \right]$$

Data set: $\mathcal{D} = \{(s_i, a_i, Q_i)\}_{i=1}^{|\mathcal{D}|}$ $\quad Q_i := \sum_t \gamma^t r(s_t, a_t)$

Assumptions:

Optimal $Q^\star$ **is smooth:** $|Q^\star(s,a) - Q^\star(s',a')| \leq L(d_{\mathcal{S}}(s,s') + d_{\mathcal{A}}(a,a'))$

distance

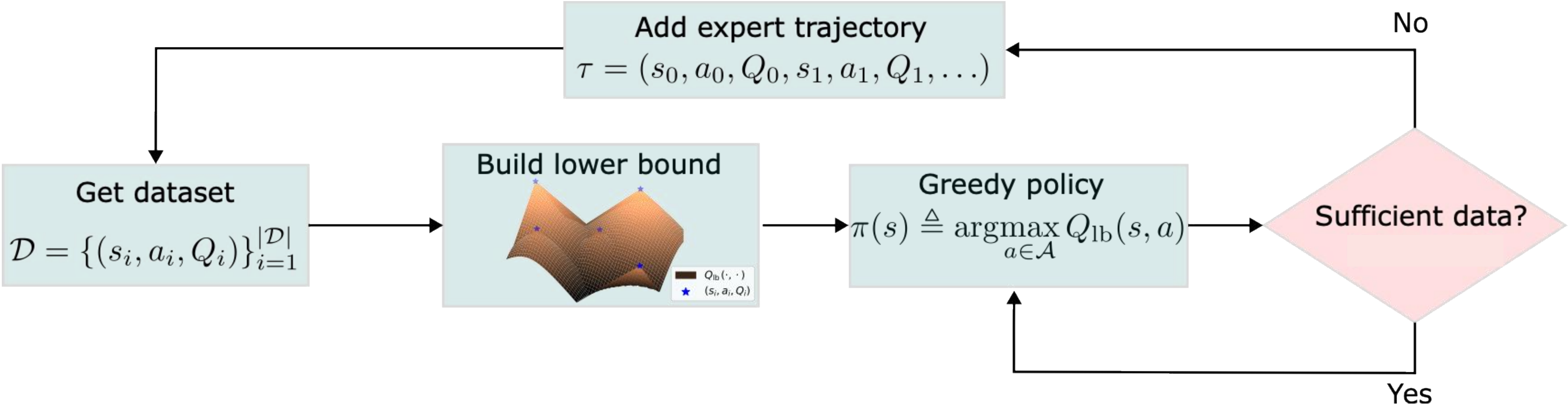
Deterministic dynamics: $s_{t+1} = f(s_t, a_t)$

Expert data: we have $\mathcal{D} = \{(s_i, a_i, Q_i)\}_{i=1}^{|\mathcal{D}|}$, where $a_i = \pi^\star(s_i); Q_i = Q^\star(s_i, a_i)$

Expert data: we have $\mathcal{D} = \{(s_i, a_i, Q_i)\}_{i=1}^{|\mathcal{D}|}$, where $a_i = \pi^\star(s_i); Q_i = Q^\star(s_i, a_i)$

1. **How** can we use these transitions to learn a nonparametric policy?
2. **What** guarantees can we get when we add more transitions?
3. **Where** should we add transitions to improve performance?

Overview of our method



1. How can we use these transitions to learn a nonparametric policy?

Building bounds & Nonparametric Policy

Expert data: we have $\mathcal{D} = \{(s_i, a_i, Q_i)\}_{i=1}^{|\mathcal{D}|}$, where $a_i = \pi^\star(s_i); Q_i = Q^\star(s_i, a_i)$

- Use the data to define **lower** bounds on optimal values:

$$V_{\text{lb}}(s) \triangleq \max_{1 \le i \le |\mathcal{D}|} \{Q_i - L \cdot d_{\mathcal{S}}(s, s_i)\} \qquad Q_{\text{lb}}(s,a) \triangleq \max_{1 \le i \le |\mathcal{D}|} \{Q_i - L \cdot (d_{\mathcal{S}}(s, s_i) + d_{\mathcal{A}}(a, a_i))\}$$

- **Nonparametric Policy:**

$$\pi(s) \triangleq \operatorname*{argmax}_{a \in \mathcal{A}} Q_{\text{lb}}(s,a) = a_{i'}$$

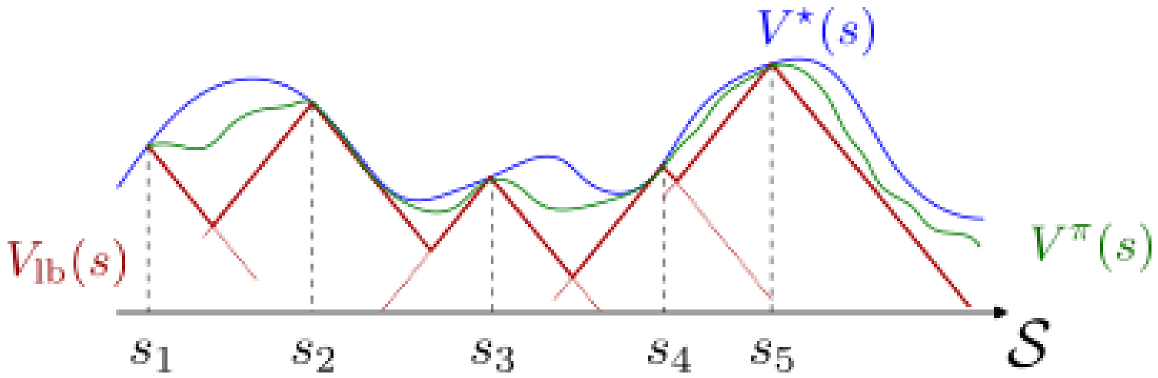
- **Remark:** Note argmax always gives actions in dataset $(s_{i'}, a_{i'}, Q_{i'})$
- **Question:** What can we say about $V^\pi(s)$?

Nonparametric policy *improves* over lower bound

Policy Evaluation:

- Nonparametric π satisfies $\forall s \in \mathcal{S}$:

$$V_{\text{lb}}(s) \leq V^{\pi}(s) \leq V^{\star}(s)$$



Policy Improvement:

- Given data sets $\mathcal{D}, \mathcal{D}'$ with $\mathcal{D} \subset \mathcal{D}'$

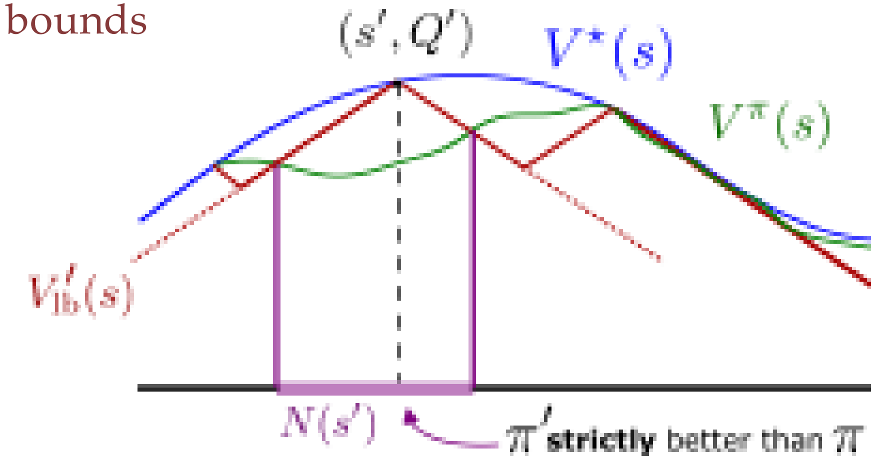
$$V_{\text{lb}}(s) \leq V'_{\text{lb}}(s) \quad \forall s \in \mathcal{S}$$

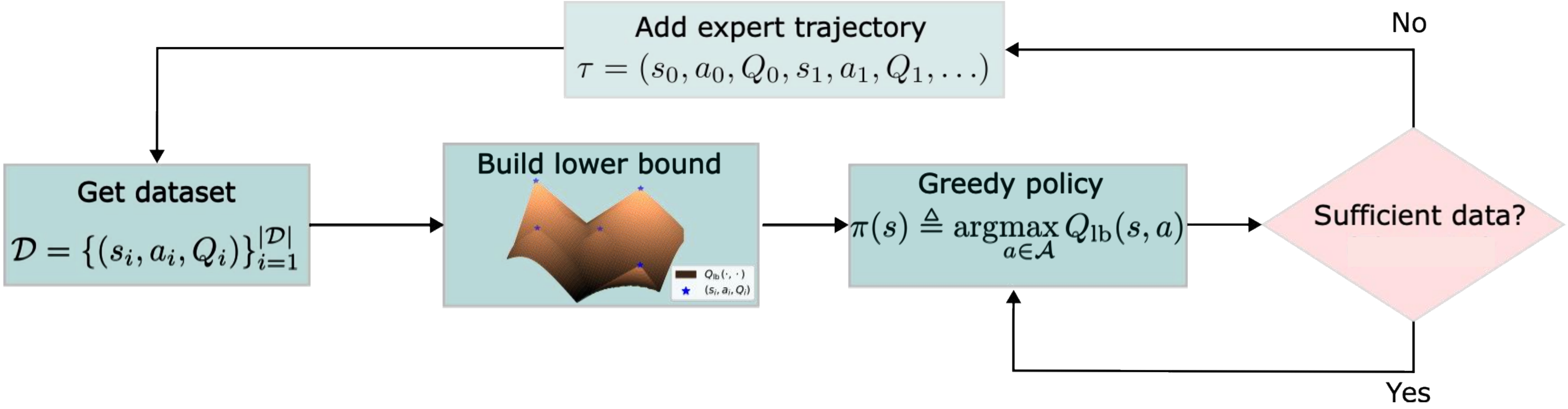
More data = better lower bounds

$$V^{\pi}(s') \leq V^{\pi'}(s') \quad \forall s' \in \mathcal{D}' \backslash \mathcal{D}$$

Improvement on added points

- **Strict** on neighbors of new data: $\forall s \in N(s')$





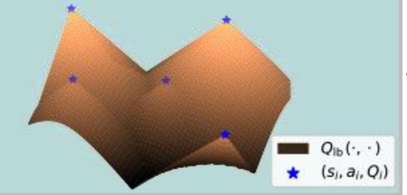
Add expert trajectory
$\tau = (s_0, a_0, Q_0, s_1, a_1, Q_1, \ldots)$
No
Get dataset
$\mathcal{D} = \{(s_i, a_i, Q_i)\}_{i=1}^{|\mathcal{D}|}$
Build lower bound
$Q_{lb}(\cdot,\cdot)$
(s_i, a_i, Q_i)
Greedy policy
$\pi(s) \triangleq \operatorname{argmax}_{a \in \mathcal{A}} Q_{\text{lb}}(s, a)$
Sufficient data?
Yes

1. **How** to learn a policy?

Get dataset

$\mathcal{D} = \{(s_i, a_i, Q_i)\}_{i=1}^{|\mathcal{D}|}$

→ **Build lower bound**



→ **Greedy policy**

$\pi(s) \triangleq \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_{\text{lb}}(s, a)$

→ **Policy Evaluation**

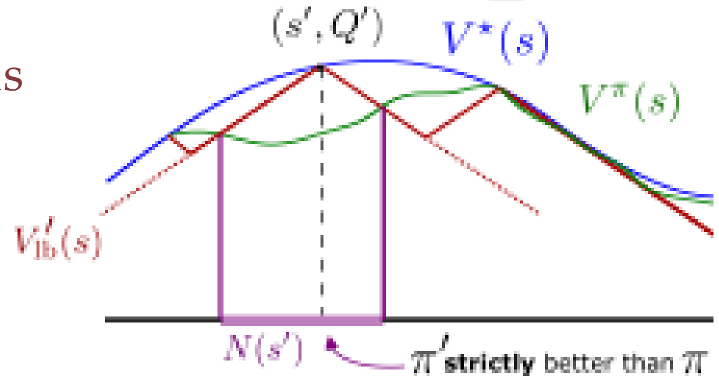
$V_{\text{lb}}(s) \le V^{\pi}(s)$

2. **What** guarantees with more transitions?

$V_{\text{lb}}(s) \le V'_{\text{lb}}(s) \quad \forall s \in \mathcal{S}$ — More data = better lower bounds

$V^{\pi}(s') \le V^{\pi'}(s') \quad \forall s' \in \mathcal{D}' \setminus \mathcal{D}$ — Improvement on added points

strict improvement



3. **Where** to add transitions?

- Only *where* **sufficient improvement** is guaranteed: $\Delta(s) := V_{\text{ub}}(s) - V_{\text{lb}}(s) > \varepsilon$

$$V_{\text{lb}}(s) \le V^{\pi}(s) \le V^{\star}(s) \le V_{\text{ub}}(s)$$

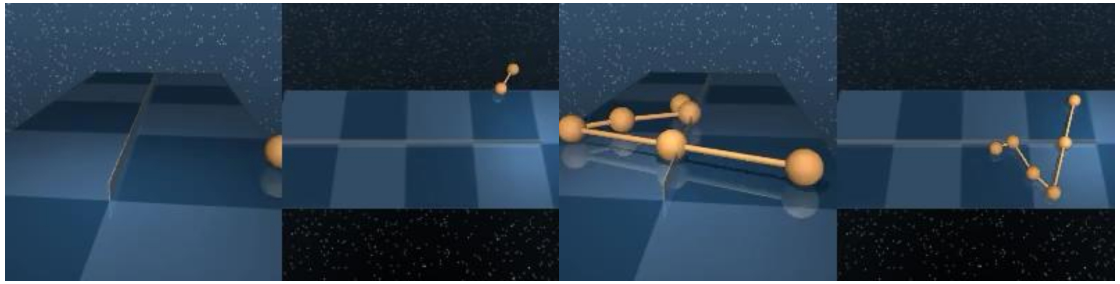
Experiments

Number of balls

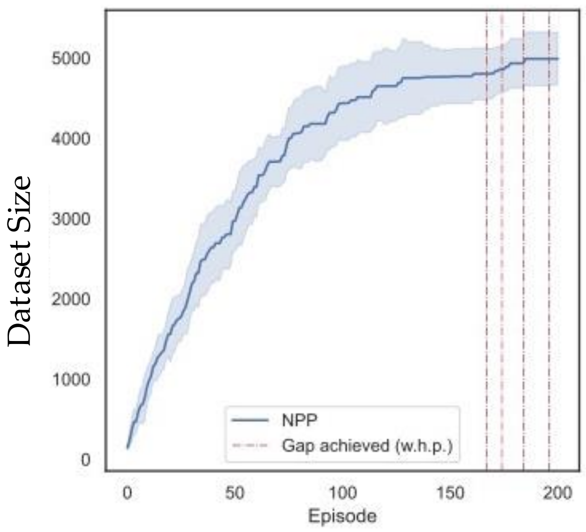
1st m actuated

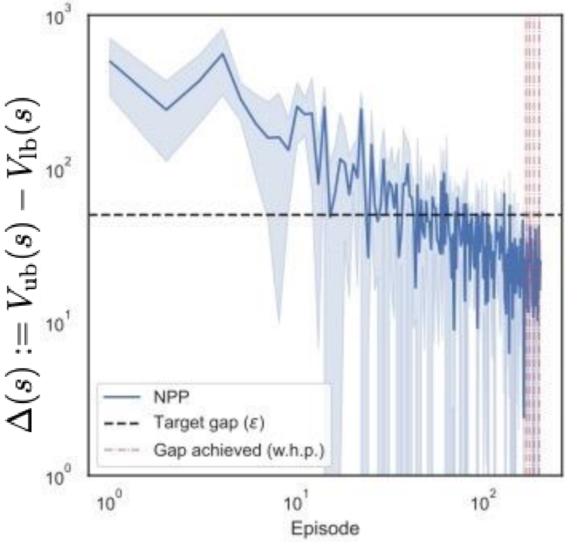
- We use the `lqr_n_m` environments from DeepMind's Control Suite

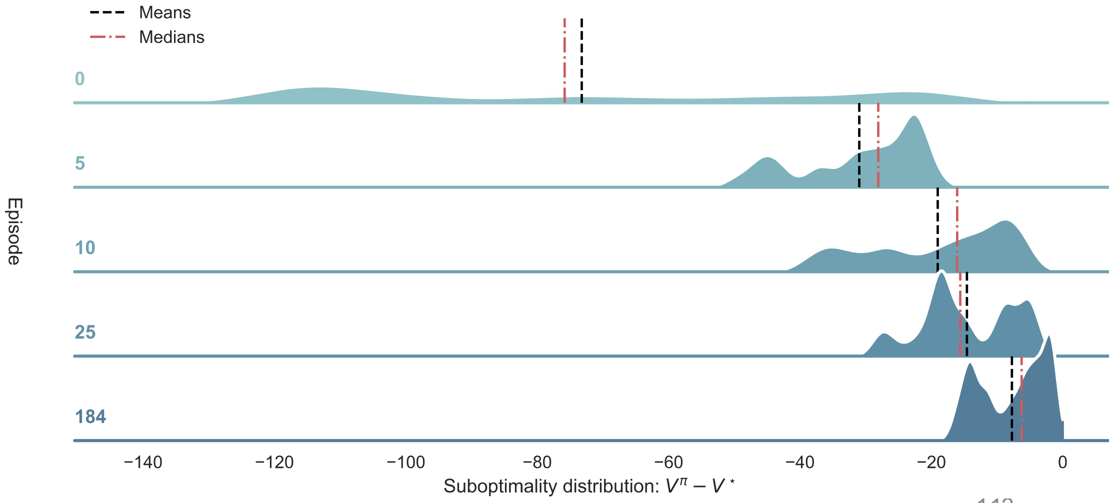
`lqr_2_1` **`lqr_6_2`**



- Results on **`lqr_2_1`**:

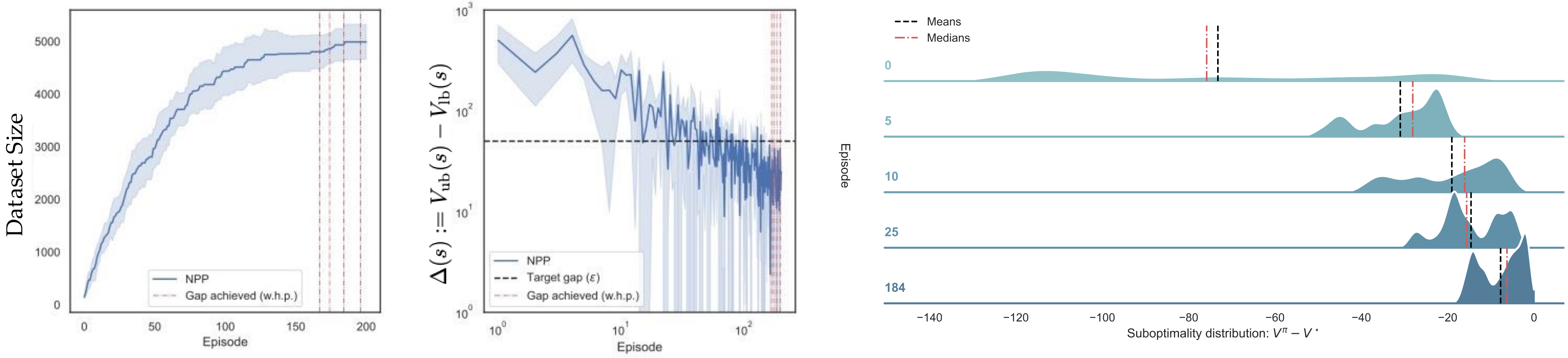






Experiments

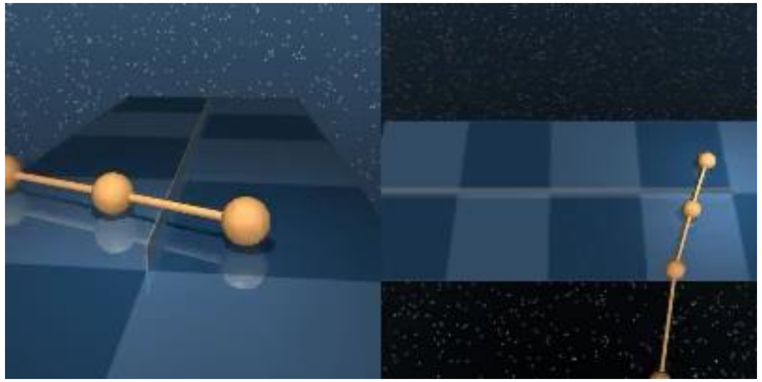
- We use the `lqr_n_m` environments from DeepMind's Control Suite
- Results on **`lqr_2_1`**:



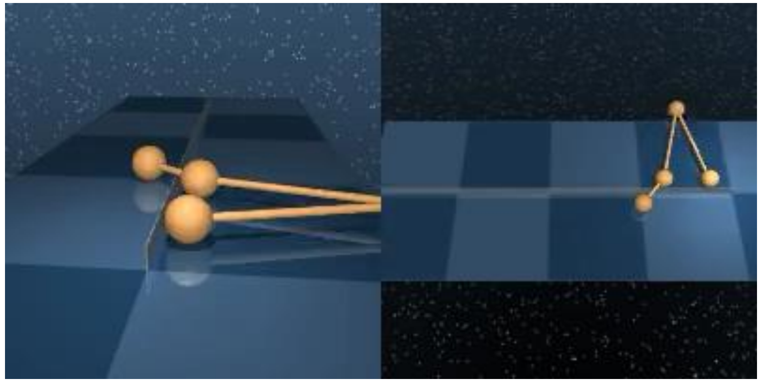
- **Remarks:**
  - **Incremental learning:** No catastrophic forgetting, or oscillations
  - Improvement across the entire state space (not in expectation)
  - Only valuable data is added (harder to find at times passes)

Incremental Learning

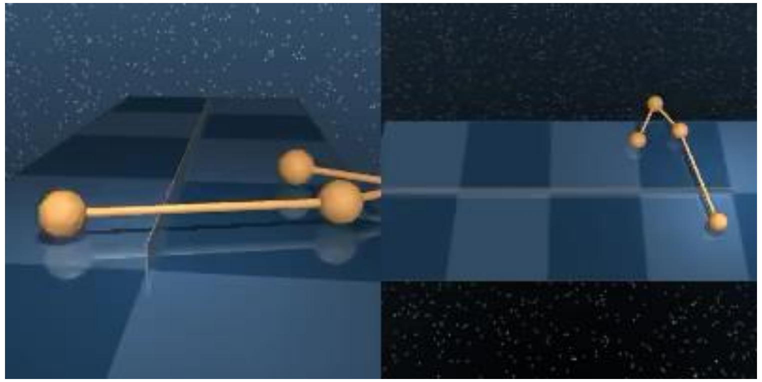
after 10 episode…



after 100 episode…



after 1000 episodes…



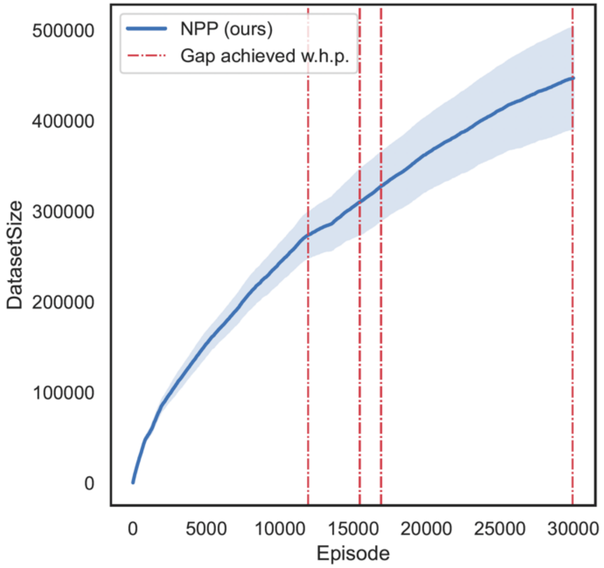
after 30K+

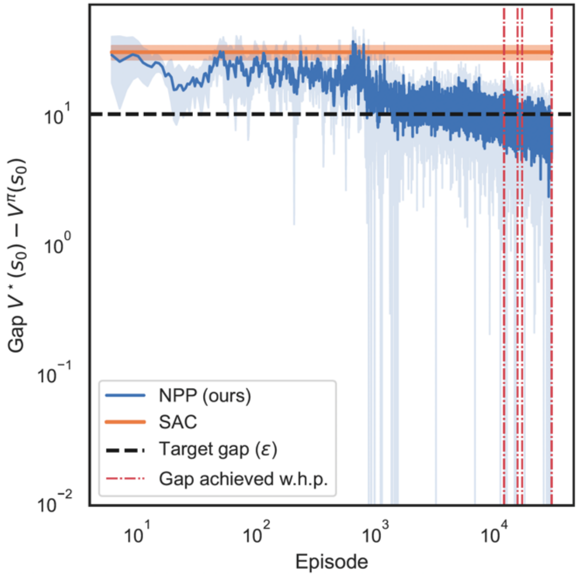
optimal control

Incremental Learning

after 30K+

optimal control





Thanks!

Enrique Mallada
mallada@jhu.edu
http://mallada.ece.jhu.edu