Introduction to Reinforcement Learning

ESE 6510
Antonio Loquercio

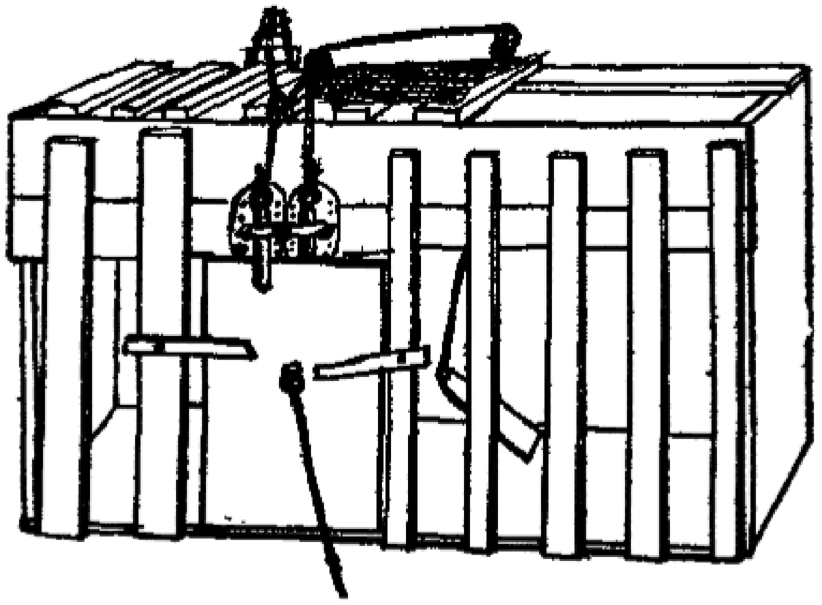
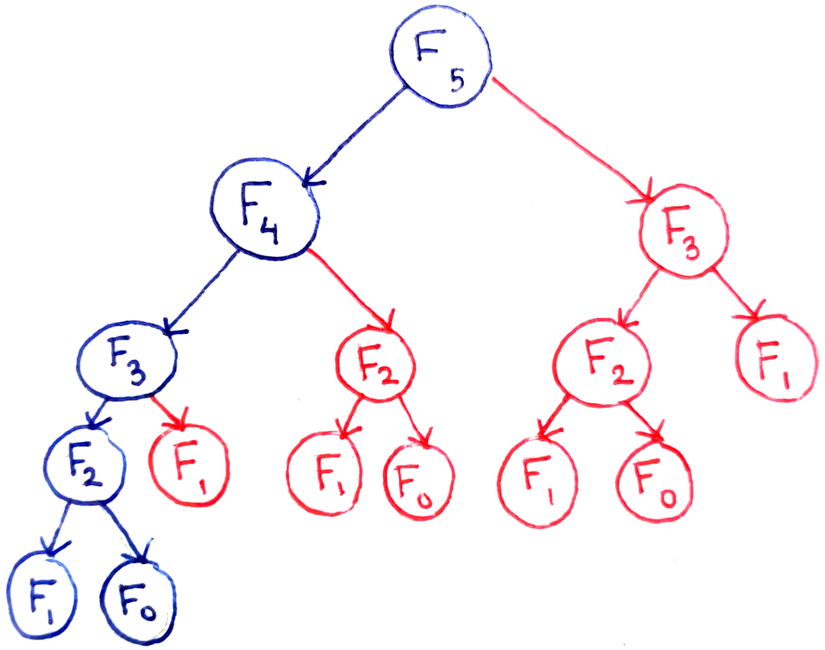


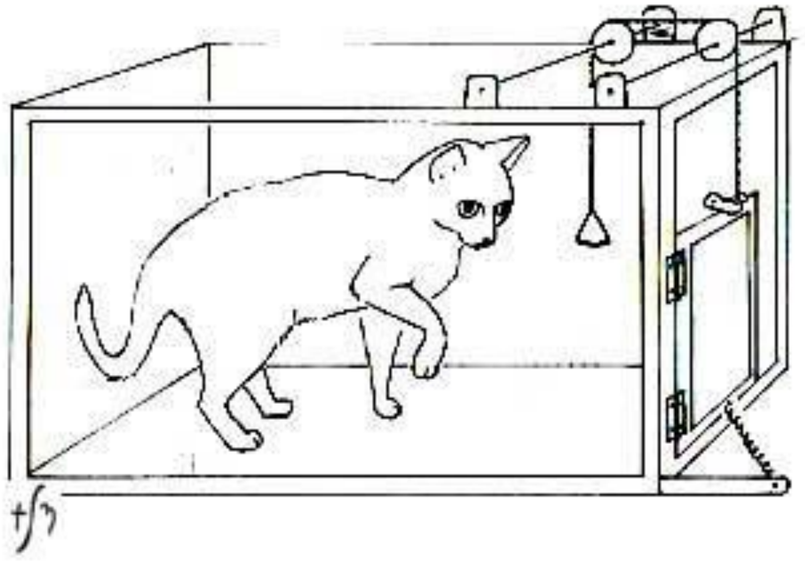
FIG. 1.

Thorndike, 1898
Animal Intelligence

Some History



Optimal Control



Trial and Error Learning

Edward Thorndike

- Interested in whether animals could learn tasks through imitation or observation
- Tested this by creating puzzles that animals had to solve to get food.
- Observed that different animals have similar learning curves (S-shaped)
- He called this “The Law of Effect”

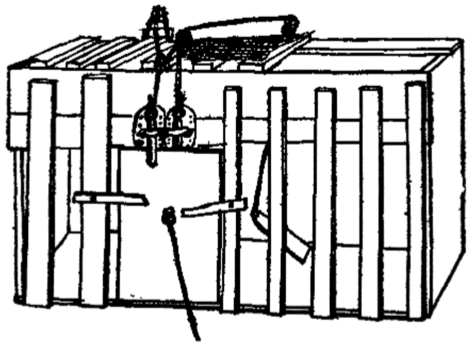


FIG. 1.

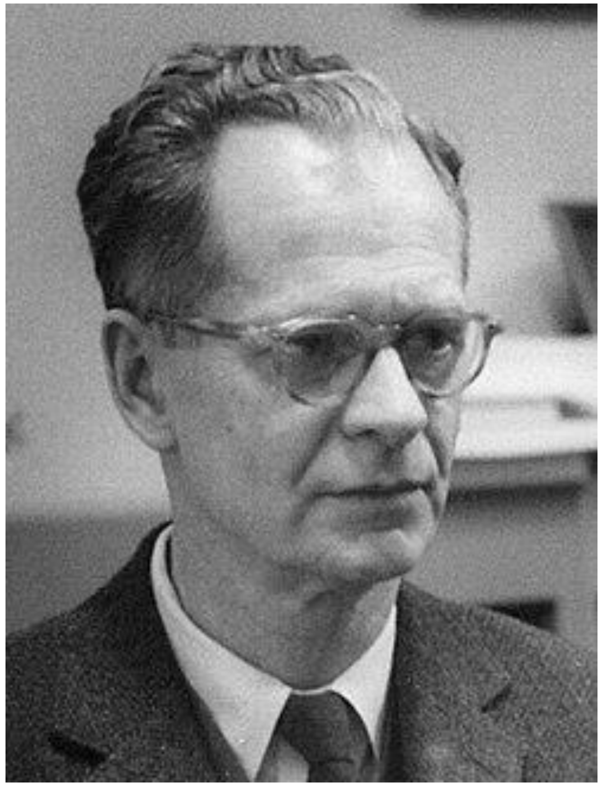
The Law of Effect and Connectionism

Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond. (Thorndike, 1911, p. 244)

Burrhus Frederic Skinner (c. 1950)

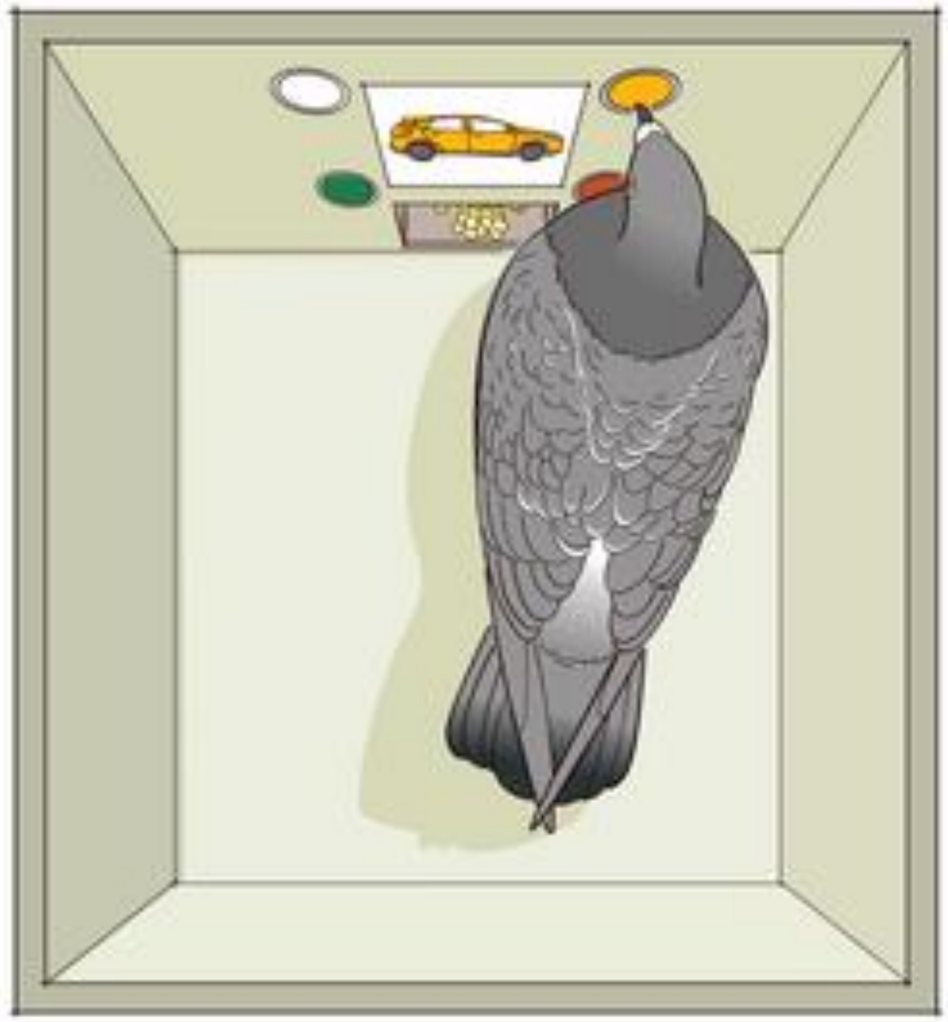
Extended the idea by asking three core questions:

- How do behaviors emerge in the first place?
  Variation and selection (shaping).
- Once in the organism's repertoire, how are they directed and used?
  The stimulus-response-reinforcer 3-way contingency
- How can very complex and seemingly novel behaviors be explained?
  Chaining. But it can't be the whole story…

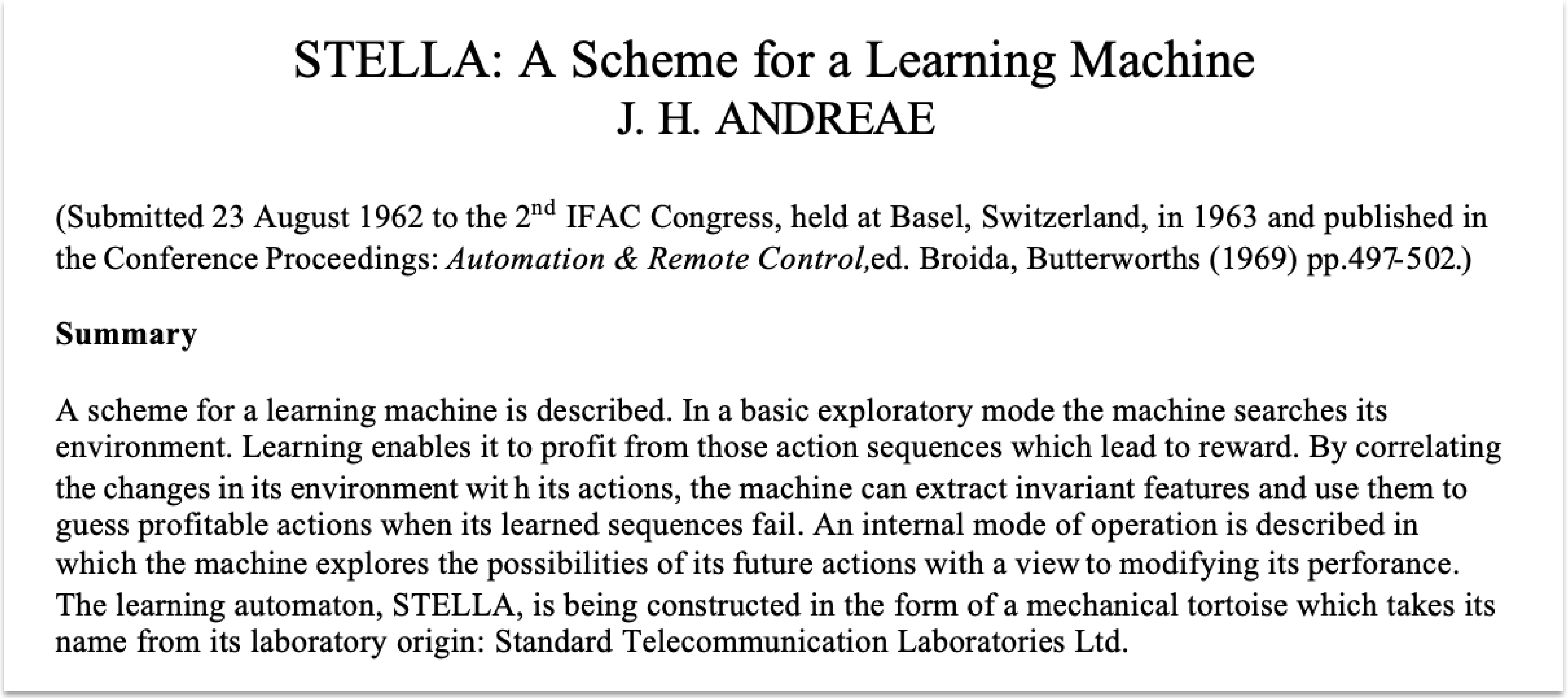


The Operant Conditioner Chamber (or Skinner Box)

- Expose the animal to carefully controlled stimuli
- Collect response rates and use them as a proxy for learning.
- Still, it oversimplifies the role of the environment on the agent.
- Trivia: Project Pigeon



Towards Engineering RL: Andreae's Stella (1962)



STELLA: A Scheme for a Learning Machine

J. H. ANDREAE

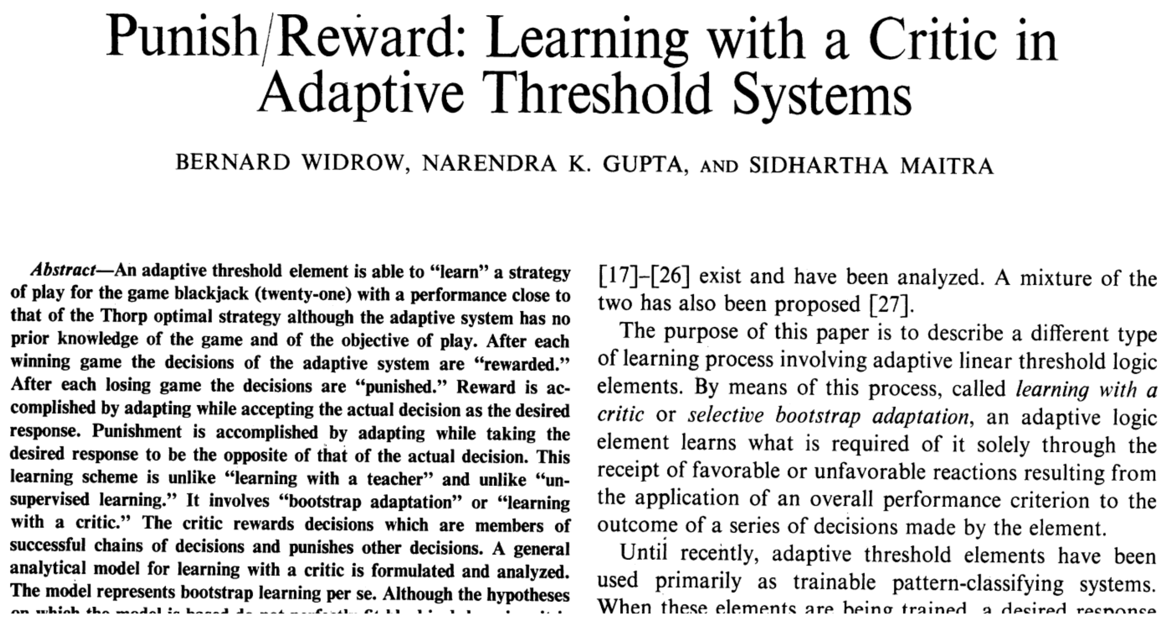
(Submitted 23 August 1962 to the 2nd IFAC Congress, held at Basel, Switzerland, in 1963 and published in the Conference Proceedings: *Automation & Remote Control,*ed. Broida, Butterworths (1969) pp.497-502.)

Summary

A scheme for a learning machine is described. In a basic exploratory mode the machine searches its environment. Learning enables it to profit from those action sequences which lead to reward. By correlating the changes in its environment wit h its actions, the machine can extract invariant features and use them to guess profitable actions when its learned sequences fail. An internal mode of operation is described in which the machine explores the possibilities of its future actions with a view to modifying its perforance. The learning automaton, STELLA, is being constructed in the form of a mechanical tortoise which takes its name from its laboratory origin: Standard Telecommunication Laboratories Ltd.

Stella (1962)

Going Beyond Supervised Learning



Punish/Reward: Learning with a Critic in Adaptive Threshold Systems

BERNARD WIDROW, NARENDRA K. GUPTA, AND SIDHARTHA MAITRA

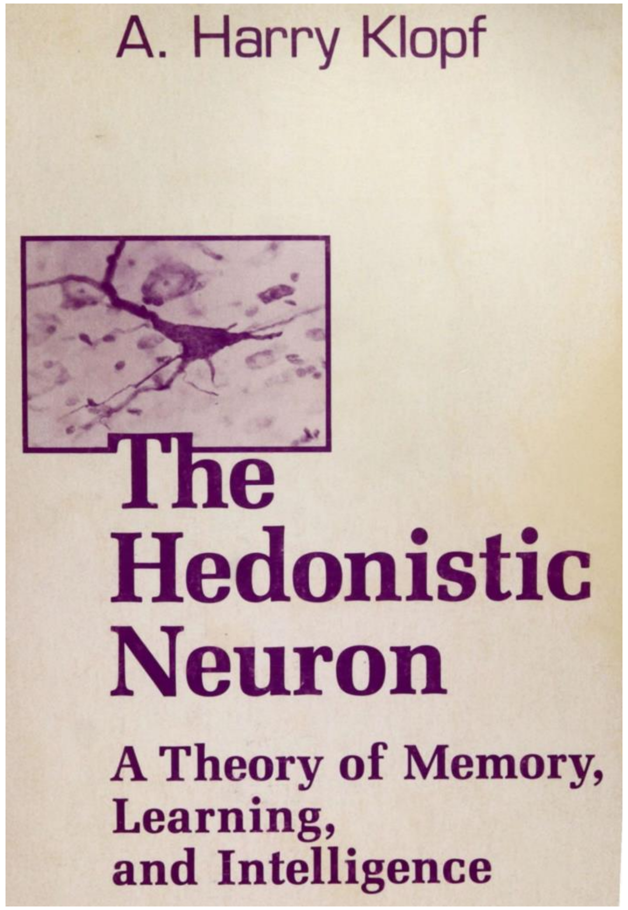
Abstract—An adaptive threshold element is able to "learn" a strategy of play for the game blackjack (twenty-one) with a performance close to that of the Thorp optimal strategy although the adaptive system has no prior knowledge of the game and of the objective of play. After each winning game the decisions of the adaptive system are "rewarded." After each losing game the decisions are "punished." Reward is accomplished by adapting while accepting the actual decision as the desired response. Punishment is accomplished by adapting while taking the desired response to be the opposite of that of the actual decision. This learning scheme is unlike "learning with a teacher" and unlike "unsupervised learning." It involves "bootstrap adaptation" or "learning with a critic." The critic rewards decisions which are members of successful chains of decisions and punishes other decisions. A general analytical model for learning with a critic is formulated and analyzed. The model represents bootstrap learning per se. Although the hypotheses

[17]–[26] exist and have been analyzed. A mixture of the two has also been proposed [27].

The purpose of this paper is to describe a different type of learning process involving adaptive linear threshold logic elements. By means of this process, called *learning with a critic* or *selective bootstrap adaptation*, an adaptive logic element learns what is required of it solely through the receipt of favorable or unfavorable reactions resulting from the application of an overall performance criterion to the outcome of a series of decisions made by the element.

Until recently, adaptive threshold elements have been used primarily as trainable pattern-classifying systems.

(1973)

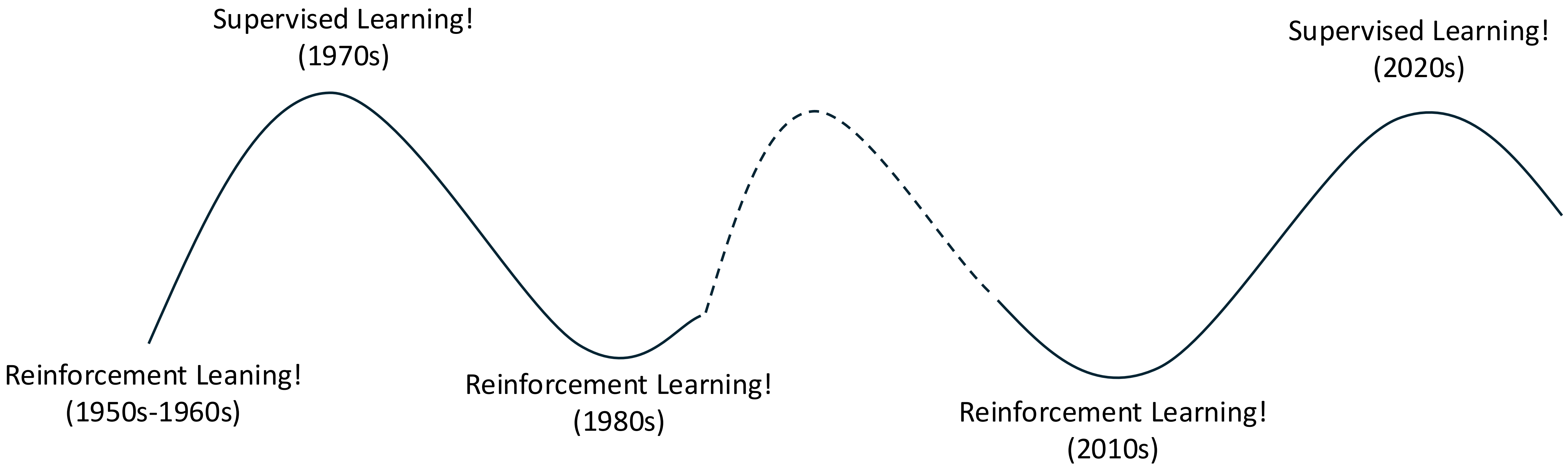


(1981)

Temporal Difference Learning

- Driven by the difference between temporally successive estimates of the same quantity (e.g., probability of winning in tic-tac-toe), connected to the idea of *secondary reinforcement* (from Klopf).
- Pioneered by Arthur Samuel in 1954.
- The *actor-critic architecture* (Sutton, Barto, Anderson, 1981): TD learning applied for trial-and-error problems (cartpole and games)
- Q-Learning (Watkins, 1989) Temporal-difference and optimal control threads are brought together.

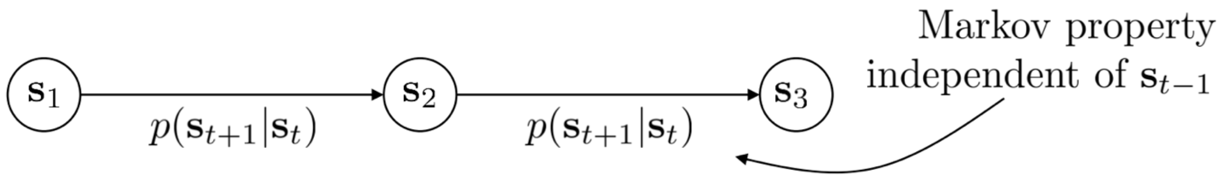
An Oversimplified History of Learning for AI Agents



Let’s Get Technical!

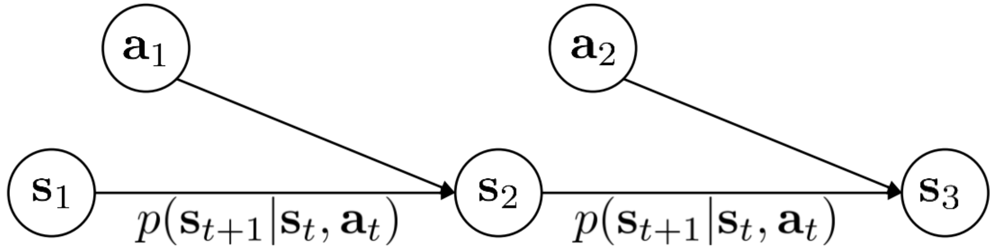
Definitions

- Markov Chain



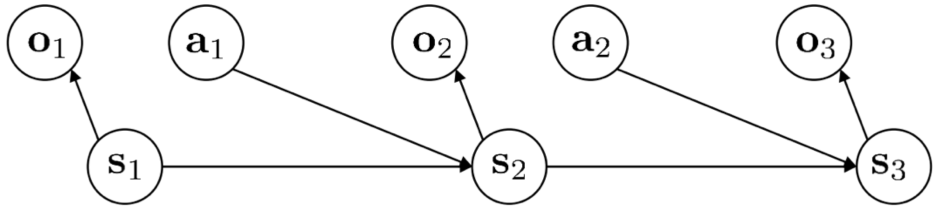
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t)$
Markov property independent of $\mathbf{s}_{t-1}$

- Markov Decision Process



$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

- Partially Observed Markov Decision Process



What do these things mean for a robot?

- What is a, p, o, r? What is s?

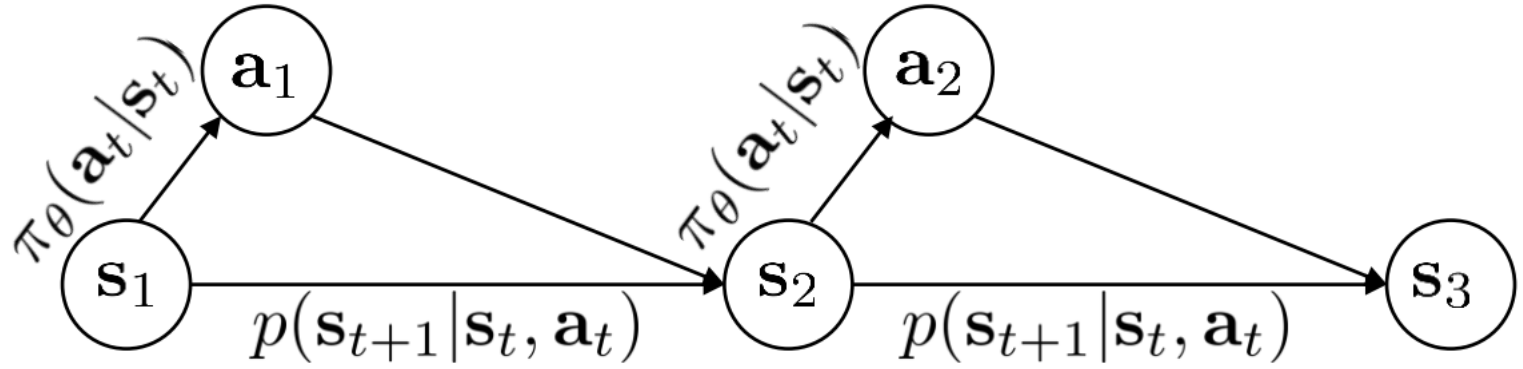
- s can be extremely large (possibly, infinite-dimensional)

Policies

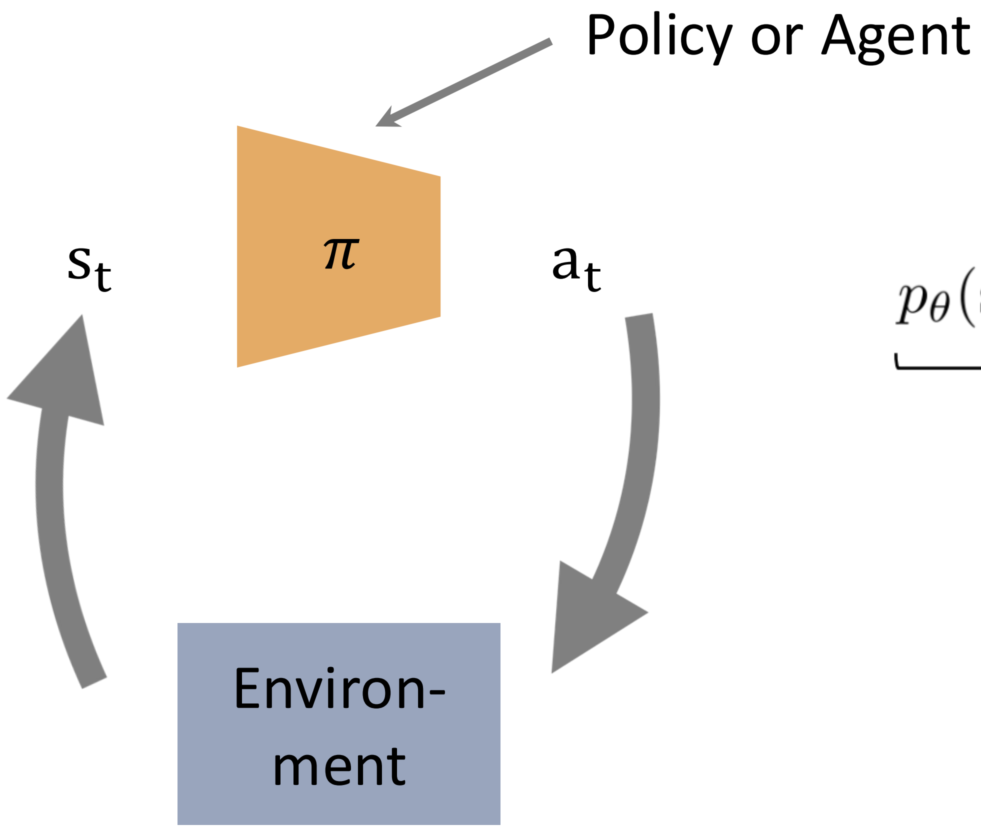
- We typically use parametrized stochastic policies, which we'll write as $\pi_\theta(\mathrm{a} \mid s)$. Can be a discrete or continuous distribution.

- θ is a parameter vector that specifies the policy.

- For multi-dimensional output, we rarely model off-diagonal terms, since the optimal policy of an MDP or POMDP is deterministic.


$\mathbf{a}_1$
$\mathbf{a}_2$
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

The goal of reinforcement learning



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

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

Adapted from S. Levine

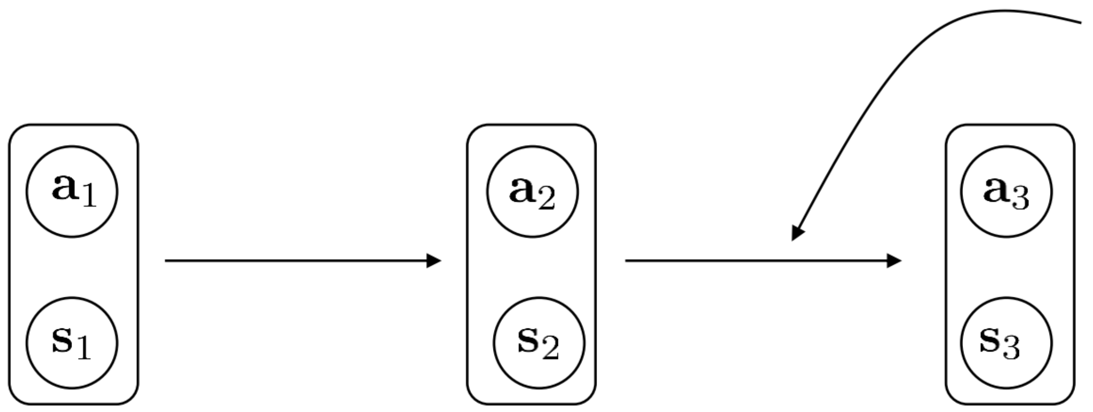
The finite horizon case

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

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

$p_\theta(\mathbf{s}_t,\mathbf{a}_t)$ **state-action marginal**

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



Adapted from S. Levine

Alternative Formulations

- Conditional Value at Risk (CVaR)

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

- Time-Varying Weight

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

The Core Issue

$$\begin{tikzcd} s_0 \arrow[r, "a"] & s_1 \end{tikzcd}$$

$$r(s_1, a) = A_{s1}$$

$$a = \pi_\theta(s_0)$$

$$r(s_1, a) = r\left(f(s_0, \pi_\theta(s_0))\right)$$

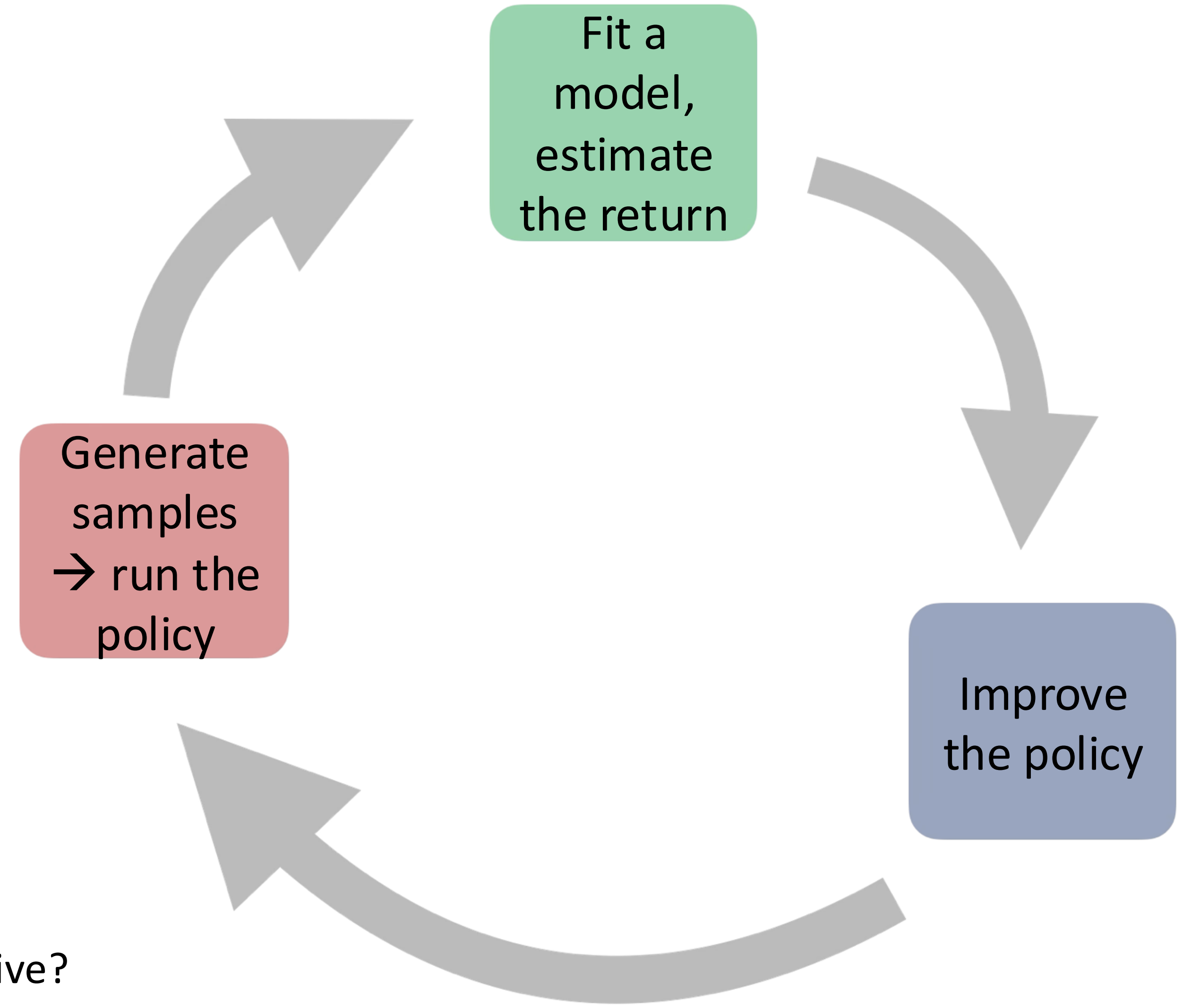
$$\frac{\partial r}{\partial \theta} = \frac{\partial r}{\pi_\theta} * \frac{\partial \pi_\theta}{\partial \theta}$$

$$\frac{\partial r}{\partial \pi} = \frac{\partial r}{\partial f} * \boxed{\frac{\partial f}{\pi}}$$

I don’t know f

f might not be differentiable

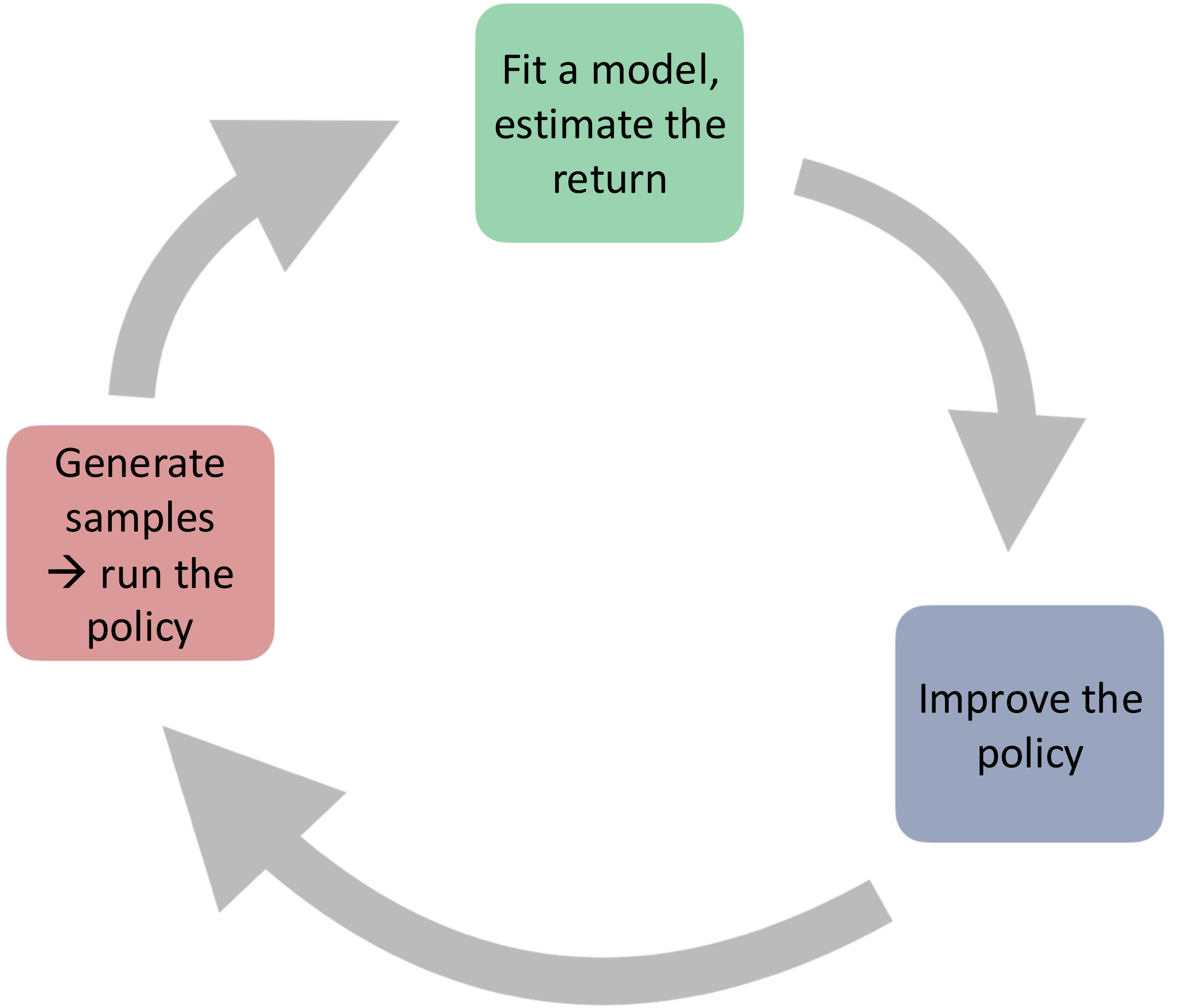
The general structure of an RL algorithm


Fit a model, estimate the return
Improve the policy
Generate samples → run the policy

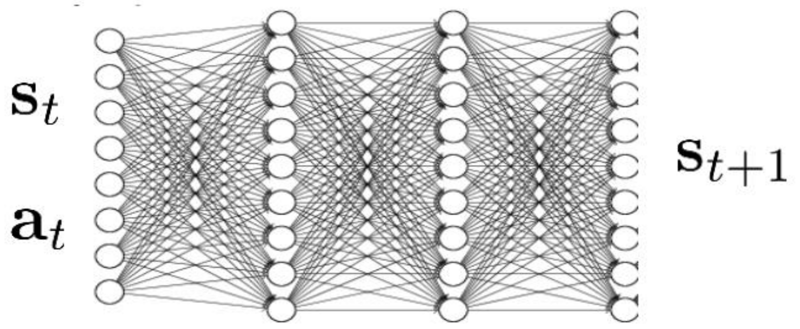
Which parts are expensive?

Adapted from S. Levine

Example: Learning the Model (Model-Based RL)


Fit a model, estimate the return
Improve the policy
Generate samples → run the policy

learn f_ϕ such that $\mathbf{s}_{t+1} \approx f_\phi(\mathbf{s}_t, \mathbf{a}_t)$



backprop through f_ϕ and r to train $\pi_\theta(\mathbf{s}_t) = \mathbf{a}_t$

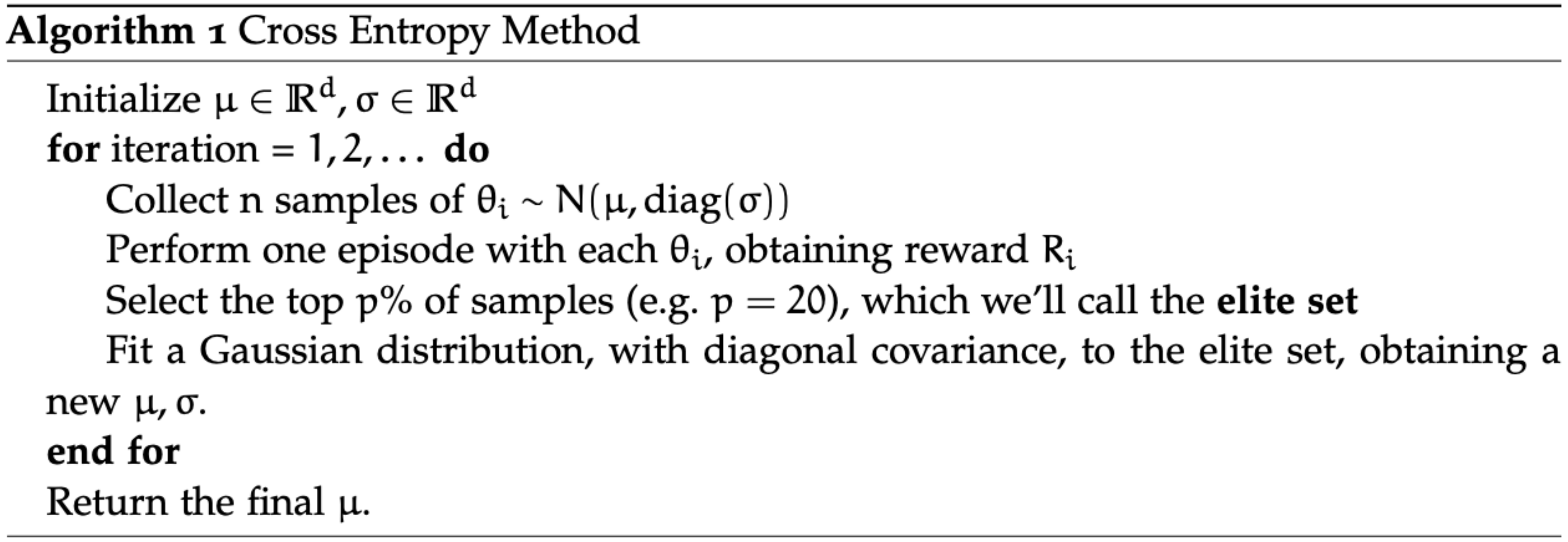
Adapted from S. Levine

Example: Evolutionary Algorithms

- Treat the whole process of turning parameters into a reward R as a black box.

- $\theta \rightarrow \blacksquare \rightarrow \sum_t r(s_t, a_t)$

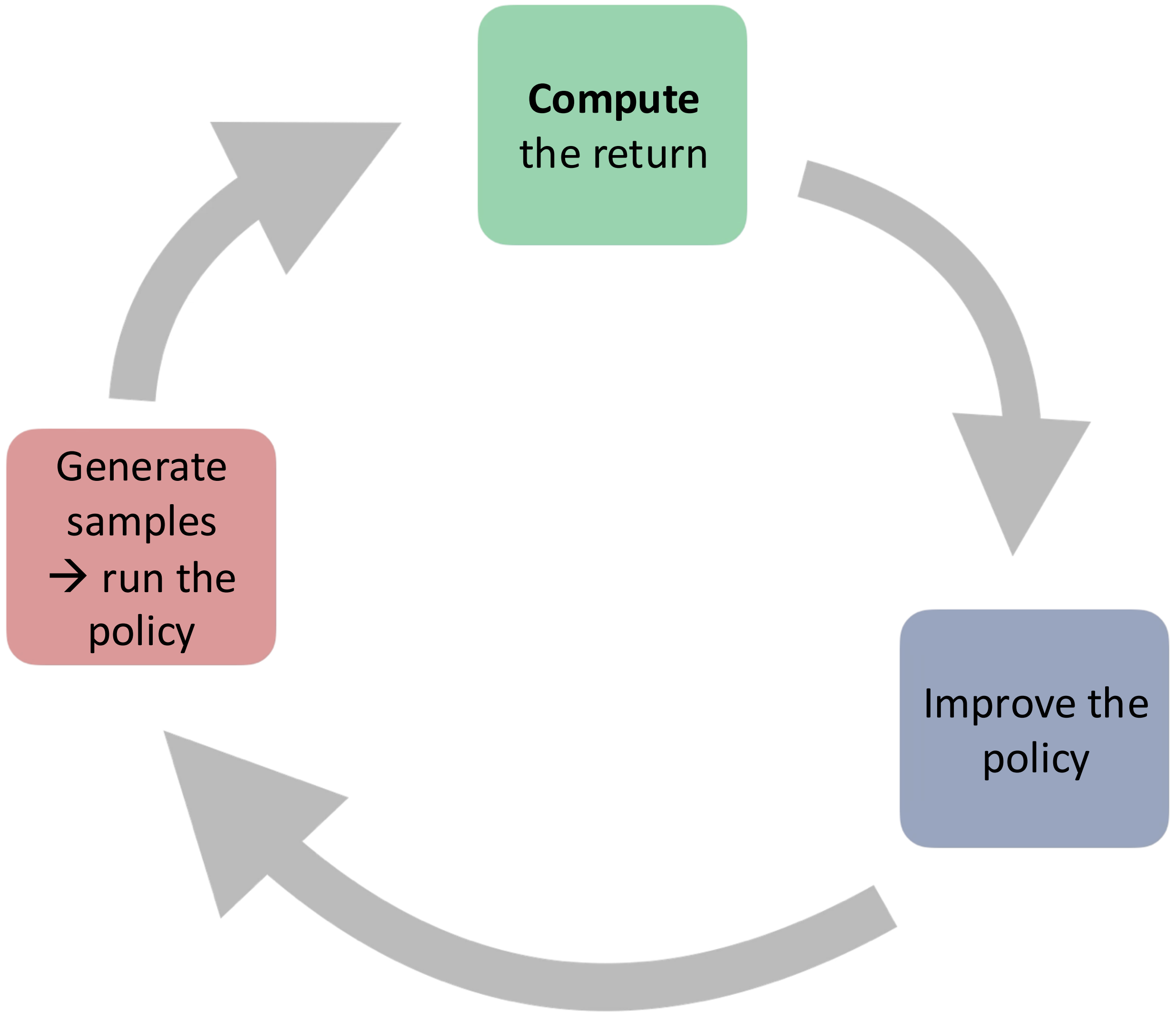
- Possible approach: Evolutionary Algorithms (e.g., CEM).



Algorithm 1 Cross Entropy Method

```
Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d$
**for** iteration = $1, 2, \ldots$ **do**
    Collect n samples of $\theta_i \sim N(\mu, \text{diag}(\sigma))$
    Perform one episode with each $\theta_i$, obtaining reward $R_i$
    Select the top p% of samples (e.g. $p = 20$), which we'll call the **elite set**
    Fit a Gaussian distribution, with diagonal covariance, to the elite set, obtaining a new $\mu, \sigma$.
**end for**
Return the final $\mu$.
```

Cross-Entropy Methods



What's this in CEM?

Can we do better?

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

Adapted from S. Levine

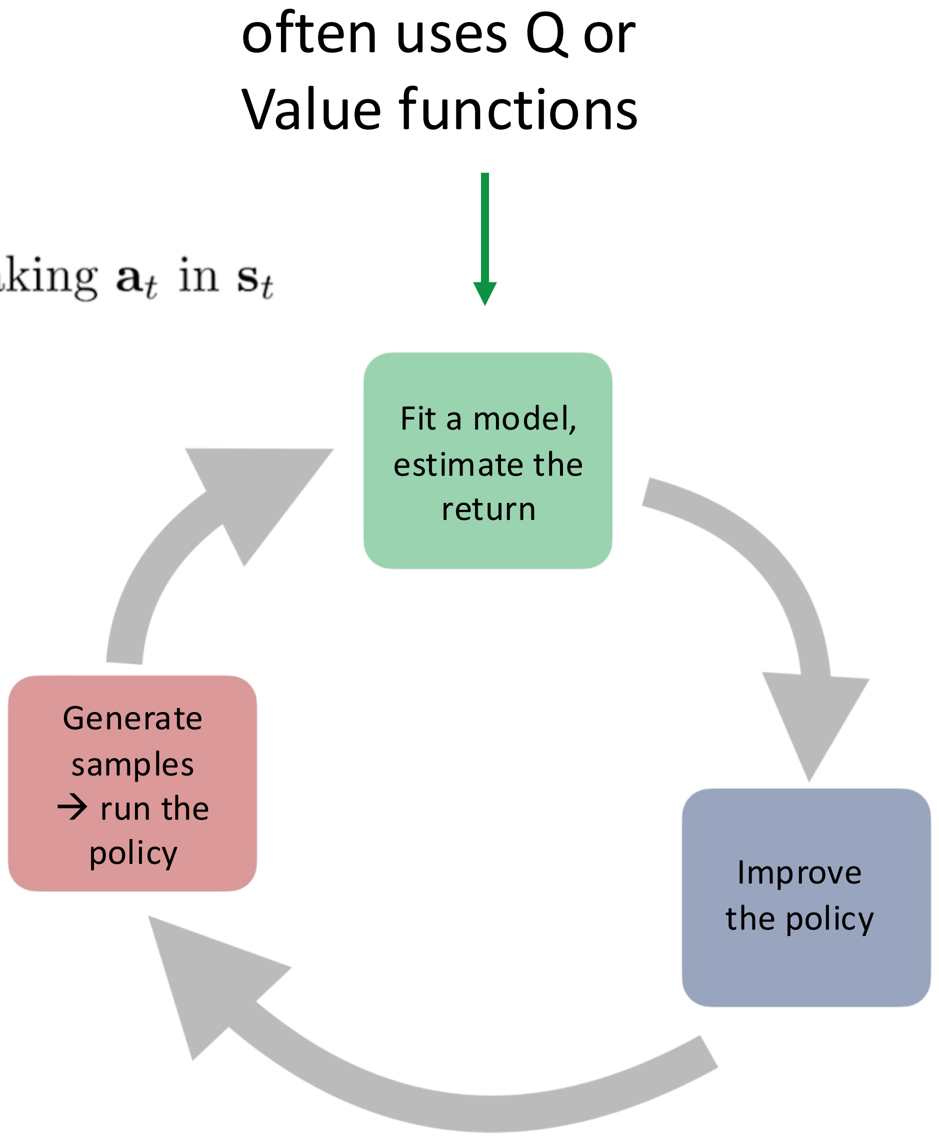
Definitions

Q-Function

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

Value Function

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


often uses Q or Value functions
Fit a model, estimate the return
Improve the policy
Generate samples → run the policy

Adapted from S. Levine

Using Value Function and Q Functions

Idea 1: if we have policy π, and we know $Q^{\pi}(\mathbf{s},\mathbf{a})$, then we can *improve* π :

set $\pi'(\mathbf{a} \mid \mathbf{s}) = 1$ if $\mathbf{a} = \mathrm{argmax}_{\mathrm{a}}\, Q^{\pi}(\mathbf{s},\mathbf{a})$

this policy is at least as good as π (and probably better)!
and it doesn't matter what π is

Idea 2: compute gradient to increase probability of good actions $\mathbf{a}$:

if $Q^{\pi}(\mathbf{s},\mathbf{a}) > V^{\pi}(\mathbf{s})$, then $\mathbf{a}$ is *better than average*
(recall that $V^{\pi}(\mathbf{s}) = E[Q^{\pi}(\mathbf{s},\mathbf{a})]$ under $\pi(\mathbf{a} \mid \mathbf{s})$)
modify $\pi(\mathbf{a} \mid \mathbf{s})$ to increase probability of $\mathbf{a}$ if $Q^{\pi}(\mathbf{s},\mathbf{a}) > V^{\pi}(\mathbf{s})$

These ideas are *very* important in RL!

Adapted from S. Levine

Why so many RL algorithms?

Different **tradeoffs**

- Sample efficiency
- Stability & ease of use

Different **assumptions**

- Stochastic or deterministic?
- Continuous or discrete policy?
- Episodic or infinite horizon?

Different things are **easy or hard in different settings**

- Easier to represent the **policy**?
- Easier to represent the model?

Adapted from S. Levine

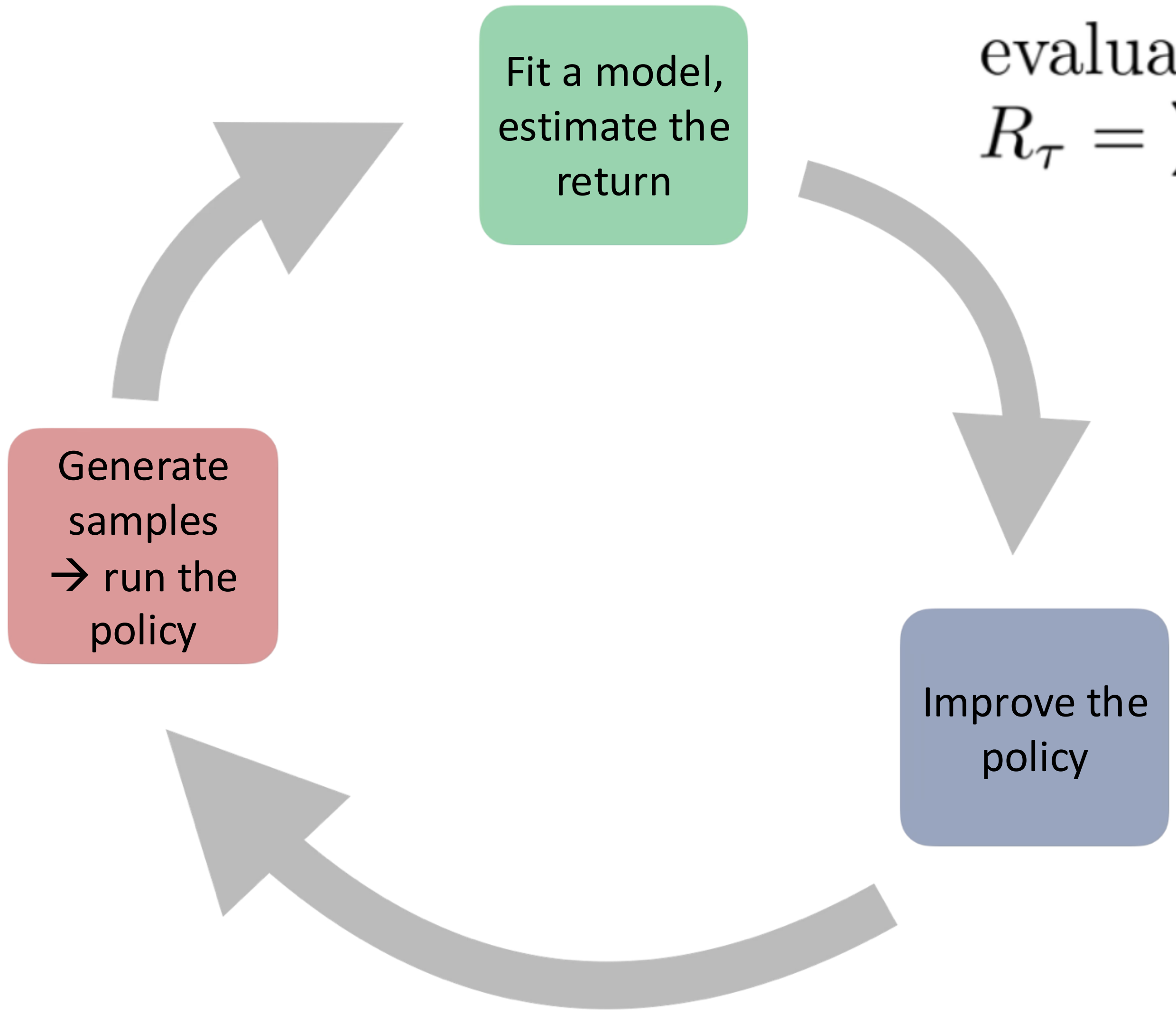
Types of RL Algorithms (Optimization type)

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

- **Policy gradients**: directly differentiate the above objective
- **Value-based**: estimate value function or Q-function of the **optimal** policy
  (no explicit policy)
- **Actor-critic**: estimate value function or Q-function of the **current** policy,
  use it to improve policy
- **Model-based RL**: estimate the transition model, and then…
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
  - Something else

Adapted from S. Levine

Direct Policy Gradients

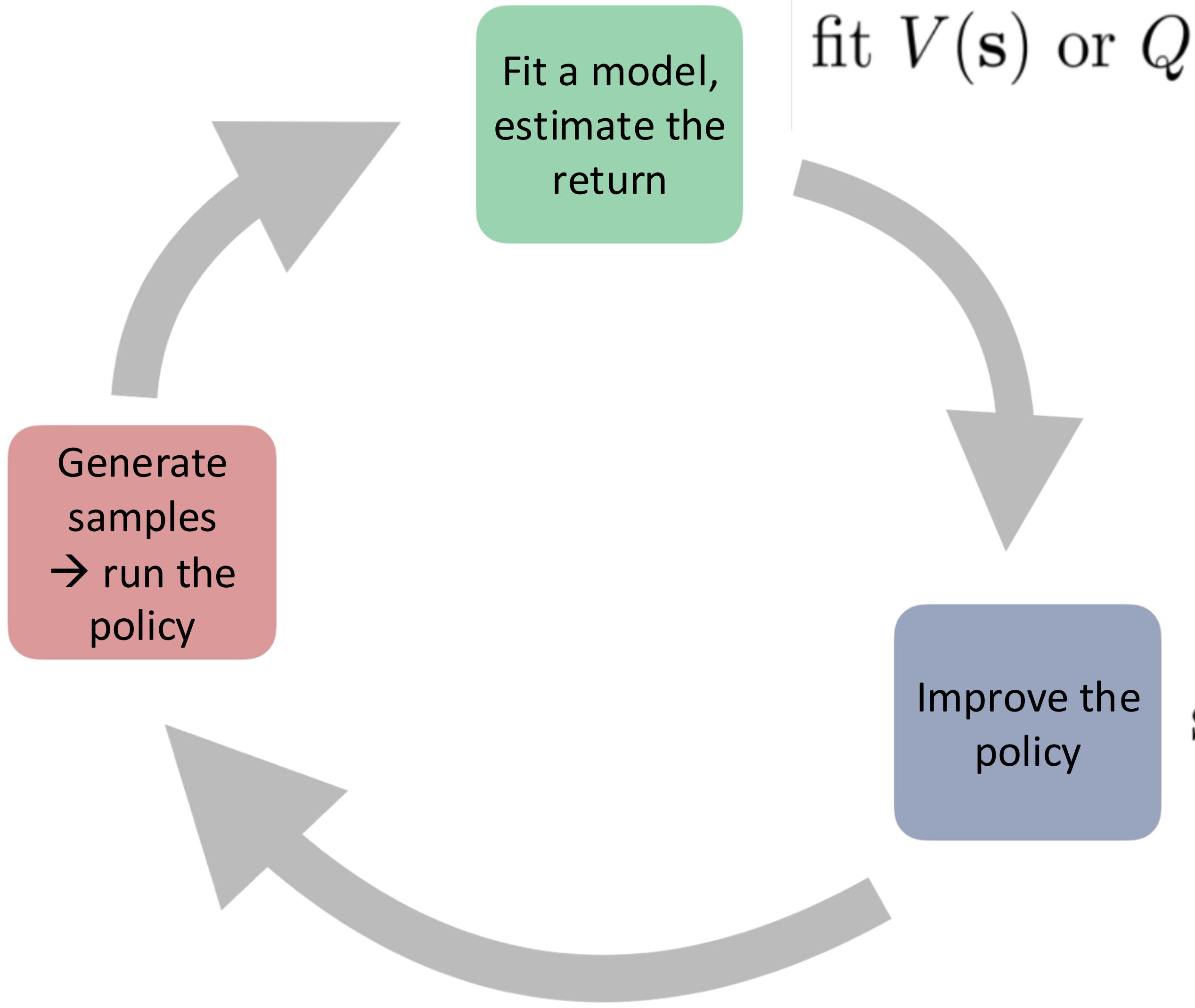

Fit a model, estimate the return
Generate samples → run the policy
Improve the policy

evaluate returns $R_\tau = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

$$\theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)]$$

Adapted from S. Levine

Value-Function Based

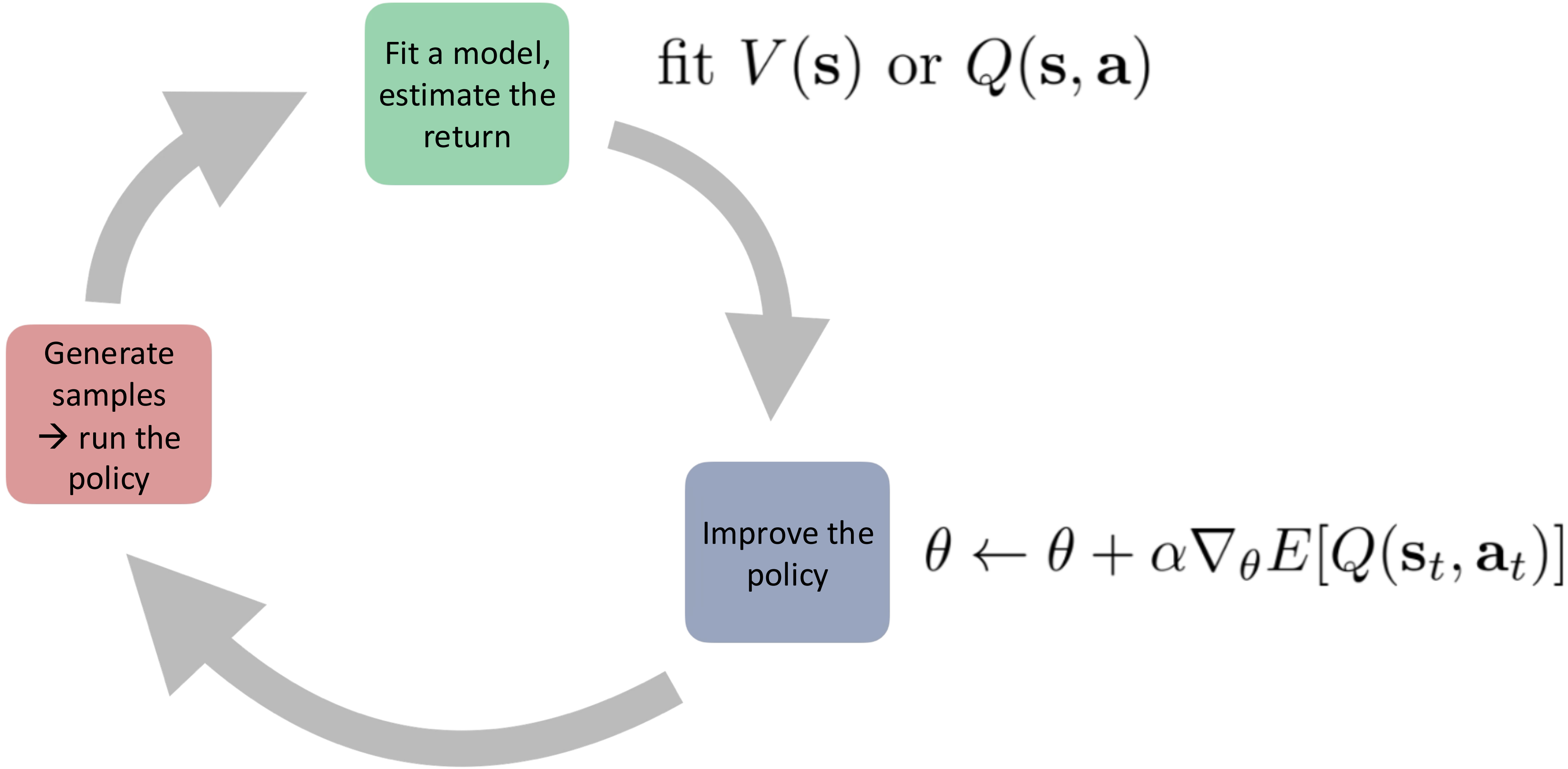

Fit a model, estimate the return
Improve the policy
Generate samples → run the policy

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

set $\pi(\mathbf{s}) = \arg\max_{\mathbf{a}} Q(\mathbf{s},\mathbf{a})$

Adapted from S. Levine

Actor-Critic: Doing Both


Fit a model, estimate the return
Improve the policy
Generate samples → run the policy

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

$\theta \leftarrow \theta + \alpha \nabla_\theta E[Q(\mathbf{s}_t,\mathbf{a}_t)]$

Adapted from S. Levine

Model-Based RL Algorithms

Just use the model to plan (no policy)

- Trajectory optimization/optimal control (primarily in continuous spaces) – essentially backpropagation to optimize over actions
- Discrete planning in discrete action spaces – e.g., Monte Carlo tree search

Backpropagate gradients into the policy

- Requires some tricks to make it work

Use the model to learn a value function or a policy

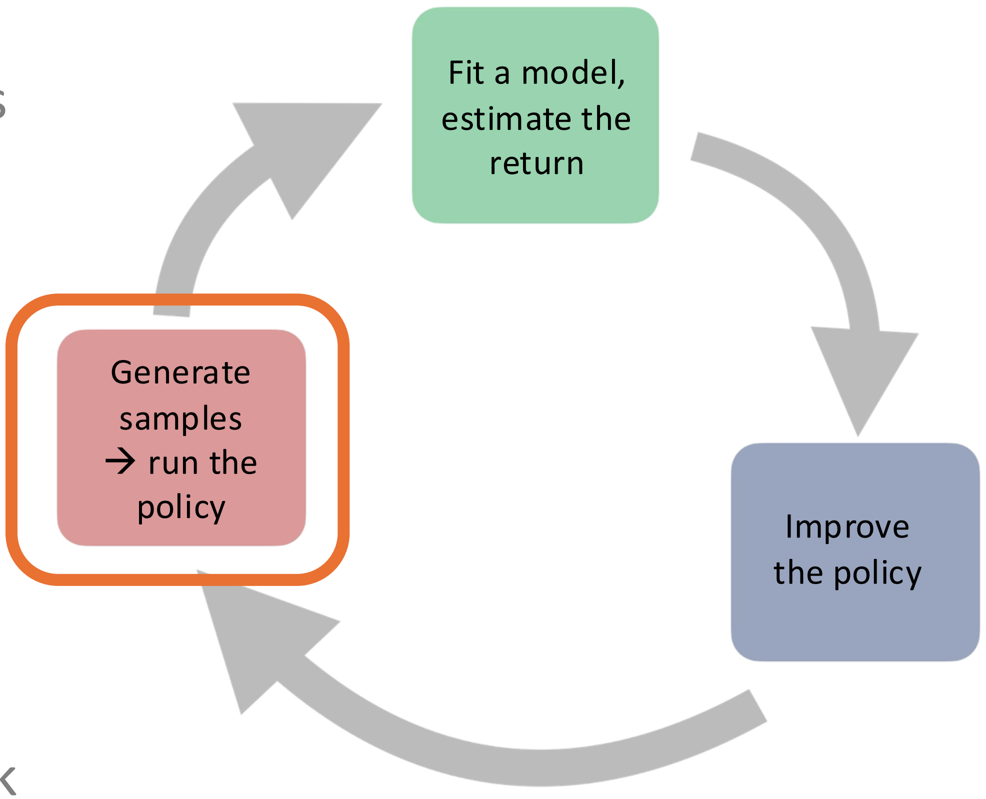
- Dynamic programming
- Generate simulated experience for a model-free learner

A better model does not necessarily imply a better policy!

Adapted from S. Levine

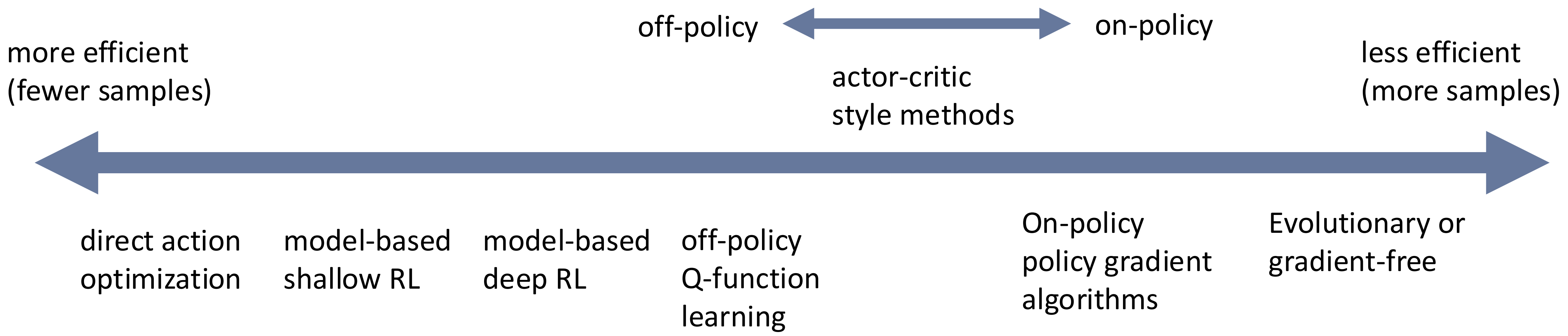
Types of RL Algorithms (Sample efficiency)

- On-policy algorithms:
  - Use data from the current policy to estimate returns. Every time the policy changes, we need new samples from that policy.
- Off-policy algorithms:
  - Can improve the policy with data from current and previous policies.
- Offline reinforcement learning:
  - Utilize previously collected data, without additional online data.
  - Conceptually similar to imitation learning, it can work better in the low-data regime.


Fit a model, estimate the return
Improve the policy
Generate samples → run the policy

Adapted from S. Levine

Types of RL Algorithms (Sample efficiency)


off-policy
on-policy
more efficient (fewer samples)
actor-critic style methods
less efficient (more samples)
direct action optimization
model-based shallow RL
model-based deep RL
off-policy Q-function learning
On-policy policy gradient algorithms
Evolutionary or gradient-free

Why not always use the most sample-efficient algorithm available?

Adapted from S. Levine

Stability and ease-of-use

- Does it **converge**?
- And if it converges, **to what**?
- And does it converge **every time**?

- **Supervised learning**: almost always gradient descent
  - Almost always converges
- **Reinforcement learning**: often not gradient descent!

Adapted from S. Levine

Stability and ease-of-use

Value function fitting

- At best, minimizes error of fit ("Bellman error")
  - Not the same as expected reward
- At worst, doesn't optimize anything
  - Many popular deep RL value fitting algorithms are not guaranteed to converge to anything in the nonlinear case

Model-based RL

- Model minimizes error of fit
  - This will converge
- No guarantee that better model = better policy

Policy gradient

- The only one that actually performs gradient descent (ascent) on the true objective

Adapted from S. Levine

Common assumptions in policy learning

#1: **full observability**

- Generally assumed by value function fitting methods
- Can be mitigated by adding recurrence

#2: **episodic learning**

- Often assumed by pure policy gradient methods
- Less common in model-based RL

#3: **continuity or smoothness**

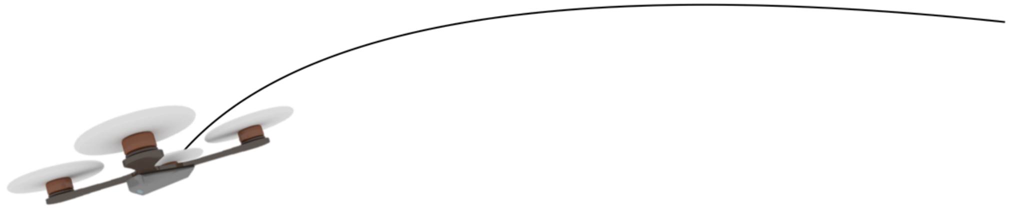
- Assumed by some continuous value function learning methods
- Often assumed by some model-based RL methods

Adapted from S. Levine

A Few Use Cases

Example: Trajectory Tracking with Drones

- Relatively slow motion
- The cost function is the difference between observed and desired states



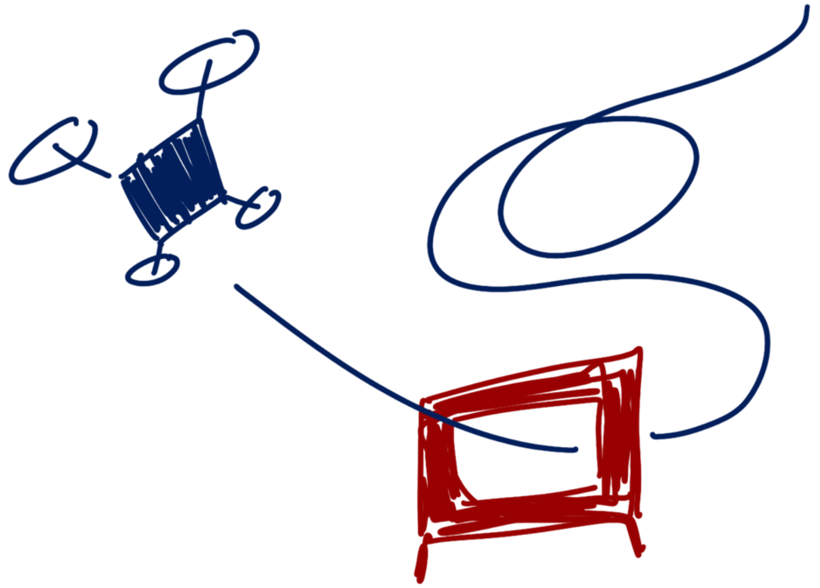
Problem characteristics:

- Easy to model the dynamics (little to no data required)
- Easy to write a smooth cost function specifying the desired behavior

Perfect fit for direct optimization of actions, e.g., MPC, LQR, etc.

Example: Drone Racing

- High-speed motion
- The goal is to win the race



Problem characteristics:

- Complex dynamics, but not too bad (some data required)
- Not clear how to write a smooth cost function specifying the desired behavior

Perfect fit for model-based RL, e.g., sim2real, dyna, dreamer, etc.

Example: Pick-and-place with an underactuated (soft) hand

- The goal is to pick up something

Problem characteristics:

- Dynamics is terribly complex (requires a lot of data)
- Not clear how to write a smooth cost function
- The robot is relatively safe
- Policy (might) not be too hard to learn: reach and close all fingers

Good fit (but hard in practice) for model-free RL, e.g., PPO, Q-learning.

Non-Robotic Example: LLM post-training

- The goal is to turn a base model into something humans like (chatbots)

Problem characteristics:

- Dynamics is challenging, but it can leverage pretraining.
- Writing a cost function is extremely hard (it has to be learned from data)

Great fit for model-based RL: Iteratively learn a reward predictor and optimize a policy (e.g., by doing PPO with the reward model on LLM predictions).