

What we have seen:

- RLMP, why does it work & different flavors

What we have seen:

- RL MPC, why does it work & different flavors

What we will do now:

- The theory is not about MPC, it is about MDPs (MPC is a special case)
- It has broader implications for AI and model-based decision making
- Provide (tentative) practical ideas for Sim2Real & Co-design problems

AI for Operational Decisions

Some perspectives from recent research

Prof. Sebastien Gros

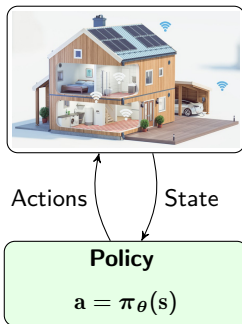
Department of Cybernetic
Faculty of Information Technology
NTNU

Freiburg PhD School

Outline

- 1 Decisions from data
- 2 AI models for Decision
- 3 Differentiating Policy Training
- 4 Examples

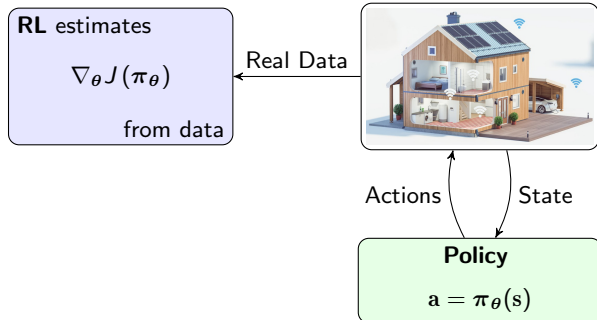
Model-Free Pathway - “Pure” Reinforcement Learning



- Define policy π_{θ} as a parametrized function
- Optimize policy as:

$$\min_{\theta} J(\pi_{\theta})$$

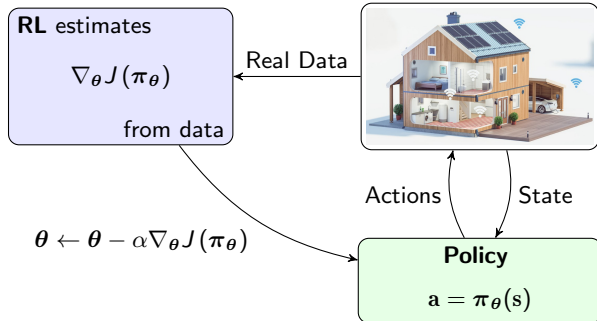
Model-Free Pathway - “Pure” Reinforcement Learning



- Define policy π_{θ} as a parametrized function
- Optimize policy as:

$$\min_{\theta} J(\pi_{\theta})$$

Model-Free Pathway - “Pure” Reinforcement Learning

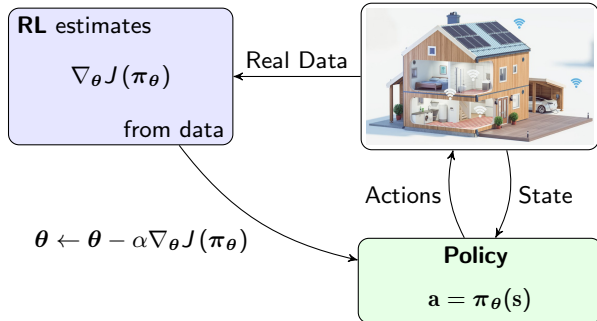


- Define policy π_{θ} as a parametrized function
- Optimize policy as:

$$\min_{\theta} J(\pi_{\theta})$$

- Gradient descent can be used to optimize policy

Model-Free Pathway - “Pure” Reinforcement Learning



- Define policy π_{θ} as a parametrized function
- Optimize policy as:

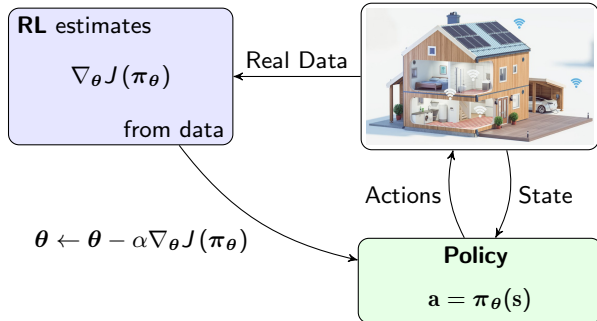
$$\min_{\theta} J(\pi_{\theta})$$

- Gradient descent can be used to optimize policy

Remarks

- $\pi_{\theta}(s)$ often from DNN
- End-to-end training requires a lot of data and effort
- Difficult to provide explainability and guarantees on the policy we obtain
- Poor track record of industrial adoption (exception chatbots)

Model-Free Pathway - “Pure” Reinforcement Learning



- Define policy π_{θ} as a parametrized function

- Optimize policy as:

$$\min_{\theta} J(\pi_{\theta})$$

- Gradient descent can be used to optimize policy

Remarks

- $\pi_{\theta}(s)$ often from DNN
- End-to-end training requires a lot of data and effort
- Difficult to provide explainability and guarantees on the policy we obtain
- Poor track record of industrial adoption (exception chatbots)

→ policy is often trained on **high-fidelity simulations**: embed knowledge, reduce costs, manage safety, promote explainability, control the training

Model-based Pathway - Predictive AI Models (data-driven, ML, etc)

Real world: $s_{k+1} \sim \rho(\cdot | s_k, a_k)$

- Data with enough “richness”

Model-based Pathway - Predictive AI Models (data-driven, ML, etc)

One-step model

AI model e.g.

$$\hat{s}_{k+1} \sim \rho_{\theta}(\cdot | s_k, a_k)$$

fitting data

Real world: $s_{k+1} \sim \rho(\cdot | s_k, a_k)$

- Data with enough “richness”

Here we no longer need the model ρ_{θ} to be “optimization-friendly”

Model-based Pathway - Predictive AI Models (data-driven, ML, etc)

One-step model

AI model e.g.

$$\hat{s}_{k+1} \sim \rho_{\theta}(\cdot | s_k, a_k)$$

fitting data

Real world: $s_{k+1} \sim \rho(\cdot | s_k, a_k)$

- Data with enough “richness”

Here we no longer need the model ρ_{θ} to be “optimization-friendly”

Model parameters θ

- Selected such that model ρ_{θ} “resembles” reality ρ , using data (+ physics)
- Deterministic models are a special case of ρ_{θ}
- Many methods, from Least Squares and MLE to Bayesian and Adversarial Learning
- Still, in most cases ρ_{θ} “simplifies” ρ because
 - ▶ need for huge amounts of data to decide θ if model is very rich
 - ▶ computationally demanding simulations if ρ_{θ} is expensive to sample from
- Then
 - ▶ biases in case of insufficiently rich model structure
 - ▶ validity is limited to some parts of the state-action space
 - ▶ only few first moments are correct (typ. mean + variance)

Model-based Pathway - Predictive AI Models (data-driven, ML, etc)

One-step model

AI model e.g.

$$\hat{s}_{k+1} \sim \rho_{\theta}(\cdot | s_k, a_k)$$

fitting data

Virtual data from “simulating”
 $\hat{s}_{k+1} \sim \rho_{\theta}(\cdot | s_k, a_k)$ forward (Monte Carlo)

Here we no longer need the model ρ_{θ} to be “optimization-friendly”

Model parameters θ

- Selected such that model ρ_{θ} “resembles” reality ρ , using data (+ physics)
- Deterministic models are a special case of ρ_{θ}
- Many methods, from Least Squares and MLE to Bayesian and Adversarial Learning
- Still, in most cases ρ_{θ} “simplifies” ρ because
 - ▶ need for huge amounts of data to decide θ if model is very rich
 - ▶ computationally demanding simulations if ρ_{θ} is expensive to sample from
- Then
 - ▶ biases in case of insufficiently rich model structure
 - ▶ validity is limited to some parts of the state-action space
 - ▶ only few first moments are correct (typ. mean + variance)

Model-based Pathway - Predictive AI Models (data-driven, ML, etc)

One-step model

AI model e.g.

$$\hat{s}_{k+1} \sim \rho_{\theta}(\cdot | s_k, a_k)$$

fitting data

Virtual data from “simulating”
 $\hat{s}_{k+1} \sim \rho_{\theta}(\cdot | s_k, a_k)$ forward (Monte Carlo)

In most cases, ρ_{θ} is a fairly simple representation of reality ρ , easy to sample from

Here we no longer need the model ρ_{θ} to be “optimization-friendly”

Model parameters θ

- Selected such that model ρ_{θ} “resembles” reality ρ , using data (+ physics)
- Deterministic models are a special case of ρ_{θ}
- Many methods, from Least Squares and MLE to Bayesian and Adversarial Learning
- Still, in most cases ρ_{θ} “simplifies” ρ because
 - ▶ need for huge amounts of data to decide θ if model is very rich
 - ▶ computationally demanding simulations if ρ_{θ} is expensive to sample from
- Then
 - ▶ biases in case of insufficiently rich model structure
 - ▶ validity is limited to some parts of the state-action space
 - ▶ only few first moments are correct (typ. mean + variance)

Back to MPC - A “Classical” View on Decision Making

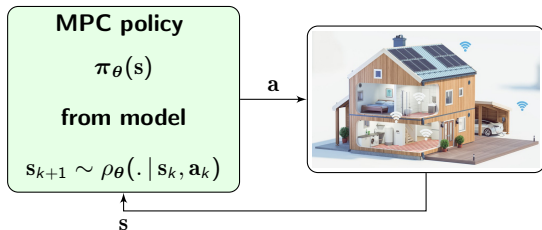
Policy $\pi_{\theta}(s)$ given implicitly by

$$\min_{\mathbf{x}, \mathbf{u}} \quad \mathbb{E} \left[T(\mathbf{x}_N) + \sum_{k=0}^{N-1} L(\mathbf{x}_k, \mathbf{u}_k) \right]$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} \sim \rho_{\theta}(\mathbf{x}_k, \mathbf{u}_k)$$

$$\mathbf{x}_0 = \mathbf{s}$$

use $\pi_{\theta}(s) = \mathbf{u}_0^*$ on the system



Back to MPC - A “Classical” View on Decision Making

Policy $\pi_{\theta}(s)$ given implicitly by

$$\min_{\mathbf{x}, \mathbf{u}} \quad \mathbb{E} \left[T(\mathbf{x}_N) + \sum_{k=0}^{N-1} L(\mathbf{x}_k, \mathbf{u}_k) \right]$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} \sim \rho_{\theta}(\mathbf{x}_k, \mathbf{u}_k)$$

$$\mathbf{x}_0 = \mathbf{s}$$

use $\pi_{\theta}(s) = \mathbf{u}_0^*$ on the system



Back to MPC - A “Classical” View on Decision Making

Policy $\pi_{\theta}(s)$ given implicitly by

$$\min_{\mathbf{x}, \mathbf{u}} \quad \mathbb{E} \left[T(\mathbf{x}_N) + \sum_{k=0}^{N-1} L(\mathbf{x}_k, \mathbf{u}_k) \right]$$

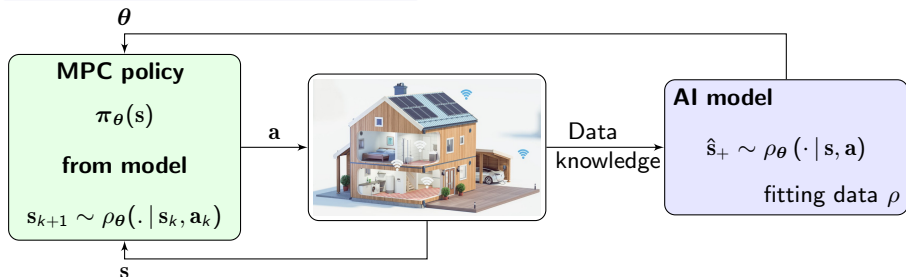
$$\text{s.t.} \quad \mathbf{x}_{k+1} \sim \rho_{\theta}(\mathbf{x}_k, \mathbf{u}_k)$$

$$\mathbf{x}_0 = \mathbf{s}$$

use $\pi_{\theta}(s) = \mathbf{u}_0^*$ on the system

Defines a paradigm...

- Performance from model accuracy
i.e. ρ_{θ} “close enough” to ρ
- “Ignore” that we replan all the time



Back to MPC - A “Classical” View on Decision Making

Policy $\pi_{\theta}(s)$ given implicitly by

$$\min_{\mathbf{x}, \mathbf{u}} \quad \mathbb{E} \left[T(\mathbf{x}_N) + \sum_{k=0}^{N-1} L(\mathbf{x}_k, \mathbf{u}_k) \right]$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} \sim \rho_{\theta}(\mathbf{x}_k, \mathbf{u}_k)$$

$$\mathbf{x}_0 = \mathbf{s}$$

use $\pi_{\theta}(s) = \mathbf{u}_0^*$ on the system

Relationship π_{θ} to π^* ??

- They match if ρ_{θ} is exact and deterministic
- But it usually cannot be...



Back to MPC - A “Classical” View on Decision Making

Policy $\pi_{\theta}(s)$ given implicitly by

$$\min_{\mathbf{x}, \mathbf{u}} \quad \mathbb{E} \left[T(\mathbf{x}_N) + \sum_{k=0}^{N-1} L(\mathbf{x}_k, \mathbf{u}_k) \right]$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} \sim \rho_{\theta}(\mathbf{x}_k, \mathbf{u}_k)$$

$$\mathbf{x}_0 = \mathbf{s}$$

use $\pi_{\theta}(s) = \mathbf{u}_0^*$ on the system

Relationship π_{θ} to π^* ??

- They match if ρ_{θ} is exact and deterministic
- But it usually cannot be...

“Standard methods” for choosing θ in general do not yield the best π_{θ} .
RL over MPC can fix that, L becomes part of the model



Back to MPC - A “Classical” View on Decision Making

Policy $\pi_{\theta}(s)$ given implicitly by

$$\min_{\mathbf{x}, \mathbf{u}} \quad \mathbb{E} \left[T(\mathbf{x}_N) + \sum_{k=0}^{N-1} L(\mathbf{x}_k, \mathbf{u}_k) \right]$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} \sim \rho_{\theta}(\mathbf{x}_k, \mathbf{u}_k)$$

$$\mathbf{x}_0 = \mathbf{s}$$

use $\pi_{\theta}(s) = \mathbf{u}_0^*$ on the system

Relationship π_{θ} to π^* ??

- They match if ρ_{θ} is exact and deterministic
- But it usually cannot be...

**“Standard methods” for choosing θ in general do not yield the best π_{θ} .
RL over MPC can fix that, L becomes part of the model**



This is not the only way of taking decisions from models

Decision policy from AI models (Sim2Real)

AI model

$$\hat{s}_+ \sim \rho_{\theta}(\cdot | s, a)$$

fitting data ρ

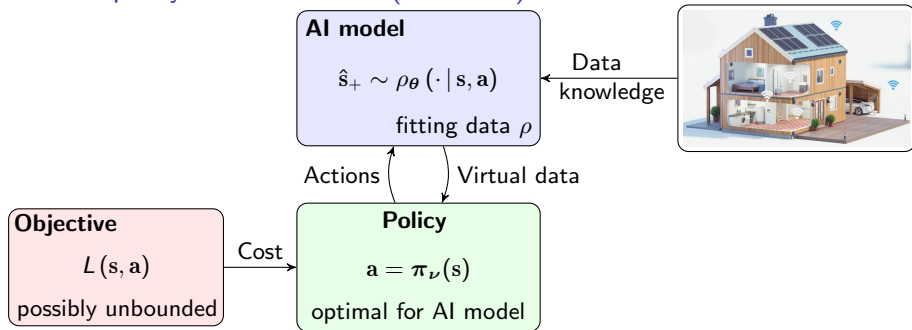
Data
knowledge



Classical Process:

- Fit AI model ρ_{θ} to real data

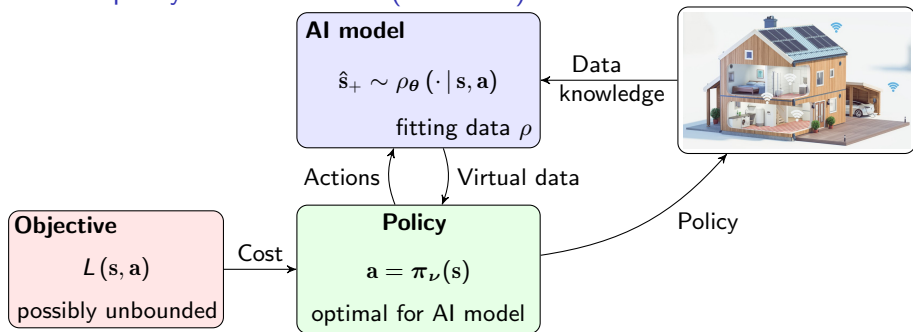
Decision policy from AI models (Sim2Real)



Classical Process:

- Fit AI model ρ_θ to real data
- Develop optimal policy for L and ρ_θ from AI model, e.g. using In-Sim RL
 - ▶ Define parametrized policy π_ν
 - ▶ Optimize policy parameters ν for performance w.r.t. AI model: $\rho_\theta, L \rightarrow \nu^*$

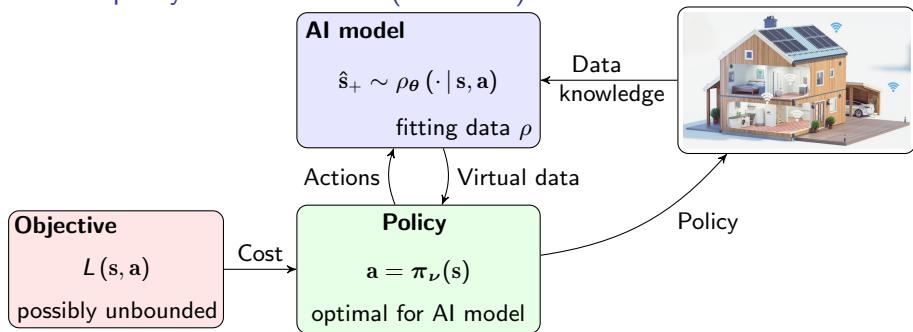
Decision policy from AI models (Sim2Real)



Classical Process:

- Fit AI model ρ_{θ} to real data
- Develop optimal policy for L and ρ_{θ} from AI model, e.g. using In-Sim RL
 - ▶ Define parametrized policy π_{ν}
 - ▶ Optimize policy parameters ν for performance w.r.t. AI model: $\rho_{\theta}, L \rightarrow \nu^*$
- Transfer policy π_{ν^*} into the real world

Decision policy from AI models (Sim2Real)

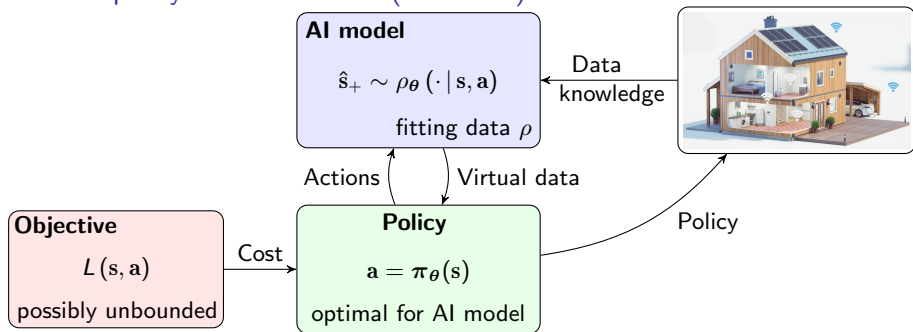


Classical Process:

- Fit AI model ρ_θ to real data
- Develop optimal policy for L and ρ_θ from AI model, e.g. using In-Sim RL
 - ▶ Define parametrized policy π_ν
 - ▶ Optimize policy parameters ν for performance w.r.t. AI model: $\rho_\theta, L \rightarrow \nu^*$
- Transfer policy π_{ν^*} into the real world

Note: policy parameters ν^* optimal for AI model become function of θ
Let's then label $\pi_\theta = \pi_{\nu^*}$ for simplicity

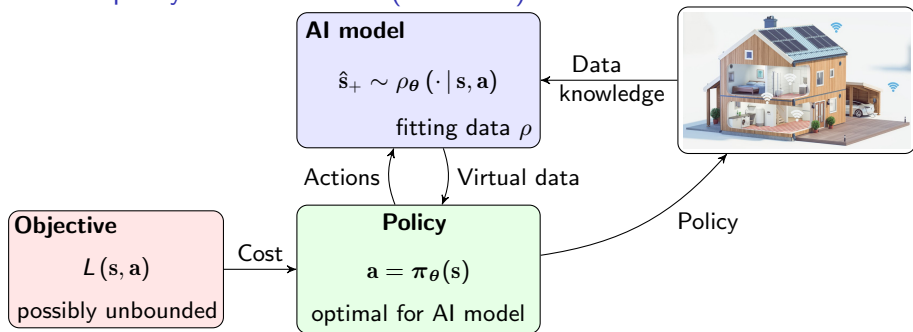
Decision policy from AI models (Sim2Real)



Remarks

- Relationship π_θ to π^* ??
 - ▶ They match if $\rho_\theta = \rho$, i.e. if model is exact
 - ▶ But ρ_θ is (almost always) an approximation

Decision policy from AI models (Sim2Real)



Remarks

Relationship π_θ to π^* ??

- ▶ They match if $\rho_\theta = \rho$, i.e. if model is exact
- ▶ But ρ_θ is (almost always) an approximation

Are “standard methods” for choosing θ resulting in a good policy π_θ ??
Empirically the answer is “no”. Then how to choose θ to get a good policy?

MPC or Sim2Real - It's the same question...

MPC or Sim2Real - It's the same question...

The theory is about model-based decisions, and equivalences between MDPs

MPC or Sim2Real - It's the same question...

The theory is about model-based decisions, and equivalences between MDPs

World MDP

World MDP definition

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

MPC or Sim2Real - It's the same question...

The theory is about model-based decisions, and equivalences between MDPs

World MDP

World MDP definition

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Optimal value functions

$$V^*(s) = \min_a Q^*(s, a)$$

$$Q^*(s, a) = L(s, a) + \gamma \mathbb{E}_{s_+ \sim \rho} [V^*(s_+) | s, a]$$

MPC or Sim2Real - It's the same question...

The theory is about model-based decisions, and equivalences between MDPs

World MDP

World MDP definition

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Optimal value functions

$$V^*(s) = \min_a Q^*(s, a)$$

$$Q^*(s, a) = L(s, a) + \gamma \mathbb{E}_{s_+ \sim \rho} [V^*(s_+) | s, a]$$

Optimal policy

$$\pi^*(s) = \arg \min_a Q^*(s, a)$$

MPC or Sim2Real - It's the same question...

The theory is about model-based decisions, and equivalences between MDPs

World MDP

World MDP definition

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Optimal value functions

$$V^*(s) = \min_a Q^*(s, a)$$

$$Q^*(s, a) = L(s, a) + \gamma \mathbb{E}_{s_+ \sim \rho} [V^*(s_+) | s, a]$$

Optimal policy

$$\pi^*(s) = \arg \min_a Q^*(s, a)$$

Model MDP (simulation environment)

Model MDP definition

- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Optimal value functions

$$\hat{V}^*(s) = \min_a \hat{Q}^*(s, a)$$

$$\hat{Q}^*(s, a) = \hat{L}(s, a) + \gamma \mathbb{E}_{s_+ \sim \hat{\rho}} [V^*(\hat{s}_+) | s, a]$$

Optimal policy

$$\hat{\pi}^*(s) = \arg \min_a \hat{Q}^*(s, a)$$

MPC or Sim2Real - It's the same question...

The theory is about model-based decisions, and equivalences between MDPs

World MDP

World MDP definition

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Optimal value functions

$$V^*(s) = \min_a Q^*(s, a)$$

$$Q^*(s, a) = L(s, a) + \gamma \mathbb{E}_{s_+ \sim \rho} [V^*(s_+) | s, a]$$

Optimal policy

$$\pi^*(s) = \arg \min_a Q^*(s, a)$$

Model MDP (simulation environment)

Model MDP definition

- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Optimal value functions

$$\hat{V}^*(s) = \min_a \hat{Q}^*(s, a)$$

$$\hat{Q}^*(s, a) = \hat{L}(s, a) + \gamma \mathbb{E}_{s_+ \sim \hat{\rho}} [V^*(\hat{s}_+) | s, a]$$

Optimal policy

$$\hat{\pi}^*(s) = \arg \min_a \hat{Q}^*(s, a)$$

Theory says that - under some technical conditions - there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$

MPC or Sim2Real - It's the same question...

The theory is about model-based decisions, and equivalences between MDPs

World MDP

World MDP definition

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Optimal value functions

$$V^*(s) = \min_a Q^*(s, a)$$

$$Q^*(s, a) = L(s, a) + \gamma \mathbb{E}_{s_+ \sim \rho} [V^*(s_+) | s, a]$$

Optimal policy

$$\pi^*(s) = \arg \min_a Q^*(s, a)$$

Model MDP (simulation environment)

Model MDP definition

- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Optimal value functions

$$\hat{V}^*(s) = \min_a \hat{Q}^*(s, a)$$

$$\hat{Q}^*(s, a) = \hat{L}(s, a) + \gamma \mathbb{E}_{s_+ \sim \hat{\rho}} [V^*(\hat{s}_+) | s, a]$$

Optimal policy

$$\hat{\pi}^*(s) = \arg \min_a \hat{Q}^*(s, a)$$

Theory says that - under some technical conditions - there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$

Proof: telescopic sum, some non-trivial assumptions to prevent $\infty - \infty$ cancellations

More on MDP Equivalence

World MDP

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Model MDP

- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

More on MDP Equivalence

World MDP

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Model MDP

- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Theory says that

- Under some technical conditions there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$

More on MDP Equivalence

World MDP

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Model MDP

- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Theory says that

- Under some technical conditions there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$
- For $\hat{L} = L$ there is a (non-unique) “optimal” model $\hat{\rho}$ such that $\hat{Q}^* = Q^*$

More on MDP Equivalence

World MDP

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Model MDP

- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Theory says that

- Under some technical conditions there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$
- For $\hat{L} = L$ there is a (non-unique) “optimal” model $\hat{\rho}$ such that $\hat{Q}^* = Q^*$
- Conditions for model $\hat{\rho}$ “optimality” $\neq$ min of classical loss functions (except. LQR)

More on MDP Equivalence

World MDP

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Model MDP

- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Theory says that

- Under some technical conditions there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$
- For $\hat{L} = L$ there is a (non-unique) “optimal” model $\hat{\rho}$ such that $\hat{Q}^* = Q^*$
- Conditions for model $\hat{\rho}$ “optimality” $\neq$ min of classical loss functions (except. LQR)
- World MDP and Model MDP do not need to use the same discount γ

More on MDP Equivalence

World MDP

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Model MDP

- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Theory says that

- Under some technical conditions there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$
- For $\hat{L} = L$ there is a (non-unique) “optimal” model $\hat{\rho}$ such that $\hat{Q}^* = Q^*$
- Conditions for model $\hat{\rho}$ “optimality” $\neq$ min of classical loss functions (except. LQR)
- World MDP and Model MDP do not need to use the same discount γ

In-Sim policy training uses a Model MDP

- Practitioners mostly work on model $\hat{\rho}$, but “unsure” on how to tune it
- In the ML communities, people talk about “value alignment”, we are trying to make this “MDP equivalence” understood

More on MDP Equivalence

World MDP

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Model MDP

- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Theory says that

- Under some technical conditions there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$
- For $\hat{L} = L$ there is a (non-unique) “optimal” model $\hat{\rho}$ such that $\hat{Q}^* = Q^*$
- Conditions for model $\hat{\rho}$ “optimality” $\neq$ min of classical loss functions (except. LQR)
- World MDP and Model MDP do not need to use the same discount γ

Remarks

- Non-unique optimal model $\hat{\rho}$ leaves room for aligning it to the real world (classical fitting)

More on MDP Equivalence

World MDP

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Model MDP

- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Theory says that

- Under some technical conditions there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$
- For $\hat{L} = L$ there is a (non-unique) “optimal” model $\hat{\rho}$ such that $\hat{Q}^* = Q^*$
- Conditions for model $\hat{\rho}$ “optimality” $\neq$ min of classical loss functions (except. LQR)
- World MDP and Model MDP do not need to use the same discount γ

Remarks

- Non-unique optimal model $\hat{\rho}$ leaves room for aligning it to the real world (classical fitting)
- If V^* is continuous and support of ρ is bounded(?) and path-connected, then we can have support $\hat{\rho} \subset$ support of ρ

More on MDP Equivalence

World MDP

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Model MDP

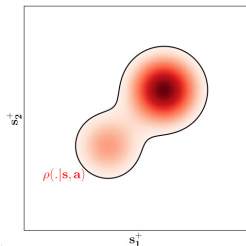
- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Theory says that

- Under some technical conditions there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$
- For $\hat{L} = L$ there is a (non-unique) “optimal” model $\hat{\rho}$ such that $\hat{Q}^* = Q^*$
- Conditions for model $\hat{\rho}$ “optimality” $\neq$ min of classical loss functions (except. LQR)
- World MDP and Model MDP do not need to use the same discount γ

Remarks

- Non-unique optimal model $\hat{\rho}$ leaves room for aligning it to the real world (classical fitting)
- If V^* is continuous and support of ρ is bounded(?) and path-connected, then we can have support $\hat{\rho} \subset$ support of ρ



More on MDP Equivalence

World MDP

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Model MDP

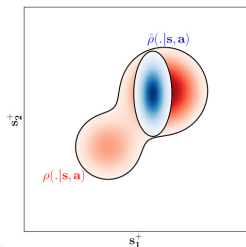
- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Theory says that

- Under some technical conditions there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$
- For $\hat{L} = L$ there is a (non-unique) “optimal” model $\hat{\rho}$ such that $\hat{Q}^* = Q^*$
- Conditions for model $\hat{\rho}$ “optimality” $\neq$ min of classical loss functions (except. LQR)
- World MDP and Model MDP do not need to use the same discount γ

Remarks

- Non-unique optimal model $\hat{\rho}$ leaves room for aligning it to the real world (classical fitting)
- If V^* is continuous and support of ρ is bounded(?) and path-connected, then we can have support $\hat{\rho} \subset$ support of ρ



More on MDP Equivalence

World MDP

- States s and actions a
- Cost $L(s, a)$
- Transition $s_+ \sim \rho[\cdot | s, a]$

Model MDP

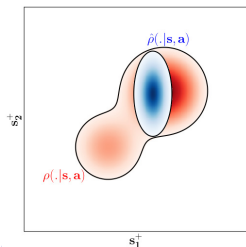
- States s and actions a
- Cost $\hat{L}(s, a)$
- Transition $s_+ \sim \hat{\rho}[\cdot | s, a]$

Theory says that

- Under some technical conditions there is a $\hat{L}$ such that $\hat{Q}^* = Q^*$
- For $\hat{L} = L$ there is a (non-unique) “optimal” model $\hat{\rho}$ such that $\hat{Q}^* = Q^*$
- Conditions for model $\hat{\rho}$ “optimality” $\neq$ min of classical loss functions (except. LQR)
- World MDP and Model MDP do not need to use the same discount γ

Remarks

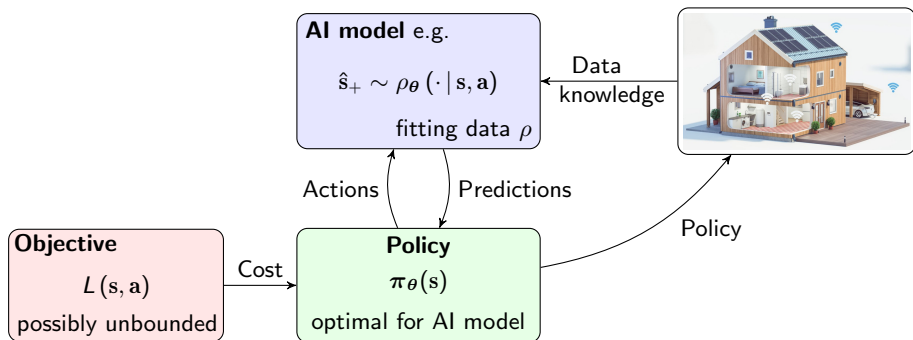
- Non-unique optimal model $\hat{\rho}$ leaves room for aligning it to the real world (classical fitting)
- If V^* is continuous and support of ρ is bounded(?) and path-connected, then we can have support $\hat{\rho} \subset \text{support of } \rho$
- *More simply said:* we can build optimal models $\hat{\rho}$ that make “plausible” predictions about the real world.



Outline

- 1 Decisions from data
- 2 AI models for Decision
- 3 Differentiating Policy Training
- 4 Examples

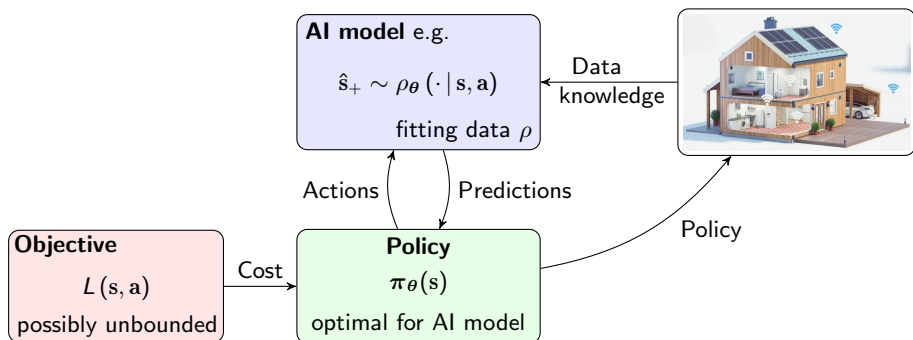
AI models for Decisions?



Learning model ρ_θ should aim at identifying

$$\theta^* = \arg \min_{\theta} J(\pi_\theta) \quad (1)$$

AI models for Decisions?

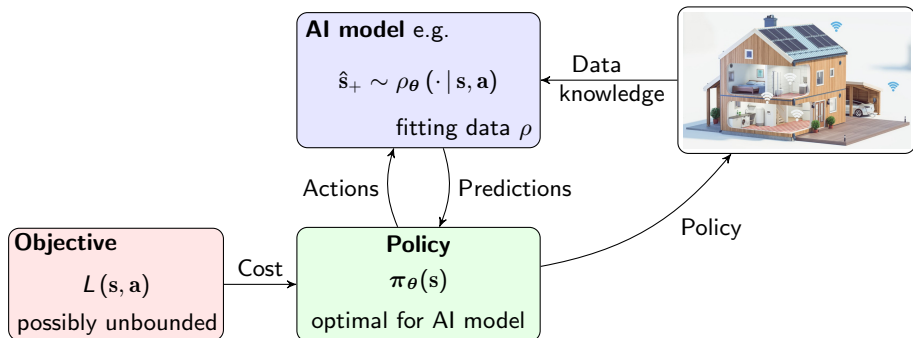


Learning model ρ_θ should aim at identifying

$$\theta^* = \arg \min_{\theta} J(\pi_\theta) \quad (1)$$

- Closed-loop performance J is intricate ($\theta \rightarrow \text{simulations} \rightarrow \text{policy} \rightarrow \text{real system}$)

AI models for Decisions?

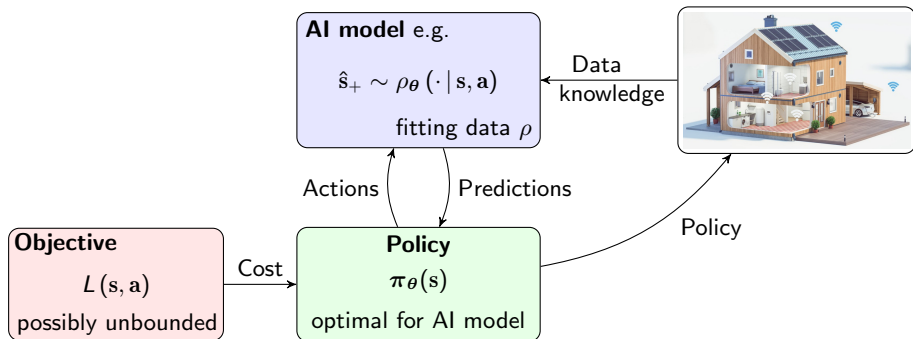


Learning model ρ_θ should aim at identifying

$$\theta^* = \arg \min_{\theta} J(\pi_\theta) \quad (1)$$

- Closed-loop performance J is intricate ($\theta \rightarrow \text{simulations} \rightarrow \text{policy} \rightarrow \text{real system}$)
- $\nabla_{\theta} J(\pi_{\theta})$ is to be estimated from data, i.e. data hungry, noisy, possible biases

AI models for Decisions?

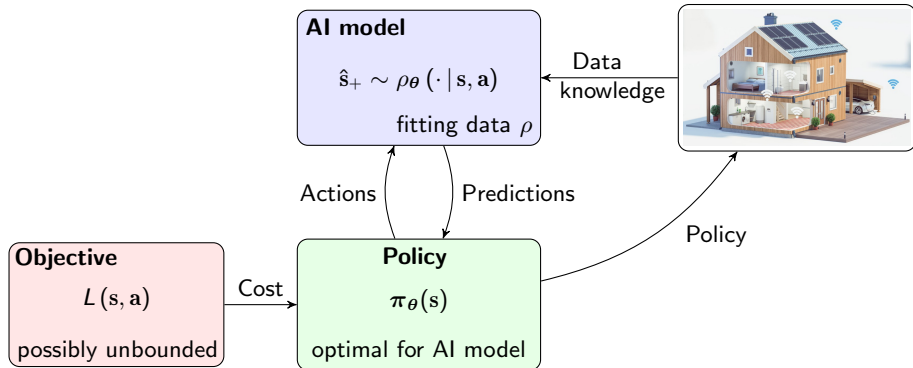


Learning model ρ_θ should aim at identifying

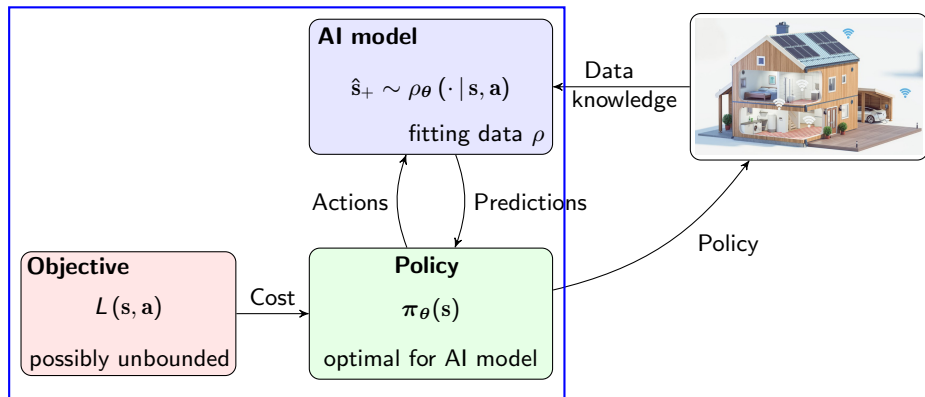
$$\theta^* = \arg \min_{\theta} J(\pi_\theta) \quad (1)$$

- Closed-loop performance J is intricate ($\theta \rightarrow \text{simulations} \rightarrow \text{policy} \rightarrow \text{real system}$)
- $\nabla_{\theta} J(\pi_\theta)$ is to be estimated from data, i.e. data hungry, noisy, possible biases
- (1) is in general different than model fitting, i.e. no loss function does (1)

Optimal AI models for Decision - A Paradigm Shift

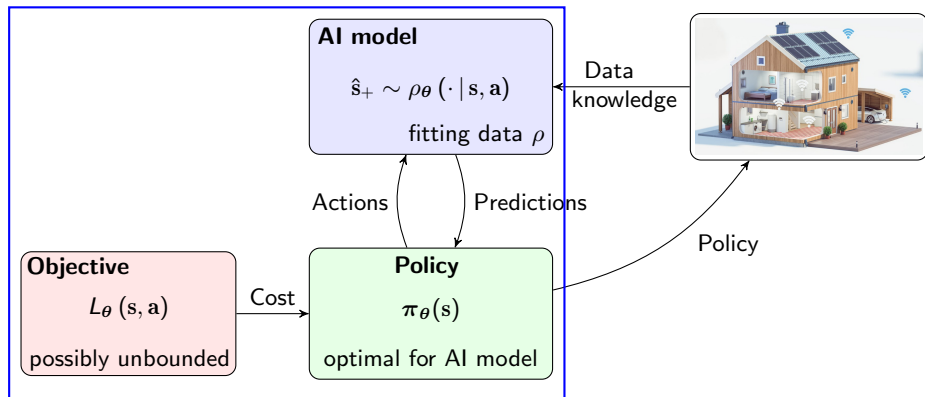


Optimal AI models for Decision - A Paradigm Shift



- The model is not ρ_θ , it is the entire "decision-making box"

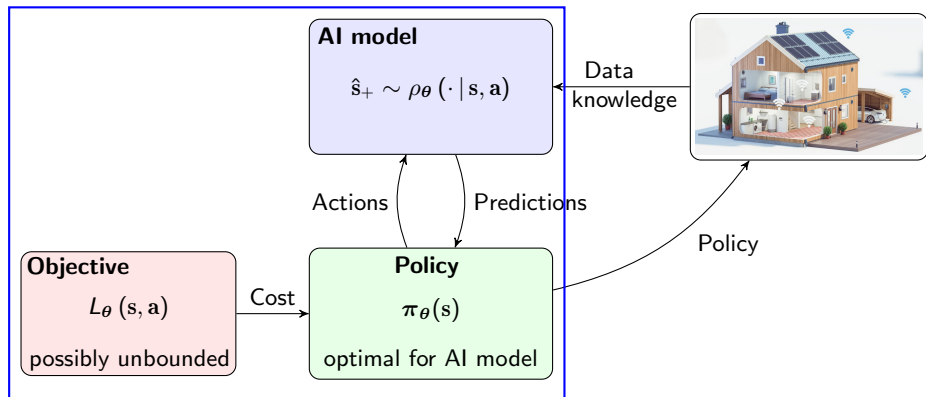
Optimal AI models for Decision - A Paradigm Shift



- The model is not ρ_{θ} , it is the entire “decision-making box”
- The model includes objective L_{θ} used to build the policy*

* policy performance on real system still assessed via L

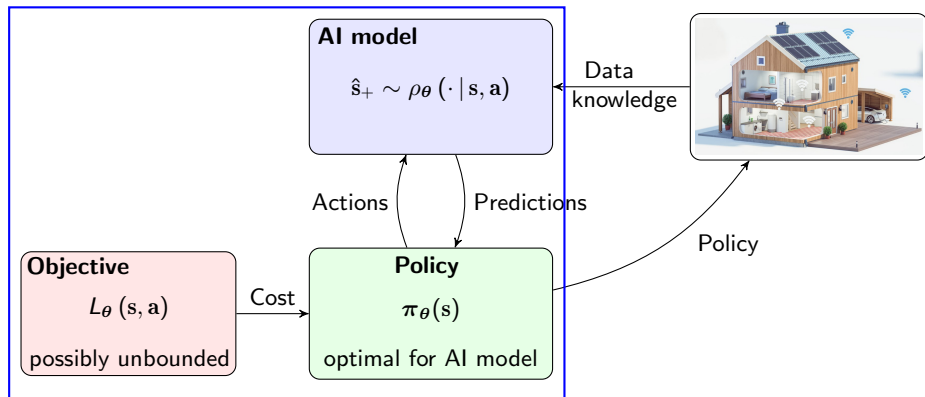
Optimal AI models for Decision - A Paradigm Shift



- The model is not ρ_{θ} , it is the entire “decision-making box”
- The model includes objective L_{θ} used to build the policy*
- Best model ρ_{θ} should *not necessarily* represent the data in a “classical sense”

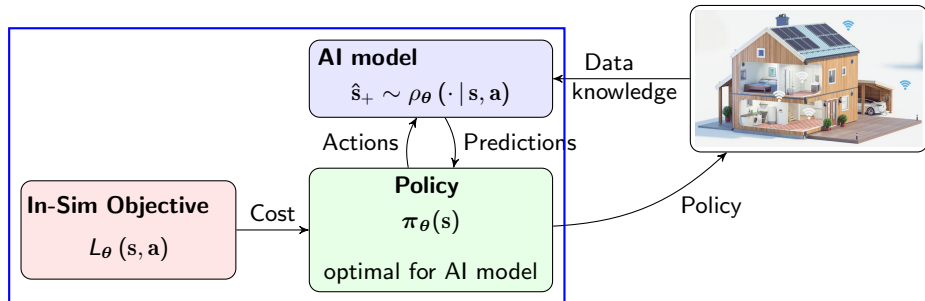
* policy performance on real system still assessed via L

Optimal AI models for Decision - A Paradigm Shift

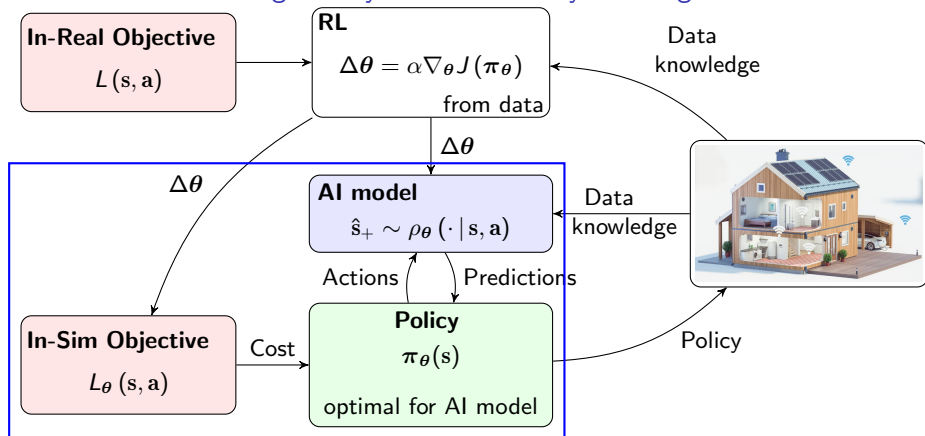


Theorem: under some (technical) assumptions, there is a L_θ such that $\pi_\theta = \pi_*$, even if ρ_θ does not represent real world ρ correctly

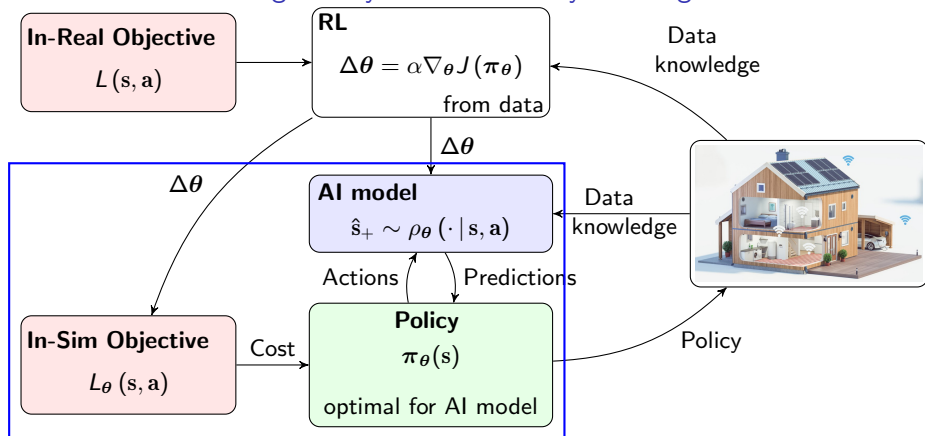
How to use this more generally? RL over Policy Training



How to use this more generally? RL over Policy Training

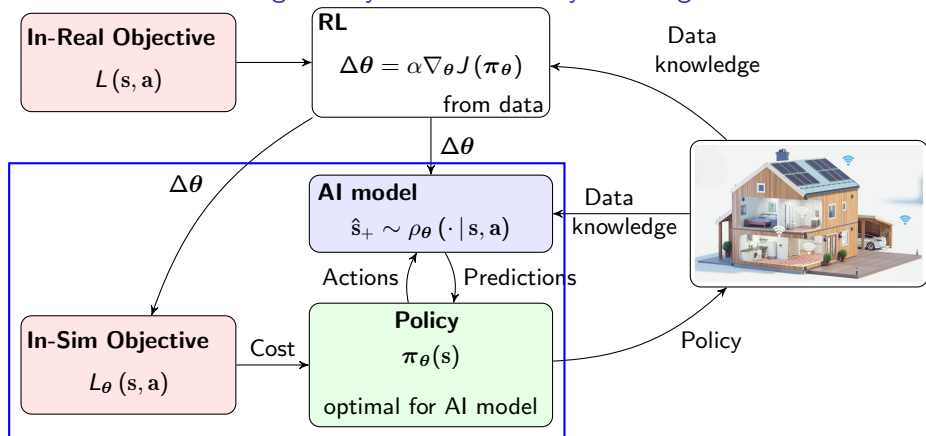


How to use this more generally? RL over Policy Training



- $Q^{\pi_{\theta}}$ is the critic, well-established RL tool
- $\nabla_{\theta} \pi_{\theta}$ requires differentiating the closed-loop simulation...

How to use this more generally? RL over Policy Training



Difficulty: computing $\nabla_{\theta} \pi_{\theta}$ requires total differentiation through the simulation-based policy optimization process:

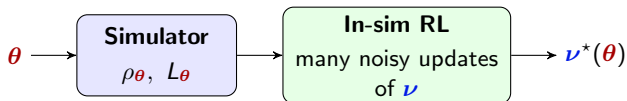
$\theta \rightarrow \text{simulations} \rightarrow \text{optimal policy for } \rho_{\theta}$

Differentiating efficiently through simulations + policy optimization can be technical...

Outline

- 1 Decisions from data
- 2 AI models for Decision
- 3 Differentiating Policy Training
- 4 Examples

What we need: the sensitivity of a trained policy

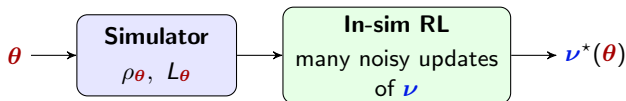


In-sim policy training, for a given θ

$$\nu^*(\theta) = \arg \min_{\nu} \hat{J}_\theta(\nu)$$

$$\hat{J}_\theta(\nu) = \mathbb{E}_{\rho_\theta, \pi_\nu} \left[\sum_k \gamma^k L_\theta(s_k, a_k) \right]$$

What we need: the sensitivity of a trained policy



In-sim policy training, for a given θ

Policy sensitivity

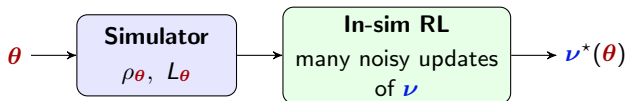
$$\nu^*(\theta) = \arg \min_{\nu} \hat{J}_\theta(\nu)$$

$$\kappa := \frac{d\nu^*}{d\theta} \in \mathbb{R}^{\dim \nu \times \dim \theta}$$

$$\hat{J}_\theta(\nu) = \mathbb{E}_{\rho_\theta, \pi_\nu} \left[\sum_k \gamma^k L_\theta(s_k, a_k) \right]$$

Needed in previous slide: $\nabla_\theta \pi_\theta = \nabla_\nu \pi_\nu \big|_{\nu^*} \kappa$

What we need: the sensitivity of a trained policy



In-sim policy training, for a given θ

Policy sensitivity

$$\nu^*(\theta) = \arg \min_{\nu} \hat{J}_{\theta}(\nu)$$

$$\kappa := \frac{d\nu^*}{d\theta} \in \mathbb{R}^{\dim \nu \times \dim \theta}$$

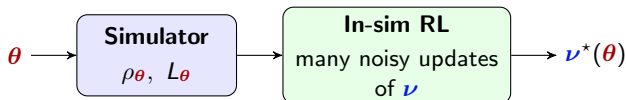
$$\hat{J}_{\theta}(\nu) = \mathbb{E}_{\rho_{\theta}, \pi_{\nu}} \left[\sum_k \gamma^k L_{\theta}(s_k, a_k) \right]$$

Needed in previous slide: $\nabla_{\theta} \pi_{\theta} = \nabla_{\nu} \pi_{\nu} \Big|_{\nu^*} \kappa$

How to get κ ?

- 1 **Unroll training:** (pyTorch) backpropagate through all updates $\nu_{j+1} = \nu_j - \beta \hat{g}_j$
✗ cost $\propto$ number of updates

What we need: the sensitivity of a trained policy



In-sim policy training, for a given θ

Policy sensitivity

$$\nu^*(\theta) = \arg \min_{\nu} \hat{J}_{\theta}(\nu)$$

$$\kappa := \frac{d\nu^*}{d\theta} \in \mathbb{R}^{\dim \nu \times \dim \theta}$$

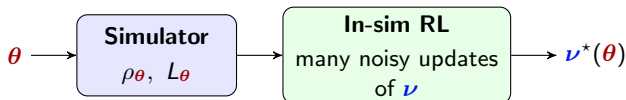
$$\hat{J}_{\theta}(\nu) = \mathbb{E}_{\rho_{\theta}, \pi_{\nu}} \left[\sum_k \gamma^k L_{\theta}(s_k, a_k) \right]$$

Needed in previous slide: $\nabla_{\theta} \pi_{\theta} = \nabla_{\nu} \pi_{\nu} \big|_{\nu^*} \kappa$

How to get κ ?

- 1 **Unroll training:** (pyTorch) backpropagate through all updates $\nu_{j+1} = \nu_j - \beta \hat{g}_j$
✗ cost $\propto$ number of updates
- 2 **Differentiate the Bellman optimality equations** (Nikishin et al., 2022)
✗ uses $\pi = \text{softmax}(Q^*)$ to manage $\arg \max_a Q^*(s, a)$

What we need: the sensitivity of a trained policy



In-sim policy training, for a given θ

Policy sensitivity

$$\nu^*(\theta) = \arg \min_{\nu} \hat{J}_{\theta}(\nu)$$

$$\kappa := \frac{d\nu^*}{d\theta} \in \mathbb{R}^{\dim \nu \times \dim \theta}$$

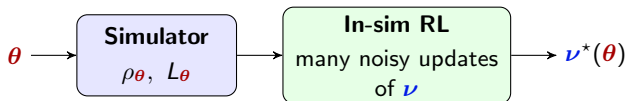
$$\hat{J}_{\theta}(\nu) = \mathbb{E}_{\rho_{\theta}, \pi_{\nu}} \left[\sum_k \gamma^k L_{\theta}(s_k, a_k) \right]$$

Needed in previous slide: $\nabla_{\theta} \pi_{\theta} = \nabla_{\nu} \pi_{\nu} \big|_{\nu^*} \kappa$

How to get κ ?

- 1 **Unroll training:** (pyTorch) backpropagate through all updates $\nu_{j+1} = \nu_j - \beta \hat{g}_j$
✗ cost $\propto$ number of updates
- 2 **Differentiate the Bellman optimality equations** (Nikishin et al., 2022)
✗ uses $\pi = \text{softmax}(Q^*)$ to manage $\arg \max_a Q^*(s, a)$
- 3 **Implicitly differentiate the end point of training**
✓ any policy class, any policy gradient

What we need: the sensitivity of a trained policy



In-sim policy training, for a given θ

Policy sensitivity

$$\nu^*(\theta) = \arg \min_{\nu} \hat{J}_{\theta}(\nu)$$

$$\kappa := \frac{d\nu^*}{d\theta} \in \mathbb{R}^{\dim \nu \times \dim \theta}$$

$$\hat{J}_{\theta}(\nu) = \mathbb{E}_{\rho_{\theta}, \pi_{\nu}} \left[\sum_k \gamma^k L_{\theta}(s_k, a_k) \right]$$

Needed in previous slide: $\nabla_{\theta} \pi_{\theta} = \nabla_{\nu} \pi_{\nu} \Big|_{\nu^*} \kappa$

How to get κ ?

- 1 **Unroll training:** (pyTorch) backpropagate through all updates $\nu_{j+1} = \nu_j - \beta \hat{g}_j$
✗ cost $\propto$ number of updates
- 2 **Differentiate the Bellman optimality equations** (Nikishin et al., 2022)
✗ uses $\pi = \text{softmax}(Q^*)$ to manage $\arg \max_a Q^*(s, a)$
- 3 **Implicitly differentiate the end point of training**
✓ any policy class, any policy gradient

How to differentiate through a (simulation-based) MDP?

Two ways to differentiate an expectation

- **Problem:** estimate $\nabla_{\psi} \mathbb{E}_{x \sim p_{\psi}} [f(x)]$ from samples $x_1, \dots, x_n \sim p_{\psi}$ (ψ : later θ or ν)

Two ways to differentiate an expectation

- **Problem:** estimate $\nabla_{\psi} \mathbb{E}_{x \sim p_{\psi}} [f(x)]$ from samples $x_1, \dots, x_n \sim p_{\psi}$ (ψ : later θ or ν)
- **Idea:** a gradient is the change of the sample average when ψ moves a little

$$\nabla_{\psi} \mathbb{E}[f(x)] \approx \frac{\mathbb{E}_{\psi+d\psi}[f] - \mathbb{E}_{\psi}[f]}{d\psi}, \quad \mathbb{E}_{\psi}[f] \approx \frac{1}{n} \sum_i f(x_i)$$

- **Question:** how to get the *new* average $\mathbb{E}_{\psi+d\psi}[f]$ from the *same* samples?

Two ways to differentiate an expectation

- **Problem:** estimate $\nabla_{\psi} \mathbb{E}_{x \sim p_{\psi}} [f(x)]$ from samples $x_1, \dots, x_n \sim p_{\psi}$ (ψ : later θ or ν)

- **Idea:** a gradient is the change of the sample average when ψ moves a little

$$\nabla_{\psi} \mathbb{E}[f(x)] \approx \frac{\mathbb{E}_{\psi+d\psi}[f] - \mathbb{E}_{\psi}[f]}{d\psi}, \quad \mathbb{E}_{\psi}[f] \approx \frac{1}{n} \sum_i f(x_i)$$

- **Question:** how to get the *new* average $\mathbb{E}_{\psi+d\psi}[f]$ from the *same* samples?

LR (likelihood ratio): **re-weight** samples

$$\nabla_{\psi} \mathbb{E}[f(x)] = \mathbb{E}\left[f(x) \nabla_{\psi} \log p_{\psi}(x)\right]$$



- samples $x_i \sim p_{\psi}$ stay where they are

Pictures: $x \sim \mathcal{N}(\psi, 1)$; gray = p_{ψ} , dashed = $p_{\psi+d\psi}$

Two ways to differentiate an expectation

- **Problem:** estimate $\nabla_{\psi} \mathbb{E}_{x \sim p_{\psi}} [f(x)]$ from samples $x_1, \dots, x_n \sim p_{\psi}$ (ψ : later θ or ν)

- **Idea:** a gradient is the change of the sample average when ψ moves a little

$$\nabla_{\psi} \mathbb{E}[f(x)] \approx \frac{\mathbb{E}_{\psi+d\psi}[f] - \mathbb{E}_{\psi}[f]}{d\psi}, \quad \mathbb{E}_{\psi}[f] \approx \frac{1}{n} \sum_i f(x_i)$$

- **Question:** how to get the *new* average $\mathbb{E}_{\psi+d\psi}[f]$ from the *same* samples?

LR (likelihood ratio): **re-weight** samples

$$\nabla_{\psi} \mathbb{E}[f(x)] = \mathbb{E}\left[f(x) \nabla_{\psi} \log p_{\psi}(x)\right]$$



- samples $x_i \sim p_{\psi}$ stay where they are

- $w_i = \frac{p_{\psi+d\psi}(x_i)}{p_{\psi}} \approx 1 + \nabla_{\psi} \log p_{\psi}(x_i) d\psi$

Pictures: $x \sim \mathcal{N}(\psi, 1)$; gray = p_{ψ} , dashed = $p_{\psi+d\psi}$

Two ways to differentiate an expectation

- **Problem:** estimate $\nabla_{\psi} \mathbb{E}_{x \sim p_{\psi}} [f(x)]$ from samples $x_1, \dots, x_n \sim p_{\psi}$ (ψ : later θ or ν)

- **Idea:** a gradient is the change of the sample average when ψ moves a little

$$\nabla_{\psi} \mathbb{E}[f(x)] \approx \frac{\mathbb{E}_{\psi+d\psi}[f] - \mathbb{E}_{\psi}[f]}{d\psi}, \quad \mathbb{E}_{\psi}[f] \approx \frac{1}{n} \sum_i f(x_i)$$

- **Question:** how to get the *new* average $\mathbb{E}_{\psi+d\psi}[f]$ from the *same* samples?

LR (likelihood ratio): **re-weight** samples

$$\nabla_{\psi} \mathbb{E}[f(x)] = \mathbb{E}\left[f(x) \nabla_{\psi} \log p_{\psi}(x)\right]$$



- samples $x_i \sim p_{\psi}$ stay where they are
- $w_i = \frac{p_{\psi+d\psi}(x_i)}{p_{\psi}(x_i)} \approx 1 + \nabla_{\psi} \log p_{\psi}(x_i) d\psi$
- new average $\approx \frac{1}{n} \sum_i w_i f(x_i)$

Pictures: $x \sim \mathcal{N}(\psi, 1)$; gray = p_{ψ} , dashed = $p_{\psi+d\psi}$

Two ways to differentiate an expectation

- **Problem:** estimate $\nabla_{\psi} \mathbb{E}_{x \sim p_{\psi}} [f(x)]$ from samples $x_1, \dots, x_n \sim p_{\psi}$ (ψ : later θ or ν)

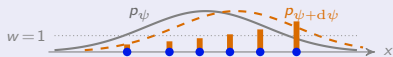
- **Idea:** a gradient is the change of the sample average when ψ moves a little

$$\nabla_{\psi} \mathbb{E}[f(x)] \approx \frac{\mathbb{E}_{\psi+d\psi}[f] - \mathbb{E}_{\psi}[f]}{d\psi}, \quad \mathbb{E}_{\psi}[f] \approx \frac{1}{n} \sum_i f(x_i)$$

- **Question:** how to get the *new* average $\mathbb{E}_{\psi+d\psi}[f]$ from the *same* samples?

LR (likelihood ratio): re-weight samples

$$\nabla_{\psi} \mathbb{E}[f(x)] = \mathbb{E} \left[f(x) \nabla_{\psi} \log p_{\psi}(x) \right]$$



- samples $x_i \sim p_{\psi}$ stay where they are
- $w_i = \frac{p_{\psi+d\psi}}{p_{\psi}}(x_i) \approx 1 + \nabla_{\psi} \log p_{\psi}(x_i) d\psi$
- new average $\approx \frac{1}{n} \sum_i w_i f(x_i)$

PD (reparameterization): move samples

$$\nabla_{\psi} \mathbb{E}[f(x)] = \mathbb{E}_{\varepsilon} \left[\nabla_x f(x) \nabla_{\psi} g_{\psi}(\varepsilon) \right]$$



- samples $x_i = g_{\psi}(\varepsilon_i)$, noise $\varepsilon_i \sim p(\varepsilon)$

Pictures: $x \sim \mathcal{N}(\psi, 1)$; gray = p_{ψ} , dashed = $p_{\psi+d\psi}$

Two ways to differentiate an expectation

- **Problem:** estimate $\nabla_{\psi} \mathbb{E}_{x \sim p_{\psi}} [f(x)]$ from samples $x_1, \dots, x_n \sim p_{\psi}$ (ψ : later θ or ν)

- **Idea:** a gradient is the change of the sample average when ψ moves a little

$$\nabla_{\psi} \mathbb{E}[f(x)] \approx \frac{\mathbb{E}_{\psi+d\psi}[f] - \mathbb{E}_{\psi}[f]}{d\psi}, \quad \mathbb{E}_{\psi}[f] \approx \frac{1}{n} \sum_i f(x_i)$$

- **Question:** how to get the *new* average $\mathbb{E}_{\psi+d\psi}[f]$ from the *same* samples?

LR (likelihood ratio): re-weight samples

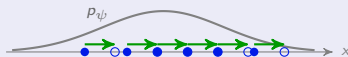
$$\nabla_{\psi} \mathbb{E}[f(x)] = \mathbb{E}[f(x) \nabla_{\psi} \log p_{\psi}(x)]$$



- samples $x_i \sim p_{\psi}$ stay where they are
- $w_i = \frac{p_{\psi+d\psi}(x_i)}{p_{\psi}} \approx 1 + \nabla_{\psi} \log p_{\psi}(x_i) d\psi$
- new average $\approx \frac{1}{n} \sum_i w_i f(x_i)$

PD (reparameterization): move samples

$$\nabla_{\psi} \mathbb{E}[f(x)] = \mathbb{E}_{\varepsilon} [\nabla_x f(x) \nabla_{\psi} g_{\psi}(\varepsilon)]$$



- samples $x_i = g_{\psi}(\varepsilon_i)$, noise $\varepsilon_i \sim p(\varepsilon)$
- keep ε_i , move: $x'_i \approx x_i + \nabla_{\psi} g_{\psi}(\varepsilon_i) d\psi$

Pictures: $x \sim \mathcal{N}(\psi, 1)$; gray = p_{ψ} , dashed = $p_{\psi+d\psi}$

Two ways to differentiate an expectation

- **Problem:** estimate $\nabla_{\psi} \mathbb{E}_{x \sim p_{\psi}} [f(x)]$ from samples $x_1, \dots, x_n \sim p_{\psi}$ (ψ : later θ or ν)

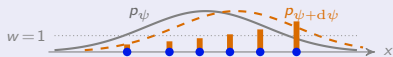
- **Idea:** a gradient is the change of the sample average when ψ moves a little

$$\nabla_{\psi} \mathbb{E}[f(x)] \approx \frac{\mathbb{E}_{\psi+d\psi}[f] - \mathbb{E}_{\psi}[f]}{d\psi}, \quad \mathbb{E}_{\psi}[f] \approx \frac{1}{n} \sum_i f(x_i)$$

- **Question:** how to get the *new average* $\mathbb{E}_{\psi+d\psi}[f]$ from the *same samples*?

LR (likelihood ratio): re-weight samples

$$\nabla_{\psi} \mathbb{E}[f(x)] = \mathbb{E} \left[f(x) \nabla_{\psi} \log p_{\psi}(x) \right]$$



- samples $x_i \sim p_{\psi}$ stay where they are
- $w_i = \frac{p_{\psi+d\psi}(x_i)}{p_{\psi}(x_i)} \approx 1 + \nabla_{\psi} \log p_{\psi}(x_i) d\psi$
- new average $\approx \frac{1}{n} \sum_i w_i f(x_i)$

PD (reparameterization): move samples

$$\nabla_{\psi} \mathbb{E}[f(x)] = \mathbb{E}_{\varepsilon} \left[\nabla_x f(x) \nabla_{\psi} g_{\psi}(\varepsilon) \right]$$



- samples $x_i = g_{\psi}(\varepsilon_i)$, noise $\varepsilon_i \sim p(\varepsilon)$
- keep ε_i , move: $x'_i \approx x_i + \nabla_{\psi} g_{\psi}(\varepsilon_i) d\psi$
- new average $\approx \frac{1}{n} \sum_i f(x'_i)$

Pictures: $x \sim \mathcal{N}(\psi, 1)$; gray = p_{ψ} , dashed = $p_{\psi+d\psi}$

Two families of policy gradients

RL: $x \rightarrow \text{trajectory } \tau = (s_0, a_0, s_1, \dots)$, $\psi \rightarrow \boldsymbol{\nu}, \boldsymbol{\theta}$, $f \rightarrow \sum_k \gamma^k L_{\boldsymbol{\theta}}(s_k, a_k)$

Two families of policy gradients

RL: $x \rightarrow \text{trajectory } \tau = (s_0, a_0, s_1, \dots)$, $\psi \rightarrow \nu, \theta$, $f \rightarrow \sum_k \gamma^k L_\theta(s_k, a_k)$

LR (score function): **re-weight** trajectories

- τ has a **density**:

$$p(\tau) = \rho_0(s_0) \prod_k \pi_\nu(a_k | s_k) \rho_\theta(s_{k+1} | s_k, a_k)$$

Two families of policy gradients

RL: $x \rightarrow \text{trajectory } \tau = (s_0, a_0, s_1, \dots)$, $\psi \rightarrow \nu, \theta$, $f \rightarrow \sum_k \gamma^k L_\theta(s_k, a_k)$

LR (score function): **re-weight** trajectories

- τ has a **density**:
$$p(\tau) = \rho_0(s_0) \prod_k \pi_\nu(a_k | s_k) \rho_\theta(s_{k+1} | s_k, a_k)$$
- score: $\nabla_\nu \log p(\tau) = \sum_k \nabla_\nu \log \pi_\nu(a_k | s_k)$

Two families of policy gradients

RL: $x \rightarrow \text{trajectory } \tau = (s_0, a_0, s_1, \dots)$, $\psi \rightarrow \nu, \theta$, $f \rightarrow \sum_k \gamma^k L_\theta(s_k, a_k)$

LR (score function): re-weight trajectories

- τ has a **density**:
$$p(\tau) = \rho_0(s_0) \prod_k \pi_\nu(a_k | s_k) \rho_\theta(s_{k+1} | s_k, a_k)$$
- score: $\nabla_\nu \log p(\tau) = \sum_k \nabla_\nu \log \pi_\nu(a_k | s_k)$
 - ▶ the model ρ_θ drops out

Two families of policy gradients

RL: $x \rightarrow \text{trajectory } \tau = (s_0, a_0, s_1, \dots)$, $\psi \rightarrow \nu, \theta$, $f \rightarrow \sum_k \gamma^k L_\theta(s_k, a_k)$

LR (score function): re-weight trajectories

- τ has a **density**:
$$p(\tau) = \rho_0(s_0) \prod_k \pi_\nu(a_k | s_k) \rho_\theta(s_{k+1} | s_k, a_k)$$
- score: $\nabla_\nu \log p(\tau) = \sum_k \nabla_\nu \log \pi_\nu(a_k | s_k)$
 - ▶ the model ρ_θ drops out

PD (pathwise): move trajectories

- τ is a **function** of fixed noise (ε_k, ξ_k) :
$$a_k = g_\nu(s_k, \varepsilon_k), \quad s_{k+1} = f_\theta(s_k, a_k, \xi_k)$$

Two families of policy gradients

RL: $x \rightarrow \text{trajectory } \tau = (s_0, a_0, s_1, \dots)$, $\psi \rightarrow \nu, \theta$, $f \rightarrow \sum_k \gamma^k L_\theta(s_k, a_k)$

LR (score function): re-weight trajectories

- τ has a **density**:
 $p(\tau) = \rho_0(s_0) \prod_k \pi_\nu(a_k|s_k) \rho_\theta(s_{k+1}|s_k, a_k)$
- score: $\nabla_\nu \log p(\tau) = \sum_k \nabla_\nu \log \pi_\nu(a_k|s_k)$
 - ▶ the model ρ_θ drops out

PD (pathwise): move trajectories

- τ is a **function** of fixed noise (ε_k, ξ_k) :
 $a_k = g_\nu(s_k, \varepsilon_k)$, $s_{k+1} = f_\theta(s_k, a_k, \xi_k)$
- gradients: $\frac{ds_{k+1}}{d\nu} = \nabla_s f_\theta \frac{ds_k}{d\nu} + \nabla_a f_\theta \frac{da_k}{d\nu}$

Two families of policy gradients

RL: $x \rightarrow \text{trajectory } \tau = (s_0, a_0, s_1, \dots)$, $\psi \rightarrow \nu, \theta$, $f \rightarrow \sum_k \gamma^k L_\theta(s_k, a_k)$

LR (score function): re-weight trajectories

- τ has a **density**:
$$p(\tau) = \rho_0(s_0) \prod_k \pi_\nu(a_k | s_k) \rho_\theta(s_{k+1} | s_k, a_k)$$
- score: $\nabla_\nu \log p(\tau) = \sum_k \nabla_\nu \log \pi_\nu(a_k | s_k)$
 - ▶ the model ρ_θ drops out

PD (pathwise): move trajectories

- τ is a **function** of fixed noise (ε_k, ξ_k) :
$$a_k = g_\nu(s_k, \varepsilon_k), \quad s_{k+1} = f_\theta(s_k, a_k, \xi_k)$$
- gradients: $\frac{ds_{k+1}}{d\nu} = \nabla_s f_\theta \frac{ds_k}{d\nu} + \nabla_a f_\theta \frac{da_k}{d\nu}$
 - ▶ the model f_θ stays in

Two families of policy gradients

RL: $x \rightarrow \text{trajectory } \tau = (s_0, a_0, s_1, \dots)$, $\psi \rightarrow \nu, \theta$, $f \rightarrow \sum_k \gamma^k L_\theta(s_k, a_k)$

LR (score function): re-weight trajectories

- τ has a **density**:
 $p(\tau) = \rho_0(s_0) \prod_k \pi_\nu(a_k | s_k) \rho_\theta(s_{k+1} | s_k, a_k)$
- score: $\nabla_\nu \log p(\tau) = \sum_k \nabla_\nu \log \pi_\nu(a_k | s_k)$
 - ▶ the model ρ_θ drops out

PD (pathwise): move trajectories

- τ is a **function** of fixed noise (ε_k, ξ_k) :
 $a_k = g_\nu(s_k, \varepsilon_k)$, $s_{k+1} = f_\theta(s_k, a_k, \xi_k)$
- gradients: $\frac{ds_{k+1}}{d\nu} = \nabla_s f_\theta \frac{ds_k}{d\nu} + \nabla_a f_\theta \frac{da_k}{d\nu}$
 - ▶ the model f_θ stays in

The policy gradient in the two views

LR: $\nabla_\nu \hat{J}_\theta = \mathbb{E} \left[\sum_k \gamma^k \nabla_\nu \log \pi_\nu(a_k | s_k) Q_\theta^{\pi_\nu}(s_k, a_k) \right]$ REINFORCE, A2C, PPO

Two families of policy gradients

RL: $x \rightarrow \text{trajectory } \tau = (s_0, a_0, s_1, \dots)$, $\psi \rightarrow \nu, \theta$, $f \rightarrow \sum_k \gamma^k L_\theta(s_k, a_k)$

LR (score function): re-weight trajectories

- τ has a **density**:
 $p(\tau) = \rho_0(s_0) \prod_k \pi_\nu(a_k|s_k) \rho_\theta(s_{k+1}|s_k, a_k)$
- score: $\nabla_\nu \log p(\tau) = \sum_k \nabla_\nu \log \pi_\nu(a_k|s_k)$
 - the model ρ_θ drops out

PD (pathwise): move trajectories

- τ is a **function** of fixed noise (ε_k, ξ_k) :
 $a_k = g_\nu(s_k, \varepsilon_k)$, $s_{k+1} = f_\theta(s_k, a_k, \xi_k)$
- gradients: $\frac{ds_{k+1}}{d\nu} = \nabla_s f_\theta \frac{ds_k}{d\nu} + \nabla_a f_\theta \frac{da_k}{d\nu}$
 - the model f_θ stays in

The policy gradient in the two views

LR: $\nabla_\nu \hat{J}_\theta = \mathbb{E} \left[\sum_k \gamma^k \nabla_\nu \log \pi_\nu(a_k|s_k) Q_\theta^{\pi_\nu}(s_k, a_k) \right]$ REINFORCE, A2C, PPO

PD: $\nabla_\nu \hat{J}_\theta = \mathbb{E}_\omega \left[\sum_k \gamma^k \left(\nabla_s L_\theta \frac{ds_k}{d\nu} + \nabla_a L_\theta \frac{da_k}{d\nu} \right) \right]$ BPTT, SVG, SVG

Two families of policy gradients

RL: $x \rightarrow \text{trajectory } \tau = (s_0, a_0, s_1, \dots), \psi \rightarrow \nu, \theta, f \rightarrow \sum_k \gamma^k L_\theta(s_k, a_k)$

LR (score function): **re-weight** trajectories

- τ has a **density**:
 $p(\tau) = \rho_0(s_0) \prod_k \pi_\nu(a_k|s_k) \rho_\theta(s_{k+1}|s_k, a_k)$
- score: $\nabla_\nu \log p(\tau) = \sum_k \nabla_\nu \log \pi_\nu(a_k|s_k)$
 - ▶ the model ρ_θ drops out

PD (pathwise): **move** trajectories

- τ is a **function** of fixed noise (ε_k, ξ_k) :
 $a_k = g_\nu(s_k, \varepsilon_k), s_{k+1} = f_\theta(s_k, a_k, \xi_k)$
- gradients: $\frac{ds_{k+1}}{d\nu} = \nabla_s f_\theta \frac{ds_k}{d\nu} + \nabla_a f_\theta \frac{da_k}{d\nu}$
 - ▶ the model f_θ stays in

The policy gradient in the two views

LR: $\nabla_\nu \hat{J}_\theta = \mathbb{E} \left[\sum_k \gamma^k \nabla_\nu \log \pi_\nu(a_k|s_k) Q_\theta^{\pi_\nu}(s_k, a_k) \right]$ REINFORCE, A2C, PPO

PD: $\nabla_\nu \hat{J}_\theta = \mathbb{E}_\omega \left[\sum_k \gamma^k \left(\nabla_s L_\theta \frac{ds_k}{d\nu} + \nabla_a L_\theta \frac{da_k}{d\nu} \right) \right]$ BPTT, SVG, SVG

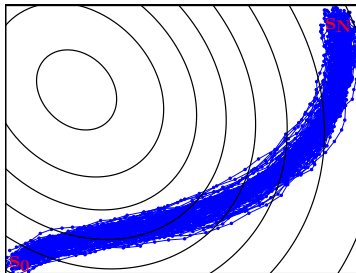
Two estimates of the same gradient, but the simulator θ sits in different places:

LR: only in the **sampling** (ρ_θ) and the critic Q_θ

PD: inside each path, through the **Jacobians** of f_θ .

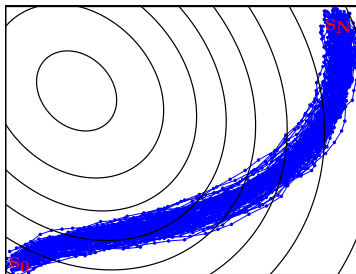
This decides how we will differentiate training with respect to θ later.

Stochastic trajectories: sampling the simulator



$$\hat{J}_{\theta}(\nu) \approx \frac{1}{n} \sum_{i=1}^n \sum_{k=0}^N \gamma^k L_{\theta}(s_k^i, a_k^i)$$

Stochastic trajectories: sampling the simulator



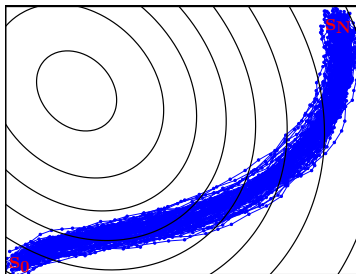
Perturb $\theta \rightarrow \theta + d\theta$: what happens to the samples?

LR: same paths, **new weights**



$$\hat{J}_{\theta}(\nu) \approx \frac{1}{n} \sum_{i=1}^n \sum_{k=0}^N \gamma^k L_{\theta}(s_k^i, a_k^i)$$

Stochastic trajectories: sampling the simulator



Perturb $\theta \rightarrow \theta + d\theta$: what happens to the samples?

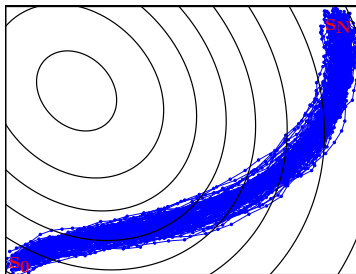
LR: same paths, **new weights**



$$w_i = \frac{p_{\theta+d\theta}(\tau_i)}{p_{\theta}(\tau_i)} \approx 1 + \sum_k \nabla_{\theta} \log \rho_{\theta}(s_{k+1}^i | s_k^i, a_k^i) d\theta$$

$$\hat{J}_{\theta}(\nu) \approx \frac{1}{n} \sum_{i=1}^n \sum_{k=0}^N \gamma^k L_{\theta}(s_k^i, a_k^i)$$

Stochastic trajectories: sampling the simulator



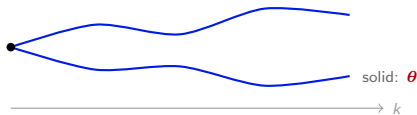
Perturb $\theta \rightarrow \theta + d\theta$: what happens to the samples?

LR: same paths, **new weights**



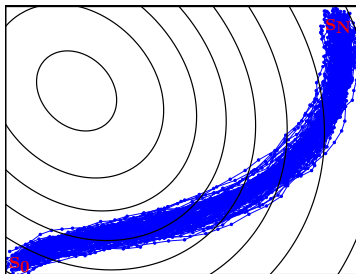
$$w_i = \frac{p_{\theta+d\theta}(\tau_i)}{p_{\theta}(\tau_i)} \approx 1 + \sum_k \nabla_{\theta} \log p_{\theta}(s_{k+1}^i | s_k^i, a_k^i) d\theta$$

PD: same noise ω_i , **moved paths**



$$\hat{J}_{\theta}(\nu) \approx \frac{1}{n} \sum_{i=1}^n \sum_{k=0}^N \gamma^k L_{\theta}(s_k^i, a_k^i)$$

Stochastic trajectories: sampling the simulator



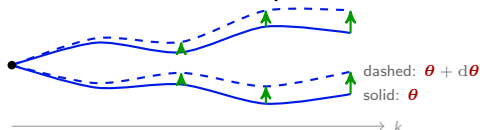
Perturb $\theta \rightarrow \theta + d\theta$: what happens to the samples?

LR: same paths, **new weights**



$$w_i = \frac{p_{\theta+d\theta}(\tau_i)}{p_{\theta}(\tau_i)} \approx 1 + \sum_k \nabla_{\theta} \log p_{\theta}(s_{k+1}^i | s_k^i, a_k^i) d\theta$$

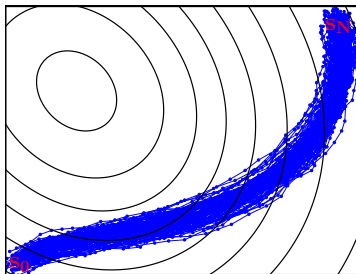
PD: same noise ω_i , **moved paths**



$s_k^i(\theta + d\theta) \approx s_k^i(\theta) + S_k^i d\theta$, with $S_k = ds_k/d\theta$
(chain rule along the path)

$$\hat{J}_{\theta}(\nu) \approx \frac{1}{n} \sum_{i=1}^n \sum_{k=0}^N \gamma^k L_{\theta}(s_k^i, a_k^i)$$

Stochastic trajectories: sampling the simulator



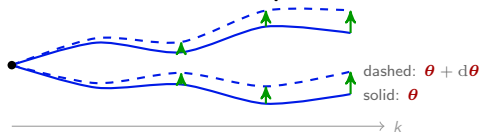
Perturb $\theta \rightarrow \theta + d\theta$: what happens to the samples?

LR: same paths, **new weights**



$$w_i = \frac{p_{\theta+d\theta}(\tau_i)}{p_{\theta}(\tau_i)} \approx 1 + \sum_k \nabla_{\theta} \log p_{\theta}(s_{k+1}^i | s_k^i, a_k^i) d\theta$$

PD: same noise ω_i , **moved paths**

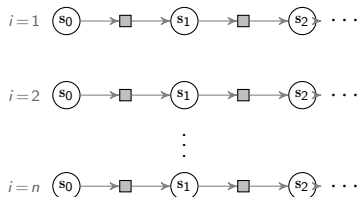


$$s_k^i(\theta + d\theta) \approx s_k^i(\theta) + S_k^i d\theta, \text{ with } S_k = ds_k/d\theta$$

(chain rule along the path)

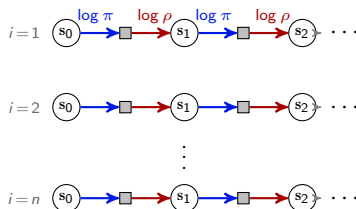
LR changes **how likely** each history is; **PD** changes **where** it goes

LR gradients: sample, score, average



1. Simulate n episodes with π_{ν} , keep the sampled (s_k, a_k) ; no gradient flows through the simulator

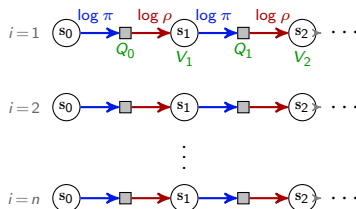
LR gradients: sample, score, average



1. Simulate n episodes with π_ν , keep the sampled (s_k, a_k) ; no gradient flows through the simulator

2. Score: the log-probability of every sampled step, $\log \pi_\nu(a_k | s_k)$ (holds ν) and $\log \rho_\theta(s_{k+1} | s_k, a_k)$ (holds θ)

LR gradients: sample, score, average

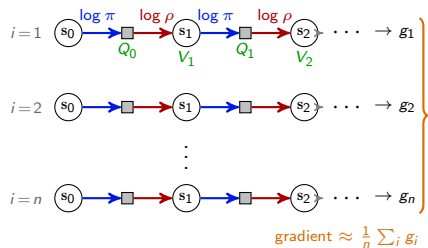


1. **Simulate** n episodes with π_{ν} , keep the sampled (s_k, a_k) ; no gradient flows through the simulator

2. **Score**: the log-probability of every sampled step, $\log \pi_{\nu}(a_k | s_k)$ (holds ν) and $\log \rho_{\theta}(s_{k+1} | s_k, a_k)$ (holds θ)

3. **Weight** each log-probability by the cost that **follows** it: Q_k or V_{k+1} (a critic or the sampled return)

LR gradients: sample, score, average



1. **Simulate** n episodes with π_ν , keep the sampled (s_k, a_k) ; no gradient flows through the simulator

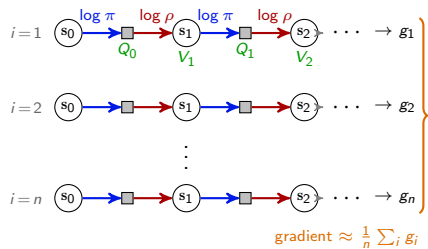
2. **Score**: the log-probability of every sampled step, $\log \pi_\nu(a_k | s_k)$ (holds ν) and $\log \rho_\theta(s_{k+1} | s_k, a_k)$ (holds θ)

3. **Weight** each log-probability by the cost that **follows** it: Q_k or V_{k+1} (a critic or the sampled return)

4. **Average** over the episodes:

$$\nabla_\nu \hat{J}_\theta \approx \frac{1}{n} \sum_{\text{episodes}} \sum_k \gamma^k \nabla_\nu \log \pi_\nu(a_k | s_k) Q_k$$

LR gradients: sample, score, average



1. **Simulate** n episodes with π_{ν} , keep the sampled (s_k, a_k) ; no gradient flows through the simulator

2. **Score**: the log-probability of every sampled step, $\log \pi_{\nu}(a_k | s_k)$ (holds ν) and $\log \rho_{\theta}(s_{k+1} | s_k, a_k)$ (holds θ)

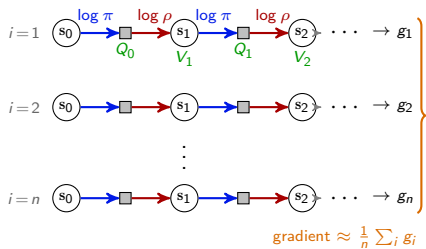
3. **Weight** each log-probability by the cost that **follows** it: Q_k or V_{k+1} (a critic or the sampled return)

4. **Average** over the episodes:

$$\nabla_{\nu} \hat{J}_{\theta} \approx \frac{1}{n} \sum_{\text{episodes}} \sum_k \gamma^k \nabla_{\nu} \log \pi_{\nu}(a_k | s_k) Q_k$$

$$\nabla_{\theta} \hat{J}_{\theta} \approx \frac{1}{n} \sum_{\text{episodes}} \sum_k \gamma^k (\nabla_{\theta} L_{\theta}(s_k, a_k) + \gamma \nabla_{\theta} \log \rho_{\theta}(s_{k+1} | s_k, a_k) V_{k+1})$$

LR gradients: sample, score, average



1. **Simulate** n episodes with π_{ν} , keep the sampled $(s_k, \mathbf{a}_k)$; no gradient flows through the simulator

2. **Score**: the log-probability of every sampled step, $\log \pi_{\nu}(\mathbf{a}_k | s_k)$ (holds ν) and $\log \rho_{\theta}(s_{k+1} | s_k, \mathbf{a}_k)$ (holds θ)

3. **Weight** each log-probability by the cost that **follows** it: Q_k or V_{k+1} (a critic or the sampled return)

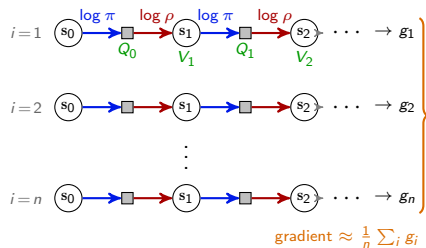
4. **Average** over the episodes:

$$\nabla_{\nu} \hat{J}_{\theta} \approx \frac{1}{n} \sum_{\text{episodes}} \sum_k \gamma^k \nabla_{\nu} \log \pi_{\nu}(\mathbf{a}_k | s_k) Q_k$$

$$\nabla_{\theta} \hat{J}_{\theta} \approx \frac{1}{n} \sum_{\text{episodes}} \sum_k \gamma^k (\nabla_{\theta} L_{\theta}(s_k, \mathbf{a}_k) + \gamma \nabla_{\theta} \log \rho_{\theta}(s_{k+1} | s_k, \mathbf{a}_k) V_{k+1})$$

In code (A2C, PPO): one backward pass on these sums without ∇ , with Q_k, V_{k+1} detached

LR gradients: sample, score, average



1. Simulate n episodes with π_ν , keep the sampled (s_k, a_k) ; no gradient flows through the simulator

2. Score: the log-probability of every sampled step, $\log \pi_\nu(a_k | s_k)$ (holds ν) and $\log \rho_\theta(s_{k+1} | s_k, a_k)$ (holds θ)

3. Weight each log-probability by the cost that **follows** it: Q_k or V_{k+1} (a critic or the sampled return)

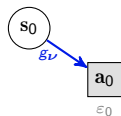
4. Average over the episodes:

$$\nabla_\nu \hat{J}_\theta \approx \frac{1}{n} \sum_{\text{episodes}} \sum_k \gamma^k \nabla_\nu \log \pi_\nu(a_k | s_k) Q_k$$

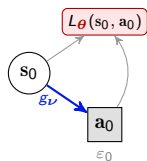
$$\nabla_\theta \hat{J}_\theta \approx \frac{1}{n} \sum_{\text{episodes}} \sum_k \gamma^k (\nabla_\theta L_\theta(s_k, a_k) + \gamma \nabla_\theta \log \rho_\theta(s_{k+1} | s_k, a_k) V_{k+1})$$

In code (A2C, PPO): one backward pass on these sums without ∇ , with Q_k, V_{k+1} detached
 ✓ no derivatives in (s, a) : black-box simulator for ∇_ν , non-smooth costs, discrete s, a
 ✗ ∇_θ needs the log-density $\log \rho_\theta$ of every simulated step ✗ variance grows with k, Q, V

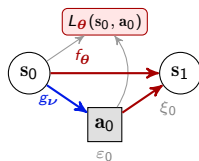
PD gradients: backpropagate through the simulator



PD gradients: backpropagate through the simulator

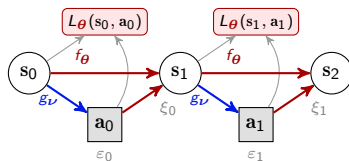


PD gradients: backpropagate through the simulator

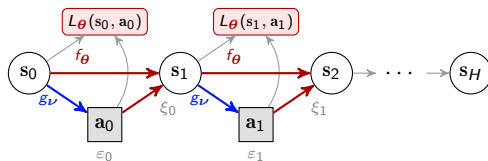


noise ε_k, ξ_k drawn once, then kept fixed: the path is a deterministic function of (ν, θ)

PD gradients: backpropagate through the simulator

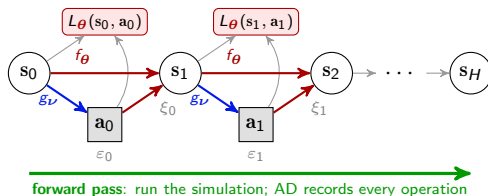


PD gradients: backpropagate through the simulator



noise ε_k, ξ_k drawn once, then kept fixed: the path is a deterministic function of (ν, θ)

PD gradients: backpropagate through the simulator



In practice (PyTorch, JAX)

draw the noise $\omega = (\varepsilon_k, \xi_k)_{k < H}$ once

forward: cost $J_{\omega}(\nu, \theta)$ of one rollout:

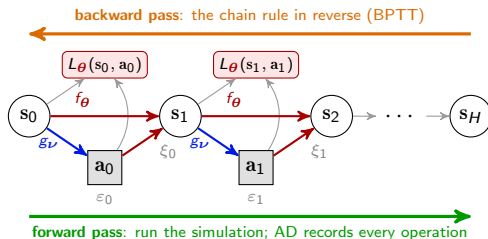
$s \leftarrow s_0$; for $k < H$:

$a \leftarrow g_{\nu}(s, \varepsilon_k)$, add $\gamma^k L_{\theta}(s, a)$,

$s \leftarrow f_{\theta}(s, a, \xi_k)$

- **forward** = the rollout itself; every call of g_{ν} , f_{θ} , L_{θ} is recorded

PD gradients: backpropagate through the simulator



In practice (PyTorch, JAX)

draw the noise $\omega = (\varepsilon_k, \xi_k)_{k < H}$ once

forward: cost $J_\omega(\boldsymbol{\nu}, \boldsymbol{\theta})$ of one rollout:

$s \leftarrow s_0$; for $k < H$:

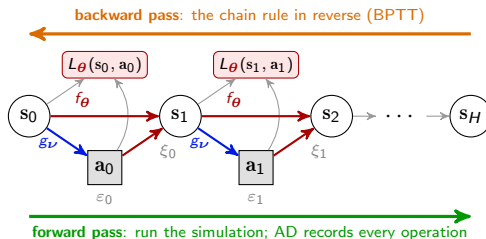
$\mathbf{a} \leftarrow g_\nu(s, \varepsilon_k)$, add $\gamma^k L_\theta(s, \mathbf{a})$,

$s \leftarrow f_\theta(s, \mathbf{a}, \xi_k)$

backward: $(\nabla_\nu J_\omega, \nabla_\theta J_\omega) = \text{grad}(J_\omega)$

- **forward** = the rollout itself; every call of g_ν , f_θ , L_θ is recorded
- **backward** = one reverse sweep through the recorded steps: ∇_ν and ∇_θ together, for any $\dim \nu$, $\dim \theta$, at the cost of about one more rollout

PD gradients: backpropagate through the simulator



In practice (PyTorch, JAX)

draw the noise $\omega = (\varepsilon_k, \xi_k)_{k < H}$ once

forward: cost $J_\omega(\nu, \theta)$ of one rollout:

$s \leftarrow s_0$; for $k < H$:

$a \leftarrow g_\nu(s, \varepsilon_k)$, add $\gamma^k L_\theta(s, a)$,

$s \leftarrow f_\theta(s, a, \xi_k)$

backward: $(\nabla_\nu J_\omega, \nabla_\theta J_\omega) = \text{grad}(J_\omega)$

average over n noise draws $\omega_i \Rightarrow \nabla_\nu \hat{J}_\theta$,

$\nabla_\theta \hat{J}_\theta$

- **forward** = the rollout itself; every call of g_ν , f_θ , L_θ is recorded
- **backward** = one reverse sweep through the recorded steps: ∇_ν and ∇_θ together, for any $\dim \nu$, $\dim \theta$, at the cost of about one more rollout
- it evaluates the **PD** fequation path by path with AD.

PD gradients: exploding variance and the terminal critic

What the backward pass multiplies: the effect of a_0 on the state k steps later

$$\frac{ds_k}{da_0} = \frac{ds_k}{ds_{k-1}} \dots \frac{ds_2}{ds_1} \frac{\partial s_1}{\partial a_0} \quad \text{each factor} = \text{one closed-loop step through } f_{\theta} \text{ and } g_{\nu}$$

PD gradients: exploding variance and the terminal critic

What the backward pass multiplies: the effect of a_0 on the state k steps later

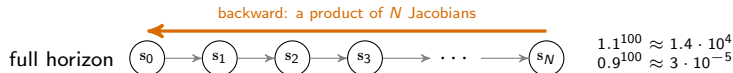
$$\frac{ds_k}{da_0} = \frac{ds_k}{ds_{k-1}} \dots \frac{ds_2}{ds_1} \frac{\partial s_1}{\partial a_0} \quad \text{each factor} = \text{one closed-loop step through } f_{\theta} \text{ and } g_{\nu}$$



PD gradients: exploding variance and the terminal critic

What the backward pass multiplies: the effect of a_0 on the state k steps later

$$\frac{ds_k}{da_0} = \frac{ds_k}{ds_{k-1}} \dots \frac{ds_2}{ds_1} \frac{\partial s_1}{\partial a_0} \quad \text{each factor} = \text{one closed-loop step through } f_{\theta} \text{ and } g_{\nu}$$

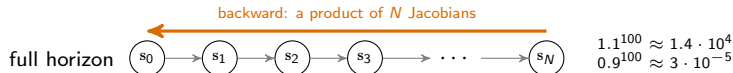


- **Catch:** for long horizons, contacts or chaotic dynamics the product **explodes or vanishes** and differs a lot between noise draws $\Rightarrow$ **high variance**

PD gradients: exploding variance and the terminal critic

What the backward pass multiplies: the effect of $\mathbf{a}_0$ on the state k steps later

$$\frac{ds_k}{da_0} = \frac{ds_k}{ds_{k-1}} \cdots \frac{ds_2}{ds_1} \frac{\partial s_1}{\partial a_0} \quad \text{each factor} = \text{one closed-loop step through } f_{\theta} \text{ and } g_{\nu}$$

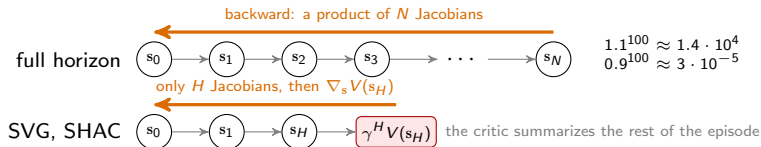


- **Catch:** for long horizons, contacts or chaotic dynamics the product **explodes or vanishes** and differs a lot between noise draws $\Rightarrow$ **high variance**
- **LR or PD?** Neither always has lower variance: **PD** wins on smooth, stable systems, **LR** on chaotic, stiff or contact-rich ones (Parmas et al., 2018; Suh et al., 2022)

PD gradients: exploding variance and the terminal critic

What the backward pass multiplies: the effect of $\mathbf{a}_0$ on the state k steps later

$$\frac{ds_k}{da_0} = \frac{ds_k}{ds_{k-1}} \dots \frac{ds_2}{ds_1} \frac{\partial s_1}{\partial a_0} \quad \text{each factor} = \text{one closed-loop step through } f_{\theta} \text{ and } g_{\nu}$$



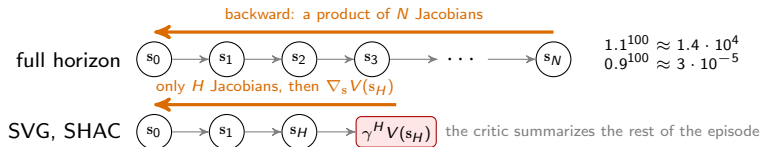
- **Catch:** for long horizons, contacts or chaotic dynamics the product **explodes or vanishes** and differs a lot between noise draws $\Rightarrow$ **high variance**
- **LR or PD?** Neither always has lower variance: **PD** wins on smooth, stable systems, **LR** on chaotic, stiff or contact-rich ones (Parmas et al., 2018; Suh et al., 2022)
- **Fix:** unroll only H steps and add a **terminal critic**:

$$J_{\omega, H} = \sum_{k < H} \gamma^k L_{\theta}(s_k, \mathbf{a}_k) + \gamma^H V(s_H)$$

PD gradients: exploding variance and the terminal critic

What the backward pass multiplies: the effect of $\mathbf{a}_0$ on the state k steps later

$$\frac{ds_k}{da_0} = \frac{ds_k}{ds_{k-1}} \cdots \frac{ds_2}{ds_1} \frac{\partial s_1}{\partial a_0} \quad \text{each factor} = \text{one closed-loop step through } f_{\theta} \text{ and } g_{\nu}$$



- **Catch:** for long horizons, contacts or chaotic dynamics the product **explodes or vanishes** and differs a lot between noise draws $\Rightarrow$ **high variance**
- **LR or PD?** Neither always has lower variance: **PD** wins on smooth, stable systems, **LR** on chaotic, stiff or contact-rich ones (Parmas et al., 2018; Suh et al., 2022)
- **Fix:** unroll only H steps and add a **terminal critic**:

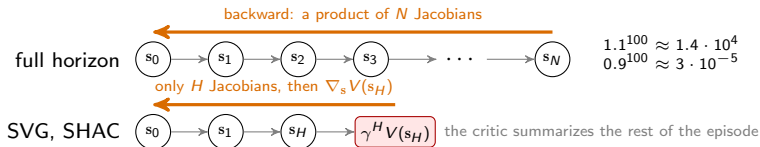
$$J_{\omega, H} = \sum_{k < H} \gamma^k L_{\theta}(s_k, \mathbf{a}_k) + \gamma^H V(s_H)$$

- short H : low variance but critic bias; SAC is case with no unroll step at all.

PD gradients: exploding variance and the terminal critic

What the backward pass multiplies: the effect of $\mathbf{a}_0$ on the state k steps later

$$\frac{ds_k}{da_0} = \frac{ds_k}{ds_{k-1}} \cdots \frac{ds_2}{ds_1} \frac{\partial s_1}{\partial a_0} \quad \text{each factor} = \text{one closed-loop step through } f_{\theta} \text{ and } g_{\nu}$$



- **Catch:** for long horizons, contacts or chaotic dynamics the product **explodes or vanishes** and differs a lot between noise draws $\Rightarrow$ **high variance**
- **LR or PD?** Neither always has lower variance: **PD** wins on smooth, stable systems, **LR** on chaotic, stiff or contact-rich ones (Parmas et al., 2018; Suh et al., 2022)
- **Fix:** unroll only H steps and add a **terminal critic**:

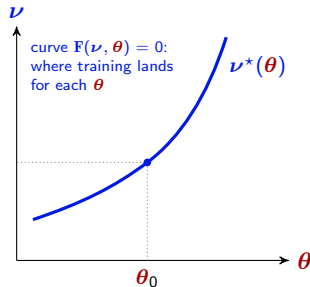
$$J_{\omega, H} = \sum_{k < H} \gamma^k L_{\theta}(s_k, \mathbf{a}_k) + \gamma^H V(s_H)$$

- short H : low variance but critic bias; SAC is case with no unroll step at all.
- ✓ exact per-path gradients; one backward pass for any $\dim \nu$, $\dim \theta$
- ✗ needs differentiable $f_{\theta}, g_{\nu}, L_{\theta}$ (smoothed contact, min, max) and continuous $\mathbf{a}, \theta$

Implicit differentiation of policy training

A trained policy is a stationary point of the in-sim objective

$$\mathbf{F}(\boldsymbol{\nu}^*, \boldsymbol{\theta}) := \nabla_{\boldsymbol{\nu}} \hat{J}_{\boldsymbol{\theta}}(\boldsymbol{\nu}^*) = 0$$



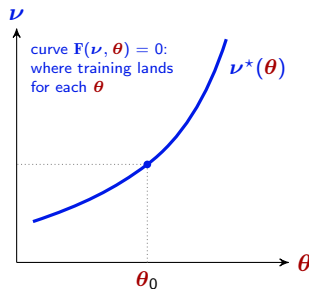
Implicit differentiation of policy training

A trained policy is a stationary point of the in-sim objective

$$\mathbf{F}(\boldsymbol{\nu}^*, \boldsymbol{\theta}) := \nabla_{\boldsymbol{\nu}} \hat{J}_{\boldsymbol{\theta}}(\boldsymbol{\nu}^*) = 0$$

LR: $\mathbb{E}[\sum_k \gamma^k \nabla_{\boldsymbol{\nu}} \log \pi_{\boldsymbol{\nu}} Q_{\boldsymbol{\theta}}^{\pi_{\boldsymbol{\nu}}}] = 0$ (PPO)

PD: $\mathbb{E}_{\omega}[\sum_k \gamma^k (\nabla_s L_{\boldsymbol{\theta}} \frac{ds_k}{d\boldsymbol{\nu}} + \nabla_a L_{\boldsymbol{\theta}} \frac{da_k}{d\boldsymbol{\nu}})] = 0$ (SVG)



Implicit differentiation of policy training

A trained policy is a stationary point of the in-sim objective

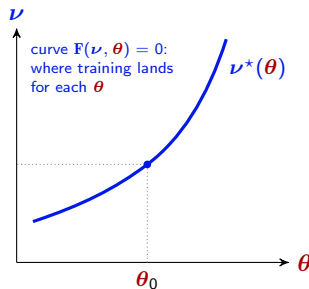
$$\mathbf{F}(\boldsymbol{\nu}^*, \boldsymbol{\theta}) := \nabla_{\boldsymbol{\nu}} \hat{J}_{\boldsymbol{\theta}}(\boldsymbol{\nu}^*) = 0$$

LR: $\mathbb{E}[\sum_k \gamma^k \nabla_{\boldsymbol{\nu}} \log \pi_{\boldsymbol{\nu}} Q_{\boldsymbol{\theta}}^{\pi_{\boldsymbol{\nu}}}] = 0$ (PPO)

PD: $\mathbb{E}_{\omega}[\sum_k \gamma^k (\nabla_{\mathbf{s}} L_{\boldsymbol{\theta}} \frac{d\mathbf{s}_k}{d\boldsymbol{\nu}} + \nabla_{\mathbf{a}} L_{\boldsymbol{\theta}} \frac{d\mathbf{a}_k}{d\boldsymbol{\nu}})] = 0$ (SVG)

Assumptions

- 1 **Smooth:** $\mathbf{F}$ is continuously differentiable in $(\boldsymbol{\nu}, \boldsymbol{\theta})$, e.g. $\pi_{\boldsymbol{\nu}}, \rho_{\boldsymbol{\theta}}$ (**LR**) or $g_{\boldsymbol{\nu}}, f_{\boldsymbol{\theta}}$ (**PD**) and $L_{\boldsymbol{\theta}}$ twice differentiable, with ∇ and $\mathbb{E}$ interchangeable



Implicit differentiation of policy training

A trained policy is a stationary point of the in-sim objective

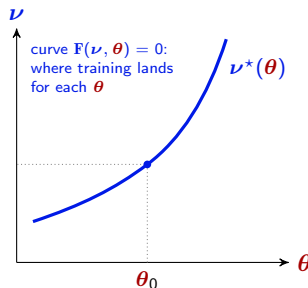
$$\mathbf{F}(\boldsymbol{\nu}^*, \boldsymbol{\theta}) := \nabla_{\boldsymbol{\nu}} \hat{J}_{\boldsymbol{\theta}}(\boldsymbol{\nu}^*) = 0$$

LR: $\mathbb{E}[\sum_k \gamma^k \nabla_{\boldsymbol{\nu}} \log \pi_{\boldsymbol{\nu}} Q_{\boldsymbol{\theta}}^{\pi_{\boldsymbol{\nu}}}] = 0$ (PPO)

PD: $\mathbb{E}_{\omega}[\sum_k \gamma^k (\nabla_s L_{\boldsymbol{\theta}} \frac{ds_k}{d\boldsymbol{\nu}} + \nabla_a L_{\boldsymbol{\theta}} \frac{da_k}{d\boldsymbol{\nu}})] = 0$ (SVG)

Assumptions

- 1 **Smooth:** $\mathbf{F}$ is continuously differentiable in $(\boldsymbol{\nu}, \boldsymbol{\theta})$, e.g. $\pi_{\boldsymbol{\nu}}, \rho_{\boldsymbol{\theta}}$ (**LR**) or $g_{\boldsymbol{\nu}}, f_{\boldsymbol{\theta}}$ (**PD**) and $L_{\boldsymbol{\theta}}$ twice differentiable, with ∇ and $\mathbb{E}$ interchangeable
- 2 **Converged:** training reached $\mathbf{F}(\boldsymbol{\nu}^*, \boldsymbol{\theta}) = 0$



Implicit differentiation of policy training

A trained policy is a stationary point of the in-sim objective

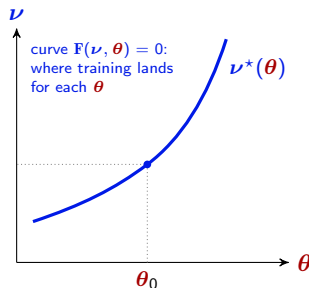
$$\mathbf{F}(\boldsymbol{\nu}^*, \boldsymbol{\theta}) := \nabla_{\boldsymbol{\nu}} \hat{J}_{\boldsymbol{\theta}}(\boldsymbol{\nu}^*) = 0$$

LR: $\mathbb{E}[\sum_k \gamma^k \nabla_{\boldsymbol{\nu}} \log \pi_{\boldsymbol{\nu}} Q_{\boldsymbol{\theta}}^{\pi_{\boldsymbol{\nu}}}] = 0$ (PPO)

PD: $\mathbb{E}_{\omega}[\sum_k \gamma^k (\nabla_s L_{\boldsymbol{\theta}} \frac{ds_k}{d\boldsymbol{\nu}} + \nabla_a L_{\boldsymbol{\theta}} \frac{da_k}{d\boldsymbol{\nu}})] = 0$ (SVG)

Assumptions

- 1 **Smooth:** $\mathbf{F}$ is continuously differentiable in $(\boldsymbol{\nu}, \boldsymbol{\theta})$, e.g. $\pi_{\boldsymbol{\nu}}, \rho_{\boldsymbol{\theta}}$ (**LR**) or $g_{\boldsymbol{\nu}}, f_{\boldsymbol{\theta}}$ (**PD**) and $L_{\boldsymbol{\theta}}$ twice differentiable, with ∇ and $\mathbb{E}$ interchangeable
- 2 **Converged:** training reached $\mathbf{F}(\boldsymbol{\nu}^*, \boldsymbol{\theta}) = 0$
- 3 **Regular:** $\nabla_{\boldsymbol{\nu}} \mathbf{F}(\boldsymbol{\nu}^*, \boldsymbol{\theta})$ is invertible



Implicit differentiation of policy training

A trained policy is a stationary point of the in-sim objective

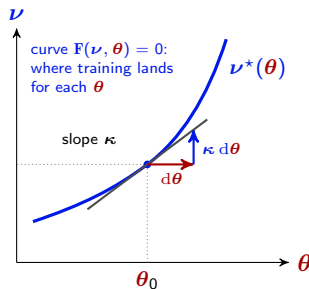
$$\mathbf{F}(\boldsymbol{\nu}^*, \boldsymbol{\theta}) := \nabla_{\boldsymbol{\nu}} \hat{J}_{\boldsymbol{\theta}}(\boldsymbol{\nu}^*) = 0$$

LR: $\mathbb{E}[\sum_k \gamma^k \nabla_{\boldsymbol{\nu}} \log \pi_{\boldsymbol{\nu}} Q_{\boldsymbol{\theta}}^{\pi_{\boldsymbol{\nu}}}] = 0$ (PPO)

PD: $\mathbb{E}_{\omega}[\sum_k \gamma^k (\nabla_{\mathbf{s}} L_{\boldsymbol{\theta}} \frac{d\mathbf{s}_k}{d\boldsymbol{\nu}} + \nabla_{\mathbf{a}} L_{\boldsymbol{\theta}} \frac{d\mathbf{a}_k}{d\boldsymbol{\nu}})] = 0$ (SVG)

Assumptions

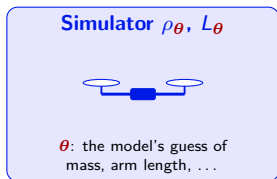
- 1 **Smooth:** $\mathbf{F}$ is continuously differentiable in $(\boldsymbol{\nu}, \boldsymbol{\theta})$, e.g. $\pi_{\boldsymbol{\nu}}, \rho_{\boldsymbol{\theta}}$ (**LR**) or $g_{\boldsymbol{\nu}}, f_{\boldsymbol{\theta}}$ (**PD**) and $L_{\boldsymbol{\theta}}$ twice differentiable, with ∇ and $\mathbb{E}$ interchangeable
- 2 **Converged:** training reached $\mathbf{F}(\boldsymbol{\nu}^*, \boldsymbol{\theta}) = 0$
- 3 **Regular:** $\nabla_{\boldsymbol{\nu}} \mathbf{F}(\boldsymbol{\nu}^*, \boldsymbol{\theta})$ is invertible



IFT: differentiate $\mathbf{F}(\boldsymbol{\nu}^*(\boldsymbol{\theta}), \boldsymbol{\theta}) \equiv 0 \Rightarrow \nabla_{\boldsymbol{\nu}} \mathbf{F} \boldsymbol{\kappa} + \nabla_{\boldsymbol{\theta}} \mathbf{F} = 0 \Rightarrow \boxed{\boldsymbol{\kappa} = -(\nabla_{\boldsymbol{\nu}} \mathbf{F})^{-1} \nabla_{\boldsymbol{\theta}} \mathbf{F}}$

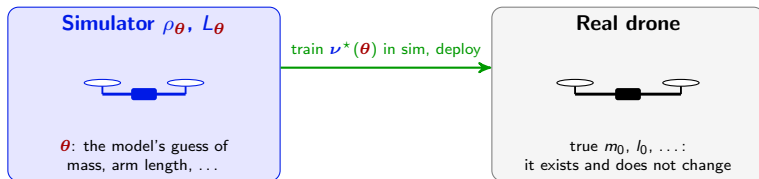
Where do we use? Sim-to-real gap: tune the simulator for performance

Motivation: a policy is trained in a simulator, then deployed on a real system.



Where do we use? Sim-to-real gap: tune the simulator for performance

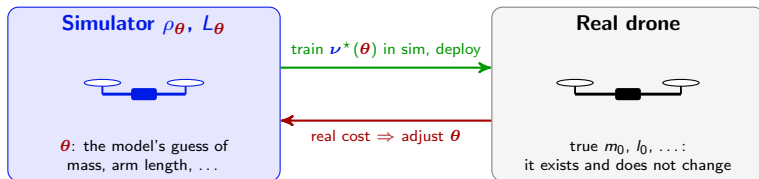
Motivation: a policy is trained in a simulator, then deployed on a real system.



- Changing θ changes the **simulator**, our model of the real world

Where do we use? Sim-to-real gap: tune the simulator for performance

Motivation: a policy is trained in a simulator, then deployed on a real system.

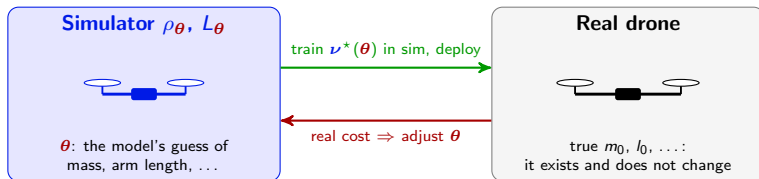


- Changing θ changes the **simulator**, our model of the real world
- Goal: policy trained in the simulator optimal **in the real robot**

$$\min_{\theta} J_{\text{real}}(\nu^*(\theta)), \quad \nu^*(\theta) \text{ trained in the simulator with parameters } \theta$$

Where do we use? Sim-to-real gap: tune the simulator for performance

Motivation: a policy is trained in a simulator, then deployed on a real system.



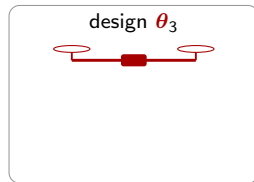
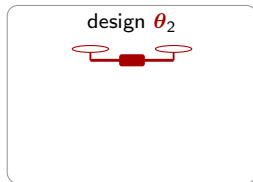
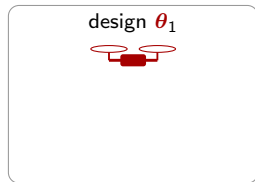
- Changing θ changes the **simulator**, our model of the real world
- Goal: policy trained in the simulator optimal **in the real robot**

$$\min_{\theta} J_{\text{real}}(\nu^*(\theta)), \quad \nu^*(\theta) \text{ trained in the simulator with parameters } \theta$$

- Not system identification: the best θ for **decisions** need not be the most accurate model

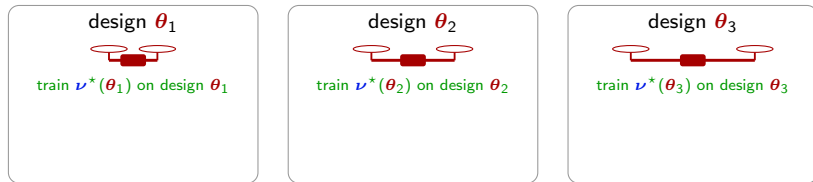
Where do we use this? Co-design: design the system and its policy

The system is **to be designed**: θ as the **design space** (structure, motor & battery size, ...)



Where do we use this? Co-design: design the system and its policy

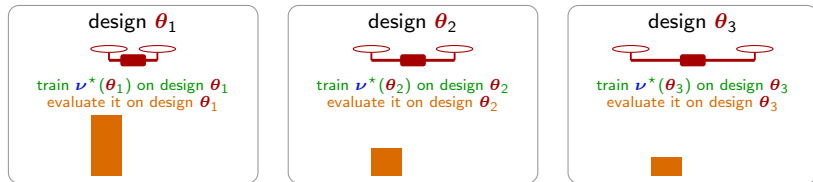
The system is **to be designed**: θ as the **design space** (structure, motor & battery size, ...)



- Changing θ changes the **real system** itself, hence its MDP: the policy must be **re-trained**, $\nu^*(\theta)$

Where do we use this? Co-design: design the system and its policy

The system is **to be designed**: θ as the **design space** (structure, motor & battery size, ...)

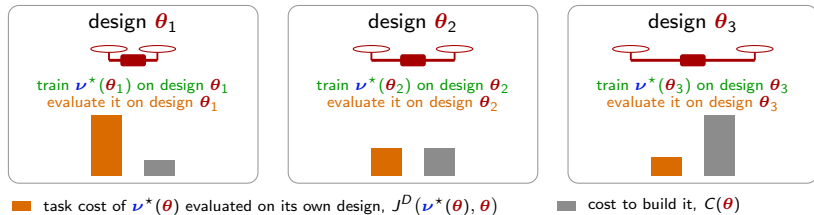


■ task cost of $\nu^*(\theta)$ evaluated on its own design, $J^D(\nu^*(\theta), \theta)$

- Changing θ changes the **real system** itself, hence its MDP: the policy must be **re-trained**, $\nu^*(\theta)$
- Each trained policy is **evaluated on its own design**: the system it is judged on changes with θ ...

Where do we use this? Co-design: design the system and its policy

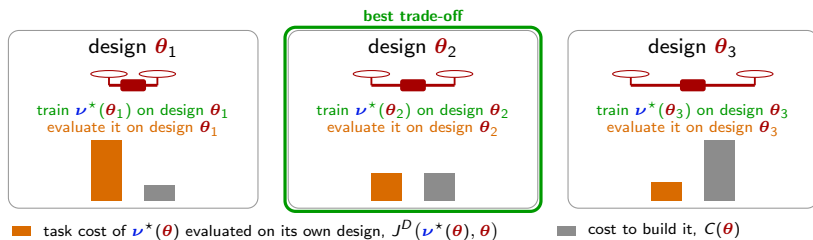
The system is **to be designed**: θ as the **design space** (structure, motor & battery size, ...)



- Changing θ changes the **real system** itself, hence its MDP: the policy must be **re-trained**, $\nu^*(\theta)$
- Each trained policy is **evaluated on its own design**: the system it is judged on changes with θ ...
- ... plus what it costs to build: $\min_{\theta} J^D(\nu^*(\theta), \theta) + C(\theta)$

Where do we use this? Co-design: design the system and its policy

The system is **to be designed**: θ as the **design space** (structure, motor & battery size, ...)



- Changing θ changes the **real system** itself, hence its MDP: the policy must be **re-trained**, $\nu^*(\theta)$
- Each trained policy is **evaluated on its own design**: the system it is judged on changes with θ ...
- ... plus what it costs to build: $\min_{\theta} J^D(\nu^*(\theta), \theta) + C(\theta)$

Sim-to-real: θ changes only the **model**.

Co-design: θ changes the **robot itself**.

Using κ : the gradient of the outer objective

In both problems we tune θ with the gradient of an **outer objective** $J^{\text{out}}(\nu^*(\theta), \theta)$ that **judges** the trained policy. The blocker was $\nabla_{\theta} \pi_{\theta} = \nabla_{\nu} \pi_{\nu} \kappa$: now computable.

$$\nabla_{\theta} J^{\text{out}}(\nu^*(\theta), \theta) = \underbrace{\partial_{\theta} J^{\text{out}}}_{\text{direct: } \nu \text{ fixed}} + \underbrace{\nabla_{\nu} J^{\text{out}} \kappa}_{\text{indirect: the policy re-adapts}}, \quad \kappa = -(\nabla_{\nu} \mathbf{F})^{-1} \nabla_{\theta} \mathbf{F}$$

	Sim-to-real	Co-design

Using κ : the gradient of the outer objective

In both problems we tune θ with the gradient of an **outer objective** $J^{\text{out}}(\nu^*(\theta), \theta)$ that **judges** the trained policy. The blocker was $\nabla_{\theta} \pi_{\theta} = \nabla_{\nu} \pi_{\nu} \kappa$: now computable.

$$\nabla_{\theta} J^{\text{out}}(\nu^*(\theta), \theta) = \underbrace{\partial_{\theta} J^{\text{out}}}_{\text{direct: } \nu \text{ fixed}} + \underbrace{\nabla_{\nu} J^{\text{out}} \kappa}_{\text{indirect: the policy re-adapts}}, \quad \kappa = -(\nabla_{\nu} \mathbf{F})^{-1} \nabla_{\theta} \mathbf{F}$$

	Sim-to-real	Co-design
θ	simulator parameters	design of the system

Using κ : the gradient of the outer objective

In both problems we tune θ with the gradient of an **outer objective** $J^{\text{out}}(\nu^*(\theta), \theta)$ that **judges** the trained policy. The blocker was $\nabla_{\theta} \pi_{\theta} = \nabla_{\nu} \pi_{\nu} \kappa$: now computable.

$$\nabla_{\theta} J^{\text{out}}(\nu^*(\theta), \theta) = \underbrace{\partial_{\theta} J^{\text{out}}}_{\text{direct: } \nu \text{ fixed}} + \underbrace{\nabla_{\nu} J^{\text{out}} \kappa}_{\text{indirect: the policy re-adapts}}, \quad \kappa = -(\nabla_{\nu} F)^{-1} \nabla_{\theta} F$$

	Sim-to-real	Co-design
θ	simulator parameters	design of the system
J^{out}	$J_{\text{real}}(\nu^*)$, on the real robot	$J^D(\nu^*, \theta) + C(\theta)$, in the simulator

Using κ : the gradient of the outer objective

In both problems we tune θ with the gradient of an **outer objective** $J^{\text{out}}(\nu^*(\theta), \theta)$ that **judges** the trained policy. The blocker was $\nabla_{\theta} \pi_{\theta} = \nabla_{\nu} \pi_{\nu} \kappa$: now computable.

$$\nabla_{\theta} J^{\text{out}}(\nu^*(\theta), \theta) = \underbrace{\partial_{\theta} J^{\text{out}}}_{\text{direct: } \nu \text{ fixed}} + \underbrace{\nabla_{\nu} J^{\text{out}} \kappa}_{\text{indirect: the policy re-adapts}}, \quad \kappa = -(\nabla_{\nu} F)^{-1} \nabla_{\theta} F$$

	Sim-to-real	Co-design
θ	simulator parameters	design of the system
J^{out}	$J_{\text{real}}(\nu^*)$, on the real robot	$J^D(\nu^*, \theta) + C(\theta)$, in the simulator
policy judged on	the fixed real robot	its own design θ (changes with θ)

Using κ : the gradient of the outer objective

In both problems we tune θ with the gradient of an **outer objective** $J^{\text{out}}(\nu^*(\theta), \theta)$ that **judges** the trained policy. The blocker was $\nabla_{\theta} \pi_{\theta} = \nabla_{\nu} \pi_{\nu} \kappa$: now computable.

$$\nabla_{\theta} J^{\text{out}}(\nu^*(\theta), \theta) = \underbrace{\partial_{\theta} J^{\text{out}}}_{\text{direct: } \nu \text{ fixed}} + \underbrace{\nabla_{\nu} J^{\text{out}} \kappa}_{\text{indirect: the policy re-adapts}}, \quad \kappa = -(\nabla_{\nu} F)^{-1} \nabla_{\theta} F$$

	Sim-to-real	Co-design
θ	simulator parameters	design of the system
J^{out}	$J_{\text{real}}(\nu^*)$, on the real robot	$J^D(\nu^*, \theta) + C(\theta)$, in the simulator
policy judged on	the fixed real robot	its own design θ (changes with θ)
direct term	0 (the real robot ignores θ)	$\neq 0$ (θ changes the system)

Using κ : the gradient of the outer objective

In both problems we tune θ with the gradient of an **outer objective** $J^{\text{out}}(\nu^*(\theta), \theta)$ that **judges** the trained policy. The blocker was $\nabla_{\theta} \pi_{\theta} = \nabla_{\nu} \pi_{\nu} \kappa$: now computable.

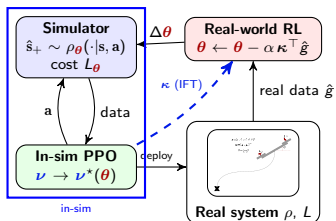
$$\nabla_{\theta} J^{\text{out}}(\nu^*(\theta), \theta) = \underbrace{\partial_{\theta} J^{\text{out}}}_{\text{direct: } \nu \text{ fixed}} + \underbrace{\nabla_{\nu} J^{\text{out}} \kappa}_{\text{indirect: the policy re-adapts}}, \quad \kappa = -(\nabla_{\nu} F)^{-1} \nabla_{\theta} F$$

	Sim-to-real	Co-design
θ	simulator parameters	design of the system
J^{out}	$J_{\text{real}}(\nu^*)$, on the real robot	$J^D(\nu^*, \theta) + C(\theta)$, in the simulator
policy judged on	the fixed real robot	its own design θ (changes with θ)
direct term	0 (the real robot ignores θ)	$\neq 0$ (θ changes the system)
Tested methods	LR, from real rollouts	PD (AD), in the simulator

Outline

- 1 Decisions from data
- 2 AI models for Decision
- 3 Differentiating Policy Training
- 4 Examples

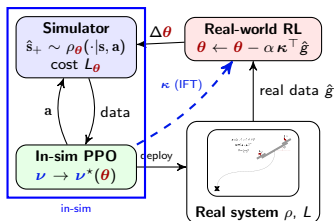
Example 1: Sim-to-real with bi-level PPO



Recall: tune θ so that the policy trained in sim is best on the robot

$$\min_{\theta} J_{\text{real}}(\nu^*(\theta)) \quad \text{s.t.} \quad \nabla_{\nu} \hat{J}_{\theta}(\nu^*) = 0$$

Example 1: Sim-to-real with bi-level PPO

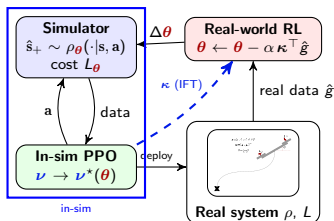


Recall: tune θ so that the policy trained in sim is best on the robot

$$\min_{\theta} J_{\text{real}}(\nu^*(\theta)) \quad \text{s.t.} \quad \nabla_{\nu} \hat{J}_{\theta}(\nu^*) = 0$$

- **In sim:** PPO trains $\nu^*(\theta)$ (warm start)

Example 1: Sim-to-real with bi-level PPO

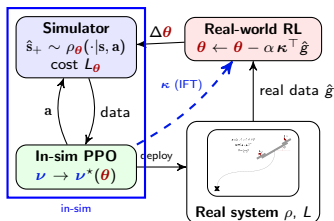


Recall: tune θ so that the policy trained in sim is best on the robot

$$\min_{\theta} J_{\text{real}}(\nu^*(\theta)) \quad \text{s.t.} \quad \nabla_{\nu} \hat{J}_{\theta}(\nu^*) = 0$$

- **In sim:** PPO trains $\nu^*(\theta)$ (warm start)
- **On the robot:** LR gradient $\hat{g}$,
 $\theta \leftarrow \theta - \alpha \kappa^\top \hat{g}$ (PPO)

Example 1: Sim-to-real with bi-level PPO

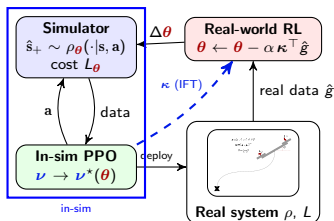


Recall: tune θ so that the policy trained in sim is best on the robot

$$\min_{\theta} J_{\text{real}}(\nu^*(\theta)) \quad \text{s.t.} \quad \nabla_{\nu} \hat{J}_{\theta}(\nu^*) = 0$$

- **In sim:** PPO trains $\nu^*(\theta)$ (warm start)
- **On the robot:** LR gradient $\hat{g}$, $\theta \leftarrow \theta - \alpha \kappa^\top \hat{g}$ (PPO)
- **Converges** a.s. (critics fast, policy medium, θ slow)

Example 1: Sim-to-real with bi-level PPO

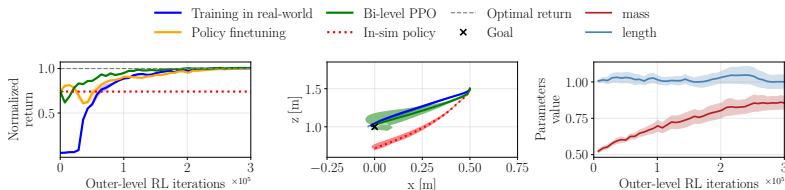


Recall: tune θ so that the policy trained in sim is best on the robot

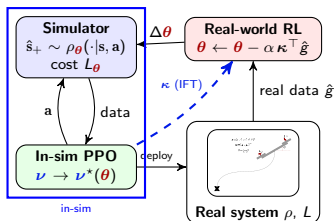
$$\min_{\theta} J_{\text{real}}(\nu^*(\theta)) \quad \text{s.t.} \quad \nabla_{\nu} \hat{J}_{\theta}(\nu^*) = 0$$

- **In sim:** PPO trains $\nu^*(\theta)$ (warm start)
- **On the robot:** LR gradient $\hat{g}$, $\theta \leftarrow \theta - \alpha \kappa^\top \hat{g}$ (PPO)
- **Converges** a.s. (critics fast, policy medium, θ slow)

2D quadrotor (safe-control-gym): the simulator starts with wrong mass and arm length



Example 1: Sim-to-real with bi-level PPO

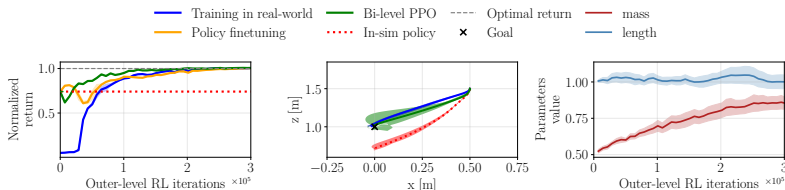


Recall: tune θ so that the policy trained in sim is best on the robot

$$\min_{\theta} J_{\text{real}}(\nu^*(\theta)) \quad \text{s.t.} \quad \nabla_{\nu} \hat{J}_{\theta}(\nu^*) = 0$$

- **In sim:** PPO trains $\nu^*(\theta)$ (warm start)
- **On the robot:** LR gradient $\hat{g}$,
 $\theta \leftarrow \theta - \alpha \kappa^{\top} \hat{g}$ (PPO)
- **Converges** a.s. (critics fast, policy medium, θ slow)

2D quadrotor (safe-control-gym): the simulator starts with wrong mass and arm length



the in-sim policy (red dotted) misses the goal; bi-level PPO closes the gap, with mass $\rightarrow \approx 0.85 m_0$: **a wrong model, but the optimal policy**

Example 2: Co-design of a system and its policy

Each design θ needs its own trained policy

Bi-level problem

$$\begin{aligned} \min_{\theta} \quad & J^D(\nu^*(\theta), \theta) + C(\theta) \\ \text{s.t.} \quad & \nabla_{\nu} J^{\pi}(\nu, \theta) = 0 \end{aligned}$$

Example 2: Co-design of a system and its policy

Each design θ needs its own trained policy

Bi-level problem

$$\min_{\theta} J^D(\nu^*(\theta), \theta) + C(\theta)$$

$$\text{s.t. } \nabla_{\nu} J^{\pi}(\nu, \theta) = 0$$

- J^{π} trains the policy; J^D **judges** the design; $C(\theta)$: cost to build it

Example 2: Co-design of a system and its policy

Each design θ needs its own trained policy

Bi-level problem

$$\min_{\theta} J^D(\nu^*(\theta), \theta) + C(\theta)$$

$$\text{s.t. } \nabla_{\nu} J^{\pi}(\nu, \theta) = 0$$

- J^{π} trains the policy; J^D **judges** the design; $C(\theta)$: cost to build it
- usual practice: BO or ES around full RL runs, scales poorly with $\dim \theta$

Example 2: Co-design of a system and its policy

Each design θ needs its own trained policy

Bi-level problem

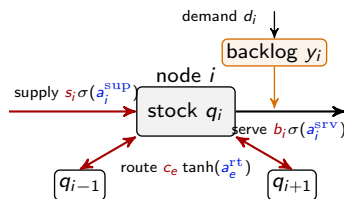
$$\min_{\theta} J^D(\nu^*(\theta), \theta) + C(\theta)$$

$$\text{s.t. } \nabla_{\nu} J^{\pi}(\nu, \theta) = 0$$

- J^{π} trains the policy; J^D **judges** the design; $C(\theta)$: cost to build it
- usual practice: BO or ES around full RL runs, scales poorly with $\dim \theta$

Example: inventory–routing network

θ = capacities, $\dim \theta$ from 15 to 150



the design bounds the **actions**;

$$J^D = J^{\pi} + \lambda_y \cdot \text{backlog}$$

Example 2: Co-design of a system and its policy

Each design θ needs its own trained policy

Bi-level problem

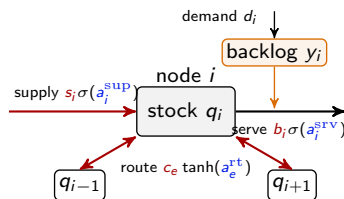
$$\min_{\theta} J^D(\nu^*(\theta), \theta) + C(\theta)$$

$$\text{s.t. } \nabla_{\nu} J^{\pi}(\nu, \theta) = 0$$

- J^{π} trains the policy; J^D **judges** the design; $C(\theta)$: cost to build it
- usual practice: BO or ES around full RL runs, scales poorly with $\dim \theta$

Example: inventory–routing network

θ = capacities, $\dim \theta$ from 15 to 150



the design bounds the **actions**;

$$J^D = J^{\pi} + \lambda_y \cdot \text{backlog}$$

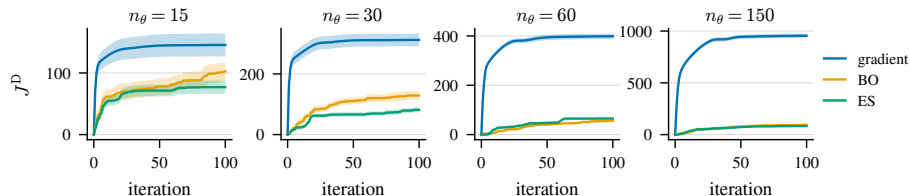
Algorithm: Gradient-based co-design (differentiable simulator: **PD** everywhere)

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

- a few warm-started SVG steps on $J^{\pi}(\cdot, \theta_k)$: they only have to **track** $\nu^*(\theta_k)$;
 - $g \leftarrow \partial_{\theta} J^D + \nabla_{\nu} J^D \kappa + \nabla_{\theta} C$ (jax.grad and the adjoint trick, slide 27);
 - $\theta_{k+1} \leftarrow \Pi_{\Theta}(\theta_k - \alpha g)$; // projection onto the bounds
-

Co-design: inventory-routing network

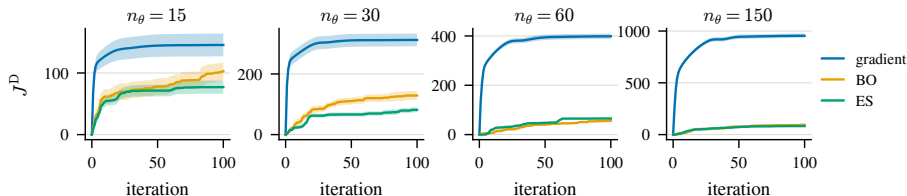
Against black-box search: 100 design evaluations each, matched objectives ($\lambda_y = 0$, so the κ -term is 0 and only the direct gradient is used)



Plots show the design return $-(J^D + C)$: higher is better.

Co-design: inventory–routing network

Against black-box search: 100 design evaluations each, matched objectives ($\lambda_y = 0$, so the κ -term is 0 and only the direct gradient is used)

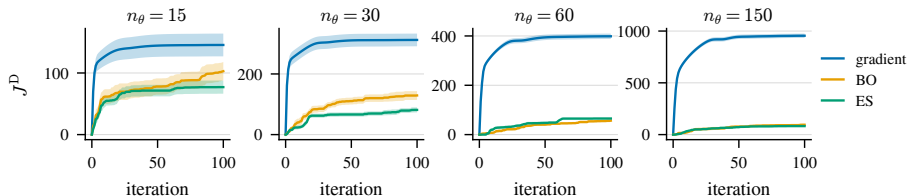


- the gradient approach outperforms BO and ES at every size, and the gap grows with $\dim \theta$ (15 to 150)

Plots show the design return $-(J^D + C)$: higher is better.

Co-design: inventory–routing network

Against black-box search: 100 design evaluations each, matched objectives ($\lambda_y = 0$, so the κ -term is 0 and only the direct gradient is used)

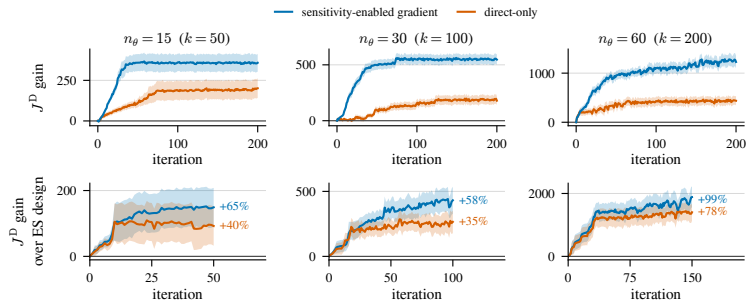


- the gradient approach outperforms BO and ES at every size, and the gap grows with $\dim \theta$ (15 to 150)
- cost per design evaluation: ≈ 2 s for the gradient, ≈ 2 s ES, ≈ 3 s BO

Plots show the design return $-(J^D + C)$: higher is better.

Co-design: inventory–routing network

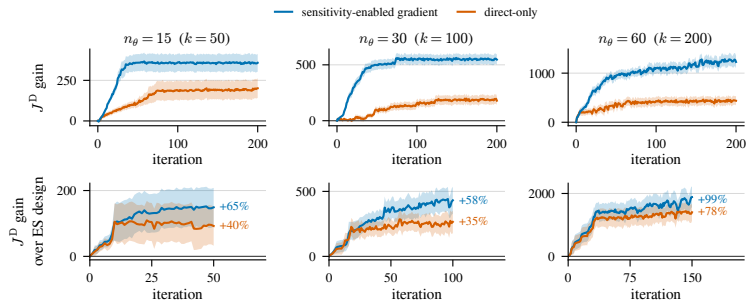
Effect of the κ -term: mismatched objectives ($\lambda_y = 0.1$, $J^D \neq J^\pi$)



Plots show the design *return* $-(J^D + C)$: higher is better.

Co-design: inventory-routing network

Effect of the κ -term: mismatched objectives ($\lambda_y = 0.1$, $J^D \neq J^\pi$)

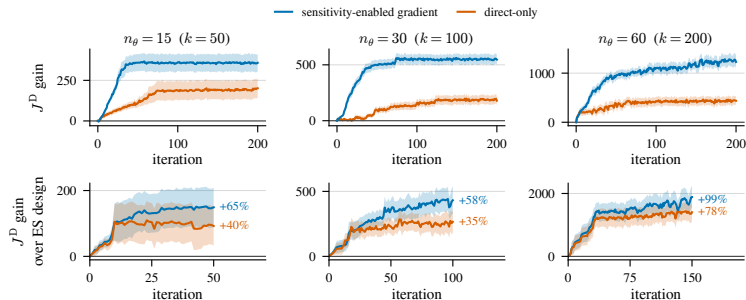


- **top:** sensitivity-enabled vs direct-only gradient, from the same initial design: the κ -term helps, more in higher dimension

Plots show the design *return* $-(J^D + C)$: higher is better.

Co-design: inventory-routing network

Effect of the κ -term: mismatched objectives ($\lambda_y = 0.1$, $J^D \neq J^\pi$)

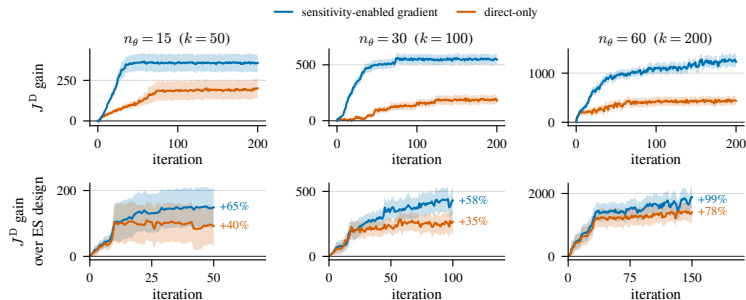


- **top:** sensitivity-enabled vs direct-only gradient, from the same initial design: the κ -term helps, more in higher dimension
- **bottom:** starting from the best design of a converged ES search: the gradient still improves it

Plots show the design *return* $-(J^D + C)$: higher is better.

Co-design: inventory-routing network

Effect of the κ -term: mismatched objectives ($\lambda_y = 0.1$, $J^D \neq J^\pi$)



- **top:** sensitivity-enabled vs direct-only gradient, from the same initial design: the κ -term helps, more in higher dimension
- **bottom:** starting from the best design of a converged ES search: the gradient still improves it
- **cost:** ≈ 6.5 s per design evaluation with a matrix-free κ -solve (vs ≈ 2 s direct-only)

Plots show the design return $-(J^D + C)$: higher is better.

Take-aways

- 1 **Models for decisions:** a policy optimal for an approximate model ρ_{θ} is in general not optimal in reality, and standard model fitting does not fix this

Take-aways

- 1 **Models for decisions:** a policy optimal for an approximate model ρ_{θ} is in general not optimal in reality, and standard model fitting does not fix this
- 2 **MDP equivalence:** some (non-unique) models and costs give the real optimal policy even if ρ_{θ} is inaccurate $\Rightarrow$ tune θ for **closed-loop performance**

Take-aways

- 1 **Models for decisions:** a policy optimal for an approximate model ρ_{θ} is in general not optimal in reality, and standard model fitting does not fix this
- 2 **MDP equivalence:** some (non-unique) models and costs give the real optimal policy even if ρ_{θ} is inaccurate $\Rightarrow$ tune θ for **closed-loop performance**
- 3 **The obstacle:** this needs $\kappa = d\nu^*/d\theta$, the sensitivity of the trained policy

Take-aways

- 1 **Models for decisions:** a policy optimal for an approximate model ρ_{θ} is in general not optimal in reality, and standard model fitting does not fix this
- 2 **MDP equivalence:** some (non-unique) models and costs give the real optimal policy even if ρ_{θ} is inaccurate $\Rightarrow$ tune θ for **closed-loop performance**
- 3 **The obstacle:** this needs $\kappa = d\nu^*/d\theta$, the sensitivity of the trained policy
- 4 **LR re-weights** paths (a surrogate loss, black-box simulator), **PD moves** them (backpropagation through the simulator); both are a forward and a backward pass, neither always has lower variance

Take-aways

- 1 **Models for decisions:** a policy optimal for an approximate model ρ_{θ} is in general not optimal in reality, and standard model fitting does not fix this
- 2 **MDP equivalence:** some (non-unique) models and costs give the real optimal policy even if ρ_{θ} is inaccurate $\Rightarrow$ tune θ for **closed-loop performance**
- 3 **The obstacle:** this needs $\kappa = d\nu^*/d\theta$, the sensitivity of the trained policy
- 4 **LR re-weights** paths (a surrogate loss, black-box simulator), **PD moves** them (backpropagation through the simulator); both are a forward and a backward pass, neither always has lower variance
- 5 **RL policy training can be differentiated:** under smoothness, convergence and regularity assumptions, the trained policy is a stationary point and the IFT gives a meaningful gradient, $\kappa = -(\nabla_{\nu}\mathbf{F})^{-1}\nabla_{\theta}\mathbf{F}$, from the end point of training alone

Take-aways

- 1 **Models for decisions:** a policy optimal for an approximate model ρ_{θ} is in general not optimal in reality, and standard model fitting does not fix this
- 2 **MDP equivalence:** some (non-unique) models and costs give the real optimal policy even if ρ_{θ} is inaccurate $\Rightarrow$ tune θ for **closed-loop performance**
- 3 **The obstacle:** this needs $\kappa = d\nu^*/d\theta$, the sensitivity of the trained policy
- 4 **LR re-weights** paths (a surrogate loss, black-box simulator), **PD moves** them (backpropagation through the simulator); both are a forward and a backward pass, neither always has lower variance
- 5 **RL policy training can be differentiated:** under smoothness, convergence and regularity assumptions, the trained policy is a stationary point and the IFT gives a meaningful gradient, $\kappa = -(\nabla_{\nu} \mathbf{F})^{-1} \nabla_{\theta} \mathbf{F}$, from the end point of training alone
- 6 **Outer gradient** = direct + $\nabla_{\nu} J^{\text{out}} \kappa$: tune a simulator for reality (sim-to-real) or a system with its policy (co-design)

Take-aways

- 1 **Models for decisions:** a policy optimal for an approximate model ρ_{θ} is in general not optimal in reality, and standard model fitting does not fix this
- 2 **MDP equivalence:** some (non-unique) models and costs give the real optimal policy even if ρ_{θ} is inaccurate $\Rightarrow$ tune θ for **closed-loop performance**
- 3 **The obstacle:** this needs $\kappa = d\nu^*/d\theta$, the sensitivity of the trained policy
- 4 **LR re-weights** paths (a surrogate loss, black-box simulator), **PD moves** them (backpropagation through the simulator); both are a forward and a backward pass, neither always has lower variance
- 5 **RL policy training can be differentiated:** under smoothness, convergence and regularity assumptions, the trained policy is a stationary point and the IFT gives a meaningful gradient, $\kappa = -(\nabla_{\nu} \mathbf{F})^{-1} \nabla_{\theta} \mathbf{F}$, from the end point of training alone
- 6 **Outer gradient** = direct + $\nabla_{\nu} J^{\text{out}} \kappa$: tune a simulator for reality (sim-to-real) or a system with its policy (co-design)

Proof of concept established on sim-to-real and co-design: a viable pathway towards decision-oriented learning of MDPs.

Take-aways

- 1 **Models for decisions:** a policy optimal for an approximate model ρ_{θ} is in general not optimal in reality, and standard model fitting does not fix this
- 2 **MDP equivalence:** some (non-unique) models and costs give the real optimal policy even if ρ_{θ} is inaccurate $\Rightarrow$ tune θ for **closed-loop performance**
- 3 **The obstacle:** this needs $\kappa = d\nu^*/d\theta$, the sensitivity of the trained policy
- 4 **LR re-weights** paths (a surrogate loss, black-box simulator), **PD moves** them (backpropagation through the simulator); both are a forward and a backward pass, neither always has lower variance
- 5 **RL policy training can be differentiated:** under smoothness, convergence and regularity assumptions, the trained policy is a stationary point and the IFT gives a meaningful gradient, $\kappa = -(\nabla_{\nu}\mathbf{F})^{-1}\nabla_{\theta}\mathbf{F}$, from the end point of training alone
- 6 **Outer gradient** = direct + $\nabla_{\nu}J^{\text{out}}\kappa$: tune a simulator for reality (sim-to-real) or a system with its policy (co-design)

Proof of concept established on sim-to-real and co-design: a viable pathway towards decision-oriented learning of MDPs.

Open questions: scaling the critic sensitivities, conditioning of $\nabla_{\nu}\mathbf{F}$ for deep networks, offline real-world updates, latent-space model learning under partial observability

Thank you!

Questions and discussion

Papers

- *Predicting What Matters: Training AI Models for Better Decisions*, IEEE TNNLS, 2025
Akhil S. Anand, Shambhuraj Sawant, Dirk Peter Reinhardt, Sebastien Gros
- *Bi-level RL pathway for sim-to-real optimality*, RLC 2026
Akhil S. Anand, Shambhuraj Sawant, Paavo Parmas, Jasper Hoffmann, Dirk Peter Reinhardt, Sebastien Gros
- *Gradient-based co-design of RL policies with differentiable simulators*, TMLR 2026
Akhil S. Anand, Xun Shen, Marc Peter Deisenroth, Sebastien Gros