Differentiable optimization for control and reinforcement learning

Brandon Amos
Meta AI NYC, Fundamental AI Research (FAIR)

brandondamos   bamos.github.io

Slides available at:
github.com/bamos/presentations

Collaborators: Akshay Agrawal, Shane Barratt, Byron Boots, Stephen Boyd, Roberto Calandra, Steven Diamond, Priya Donti, Ivan Jimenez, Zico Kolter, Nathan Lambert, Jacob Sacks, Samuel Stanton, Andrew Gordon Wilson, Omry Yadan, and Denis Yarats

Control is powerful*

*when properly set up

Setting: deterministic, discrete-time system with a **continuous state-action space**

$$x^\star_{1:T}, u^\star_{1:T} \in \operatorname*{argmin}_{x_{1:T}, u_{1:T}} \sum_t \overset{\text{cost}}{\boxed{C_\theta(x_t, u_t)}} \text{ s.t. } \overset{\text{initial state}}{\boxed{x_1 = x_{\text{init}}}} \; \overset{\text{dynamics}}{\boxed{x_{t+1} = f_\theta(x_t, u_t)}} \; \overset{\text{constraints}}{\boxed{u_t \in \mathcal{U}}}$$

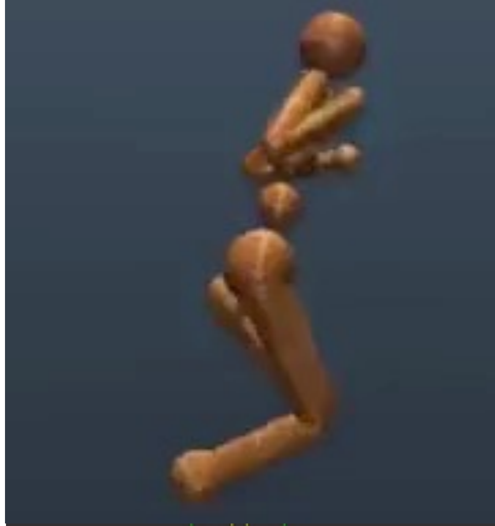
Full notation: $u^\star_{1:T}(x_{\text{init}}, \theta)$

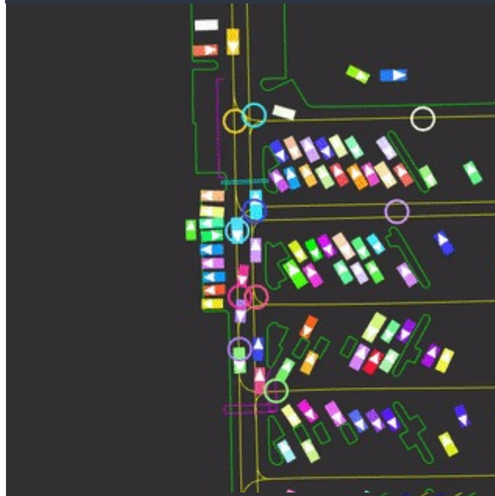
Widely deployed over the past century for aviation, robotics, autonomous driving, HVAC
Often for a **Markov decision process** but doesn't have to be

The **real-world is non-convex**, so are our controllers
Convex in some cases and subproblems, e.g., with quadratic cost/linear dynamics (LQR)

NO LEARNING NECESSARY if we know the system — just pure optimization

Notation: θ are the **parameters** of the controller (usually of the cost or dynamics)





Model-free RL and control

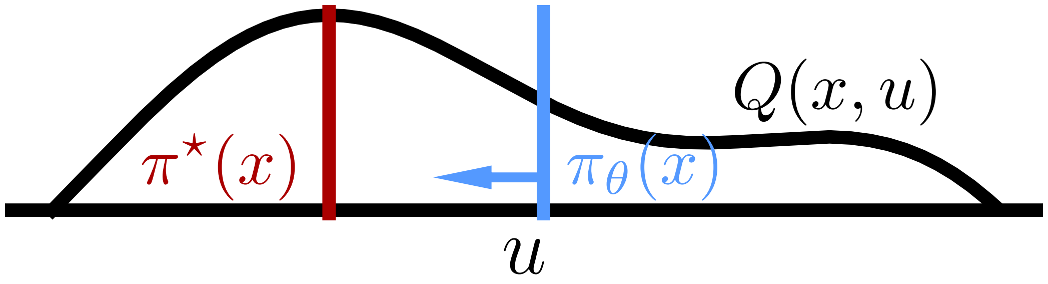
Take the **cost** to be the (negated) **value estimate**, **no dynamics**
Value estimate approximates the **model-based objective**
Policy learning performs **amortized optimization**

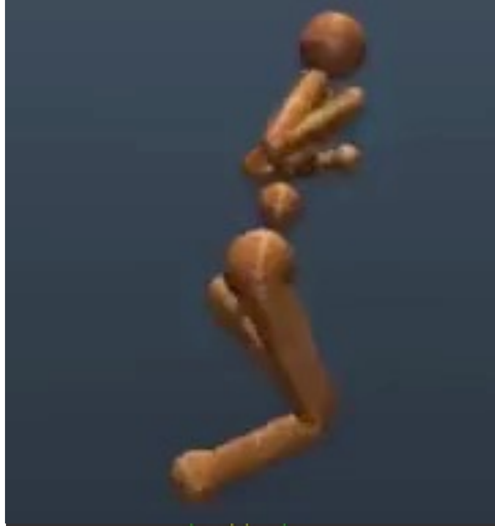
Viewpoint leads to a **model-based to model-free spectrum:**
take **short-horizon model-based rollouts** with a **value estimate at the end**

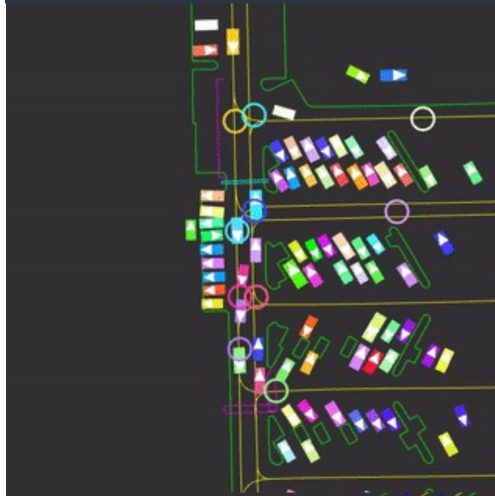
$$x^\star_{1:T}, u^\star_{1:T} \in \operatorname*{argmin}_{x_{1:T}, u_{1:T}} \sum_t \underbrace{\boxed{C_\theta(x_t, u_t)}}_{\text{cost}} \;\text{s.t.}\; \underbrace{\boxed{x_1 = x_{\text{init}}}}_{\text{initial state}} \; \underbrace{\boxed{x_{t+1} = f_\theta(x_t, u_t)}}_{\text{dynamics}} \; \underbrace{\boxed{u_t \in \mathcal{U}}}_{\text{constraints}}$$

Full notation: $u^\star_{1:T}(x_{\text{init}}, \theta)$

$$\pi^\star(x) \in \operatorname*{argmax}_u Q^\pi_\theta(x, u)$$







Control may fail for many reasons

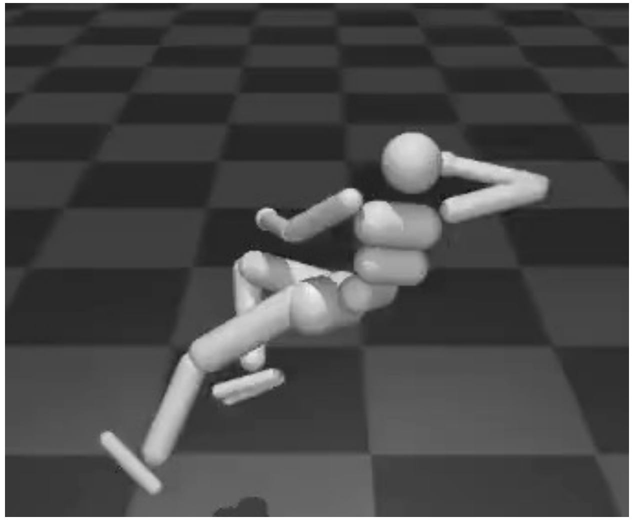
$$x^\star_{1:T}, u^\star_{1:T} \in \underset{x_{1:T}, u_{1:T}}{\operatorname{argmin}} \sum_t \underbrace{\boxed{C_\theta(x_t, u_t)}}_{\text{cost}} \text{ s.t. } \underbrace{\boxed{x_1 = x_{\text{init}}}}_{\text{initial state}} \quad \underbrace{\boxed{x_{t+1} = f_\theta(x_t, u_t)}}_{\text{dynamics}} \quad \underbrace{\boxed{u_t \in \mathcal{U}}}_{\text{constraints}}$$

Full notation: $u^\star_{1:T}(x_{\text{init}}, \theta)$

Control starts failing us when we can't describe everything
Impossible to analytically **encode every detail** of non-trivial systems

Cost and **dynamics** may be **unknown, mis-specified,** or **inaccurate**
Especially difficult in **high-dimensional state-action spaces**

Learning methods help but **are not perfect**
system identification, learning dynamics, inverse cost learning



Controllers don't live in isolation

We can often measure the **downstream performance** induced by the controller

Idea: optimize (i.e., tune/learn) the parameters for a **downstream performance metric**

Controller-design loop is **not** a new idea and has been extensively used over the past century

θ controller **parameters**
(design space of cost and model)

solve control optimization problem

$$x^\star_{1:T}, u^\star_{1:T} \in \underset{x_{1:T}, u_{1:T}}{\operatorname{argmin}} \sum_t \underbrace{C_\theta(x_t, u_t)}_{\text{cost}} \;\text{s.t.}\; \underbrace{x_1 = x_{\text{init}}}_{\text{initial state}} \quad \underbrace{x_{t+1} = f_\theta(x_t, u_t)}_{\text{dynamics}} \quad \underbrace{u_t \in \mathcal{U}}_{\text{constraints}}$$

Full notation: $u^\star_{1:T}(x_{\text{init}}, \theta)$

$\pi_\theta(x) \stackrel{\text{def}}{=} u^\star_1(x, \theta)$ **execute** control policy on the system

$\mathcal{J}(\pi_\theta)$ **measure downstream performance** of the controller

This talk: differentiate the controller!

We can often measure the **downstream performance** induced by the controller
Idea: optimize (i.e., tune/learn) the parameters for a **downstream performance metric**
by **differentiating through the control optimization problem**

θ controller **parameters**
(design space of cost and model)

solve control optimization problem

$$x^\star_{1:T}, u^\star_{1:T} \in \underset{x_{1:T}, u_{1:T}}{\operatorname{argmin}} \sum_t \underbrace{C_\theta(x_t, u_t)}_{\text{cost}} \text{ s.t. } \underbrace{x_1 = x_{\text{init}}}_{\text{initial state}} \quad \underbrace{x_{t+1} = f_\theta(x_t, u_t)}_{\text{dynamics}} \quad \underbrace{u_t \in \mathcal{U}}_{\text{constraints}}$$

Full notation: $u^\star_{1:T}(x_{\text{init}}, \theta)$

$\pi_\theta(x) \overset{\text{def}}{=} u^\star_1(x, \theta)$ **execute** control policy on the system

$\mathcal{J}(\pi_\theta)$ **measure downstream performance** of the controller

differentiable control

$\nabla_\theta \mathcal{J}(\pi_\theta)$

or $\mathrm{D}_\theta(\pi_\theta(x))$

This talk: differentiate the controller!

Foundations of differentiable optimization and control
- Unrolling or autograd (gradient descent, differentiable cross-entropy method)
- Implicit differentiation (convex and non-convex MPC)

`cvxpylayers`: **Prototyping** differentiable convex optimization and control

Applications of differentiable control
- Objective mismatch
- Amortized control

Derivatives in RL and control

The policy (or value) **gradient**

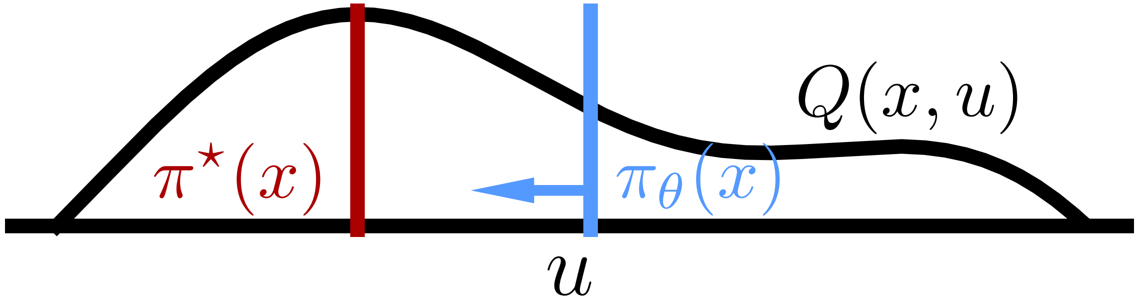
Derivative of **value** w.r.t. a **parameterized policy**:

$$\nabla_\theta \mathbb{E}_{x_t}\left[Q\big(x_t, \pi_\theta(x_t)\big)\right]$$

For policy learning via **amortized optimization**

Q-value can be model-based or model-free

Works for deterministic and stochastic policies

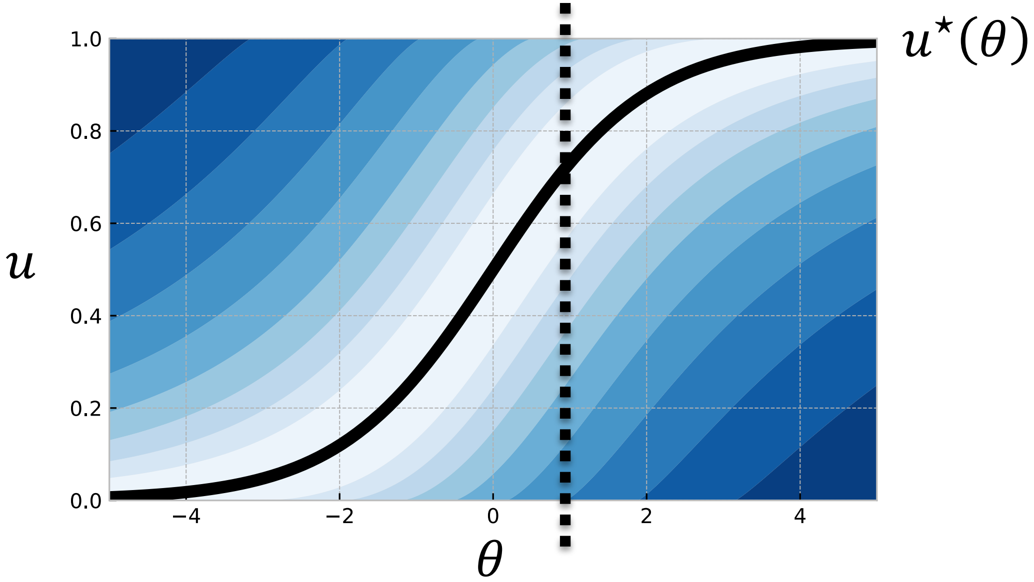


Differentiable control — this talk

Derivative of **actions** w.r.t. **controller parameters**:

$$\partial u^\star_{1:T}(\theta)/\partial\theta$$

Controller induces a **model-based policy**



Each vertical slice is a control problem

How to differentiate the controller?

Unrolling or autograd

$$\hat{u}_\theta^0 \rightarrow \hat{u}_\theta^1 \rightarrow \cdots \rightarrow \hat{u}_\theta^K \rightarrow \hat{\pi}_\theta(x) \rightarrow \mathcal{J}$$

Idea: Implement controller, let **autodiff** do the rest
Like MAML's unrolled gradient descent

Ideal when **unconstrained** with a **short horizon**
Does **not** require a fixed-point or optimal solution
Instable and resource-intensive for large horizons

Can unroll algorithms **beyond gradient descent**
The differentiable cross-entropy method

Implicit differentiation

$$\mathrm{D}_\theta u^\star(\theta) = -\mathrm{D}_u g\big(\theta, u^\star(\theta)\big)^{-1} \mathrm{D}_\theta g\big(\theta, u^\star(\theta)\big)$$

Idea: Differentiate controller's optimality conditions

Agnostic of the control algorithm
Ill-defined if controller gives **suboptimal solution**
Memory and **compute** efficient: free in some cases

How to differentiate the controller?

Unrolling or autograd

$$\hat{u}_\theta^0 \rightarrow \hat{u}_\theta^1 \rightarrow \cdots \rightarrow \hat{u}_\theta^K \rightarrow \hat{\pi}_\theta(x) \rightarrow \mathcal{J}$$

Idea: Implement controller, let **autodiff** do the rest
Like MAML's unrolled gradient descent

Ideal when **unconstrained** with a **short horizon**
Does **not** require a fixed-point or optimal solution
Instable and resource-intensive for large horizons

Can unroll algorithms **beyond gradient descent**
The differentiable cross-entropy method

Implicit differentiation

$$\mathrm{D}_\theta u^\star(\theta) = -\mathrm{D}_u g\big(\theta, u^\star(\theta)\big)^{-1} \mathrm{D}_\theta g\big(\theta, u^\star(\theta)\big)$$

Idea: Differentiate controller's optimality conditions

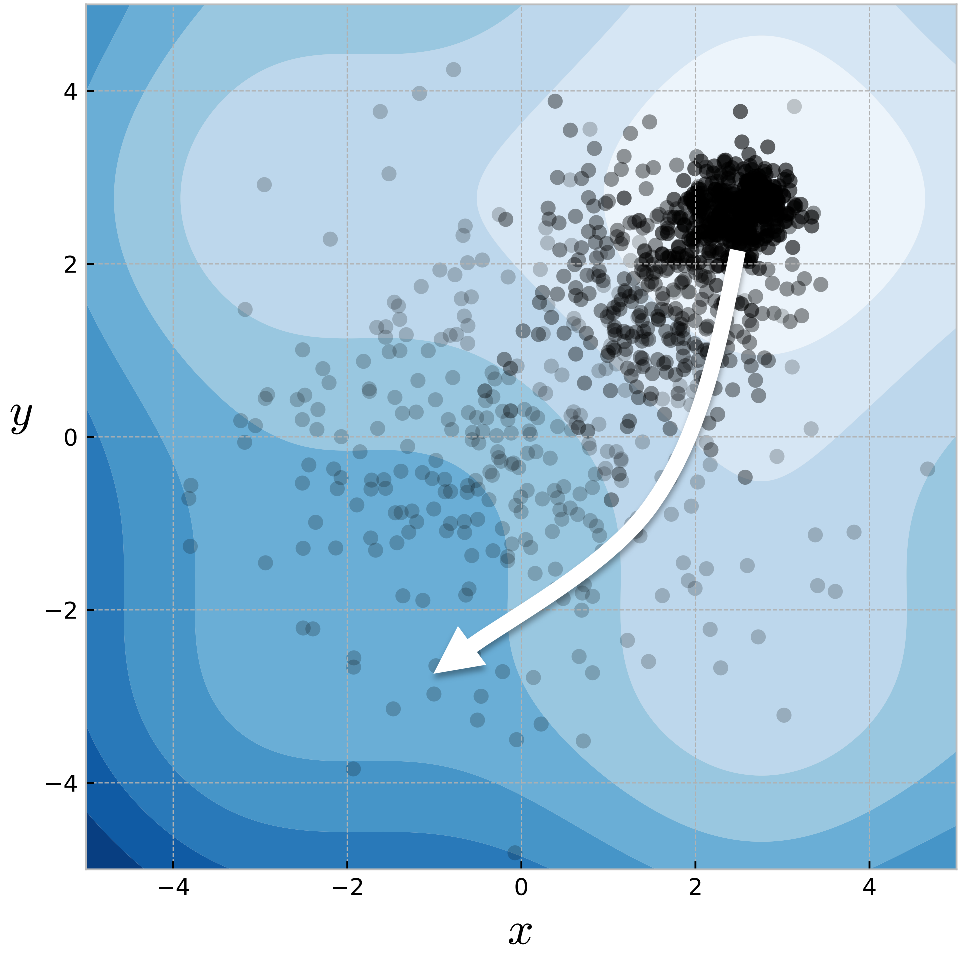
Agnostic of the control algorithm
Ill-defined if controller gives **suboptimal solution**
Memory and **compute** efficient: free in some cases

The Differentiable Cross-Entropy Method (DCEM)

The **cross-entropy method (CEM)** optimizer:
1. **Samples** from the domain with a Gaussian
2. **Updates** the Gaussian with the **top-k values**

Solves challenging **non-convex control** problems

The differentiable cross-entropy method (DCEM):
Use **unrolling** to differentiate through CEM using:
1. the **reparameterization trick** for sampling
2. a **differentiable top-k operation** (LML)



From the softmax to soft/differentiable top-k

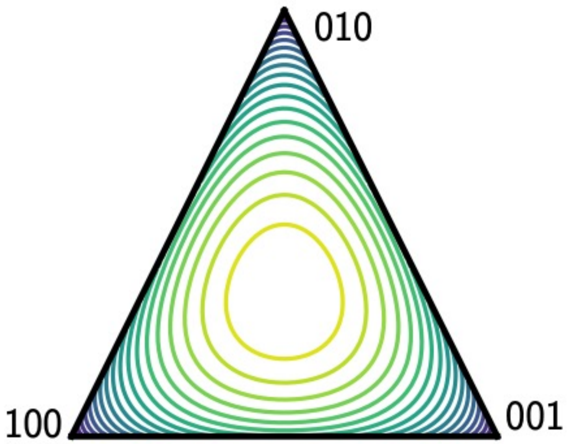
Constrained softmax, constrained sparsemax, Limited Multi-Label Projection

$$\pi_\Delta(x) = \underset{y}{\operatorname{argmin}} \; -y^\top x - H(y) \quad \text{s.t.} \; 0 \le y \le 1, \; 1^\top y = 1$$

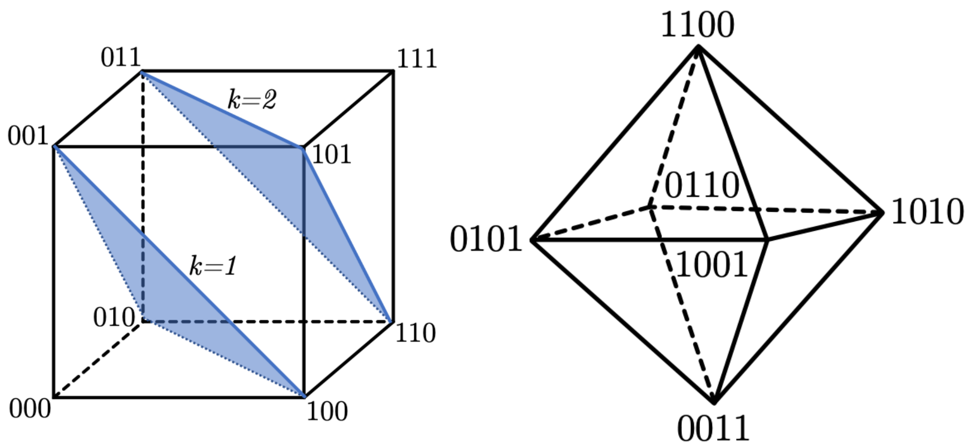
$$\pi_{\Delta_k} = \underset{y}{\operatorname{argmin}} \; -y^\top x - H_b(y) \quad \text{subject to} \; 0 \le y \le 1, \; 1^\top y = k$$

Has closed-form solution $\pi_\Delta(x) = \frac{\exp x}{\Sigma_i \exp x_i}$

No closed-form solution, can still differentiate



Entropy penalty of the softmax



top-k polytope

How to differentiate the controller?

Unrolling or autograd

$$\hat{u}_\theta^0 \to \hat{u}_\theta^1 \to \cdots \to \hat{u}_\theta^K \to \hat{\pi}_\theta(x) \to \mathcal{J}$$

$$\cdots$$

Idea: Implement controller, let **autodiff** do the rest
Like MAML's unrolled gradient descent

Ideal when **unconstrained** with a **short horizon**
Does **not** require a fixed-point or optimal solution
Instable and resource-intensive for large horizons

Can unroll algorithms **beyond gradient descent**
The differentiable cross-entropy method

Implicit differentiation

$$\mathrm{D}_\theta u^\star(\theta) = -\mathrm{D}_u g\big(\theta, u^\star(\theta)\big)^{-1} \mathrm{D}_\theta g\big(\theta, u^\star(\theta)\big)$$

Idea: Differentiate controller's optimality conditions

Agnostic of the control algorithm
Ill-defined if controller gives **suboptimal solution**
Memory and **compute** efficient: free in some cases

The Implicit Function Theorem

Dini 1877, Dontchev and Rockafellar 2009

Given an **implicit function** $\mathrm{u}^\star(\theta)\colon\ \mathbb{R}^n \to \mathbb{R}^m$
defined by $\mathrm{u}^\star(\theta) \in \{u\colon g(\theta, u) = 0\}$ where
$g(\theta, u)\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$

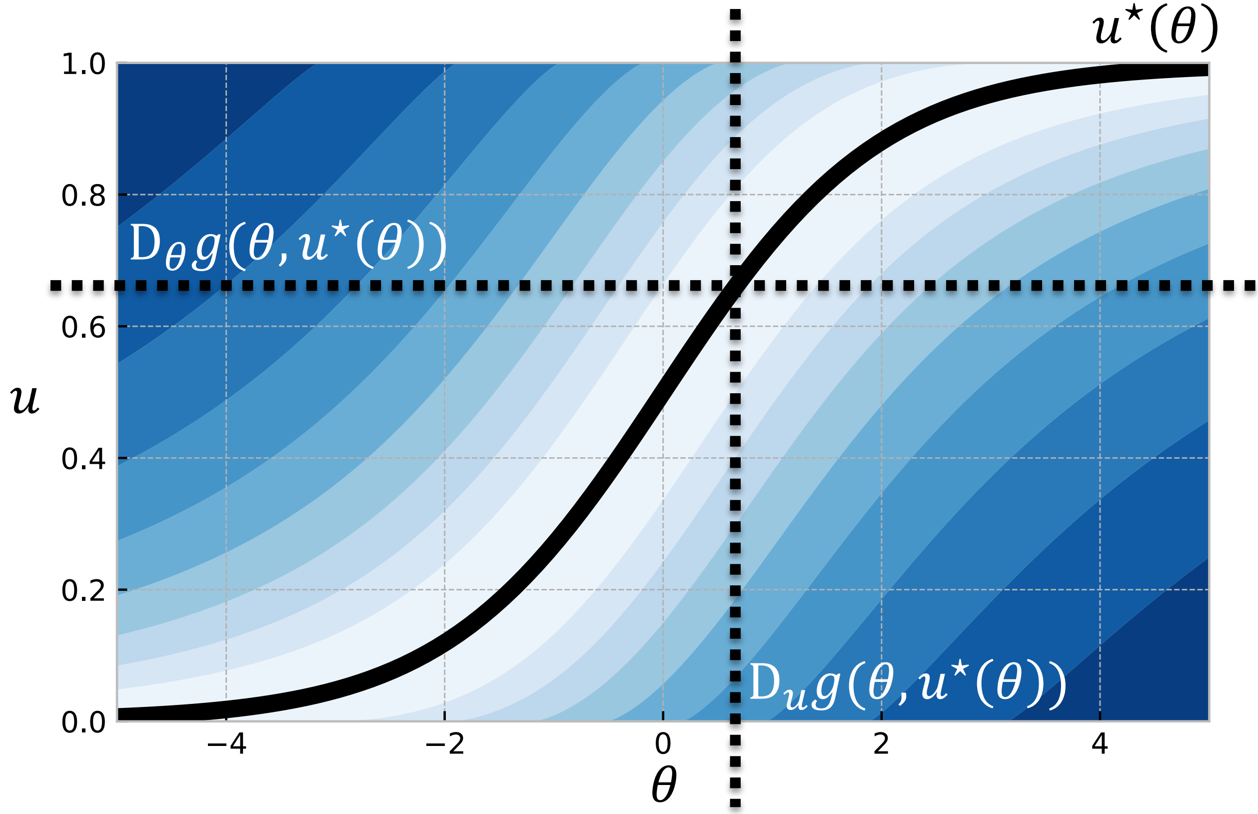
How can we compute $\mathrm{D}_\theta u^\star(\theta)$?

The **Implicit Function Theorem** gives

$$\mathrm{D}_\theta u^\star(\theta) = -\mathrm{D}_u g\big(\theta, u^\star(\theta)\big)^{-1} \mathrm{D}_\theta g\big(\theta, u^\star(\theta)\big)$$

under mild assumptions

Contour of $g(\theta, u)$ defining an **implicit function**



Each vertical slice is a control problem

Implicitly differentiating convex LQR control

$$\min_{\tau=\{x_t,u_t\}} \sum_t \tau_t^T C_t \tau_t + c_t \tau_t \;\; \text{s.t.} \;\; x_{t+1} = F_t \tau_t + f_t \;\; x_0 = x_{\text{init}}$$

Parameters: $\theta = \{C_t, c_t, F_t, F_t\}$

Define implicit function via **KKT optimality conditions**

Find $z^\star$ s.t. $Kz^\star + k = 0$ where $z^\star = [\tau^\star, \ldots]$

Solved with **Riccati recursion**

$$\overbrace{\begin{array}{c|c|c|c|c|c} & \tau_t & \lambda_t & \tau_{t+1} & \lambda_{t+1} & \\ \hline \ddots & & & & & \\ \hline & C_t & F_t^\top & & & \\ & F_t & & [-I \;\; 0] & & \\ \hline & & \begin{bmatrix} -I \\ 0 \end{bmatrix} & C_{t+1} & F_{t+1}^\top & \\ & & & F_{t+1} & & \\ \hline & & & & & \ddots \end{array}}^{K} \begin{bmatrix} \vdots \\ \tau_t^\star \\ \lambda_t^\star \\ \tau_{t+1}^\star \\ \lambda_{t+1}^\star \\ \vdots \end{bmatrix} = - \begin{bmatrix} \vdots \\ c_t \\ f_t \\ c_{t+1} \\ f_{t+1} \\ \vdots \end{bmatrix}$$

Backward pass: implicitly **differentiate** the LQR KKT conditions:

$$\frac{\partial \ell}{\partial C_t} = \frac{1}{2}\left(d_{\tau_t}^\star \otimes \tau_t^\star + \tau_t^\star \otimes d_{\tau_t}^\star\right) \qquad \frac{\partial \ell}{\partial c_t} = d_{\tau_t}^\star \qquad \frac{\partial \ell}{\partial x_{\text{init}}} = d_{\lambda_0}^\star$$

$$\frac{\partial \ell}{\partial F_t} = d_{\lambda_{t+1}}^\star \otimes \tau_t^\star + \lambda_{t+1}^\star \otimes d_{\tau_t}^\star \qquad \frac{\partial \ell}{\partial f_t} = d_{\lambda_t}^\star$$

where $K \begin{bmatrix} \vdots \\ d_{\tau_t}^\star \\ d_{\lambda_t}^\star \\ \vdots \end{bmatrix} = - \begin{bmatrix} \vdots \\ \nabla_{\tau_t^\star} \ell \\ 0 \\ \vdots \end{bmatrix}$

Just another LQR problem!

Differentiating non-convex MPC

$$x^\star_{1:T}, u^\star_{1:T} \in \operatorname*{argmin}_{x_{1:T}, u_{1:T}} \sum_t \underbrace{\boxed{C_\theta(x_t, u_t)}}_{\text{cost}} \;\text{s.t.}\; \underbrace{\boxed{x_1 = x_{\text{init}}}}_{\text{initial state}} \;\; \underbrace{\boxed{x_{t+1} = f_\theta(x_t, u_t)}}_{\text{dynamics}} \;\; \underbrace{\boxed{u_t \in \mathcal{U}}}_{\text{constraints}}$$

Solve with **sequential quadratic programming (SQP)**
Approximate non-convex argmin with the **final convex approximation**

Backward pass: differentiate the **convex approximation**, e.g., with:

$$\frac{\partial \ell}{\partial C_t} = \frac{1}{2}\left(d^\star_{\tau_t} \otimes \tau^\star_t + \tau^\star_t \otimes d^\star_{\tau_t}\right) \qquad \frac{\partial \ell}{\partial c_t} = d^\star_{\tau_t} \qquad \frac{\partial \ell}{\partial x_{\text{init}}} = d^\star_{\lambda_0}$$

$$\frac{\partial \ell}{\partial F_t} = d^\star_{\lambda_{t+1}} \otimes \tau^\star_t + \lambda^\star_{t+1} \otimes d^\star_{\tau_t} \qquad \frac{\partial \ell}{\partial f_t} = d^\star_{\lambda_t}$$

where $K \begin{bmatrix} \vdots \\ d^\star_{\tau_t} \\ d^\star_{\lambda_t} \\ \vdots \end{bmatrix} = - \begin{bmatrix} \vdots \\ \nabla_{\tau^\star_t} \ell \\ 0 \\ \vdots \end{bmatrix}$

Just an LQR problem!
(in some cases)

This talk: differentiate the controller!

Foundations of differentiable optimization and control
  Unrolling or autograd (gradient descent, differentiable cross-entropy method)
  Implicit differentiation (convex and non-convex MPC)

`cvxpylayers`: **Prototyping** differentiable convex optimization and control

Applications of differentiable control
  Objective mismatch
  Amortized control

Optimization layers need to be carefully implemented

$$
\begin{aligned}
\mathsf{d}Qz^\star + Q\mathsf{d}z + \mathsf{d}q + \mathsf{d}A^T\nu^\star + \\
A^T\mathsf{d}\nu + \mathsf{d}G^T\lambda^\star + G^T\mathsf{d}\lambda &= 0 \\
\mathsf{d}Az^\star + A\mathsf{d}z - \mathsf{d}b &= 0 \\
D(Gz^\star - h)\mathsf{d}\lambda + D(\lambda^\star)(\mathsf{d}Gz^\star + G\mathsf{d}z - \mathsf{d}h) &= 0
\end{aligned}
$$

$$
\begin{bmatrix} Q & A^\top & \tilde{G}^\top \\ A & 0 & 0 \\ \tilde{G} & 0 & 0 \end{bmatrix}
\begin{bmatrix} d_x^\star \\ d_\lambda^\star \\ d_{\tilde{\nu}}^\star \end{bmatrix}
= - \begin{bmatrix} \nabla_{x^\star}\ell \\ 0 \\ 0 \end{bmatrix}
$$

$$
\begin{bmatrix} Q & G^T & A^T \\ D(\lambda^\star)G & D(Gz^\star - h) & 0 \\ A & 0 & 0 \end{bmatrix}
\begin{bmatrix} \mathsf{d}z \\ \mathsf{d}\lambda \\ \mathsf{d}\nu \end{bmatrix}
= \begin{bmatrix} -\mathsf{d}Qz^\star - \mathsf{d}q - \mathsf{d}G^T\lambda^\star - \mathsf{d}A^T\nu^\star \\ -D(\lambda^\star)\mathsf{d}Gz^\star + D(\lambda^\star)\mathsf{d}h \\ -\mathsf{d}Az^\star + \mathsf{d}b \end{bmatrix}
$$

```
invQ_AT = A.transpose(1, 2).lu_solve(*Q_LU)
A_invQ_AT = torch.bmm(A, invQ_AT)
G_invQ_AT = torch.bmm(G, invQ_AT)

LU_A_invQ_AT = lu_hack(A_invQ_AT)
P_A_invQ_AT, L_A_invQ_AT, U_A_invQ_AT = torch.lu_unpack(*
P_A_invQ_AT = P_A_invQ_AT.type_as(A_invQ_AT)

S_LU_11 = LU_A_invQ_AT[0]
U_A_invQ_AT_inv = (P_A_invQ_AT.bmm(L_A_invQ_AT)
                   ).lu_solve(*LU_A_invQ_AT)
S_LU_21 = G_invQ_AT.bmm(U_A_invQ_AT_inv)
T = G_invQ_AT.transpose(1, 2).lu_solve(*LU_A_invQ_AT)
S_LU_12 = U_A_invQ_AT.bmm(T)
S_LU_22 = torch.zeros(nBatch, nineq, nineq).type_as(Q)
S_LU_data = torch.cat((torch.cat((S_LU_11, S_LU_12), 2),
                       torch.cat((S_LU_21, S_LU_22), 2)),
                      1)
S_LU_pivots[:, :neq] = LU_A_invQ_AT[1]

R -= G_invQ_AT.bmm(T)
```

$$
\overbrace{
\begin{array}{c}
\begin{array}{cccc} \tau_t & \lambda_t & \tau_{t+1} & \lambda_{t+1} \end{array} \\
\begin{bmatrix}
\ddots & & & & \\
& C_t & F_t^\top & & \\
& F_t & & [-I \quad 0] & \\
& & \begin{bmatrix} -I \\ 0 \end{bmatrix} & C_{t+1} & F_{t+1}^\top \\
& & & F_{t+1} & \\
& & & & \ddots
\end{bmatrix}
\end{array}
}^{K}
\begin{bmatrix} \vdots \\ \tau_t^\star \\ \lambda_t^\star \\ \tau_{t+1}^\star \\ \lambda_{t+1}^\star \\ \vdots \end{bmatrix}
= - \begin{bmatrix} \vdots \\ c_t \\ f_t \\ c_{t+1} \\ f_{t+1} \\ \vdots \end{bmatrix}
$$

$$
\nabla_Q \ell = \frac{1}{2}(d_z z^T + z d_z^T) \qquad \nabla_q \ell = d_z
$$

$$
\nabla_A \ell = d_\nu z^T + \nu d_z^T \qquad \nabla_b \ell = -d_\nu
$$

$$
\nabla_G \ell = D(\lambda^\star)(d_\lambda z^T + \lambda d_z^T) \qquad \nabla_h \ell = -D(\lambda^\star) d_\lambda
$$

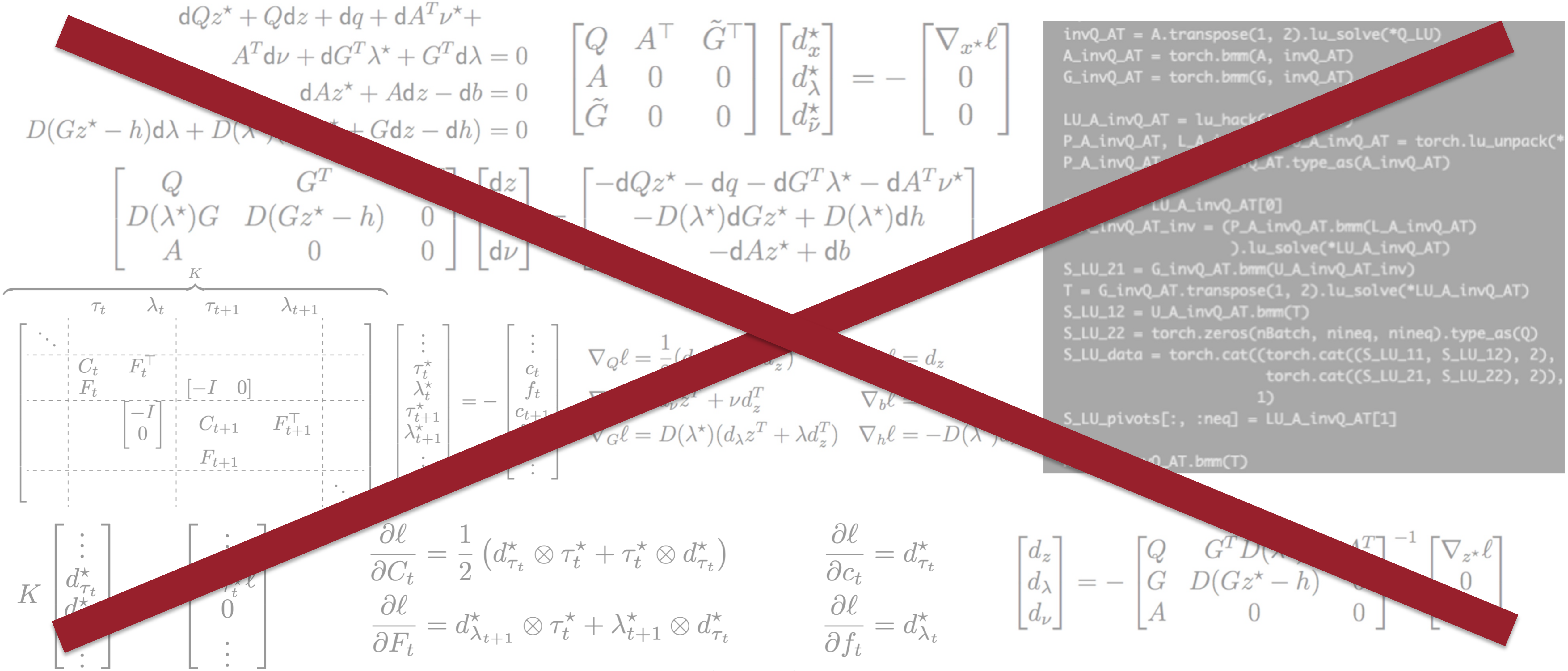
$$
K \begin{bmatrix} \vdots \\ d_{\tau_t}^\star \\ d_{\lambda_t}^\star \\ \vdots \end{bmatrix}
= - \begin{bmatrix} \vdots \\ \nabla_{\tau_t^\star}\ell \\ 0 \\ \vdots \end{bmatrix}
$$

$$
\frac{\partial \ell}{\partial C_t} = \frac{1}{2}\left(d_{\tau_t}^\star \otimes \tau_t^\star + \tau_t^\star \otimes d_{\tau_t}^\star\right) \qquad \frac{\partial \ell}{\partial c_t} = d_{\tau_t}^\star
$$

$$
\frac{\partial \ell}{\partial F_t} = d_{\lambda_{t+1}}^\star \otimes \tau_t^\star + \lambda_{t+1}^\star \otimes d_{\tau_t}^\star \qquad \frac{\partial \ell}{\partial f_t} = d_{\lambda_t}^\star
$$

$$
\begin{bmatrix} d_z \\ d_\lambda \\ d_\nu \end{bmatrix}
= - \begin{bmatrix} Q & G^T D(\lambda^\star) & A^T \\ G & D(Gz^\star - h) & 0 \\ A & 0 & 0 \end{bmatrix}^{-1}
\begin{bmatrix} \nabla_{z^\star}\ell \\ 0 \\ 0 \end{bmatrix}
$$

Why should practitioners care?



Differentiable convex optimization layers

NeurIPS 2019 and officially in CVXPY!
Joint work with A. Agrawal, S. Barratt, S. Boyd, S. Diamond, J. Z. Kolter

Useful for **convex control** problems and subproblems

CVXpy

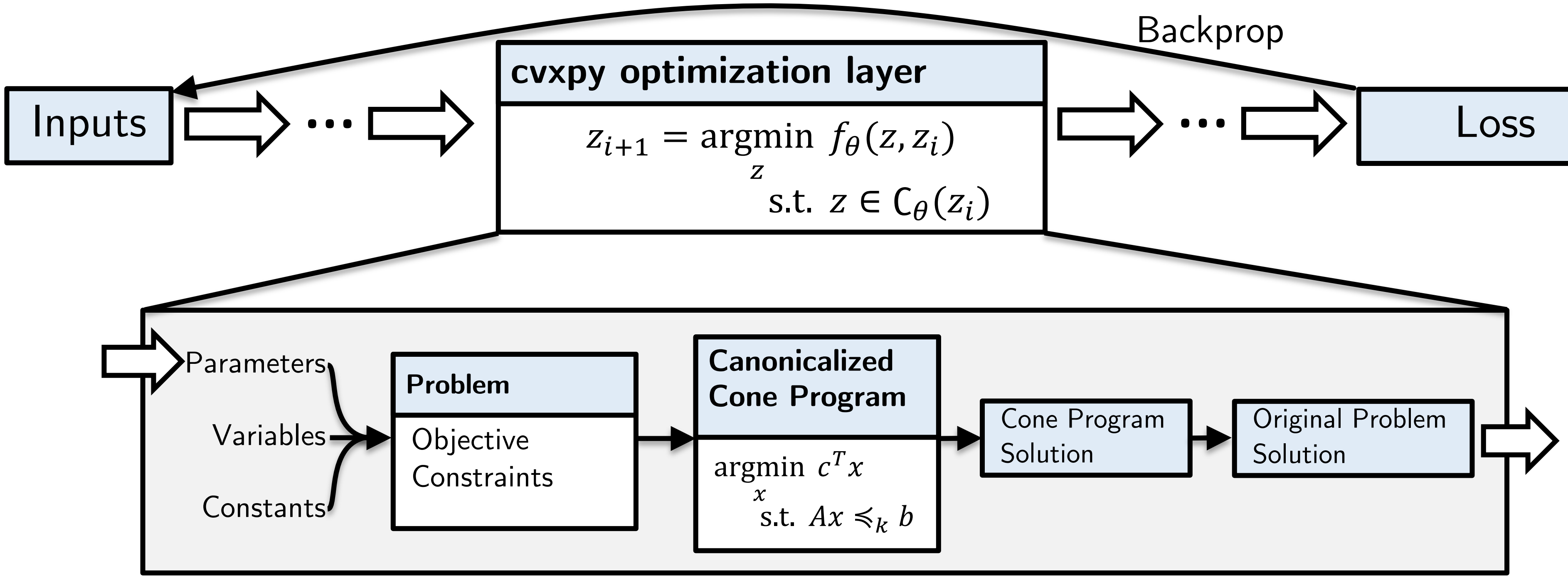
$$
\begin{aligned}
x^\star(\theta) = \ & \underset{x}{\operatorname{argmin}} \ f(x;\theta) \\
& \text{subject to} \ g(x;\theta) \leq 0 \\
& \qquad\qquad\quad h(x;\theta) = 0
\end{aligned}
$$

PyTorch

TensorFlow

locuslab.github.io/2019-10-28-cvxpylayers

Rapidly prototyping optimization layers



Code example: OptNet QP

Before: 1k lines of code
Hand-implemented and optimized PyTorch GPU-capable batched primal-dual interior point method

Now: <10 lines of code
Same speed

$$x^\star = \operatorname*{argmin}_z \frac{1}{2} x^T Q x + p^\top x \quad \text{s.t.} \quad Ax = b, \quad Gx \le h$$

$$\theta = \{Q, p, A, b, G, h\}$$

```
obj = cp.Minimize(0.5*cp.quad_form(x, Q) + p.T * x)
cons = [A*x == b, G*x <= h]
prob = cp.Problem(obj, cons)
layer = CvxpyLayer(prob, params=[Q, p, A, b, G, h], out=[x]
```

Write **standard CVXPY** problem

Export to PyTorch, TensorFlow, JAX

Under the hood: cone program differentiation

Section 7 of my thesis and in Agrawal et al.

$$x^\star = \underset{x}{\operatorname{argmin}} \; c^\top x \quad \text{subject to } b - Ax \in \mathcal{K}$$

Conic Optimality

Find $z^\star$ s.t. $\mathcal{R}(z^\star, \theta) = 0$ where $z^\star = [x^\star, \ldots]$ and $\theta = \{A, b, c\}$

Implicitly differentiating $\mathcal{R}$ gives $D_\theta(z^\star) = -\left(D_z\mathcal{R}(z^\star)\right)^{-1} D_\theta\mathcal{R}(z^\star)$

This talk: differentiate the controller!

Foundations of differentiable optimization and control
- Unrolling or autograd (gradient descent, differentiable cross-entropy method)
- Implicit differentiation (convex and non-convex MPC)

`cvxpylayers`: **Prototyping** differentiable convex optimization and control

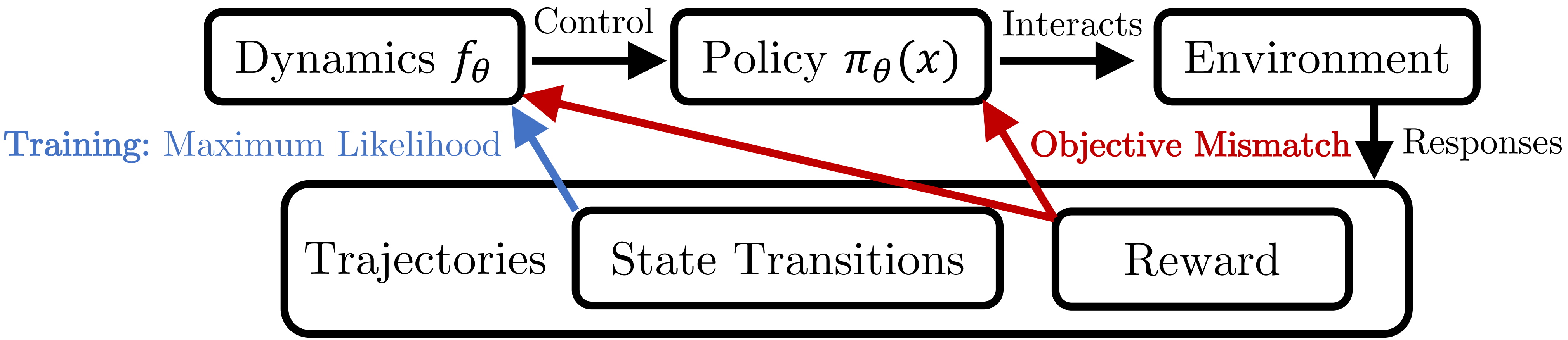
Applications of differentiable control
- Objective mismatch
- Amortized control

The Objective Mismatch Problem

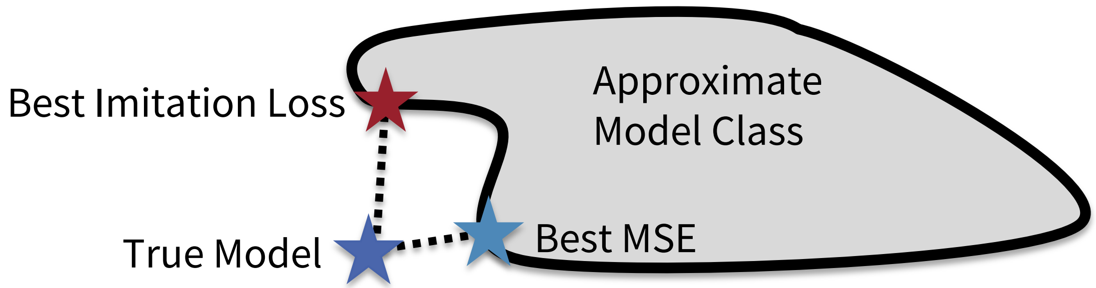
Summary: Maximum-likelihood training of dynamics separate from controlling the dynamics
Especially problematic with inaccurate models

The **controller** (i.e. policy) **optimizes over the dynamics**
Can find **adversarial trajectories** that **appear deceptively "good"**

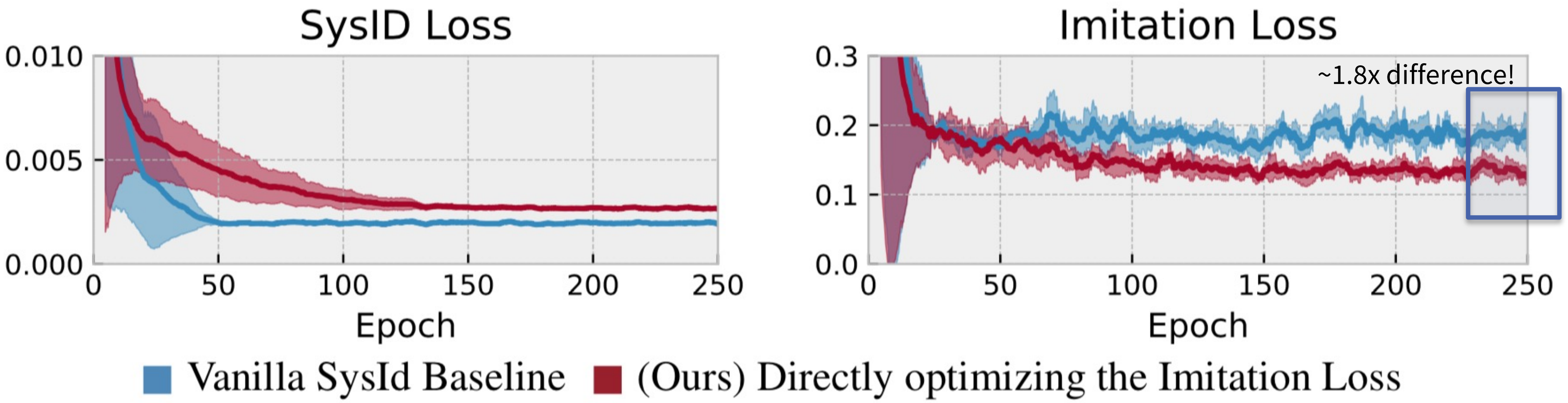
Differentiable control one potential solution, may be **combined with many others:**
advantage weighting, value-gradient weighting, value-aware model learning



Optimizing the task loss is better than SysID



True System: Pendulum environment with noise (damping and a wind force)
Approximate Model: Pendulum without the noise terms



Optimizing system models with a task loss

Among many others!

Using a Financial Training Criterion Rather than a Prediction Criterion*

Yoshua Bengio†

Gnu-RL: A Precocial Reinforcement Learning Solution for Building HVAC Control Using a Differentiable MPC Policy

Bingqing Chen
Carnegie Mellon University
Pittsburgh, PA, USA
bingqinc@andrew.cmu.edu

Zicheng Cai
Dell Technologies
Austin, TX, USA
zicheng.cai@dell.com

Mario Bergés
Carnegie Mellon University
Pittsburgh, PA, USA
mberges@andrew.cmu.edu

Smart "Predict, then Optimize"

Adam N. Elmachtoub
Department of Industrial Engineering and Operations Research and Data Science Institute, Columbia University, New York, NY 10027, adam@ieor.columbia.edu

Paul Grigas
Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720, pgrigas@berkeley.edu

Task-based End-to-end Model Learning in Stochastic Optimization

Priya L. Donti
Dept. of Computer Science
Dept. of Engr. & Public Policy
Carnegie Mellon University
Pittsburgh, PA 15213
pdonti@cs.cmu.edu

Brandon Amos
Dept. of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
bamos@cs.cmu.edu

J. Zico Kolter
Dept. of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
zkolter@cs.cmu.edu

Learning Convex Optimization Control Policies

Akshay Agrawal AKSHAYKA@CS.STANFORD.EDU
Shane Barratt SBARRATT@STANFORD.EDU
Stephen Boyd BOYD@STANFORD.EDU
450 Serra Mall, Stanford, CA, 94305

Bartolomeo Stellato* STELLATO@MIT.EDU
77 Massachusetts Ave, Cambridge, MA, 02139

This talk: differentiate the controller!

Foundations of differentiable optimization and control
  Unrolling or autograd (gradient descent, differentiable cross-entropy method)
  Implicit differentiation (convex and non-convex MPC)

`cvxpylayers`: **Prototyping** differentiable convex optimization and control

Applications of differentiable control
  Objective mismatch
  Amortized control

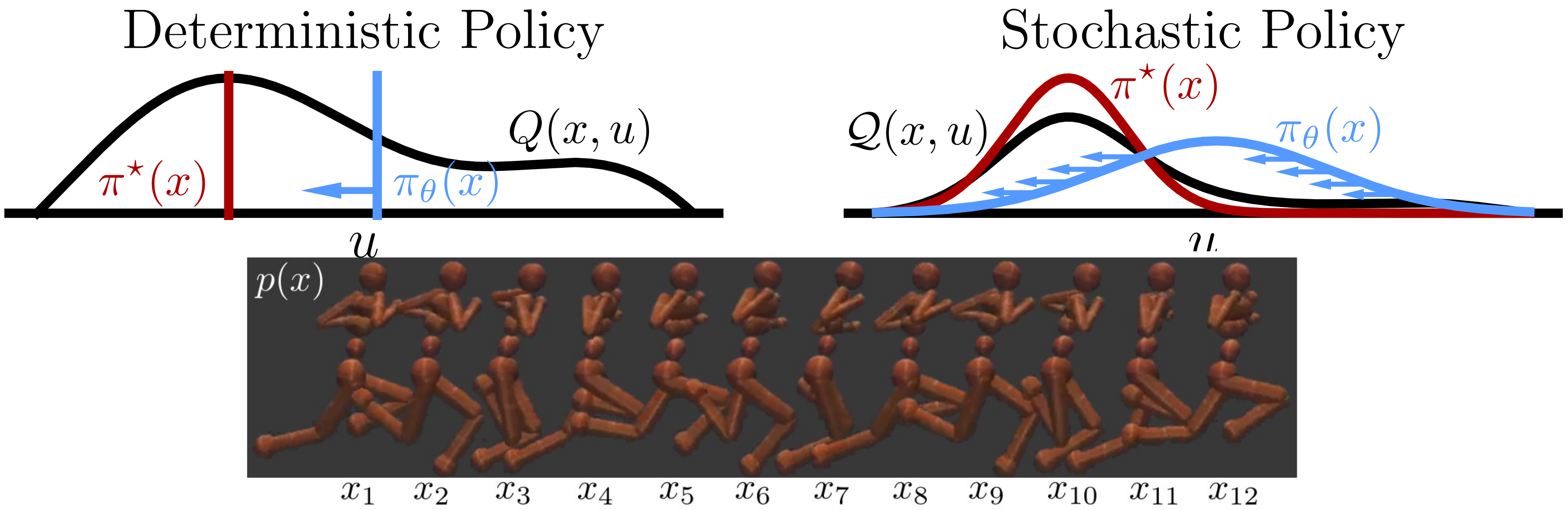
RL policy learning is amortized optimization

Setup: controlling a **continuous MDP** with a **model-free policy** $\pi_\theta(x)$

Review: Learning a policy with a **value gradient** amortizes over the Q-value:

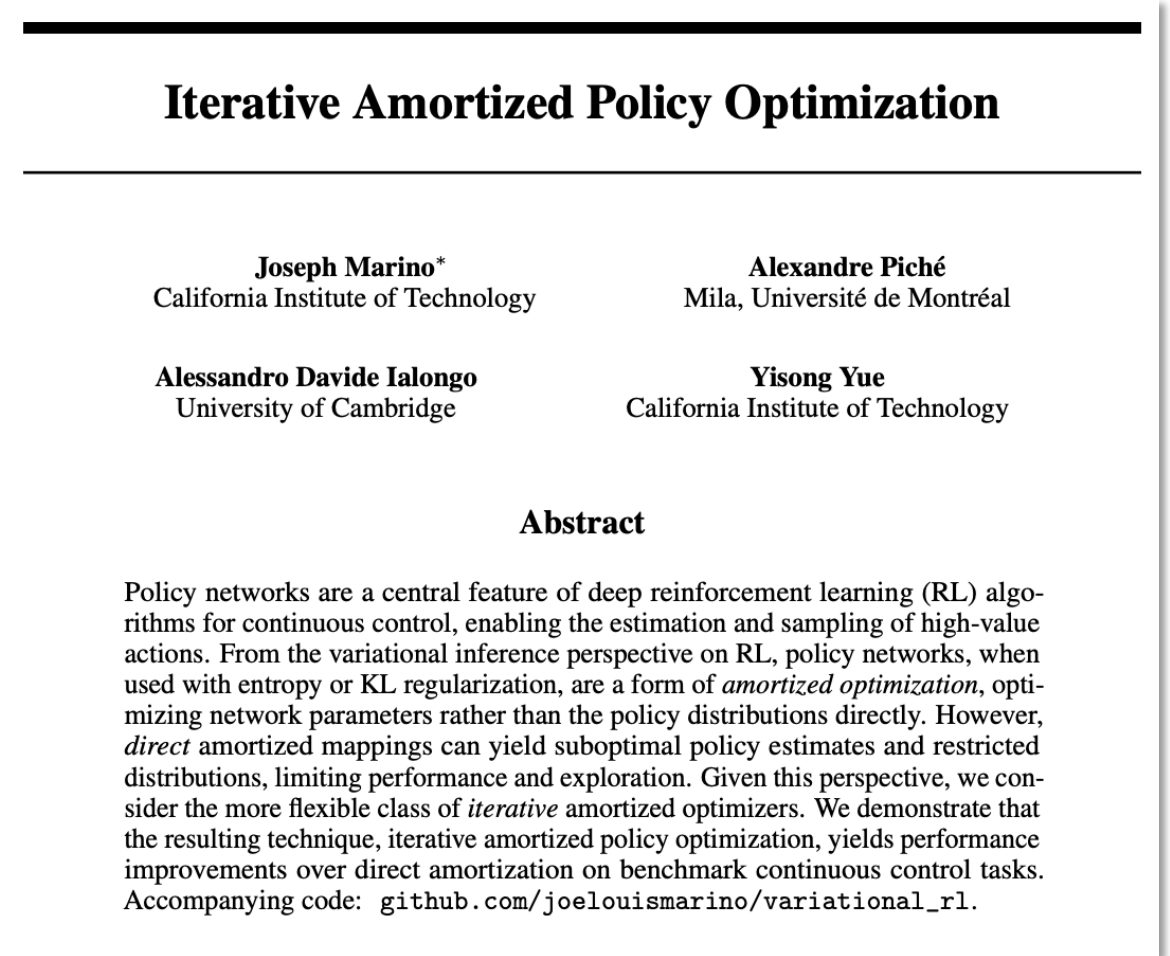
$$\underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{p(x)}\, Q(x, \pi_\theta(x))$$

$\pi_\theta(x)$ is **fully amortized**: tries to **predict** the max-Q operation **without looking at the Q function!**
The **amortization perspective** easily enables us to consider other policies



Amortized control via unrolled gradient descent

The policy's **prediction is adapted** to maximize the Q function for every state
Unrolled gradient descent: policy has **knowledge it is going to be adapted**
Can generalize to **other differentiable optimizers**, e.g., the cross-entropy method



Iterative Amortized Policy Optimization

Joseph Marino*
California Institute of Technology

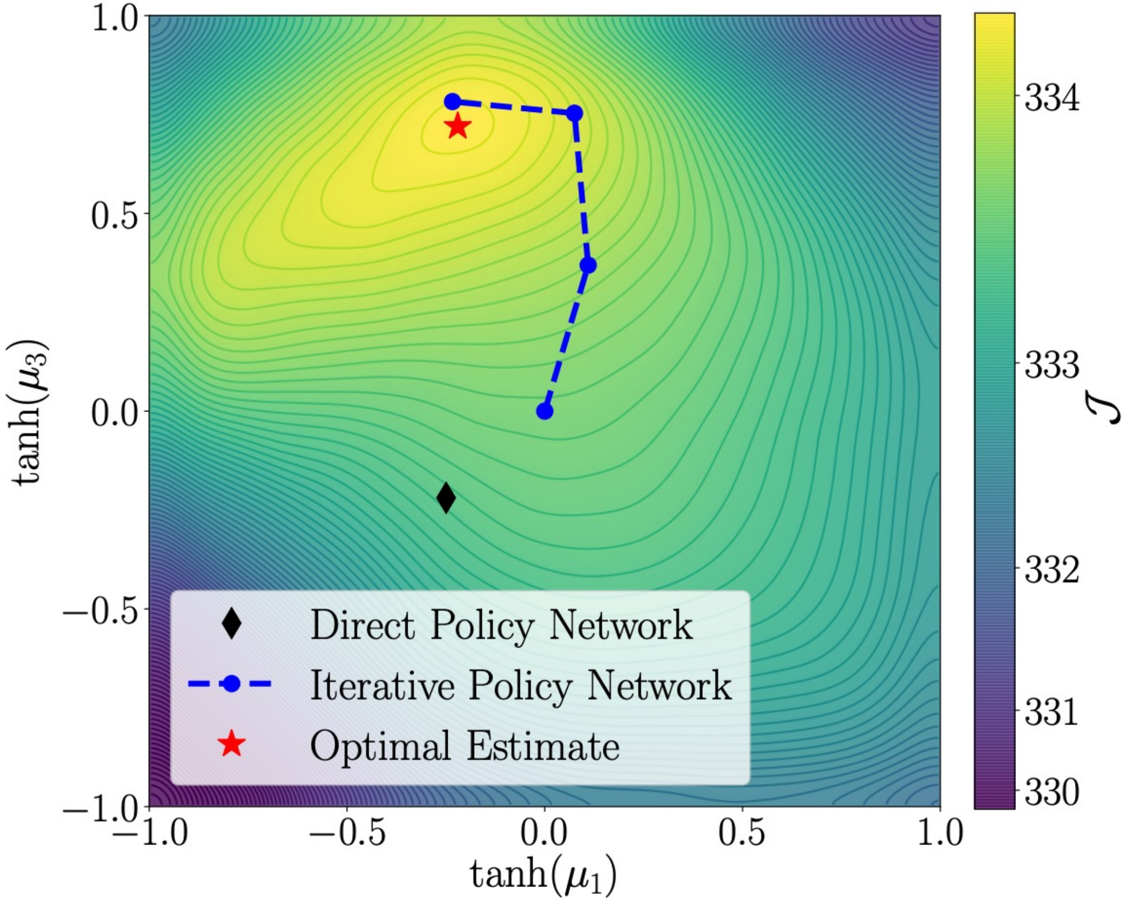
Alexandre Piché
Mila, Université de Montréal

Alessandro Davide Ialongo
University of Cambridge

Yisong Yue
California Institute of Technology

Abstract

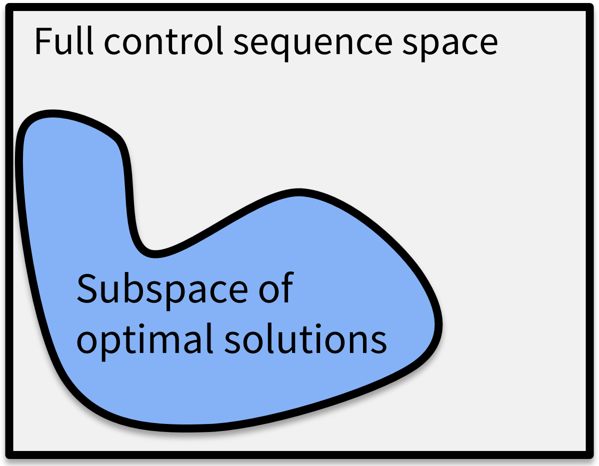
Policy networks are a central feature of deep reinforcement learning (RL) algorithms for continuous control, enabling the estimation and sampling of high-value actions. From the variational inference perspective on RL, policy networks, when used with entropy or KL regularization, are a form of *amortized optimization*, optimizing network parameters rather than the policy distributions directly. However, *direct* amortized mappings can yield suboptimal policy estimates and restricted distributions, limiting performance and exploration. Given this perspective, we consider the more flexible class of *iterative* amortized optimizers. We demonstrate that the resulting technique, iterative amortized policy optimization, yields performance improvements over direct amortization on benchmark continuous control tasks. Accompanying code: `github.com/joelouismarino/variational_rl`.



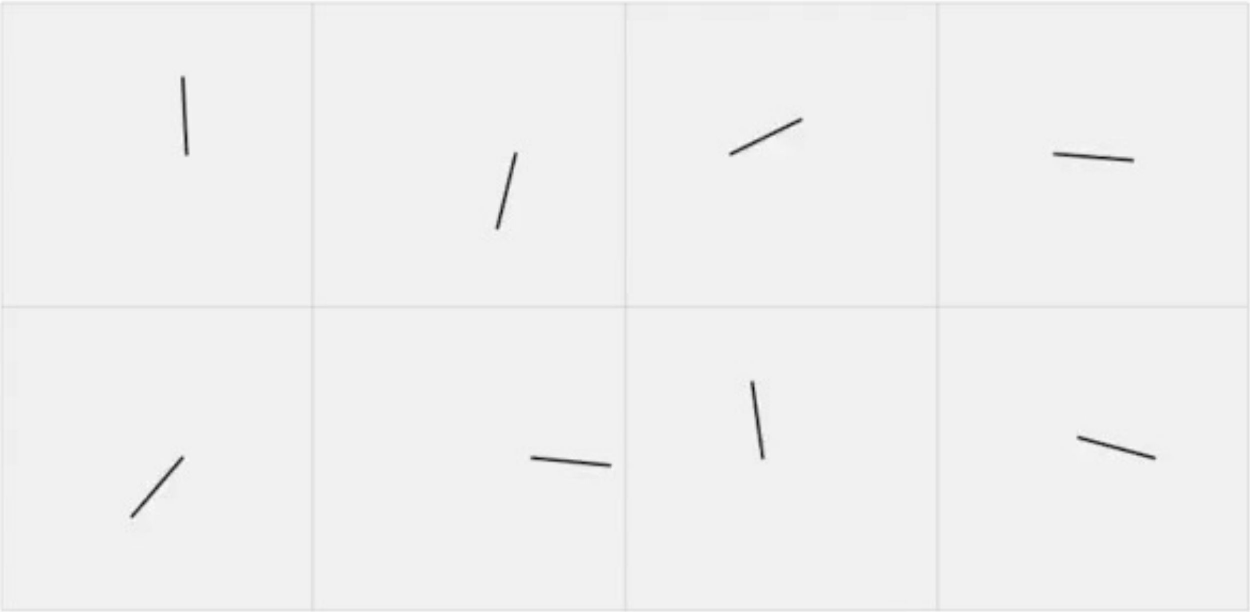
Optimal control sequences share structure

Control optimization problems are **repeatedly solved** for every state
Optimal control sequences **do not live in isolation** and **share structure**

Use **differentiable control** to **learn a latent subspace**
Only search over optimal solutions rather than the entire space
Amortizes the original control optimization problem



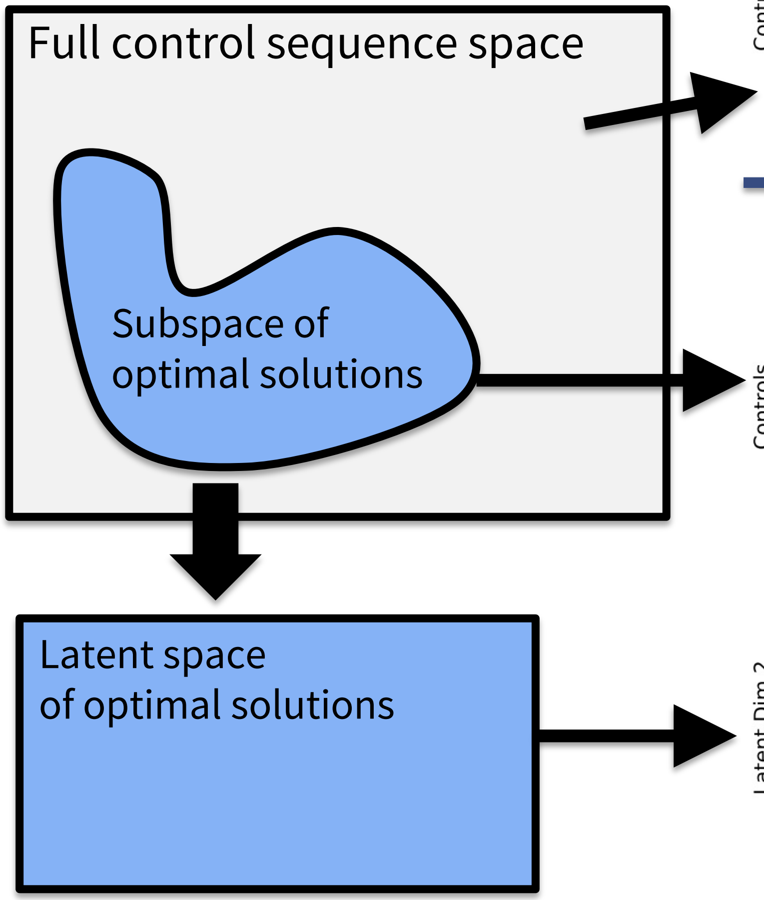
Cartpole videos

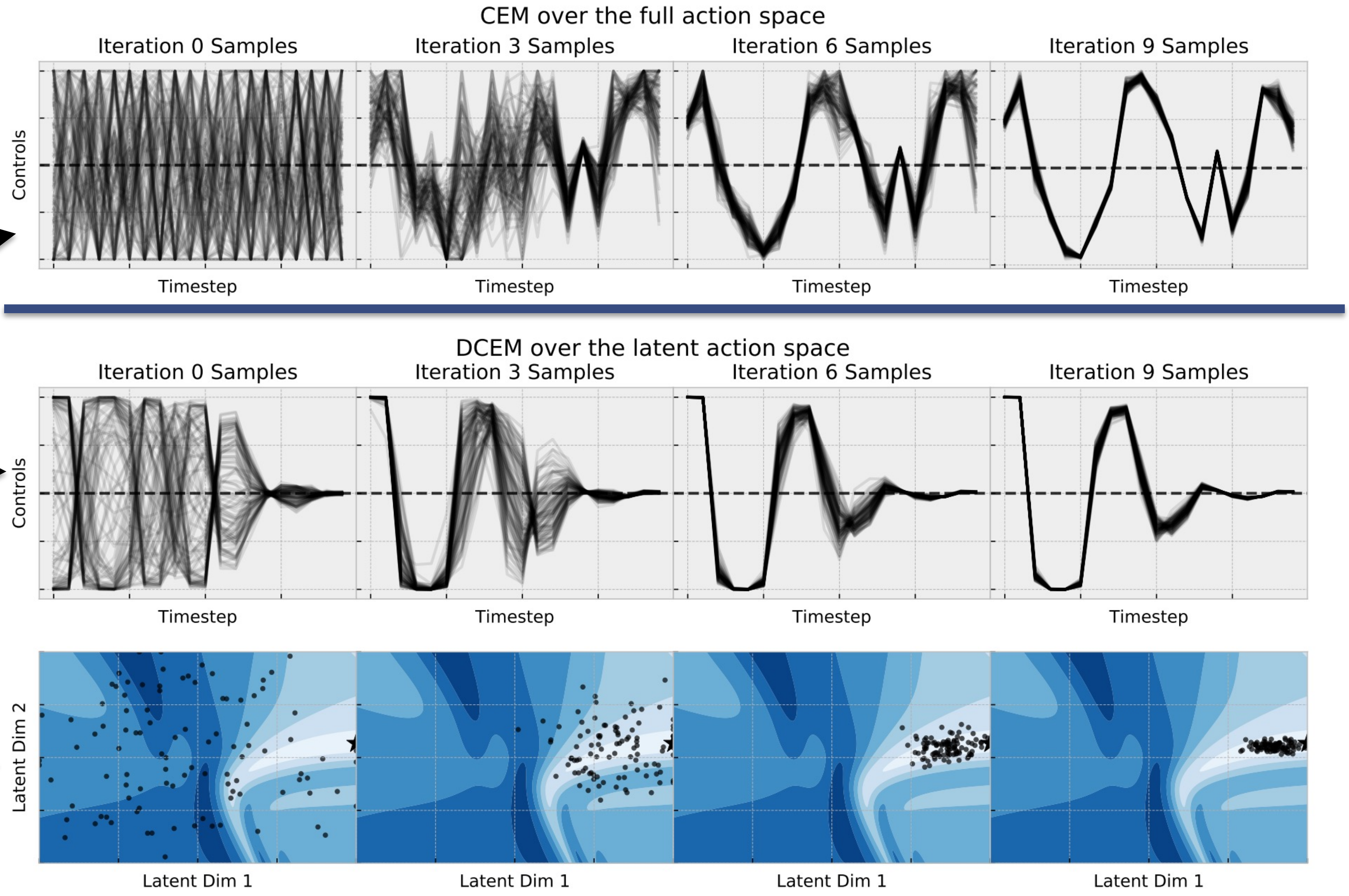


Optimal controls over time — force on the cartpole

DCEM learns the solution space structure

$$u^\star = \underset{u\in[0,1]^N}{\operatorname{argmin}}\ f(u)$$





Closing Thoughts And Future Directions

Differentiable optimization and **control** are **powerful primitives** to use within larger systems

- **Theoretical** and **engineering** foundations are here
- Works for **convex** and **non-convex** control
- Specify and hand-engineer the parts you know, **learn the rest**
- Can be **propagated through and learned**, just like any layer

Applications in:

- Objective mismatch
- Amortized optimization
- Safe and robust control
- Learning state embeddings

Differentiable optimization for control and reinforcement learning

Brandon Amos
Meta AI NYC, Fundamental AI Research (FAIR)

brandondamos bamos.github.io

Slides available at:
github.com/bamos/presentations

Differentiable QPs: OptNet [Amos and Kolter, ICML 2017]
Differentiable task-based stochastic optimization [Donti, Amos, Kolter, NeurIPS 2017]
Differentiable MPC for end-to-end planning and control [Amos, Jimenez, Sacks, Boots, Kolter, NeurIPS 2018]
Differentiable Convex Optimization Layers [Agrawal*, Amos*, Barratt*, Boyd*, Diamond*, Kolter*, NeurIPS 2019]
Differentiable optimization-based modeling for ML [Amos, Ph.D. Thesis 2019]
Differentiable Cross-Entropy Method [Amos and Yarats, ICML 2020]
Objective mismatch in model-based reinforcement learning [Lambert, Amos, Yadan, Calandra, L4DC 2020]
On the model-based stochastic value gradient [Amos, Stanton, Yarats, Wilson, L4DC 2021]
Tutorial on amortized optimization [Amos, arXiv 2022]

Collaborators: Akshay Agrawal, Shane Barratt, Byron Boots, Stephen Boyd, Roberto Calandra, Steven Diamond, Priya Donti, Ivan Jimenez, Zico Kolter, Nathan Lambert, Jacob Sacks, Samuel Stanton, Andrew Gordon Wilson, Omry Yadan, and Denis Yarats