On differentiable optimization for control and vision

Brandon Amos • Facebook AI Research

Joint with Akshay Agrawal, Shane Barratt, Byron Boots, Stephen Boyd, Roberto Calandra, Steven Diamond, Priya Donti, Ivan Jimenez, Zico Kolter, Vladlen Koltun, Nathan Lambert, Jacob Sacks, Omry Yadan, and Denis Yarats

This Talk

Foundation: Differentiable convex optimization

Differentiable continuous control

- Differentiable model predictive control
- Differentiable cross-entropy method

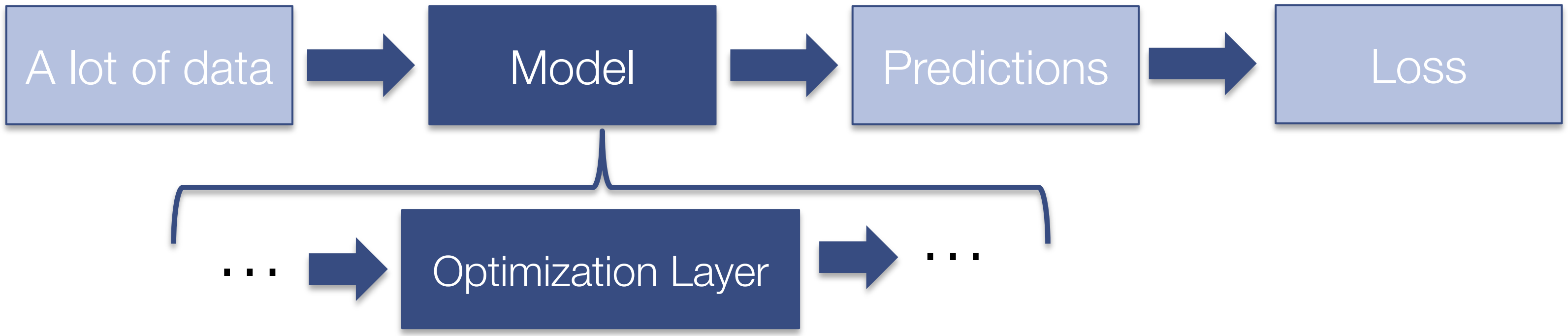
Can we throw big neural networks at every problem?

(Maybe) Neural networks are **soaring** in vision, RL, and language

A lot of data → Model → Predictions → Loss

AGI: A pile of linear algebra?

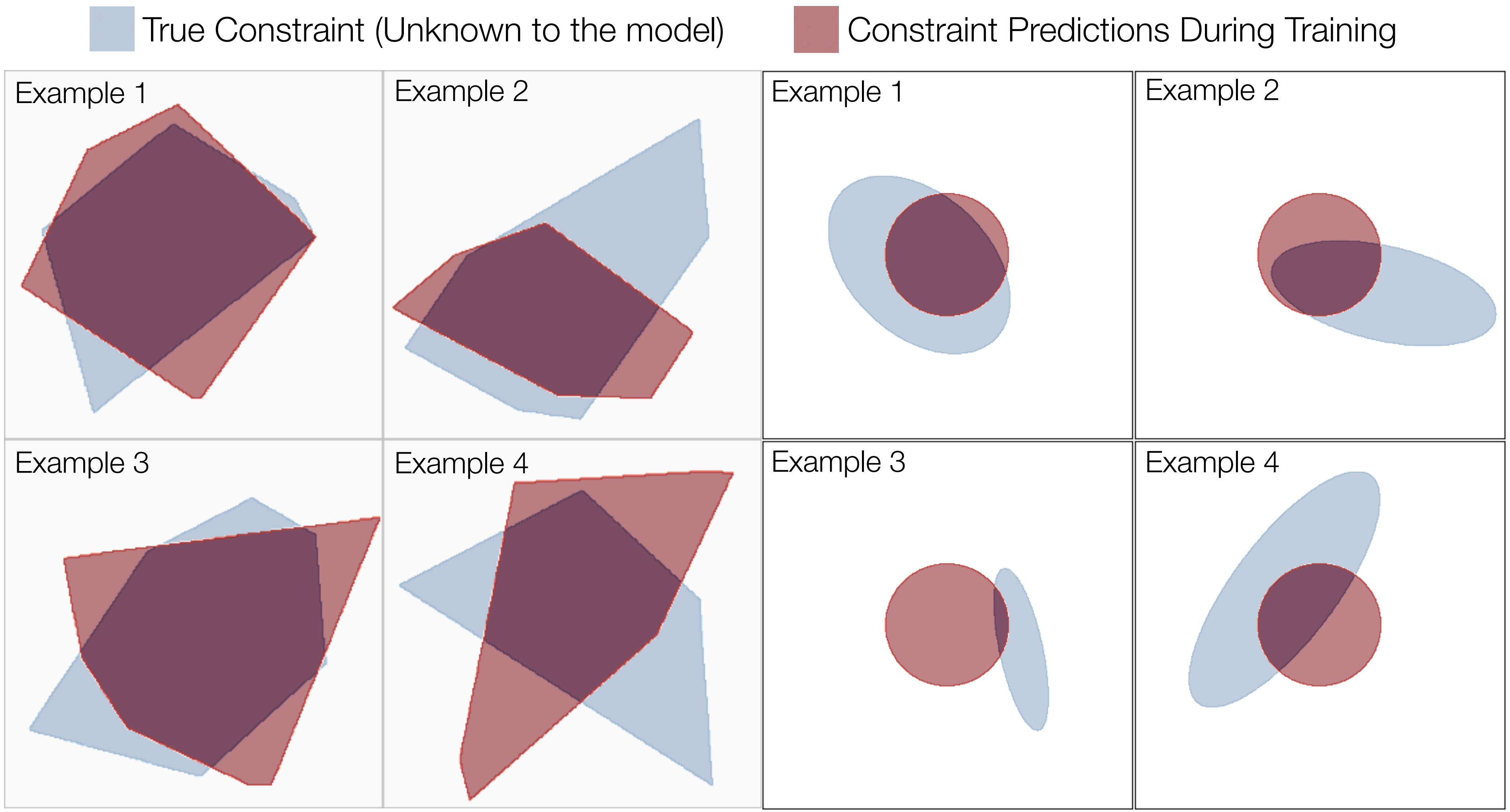
Optimization-Based Modeling for Machine Learning



$$\begin{aligned} z_{i+1} = \quad & \underset{z}{\operatorname{argmin}} \; f_\theta(z, z_i) \\ & \text{subject to } \; z \in C_\theta(z, z_i) \end{aligned}$$

- Adds **domain knowledge** and **hard constraints** to your **modeling** pipeline
- **Integrates** and **trains** nicely with your other **end-to-end** modeling components
- Applications in **RL, control, meta-learning, game theory, optimal transport**

Optimization Layers Model Constraints



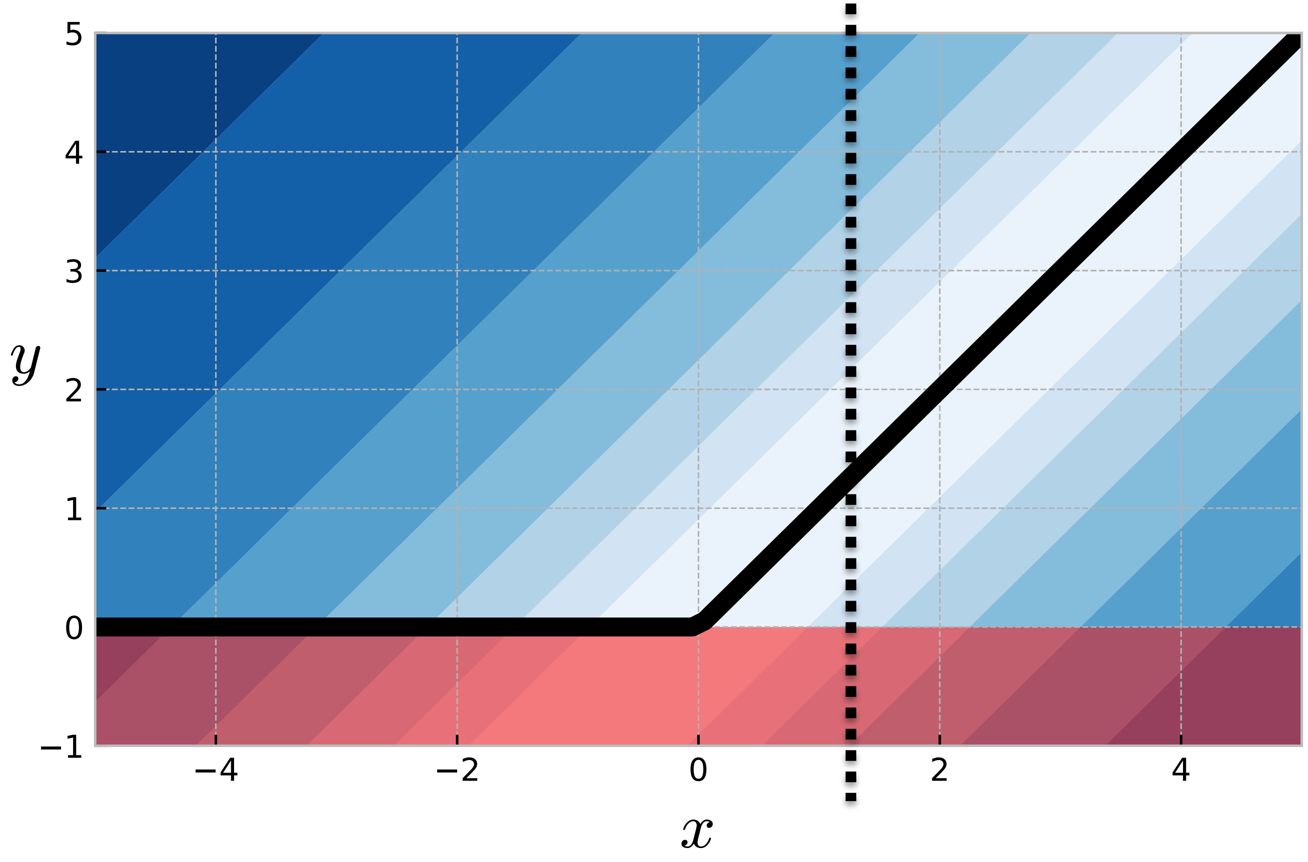
Optimization Perspective of the ReLU

Proof [S2 of my thesis]: Comes from first-order optimality

$$y = \max\{0, x\}$$

$\Updownarrow$

$$y^\star = \underset{y}{\operatorname{argmin}} \; \|y - x\|_2^2 \quad \text{subject to} \quad y \geq 0$$



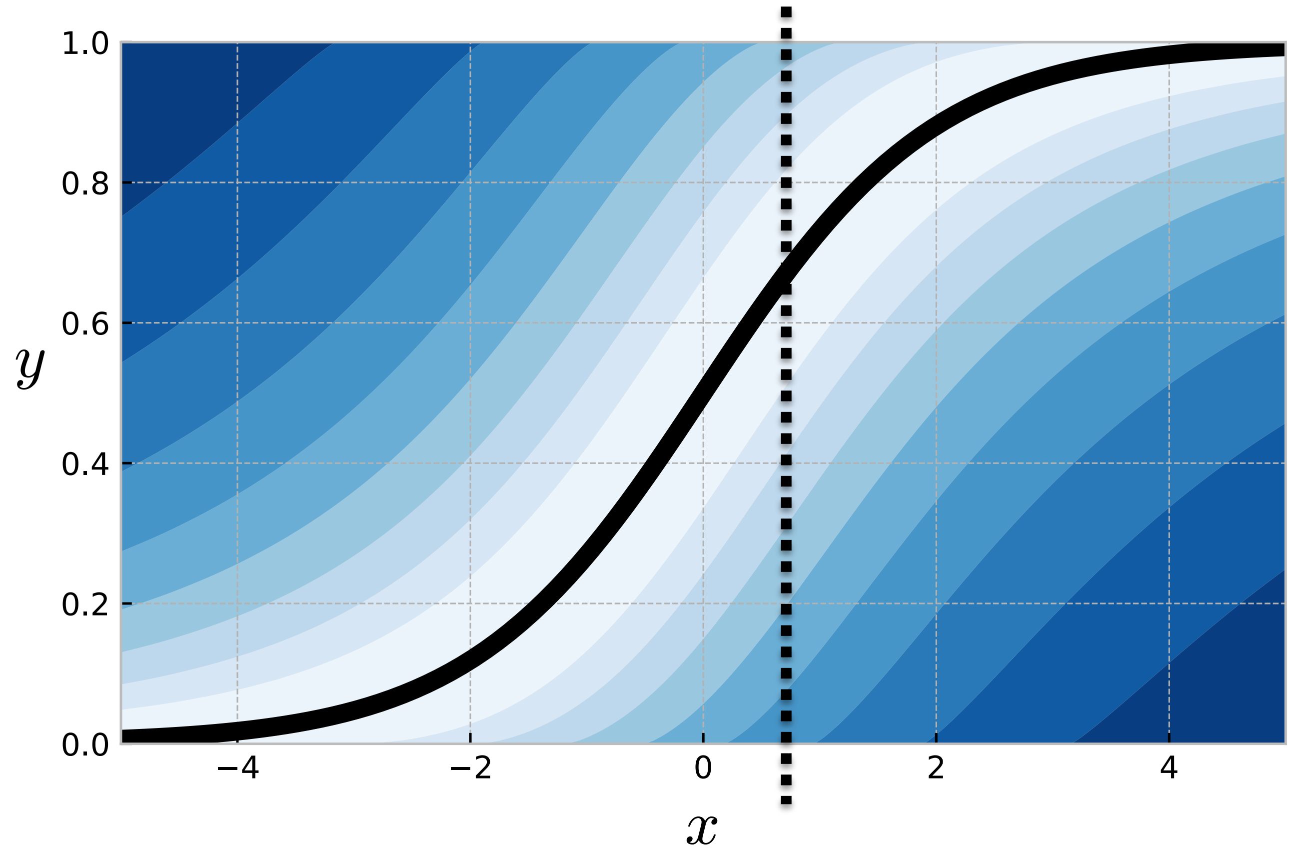
Optimization Perspective of the Sigmoid

Proof [S2 of my thesis]: Comes from first-order optimality

$$y = \frac{1}{1 + \exp\{-x\}}$$

$$\Updownarrow$$

$$y^\star = \operatorname*{argmin}_y \; -y^\top x - H_b(y) \quad \text{subject to} \quad 0 \le y \le 1$$



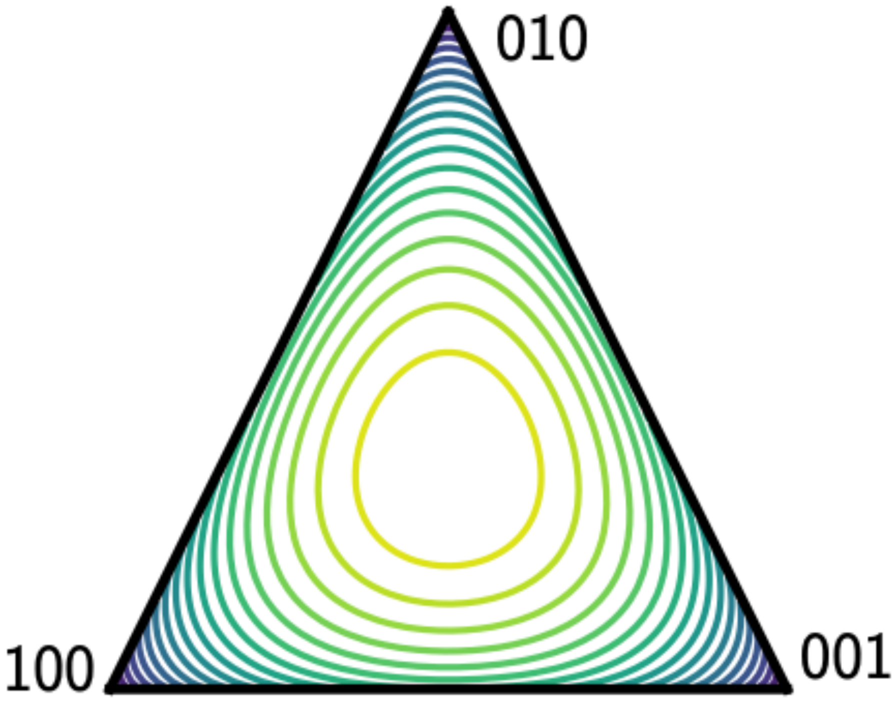
Optimization Perspective of the Softmax

Proof [S2 of my thesis]: Comes from first-order optimality

$$y = \frac{\exp x}{\Sigma_i \exp x_i}$$

$\Updownarrow$

$$\begin{aligned} y^\star = \;& \underset{y}{\operatorname{argmin}} \;\; -y^\top x - H(y) \\ & \text{subject to} \;\; 0 \le y \le 1 \\ & \qquad\qquad\quad 1^\top y = 1 \end{aligned}$$



How can we generalize this?

$$
\begin{aligned}
z_{i+1} = \quad & \underset{z}{\operatorname{argmin}} \; f_\theta(z, z_i) \\
& \text{subject to } \; z \in C_\theta(z, z_i)
\end{aligned}
$$

The Implicit Function Theorem

[Dini 1877, Dontchev and Rockafellar 2009]

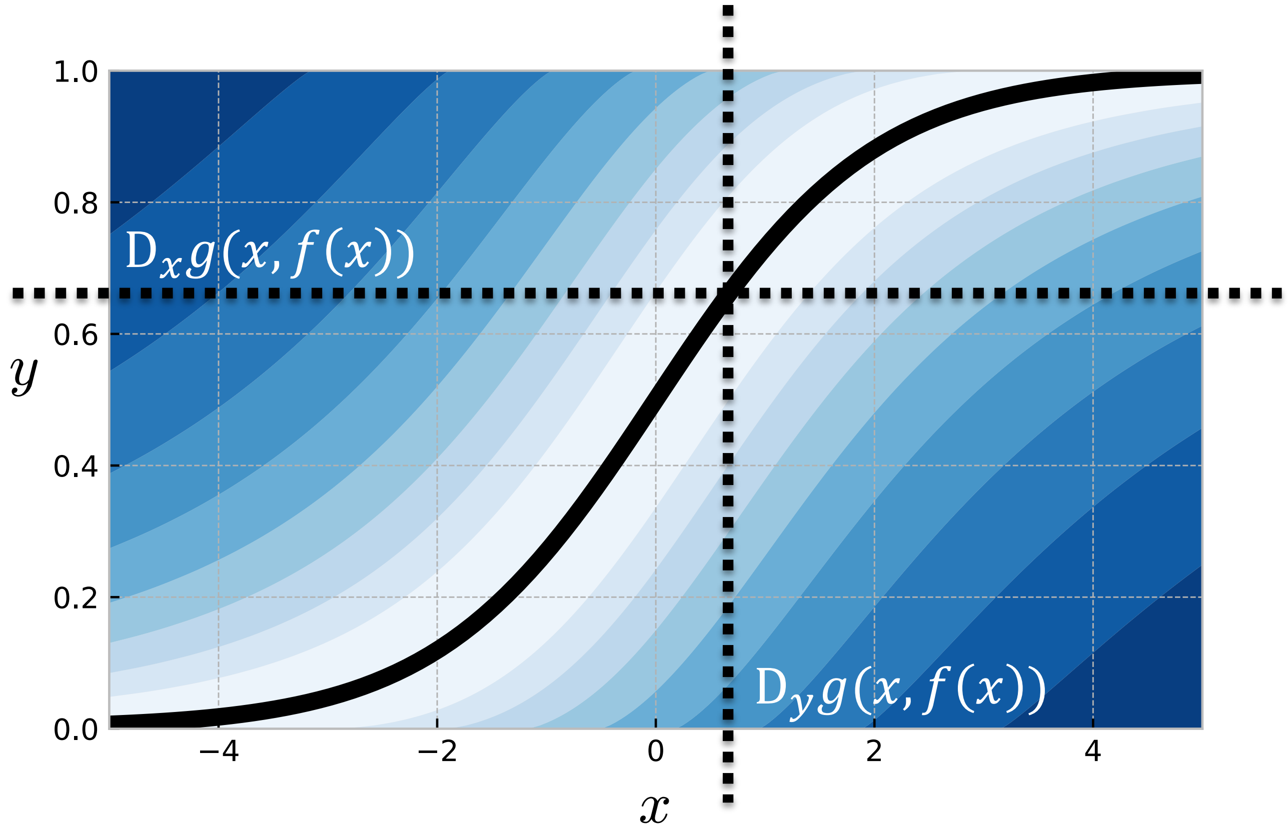
Given $g(x, y)$ and $f(x) = g(x, y')$, where $y' \in \{y: g(x, y) = 0\}$

How can we compute $\mathrm{D}_x f(x)$?

The **Implicit Function Theorem** gives

$$\mathrm{D}_x f(x) = -\mathrm{D}_y g\big(x, f(x)\big)^{-1} \mathrm{D}_x g\big(x, f(x)\big)$$

under mild assumptions



Implicitly Differentiating a Quadratic Program

[OptNet] We only consider convex QPs

$$x^\star = \underset{x}{\operatorname{argmin}} \; \frac{1}{2} x^\top Q x + p^\top x$$
$$\text{subject to } \; Ax = b \quad Gx \leq h$$

[KKT Optimality]

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

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

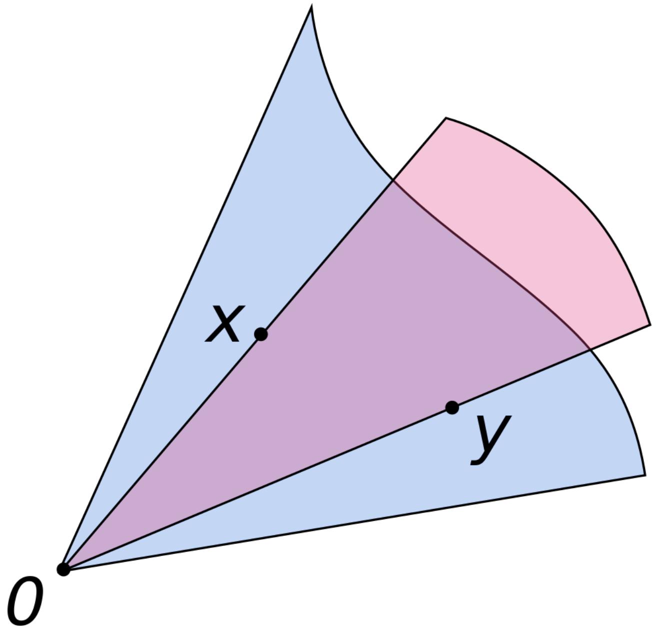
Cones and Conic Programs

Most convex optimization problems can be transformed into a (convex) conic program

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

Zero: $\{0\}$
Free: $\mathbb{R}^n$
Non-negative: $\mathbb{R}^n_+$
Second-order (Lorentz): $\{(t, x) \in \mathbb{R}_+ \times \mathbb{R}^n \,|\, \|x\|_2 \le t\}$
Semidefinite: $\mathbb{S}^n_+$
Exponential: $\{(x, y, z) \in \mathbb{R}^3 \,|\, y e^{x/y} \le z, y > 0\} \cup \mathbb{R}_- \times \{0\} \times \mathbb{R}_+$

Cartesian Products: $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_p$



Implicitly Differentiating a Conic Program

[e.g. S7 of my thesis]

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

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

Some Applications

Learning **hard constraints** (Sudoku from data)

Modeling **projections** (ReLU, sigmoid, softmax; differentiable top-k, and sorting)

Game theory (differentiable equilibrium finding)

RL and control (differentiable control-based policies)

Meta-learning (differentiable SVMs)

Energy-based learning and **structured prediction** (differentiable inference)

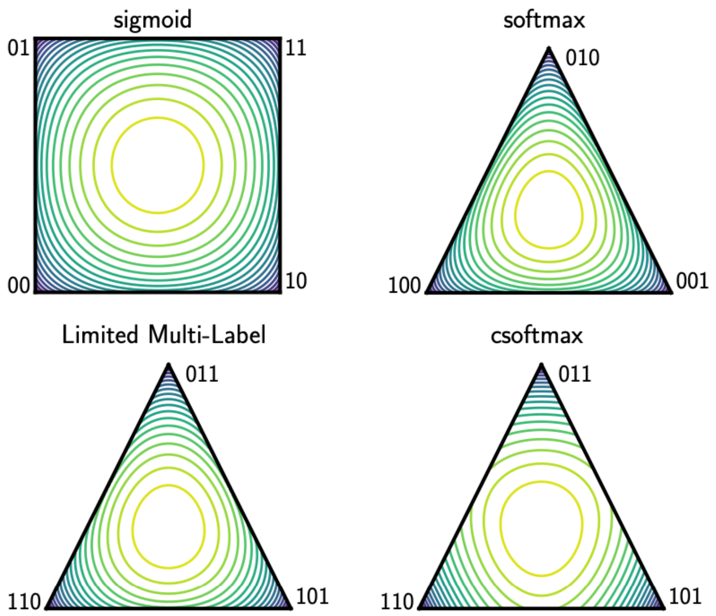
From the softmax to soft/differentiable top-k

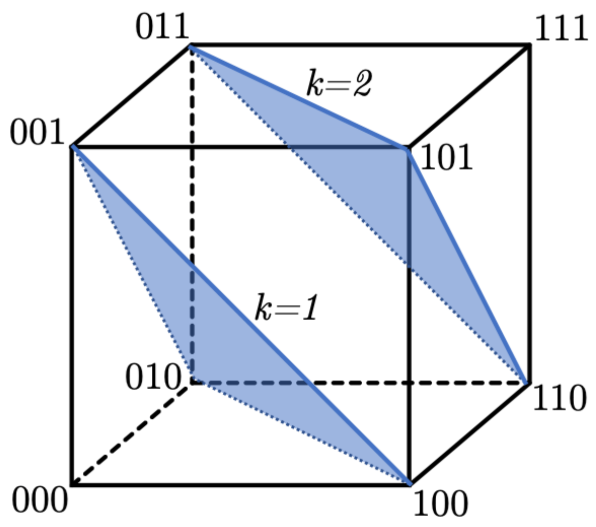
[Constrained softmax, constrained sparsemax, Limited Multi-Label Projection]

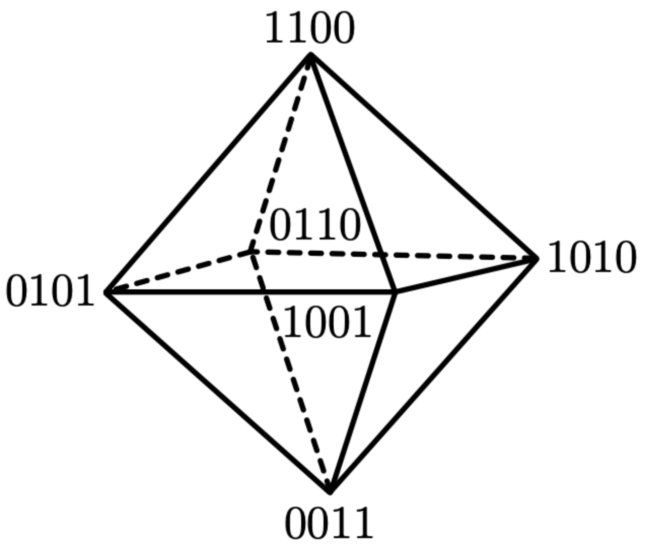
Vision application: End-to-end learn the top-k recall or predictions

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

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







Optimization layers need to be carefully implemented

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

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

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

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

$$
\begin{aligned}
\nabla_Q\ell &= \frac{1}{2}(d_z z^T + z d_z^T) & \nabla_q\ell &= d_z \\
\nabla_A\ell &= d_\nu z^T + \nu d_z^T & \nabla_b\ell &= -d_\nu \\
\nabla_G\ell &= D(\lambda^\star)(d_\lambda z^T + \lambda d_z^T) & \nabla_h\ell &= -D(\lambda^\star)d_\lambda
\end{aligned}
$$

```
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)
```

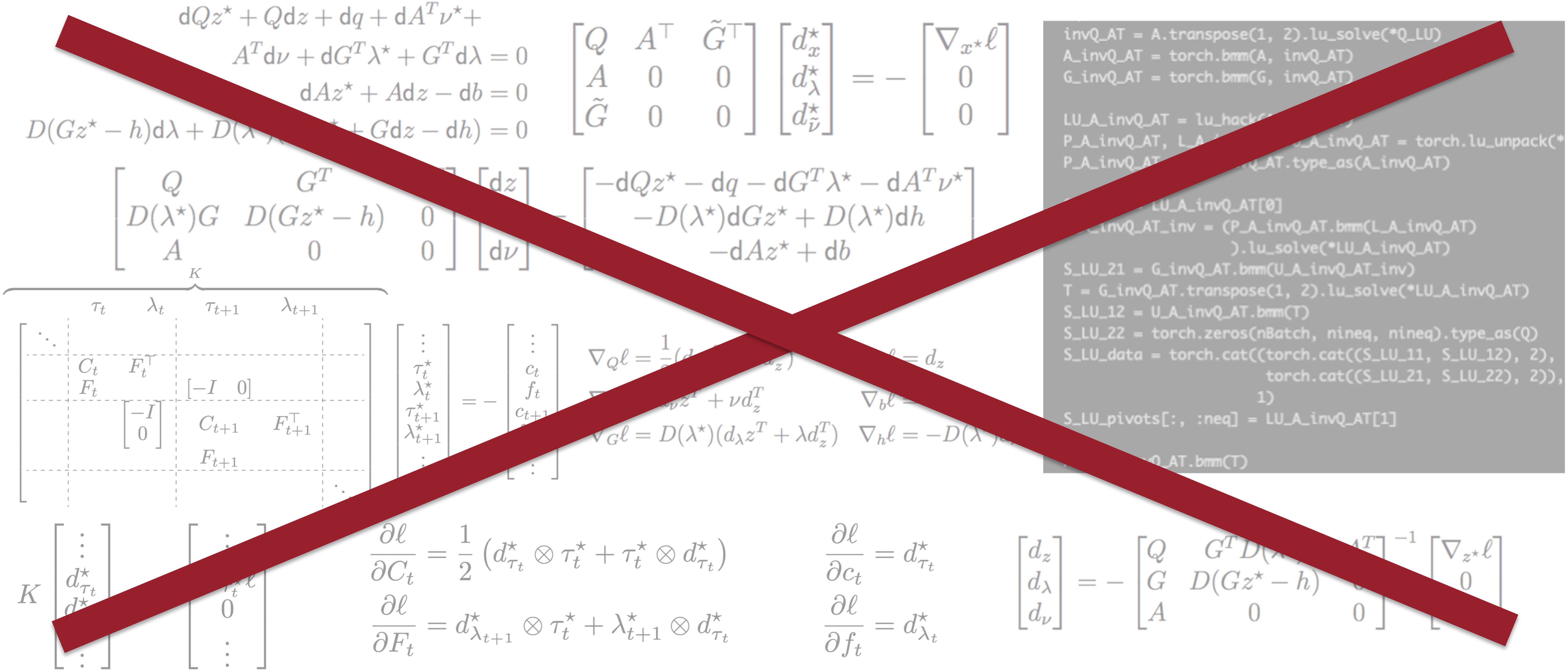
$$
K\begin{bmatrix} \vdots \\ d_{\tau_t}^\star \\ d_{\lambda_t}^\star \\ \vdots \end{bmatrix}
= -\begin{bmatrix} \vdots \\ \nabla_{\tau_t^\star}\ell \\ 0 \\ \vdots \end{bmatrix}
$$

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

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

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

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

CVXPY

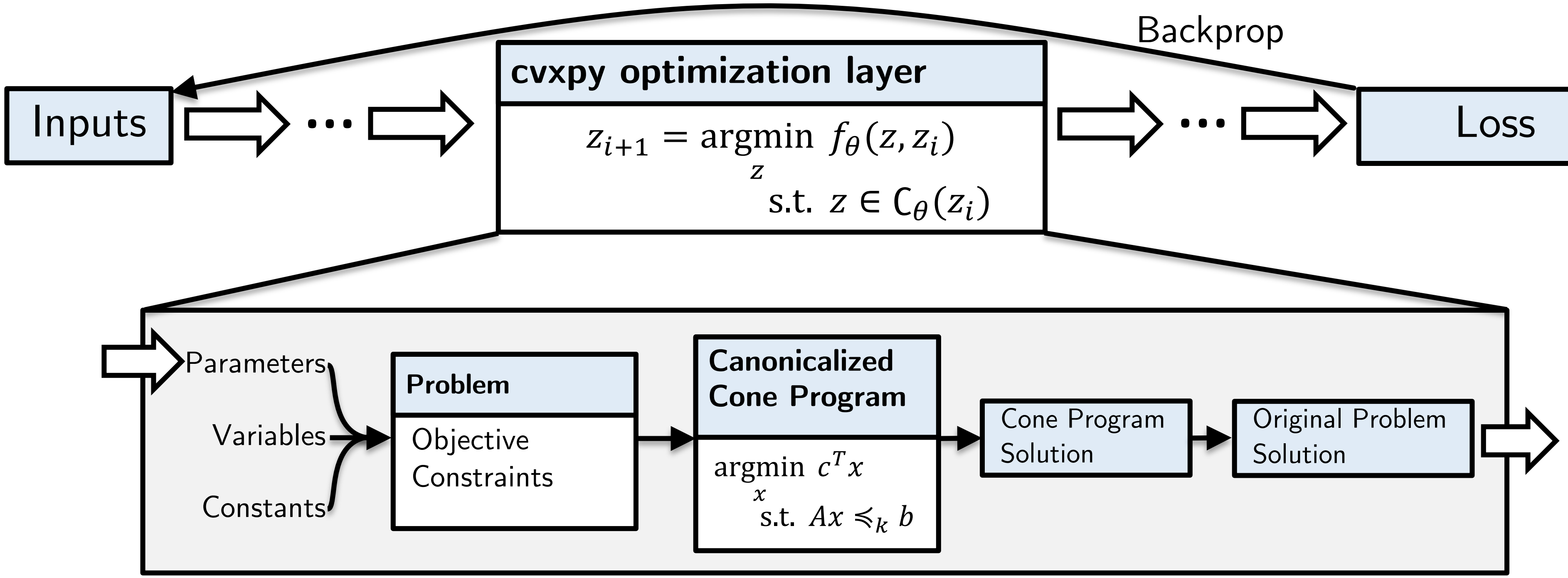
$$\begin{aligned} x^\star(\theta) = \ & \underset{x}{\operatorname{argmin}} \ f(x;\theta) \\ & \text{subject to} \ g(x;\theta) \leq 0 \\ & \qquad\qquad\ \ h(x;\theta) = 0 \end{aligned}$$

→ PyTorch

→ TensorFlow

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

A new way of rapidly prototyping optimization layers

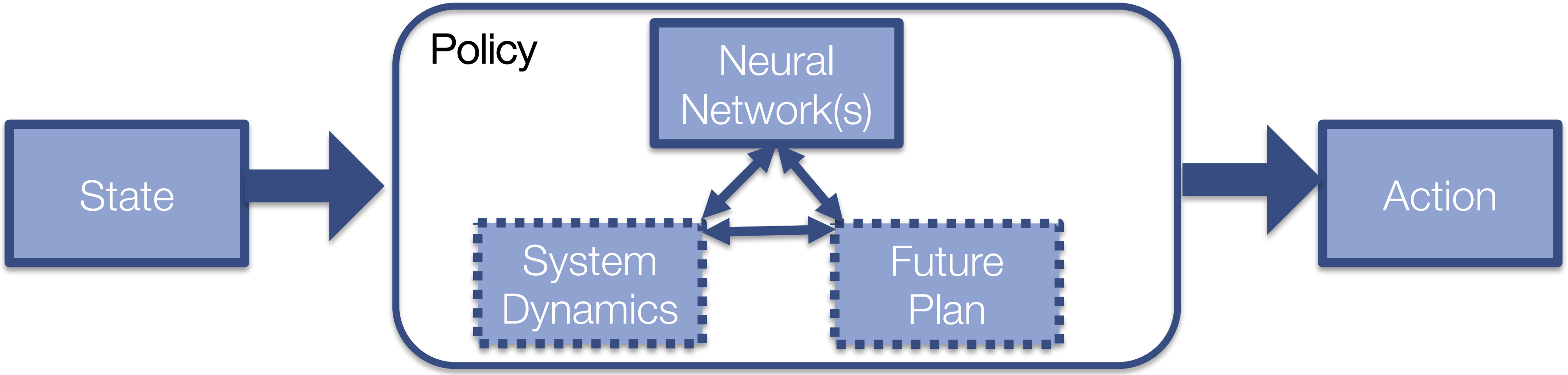


This Talk

Foundation: Differentiable convex optimization

Differentiable continuous control
- Differentiable model predictive control
- Differentiable cross-entropy method

Should RL policies have a system dynamics model or not?



Model-free RL

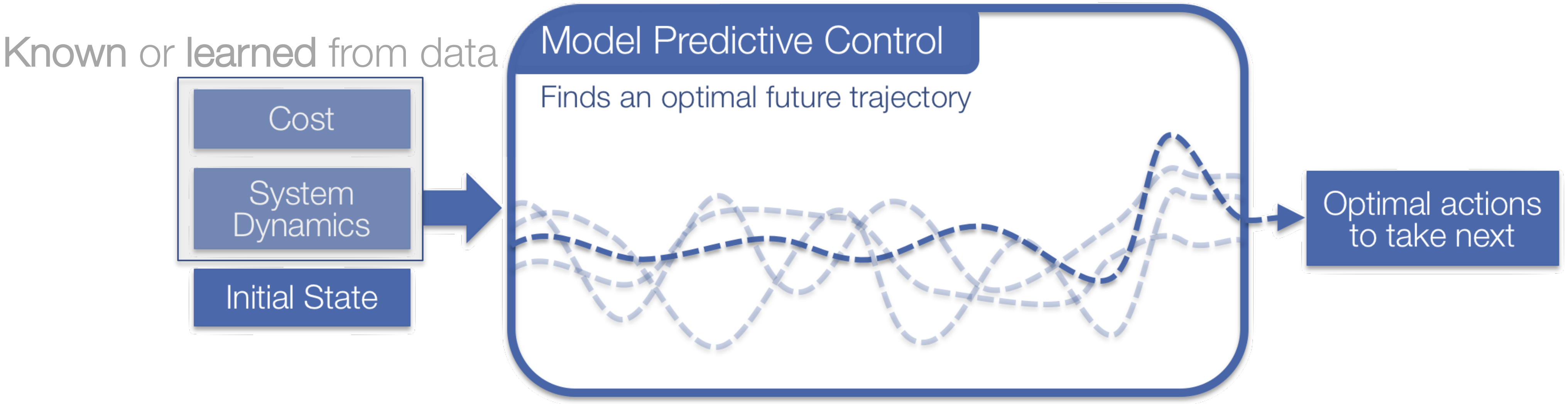
More general, doesn't make as many assumptions about the world

Rife with poor data efficiency and learning stability issues

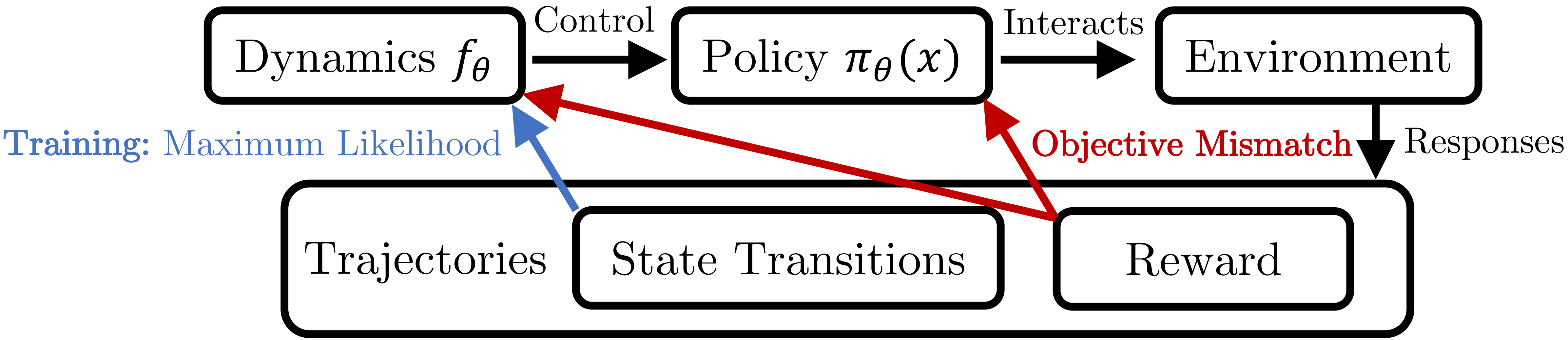
Model-based RL (or control)

A useful prior on the world if it lies within your set of assumptions

Model Predictive Control



The Objective Mismatch Problem



Differentiable Model Predictive Control

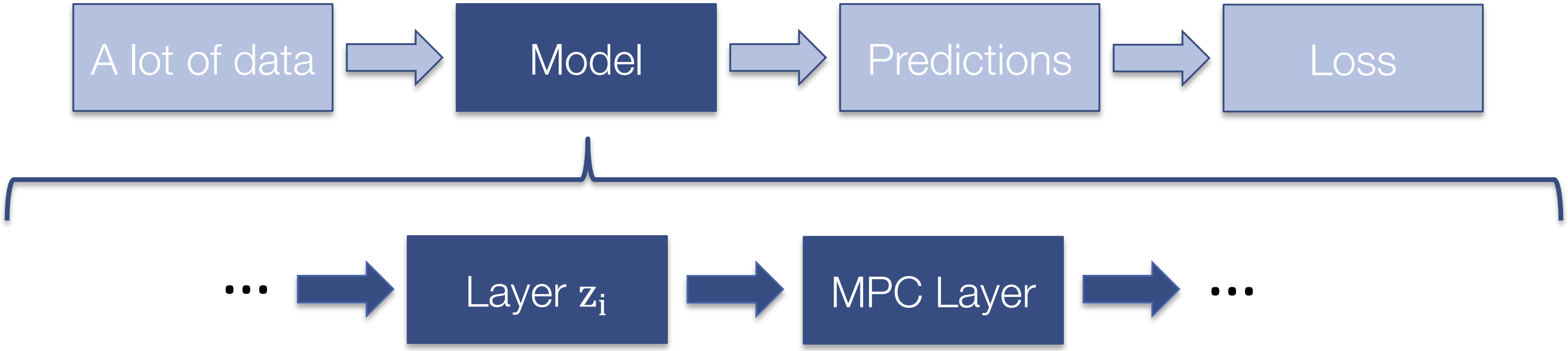
A pure **planning problem** given (potentially non-convex) **cost** and **dynamics**:

$$
\begin{aligned}
\tau_{1:T}^{\star} = \operatorname*{argmin}_{\tau_{1:T}} \; & \sum_t \boxed{C_\theta(\tau_t)} \text{ Cost} \\
\text{subject to } \; & x_1 = x_{\text{init}} \\
& x_{t+1} = \boxed{f_\theta(\tau_t)} \text{ Dynamics} \\
& \underline{u} \le u \le \overline{u}
\end{aligned}
$$

where $\tau_t = \{x_t, u_t\}$

Idea: Differentiate through this optimization problem

Differentiable Model Predictive Control



What can we do with this now?

Augment neural network policies in model-free algorithms with MPC policies
Replace the unrolled controllers in other settings (hindsight plan, universal planning networks)
Fight objective mismatch by end-to-end learning dynamics
The cost can also be end-to-end learned! No longer need to hard-code in values

Approach 1: Differentiable MPC/iLQR

Can **differentiate through the chain of QPs** or **just the last one** if it's a fixed point

…

QP Iterate i

$$\tau_{1:T}^{i} = \operatorname*{argmin}_{\tau_{1:T}} \sum_{t} \tilde{C}_{\theta}^{i}(\tau_t)$$

$$\text{subject to} \quad x_1 = x_{init}$$

$$x_{t+1} = \tilde{f}_{\theta}^{i}(\tau_t)$$

$$\underline{u} \leq u \leq \overline{u}$$

…

Differentiating LQR with LQR

Solving **LQR** with **dynamic Riccati recursion** efficiently solves the **KKT system**

$$\overbrace{\begin{array}{cccccc} & \tau_t & \lambda_t & \tau_{t+1} & \lambda_{t+1} & \\ \end{array}}^{K}$$

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

Backwards Pass: Implicitly **differentiate** the LQR KKT conditions:

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

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

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

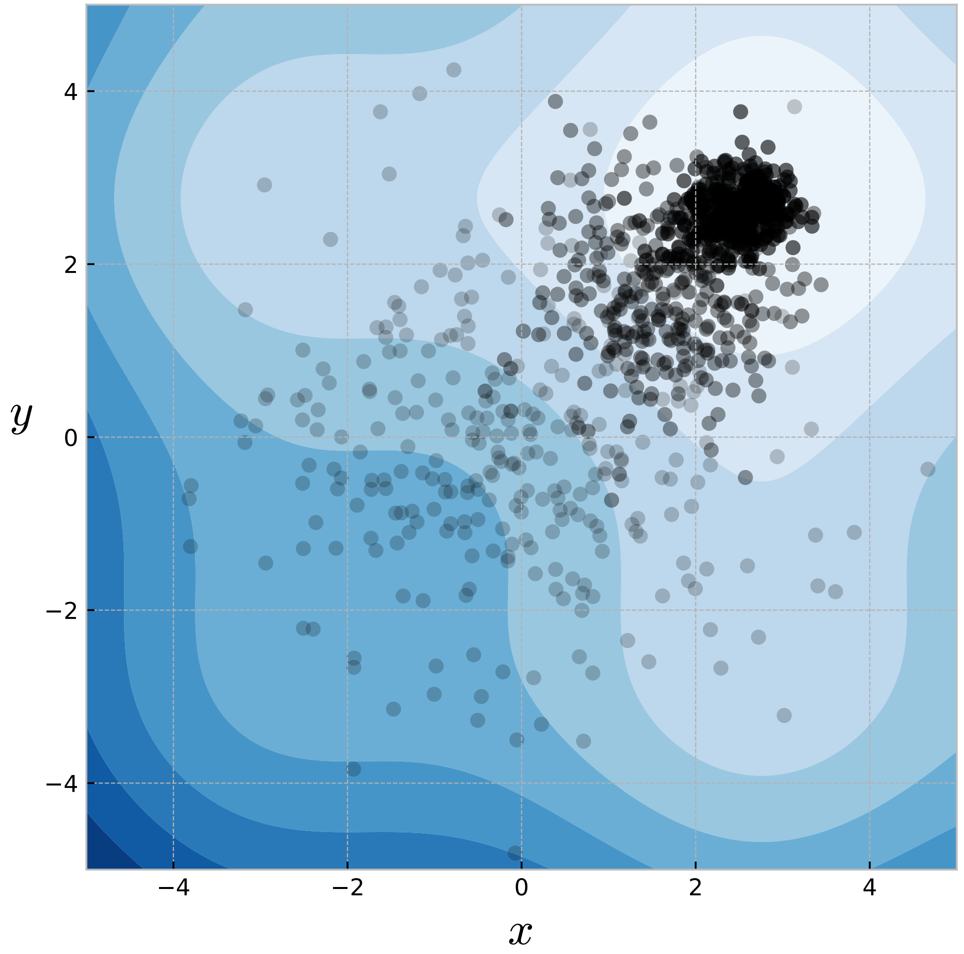
Just another LQR problem!

Approach 2: The Cross-Entropy Method

Iterative sampling-based optimizer that:

1. **Samples** from the domain
2. **Observes** the function's values
3. **Updates** the sampling distribution

SOTA optimizer for **control** and **model-based RL**

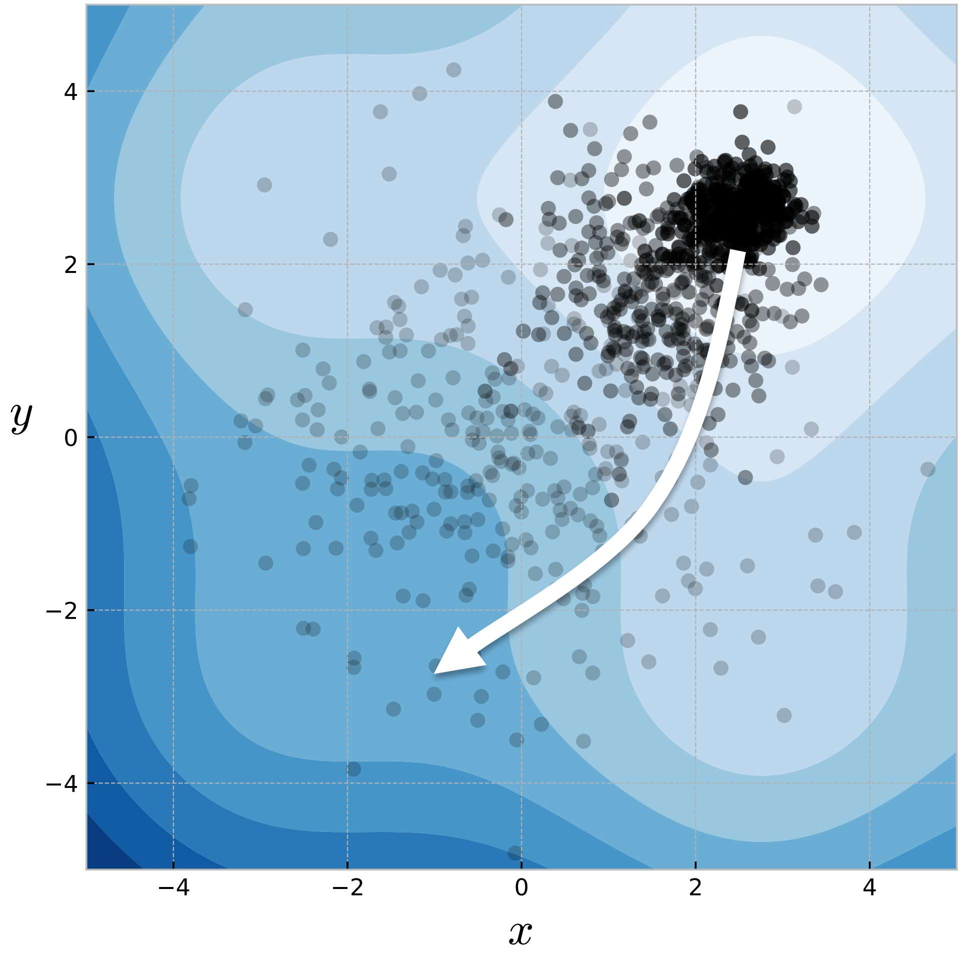


The Differentiable Cross-Entropy Method (DCEM)

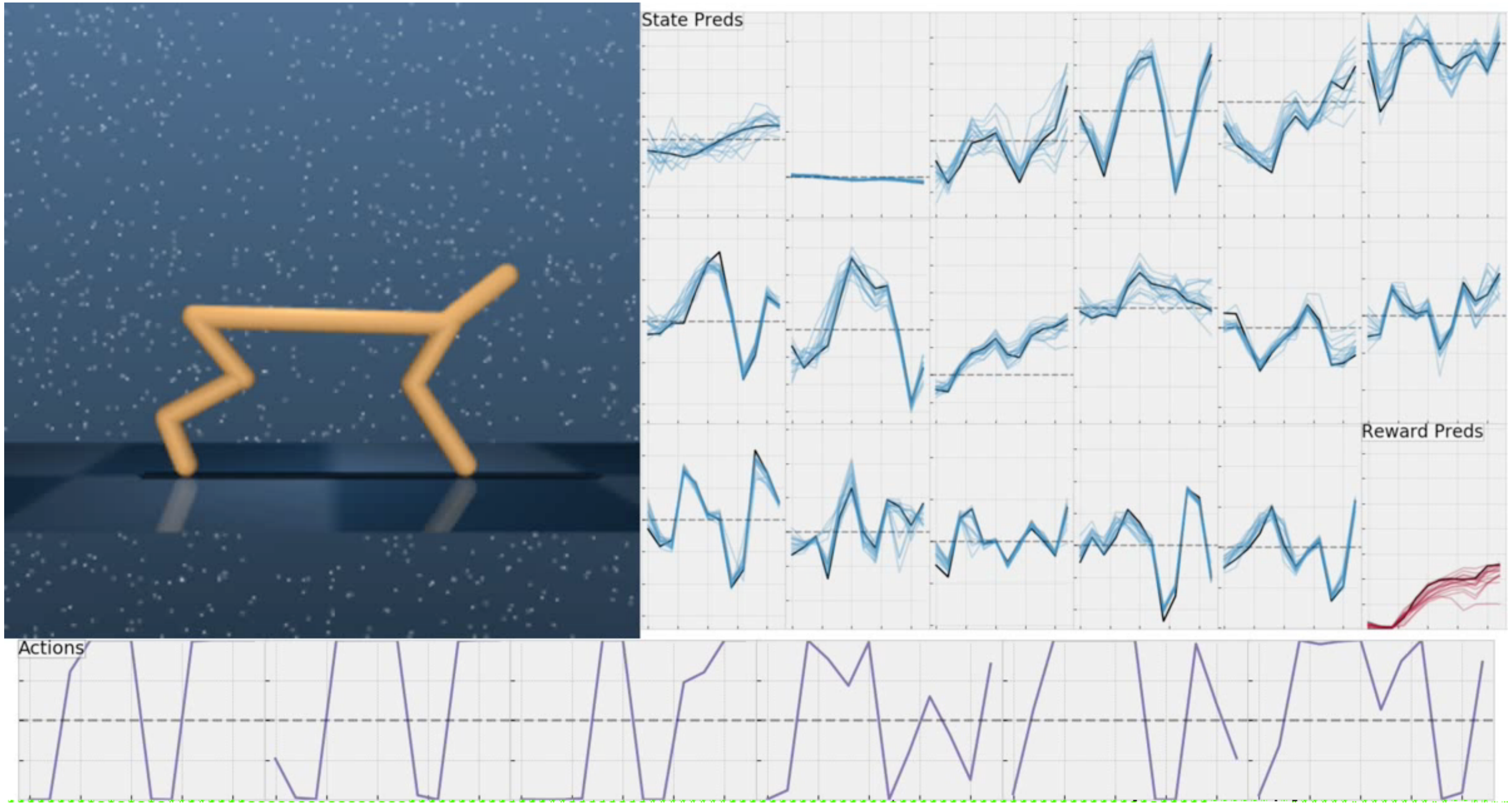
Differentiate backwards through the sequence of samples
- Using **differentiable top-k** (LML) and **reparameterization**

Useful when a fixed point is **hard to find,** or when unrolling gradient descent hits a local optimum

A **differentiable controller** in the RL setting



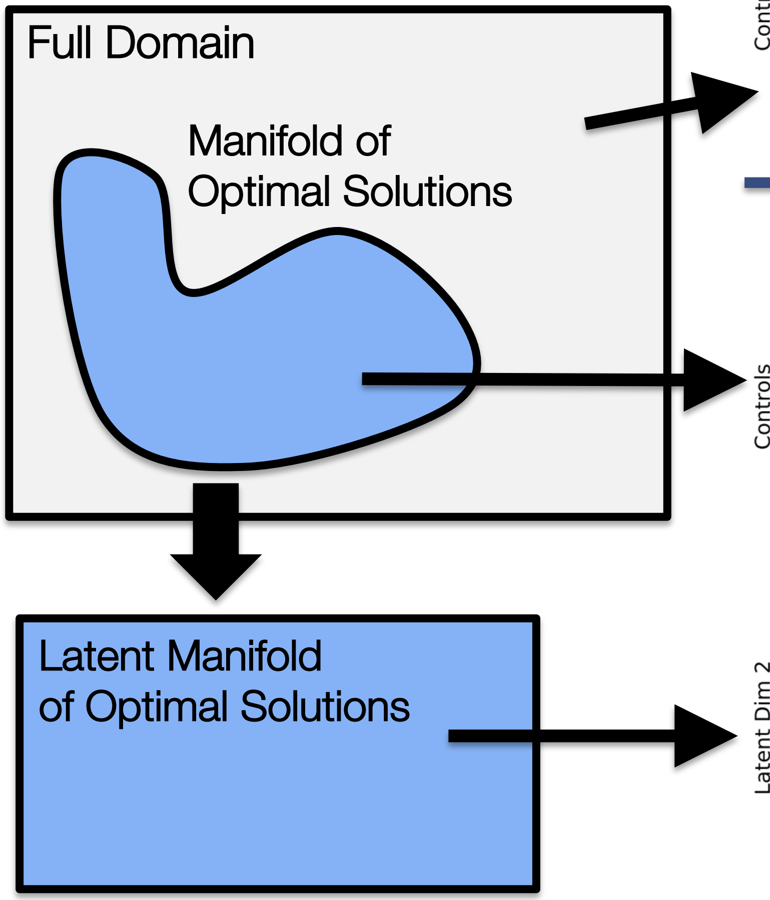
DCEM fine-tunes highly non-convex controllers

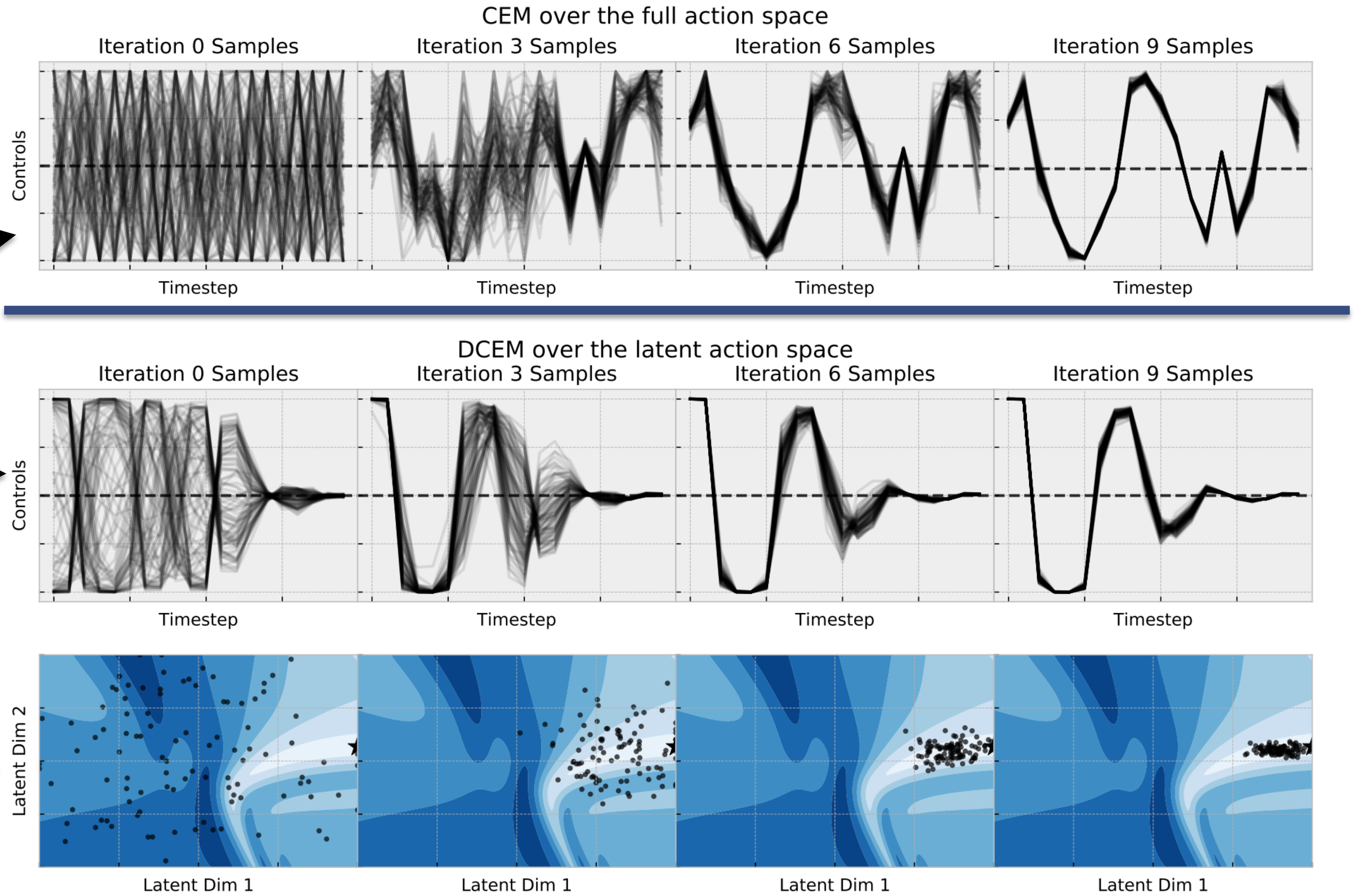


sites.google.com/view/diff-cross-entropy-method

DCEM can exploit the solution space structure

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





Differentiable Optimization-Based Modeling and Continuous Control

Brandon Amos • Facebook AI Research

brandondamos
bamos.github.io

- Differentiable QPs: OptNet [ICML 2017]
- Differentiable Stochastic Opt: Task-based Model Learning [NeurIPS 2017]
- Differentiable MPC for End-to-end Planning and Control [NeurIPS 2018]
- Differentiable Convex Optimization Layers [NeurIPS 2019]
- Differentiable Optimization-Based Modeling for ML [Ph.D. Thesis 2019]
- Differentiable Top-k and Multi-Label Projection [arXiv 2019]
- Objective Mismatch in Model-based Reinforcement Learning [L4DC 2020]
- Differentiable Cross-Entropy Method [ICML 2020]

Joint with Akshay Agrawal, Shane Barratt, Byron Boots, Stephen Boyd, Roberto Calandra, Steven Diamond, Priya Donti, Ivan Jimenez, Zico Kolter, Vladlen Koltun, Nathan Lambert, Jacob Sacks, Omry Yadan, and Denis Yarats