Learning with differentiable and amortized optimization

Brandon Amos • Meta AI (FAIR) NYC

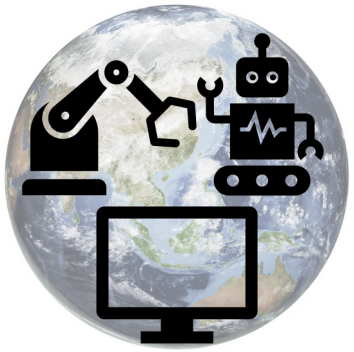
http://github.com/bamos/presentations

Optimization is crucial technology

Optimization is a **modeling** and **decision-making** paradigm and **encodes reasoning operations**

- Finds the **best way to interact** with a **representation of the world**

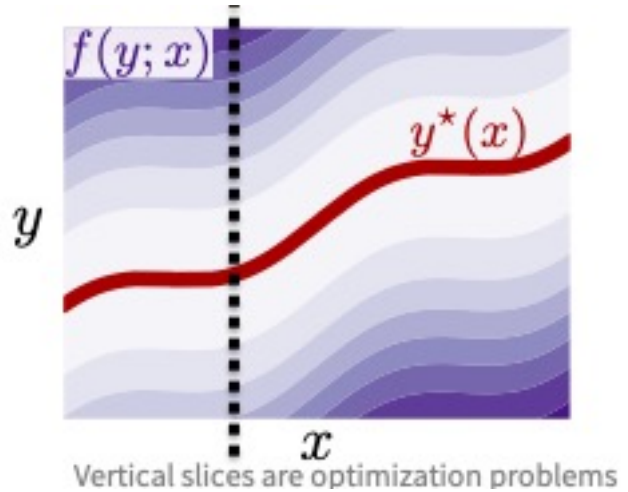
Focus: parametric optimization problems that are **repeatedly solved**



optimal solution — objective — context (or parameterization)

$$y^\star(x) \in \underset{y \in \mathcal{C}(x)}{\operatorname{argmin}} f(y; x)$$

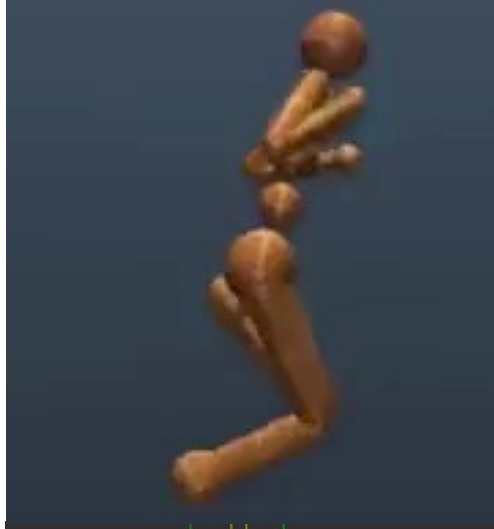
optimization variable — constraints

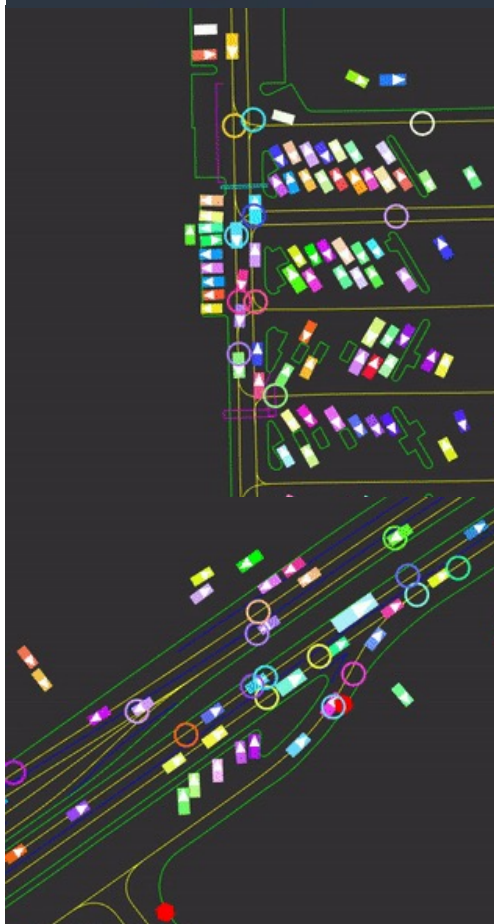


Vertical slices are optimization problems

Breakthroughs enabled by optimization

1. **controlling systems** (robotic, autonomous, mechanical, and multi-agent)
2. **making operational decisions** based on future predictions
3. efficiently **transporting** or **matching** resources, information, and measures for **optimal transport** and **generative modeling**
4. **allocating** budgets and portfolios,
5. **designing** materials, molecules, and other structures,
6. **solving inverse problems** (infers underlying hidden costs, incentives, geometries, terrains, and other structures)
7. **parameter learning** of predictive and statistical models

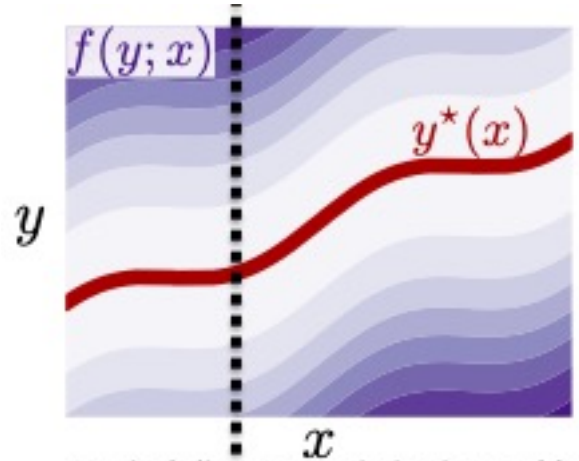




optimal solution — objective — context (or parameterization)

$$y^\star(x) \in \underset{y \in \mathcal{C}(x)}{\operatorname{argmin}} f(y; x)$$

optimization variable — constraints



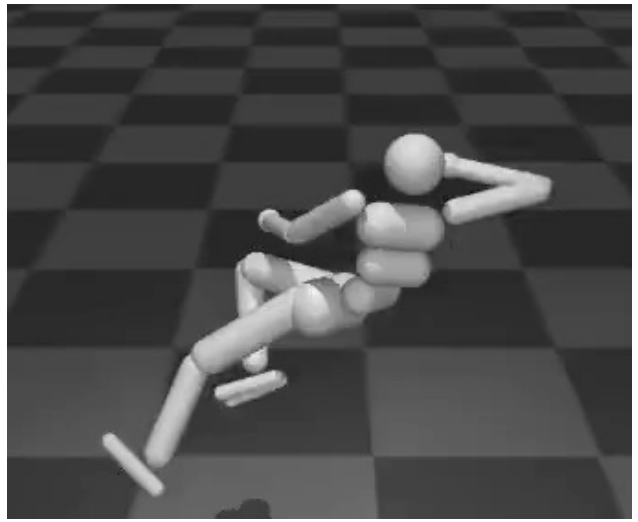
Vertical slices are optimization problems

When optimization fails, machine learning helps

$$y^\star(x) \in \underset{y \in \mathcal{C}(x)}{\operatorname{argmin}} f(y; x)$$

Optimization starts failing us when:

1. Analytically **encoding every detail** into f and $\mathcal{C}$ is impossible
   - May be **unknown, mis-specified,** or **inaccurate**
   - Especially difficult with **high-dimensional data** (text, images, video)
2. **Solving** for $y^\star(x)$ is **intractable** and **difficult to compute**

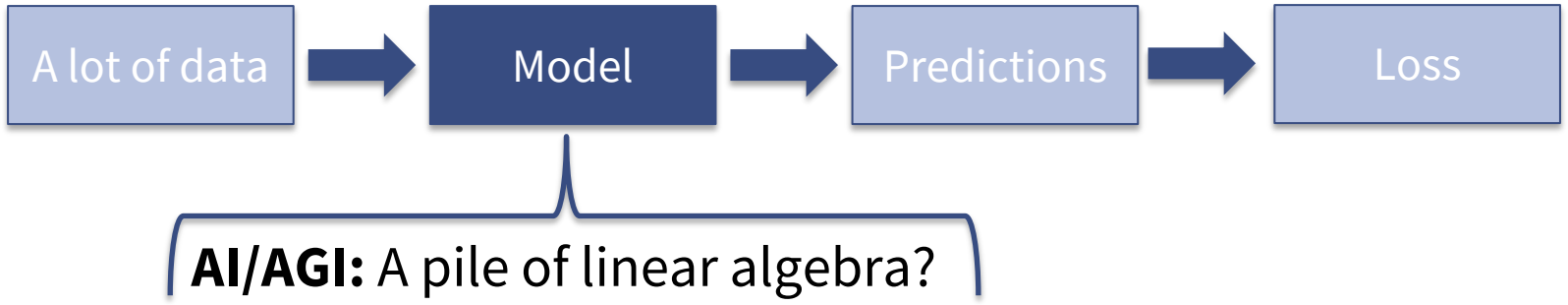


Machine learning systems are **part of the solution** and **excel** at:

1. **Modeling systems** we can extract a lot of data from
2. Extracting **meaningful latent representations**
3. Making **fast and accurate** predictions

When optimization fails, machine learning helps

When machine learning fails, optimization helps

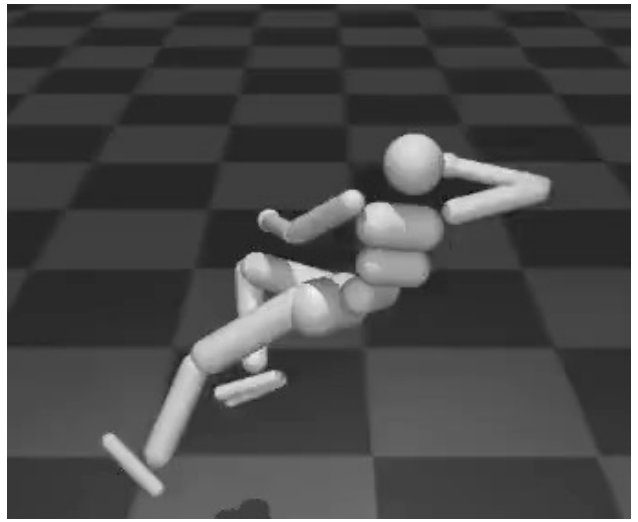


Is a **black-box of linear algebra** the **solution to AI/AGI**? *Maybe…*

Without enough data, learning systems struggle to:

1. **Encode hard constraints** and **rules** of the world crucial for safety
2. Perform **reasoning operations**
3. Isolate **interpretable** and **modular subcomponents**
4. **Generalize** to new settings beyond the training distribution

Optimization often **excels in these** and **can be brought into the model/loss**



Integrating optimization and machine learning

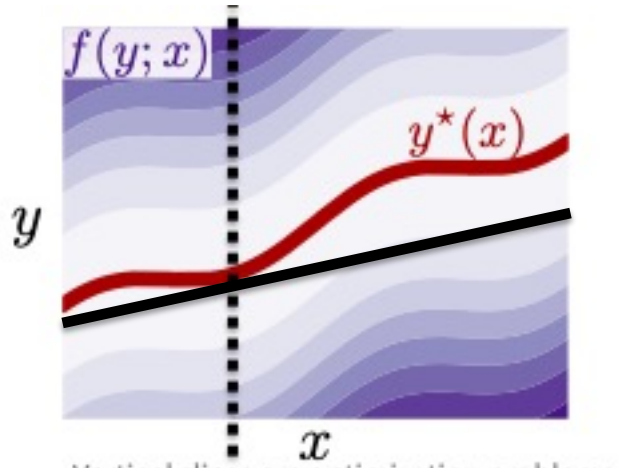
Key: view **optimization as a function** from the context x to the solution $y^\star(x) \in \underset{y \in \mathcal{C}(x)}{\operatorname{argmin}} f(y; x)$

Differentiable optimization — $\frac{\partial}{\partial x} y^\star(x)$

Optimization is a function that can be **differentiated**

Tells how **perturbing the context changes the solution**

Enables **optimization as a layer** or loss for ML



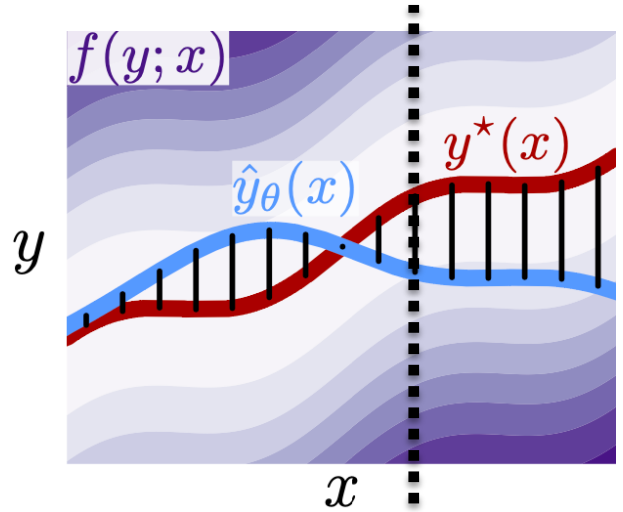
Vertical slices are optimization problems

Amortized optimization — $\hat{y}_\theta(x) \approx y^\star(x)$

Optimization is a function that can be **approximated**

Approximate the solution map with $\hat{y}_\theta$

Power of optimization, fast **speed of an ML model**



Vertical slices are optimization problems

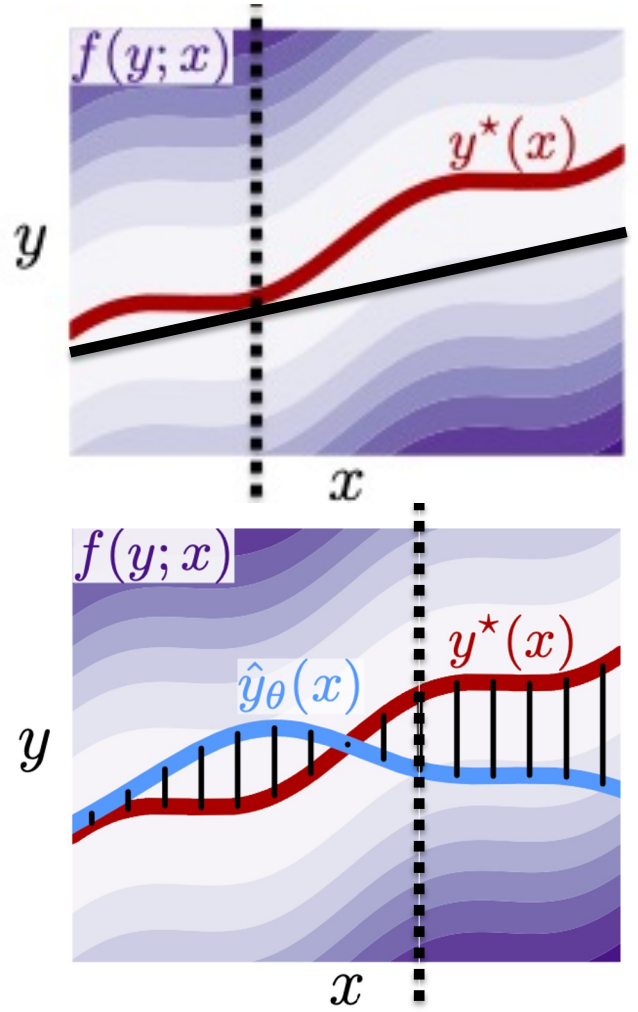
This talk

Differentiable optimization — $\frac{\partial}{\partial x} y^\star(x)$

Foundations
Differentiable convex optimization layers
Differentiable control
Optimization-based task losses

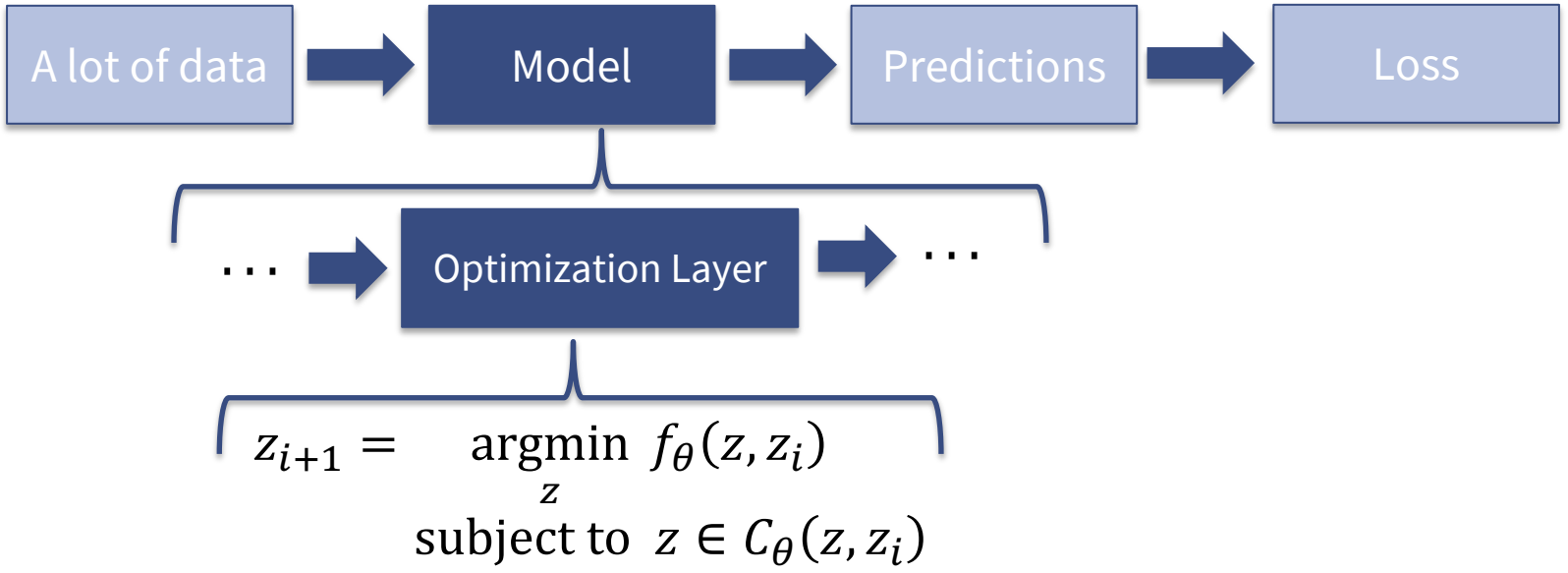
Amortized optimization — $\hat{y}_\theta(x) \approx y^\star(x)$

Foundations
RL as amortized optimization
Amortization via learning latent subspaces
Meta Optimal Transport
Amortizing convex conjugates



Differentiable optimization layers

- **Definition**. A **differentiable optimization layer** for a machine learning model internally solves an optimization problem
- **Why optimization?** Many **useful operations** can be expressed with optimization (sorting, permutations, top-k, control, planning, power generation, and others from earlier)
- **Why *differentiable* optimization?** End-to-end learn **through the optimization procedure** along with the other parts of the model using derivatives



Differentiable convex optimization layers

The **argmin** of a **convex** optimization problem is **non-convex** and **expressive**
Standard non-linearities to be seen as **solutions** to convex optimization problem

$$y^\star(x) = \underset{y}{\text{argmin}}\ f(y;x)\ \text{subject to}\ y \in C(x)$$

The ReLU is a convex optimization layer

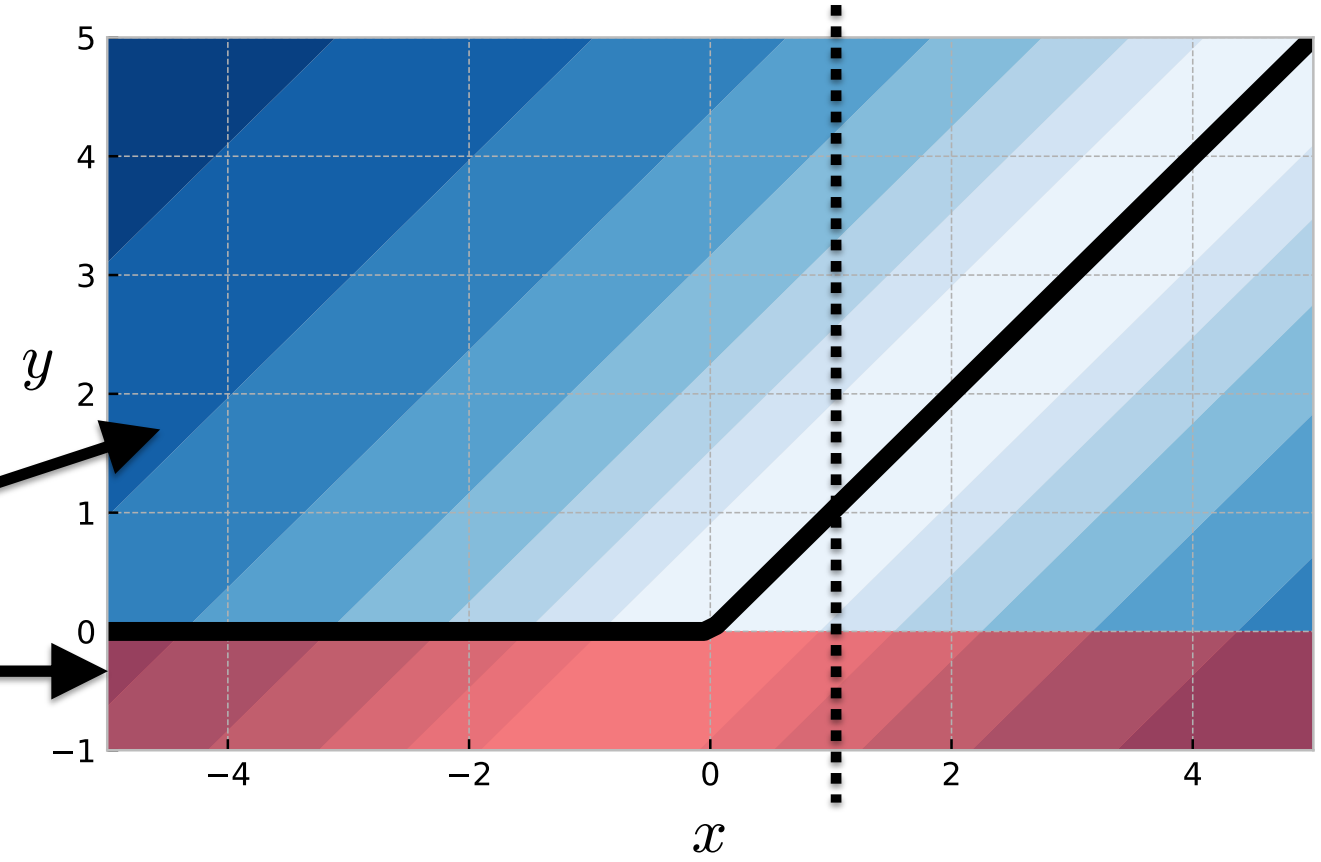
Section 2 of Differentiable optimization-based modeling for machine learning. Amos, PhD Thesis 2019

Proof: Comes from first-order optimality

$$\text{ReLU}(x) = \max\{0, x\}$$

$$\Updownarrow$$

$$\text{ReLU}(x) = \underset{y}{\text{argmin}} \quad \|y - x\|_2^2$$
$$\text{s.t.} \quad y \geq 0$$

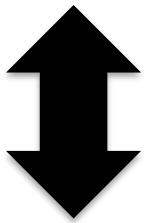


The sigmoid is a convex optimization layer

Section 2 of Differentiable optimization-based modeling for machine learning. Amos, PhD Thesis 2019

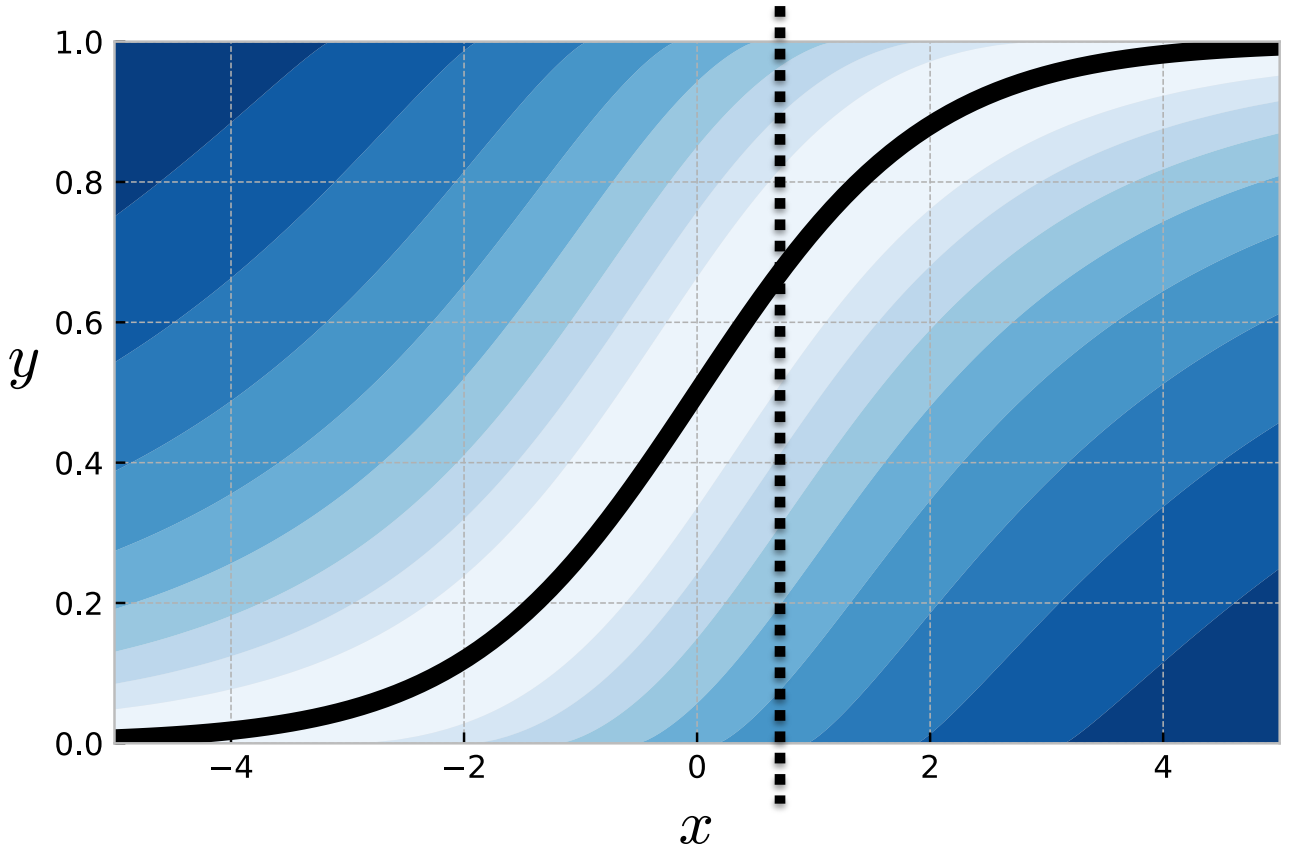
Proof: Comes from first-order optimality

$$\sigma(x) = \frac{1}{1 + \exp\{-x\}}$$



$$\sigma(x) = \underset{y}{\text{argmin}} \; -y^\top x - H_b(y)$$

$$\text{s.t.} \;\; 0 \leq y \leq 1$$



$H_b(y) \coloneqq -\sum_i (y_i \log y_i + (1 - y_i) \log(1 - y_i))$ is the binary cross-entropy function

The softargmax is a convex optimization layer

Section 2 of Differentiable optimization-based modeling for machine learning. Amos, PhD Thesis 2019

Proof: Comes from first-order optimality

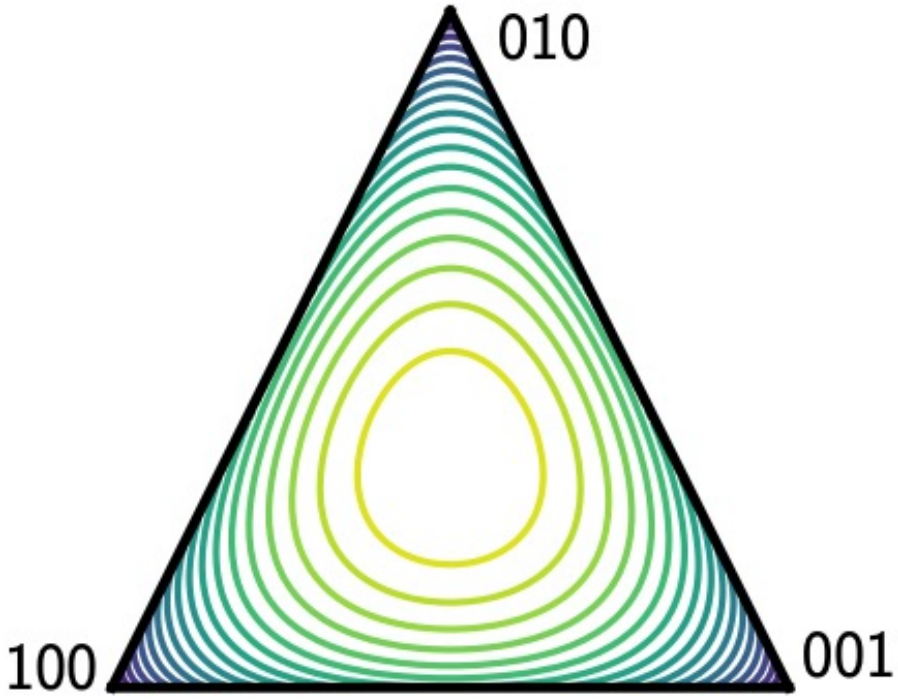
$$\pi_\Delta(x) = \frac{\exp x}{\Sigma_i \exp x_i}$$

$$\pi_\Delta(x) = \underset{y}{\operatorname{argmin}} \; -y^\top x - H(y)$$

$$\text{s.t. } 0 \leq y \leq 1$$

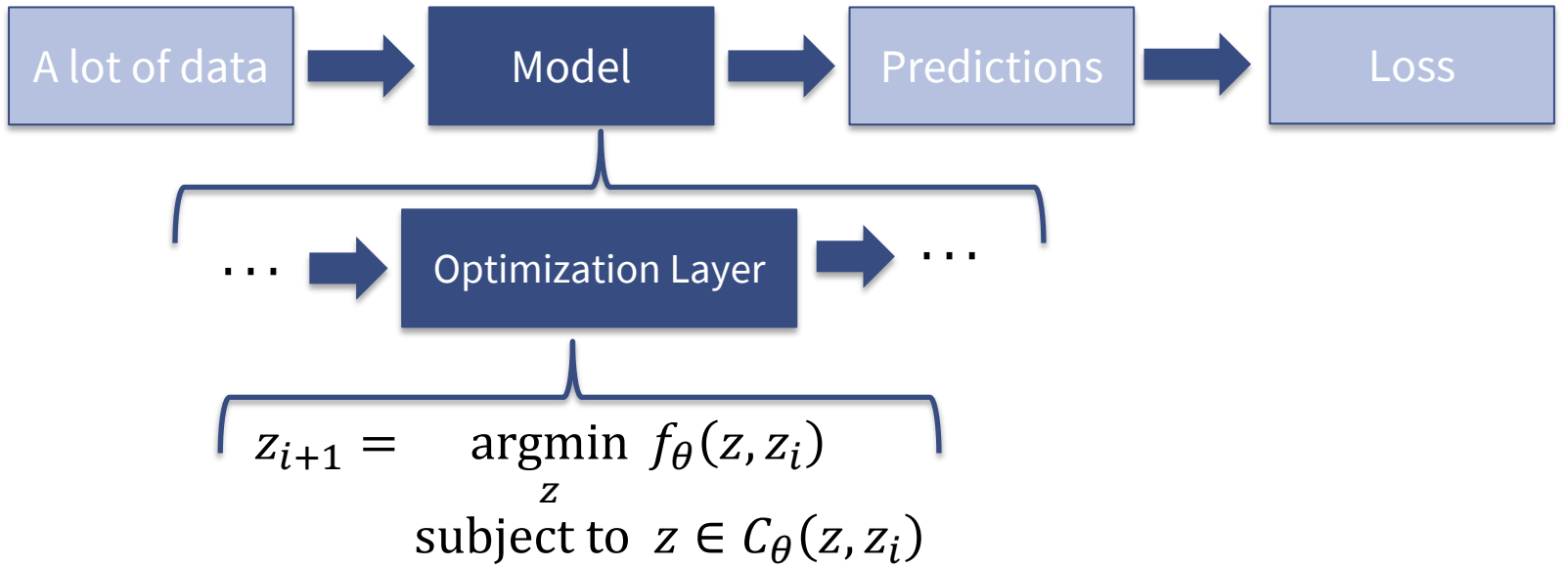
$$1^\top y = 1$$

$H(y) \coloneqq -\sum_i y_i \log y_i$ is the entropy function



Contours of the entropy $H(y)$ over the simplex

How can we generalize this?



Derivatives and backpropagation

For learning, we **differentiate** or backpropagate through these layers — **differentiable optimization**

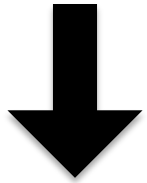
Easy if the optimization problem has an **explicit, closed-form solution** (often standard differentiation)

Otherwise, need to use **implicit differentiation,** which is also used for sensitivity analysis

Differentiable convex quadratic programs

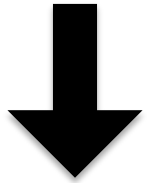
OptNet: Differentiable Optimization as a Layer in Neural Networks. Amos and Kolter, ICML 2017.

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



KKT Optimality

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



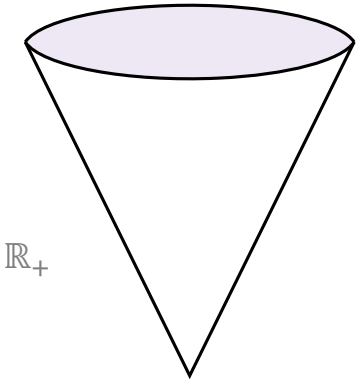
Implicitly differentiating $\mathcal{R}$ gives $D_\theta(z^\star) = -\left(D_z \mathcal{R}(z^\star)\right)^{-1} D_\theta \mathcal{R}(z^\star)$

Differentiable convex conic programs

Section 7 of Differentiable optimization-based modeling for machine learning. Amos, PhD Thesis 2019
Differentiating through a cone program. Agrawal et al., 2019
Differentiable convex optimization layers. Agrawal*, Amos*, Barratt*, Boyd*, Diamond*, Kolter*, NeurIPS 2019.

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

- **Zero:** $\{0\}$
- **Free:** $\mathbb{R}^n$
- **Non-negative:** $\mathbb{R}^n_+$
- **Second-order** (Lorentz): $\{(t, x) \in \mathbb{R}_+ \times \mathbb{R}^n \,|\, \|x\|_2 \leq t\}$
- **Semidefinite:** $\mathbb{S}^n_+$
- **Exponential:** $\{(x, y, z) \in \mathbb{R}^3 \,|\, ye^{x/y} \leq z, y > 0\} \cup \mathbb{R}_- \times \{0\} \times \mathbb{R}_+$
- **Cartesian Products:** $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_p$



Conic Optimality

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

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

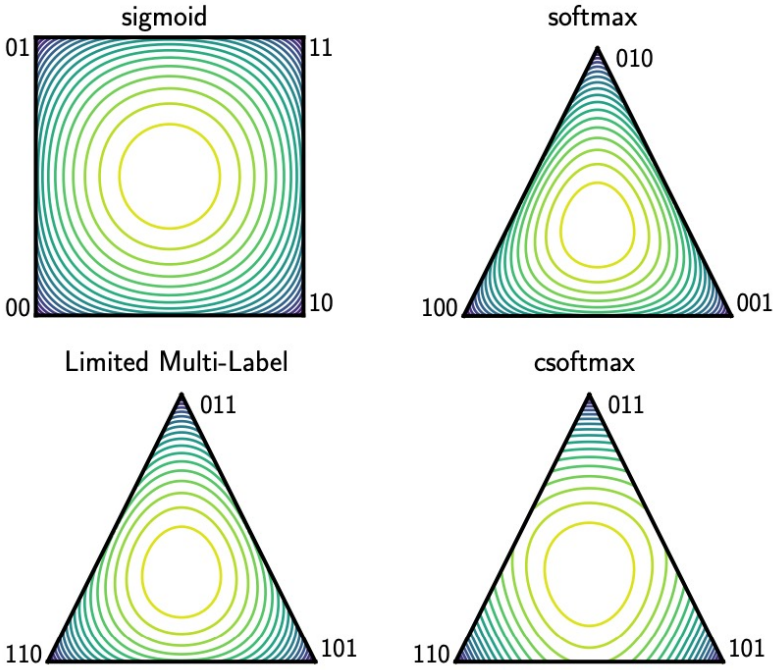
From the softmax to soft and differentiable top-k

Limited multi-label projection layer. Amos et al., 2019.

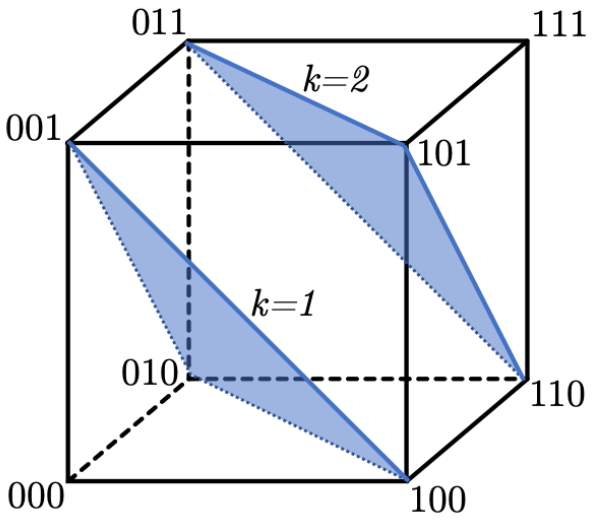
$$y^\star = \underset{y}{\text{argmin}} \; -y^\top x - H(y) \quad \text{subject to} \quad 0 \le y \le 1, \; 1^\top y = \boxed{1}$$

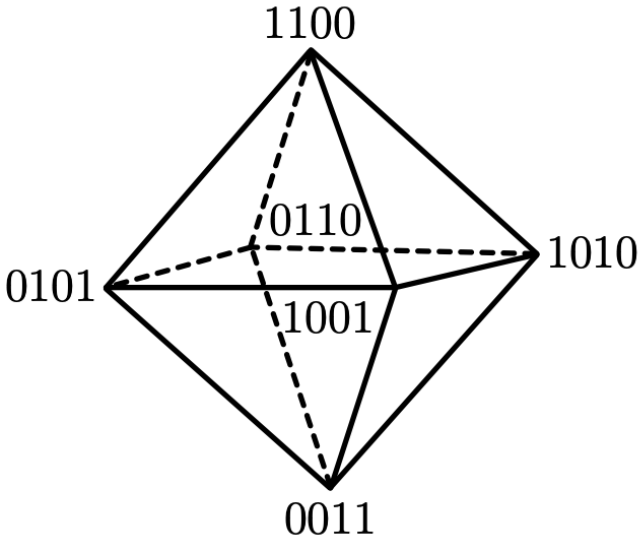
$$y^\star = \underset{y}{\text{argmin}} \; -y^\top x - H_b(y) \quad \text{subject to} \quad 0 \le y \le 1, \; 1^\top y = \boxed{k}$$

$H_b(y) \coloneqq -\sum_i (y_i \log y_i + (1-y_i)\log(1-y_i))$ is the binary cross-entropy function



Contours of the entropy penalties





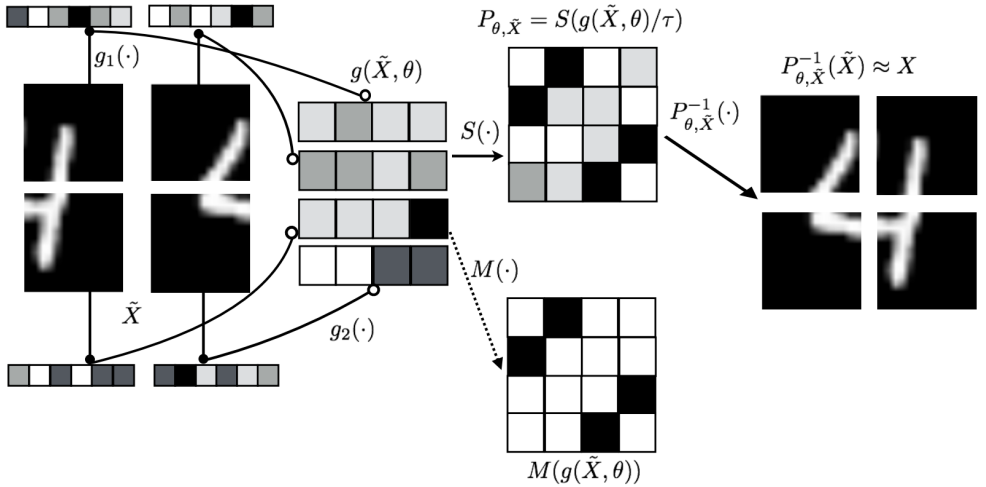
Differentiable permutations, sorting and SVMs

Differentiable permutations and sorting (Gumbel-Sinkhorn)
Projection onto the **Birkhoff polytope** $\mathcal{B}_N$**:**

$$S\left(\frac{X}{\tau}\right) = \underset{P \in \mathcal{B}_N}{\operatorname{argmax}} \ \langle P, X \rangle_F + \tau H(P)$$

$$\mathcal{B}_N = \left\{X : X \geq 0, \Sigma_i X_{ij} = \Sigma_j X_{ij} = 1\right\}$$

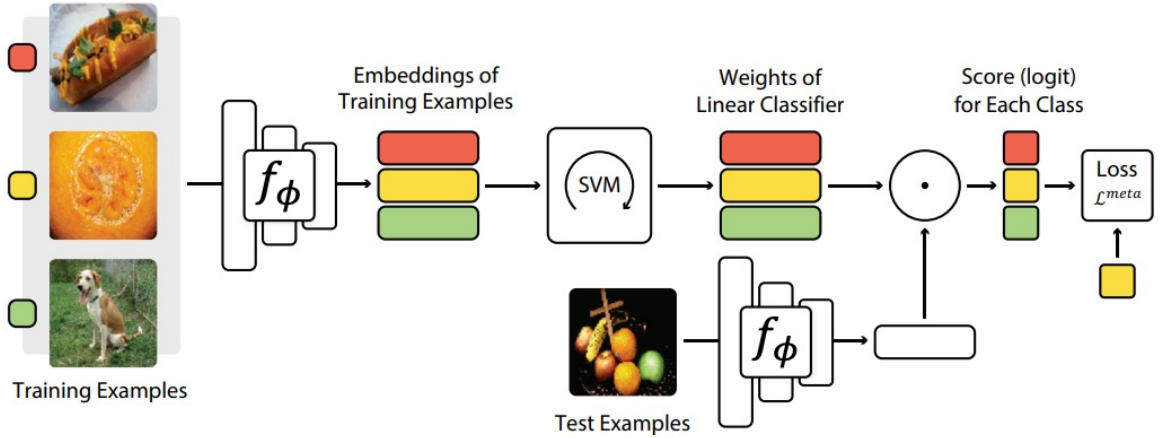
Learning latent permutations with Gumbel-Sinkhorn networks. Mena et al., ICLR 2018.



Differentiable SVMs (MetaOptNet)
Differentiate the decision boundary w.r.t. the dataset

$$w^\star(\mathcal{D}) = \underset{w}{\operatorname{argmin}} \ \|w\|^2 + C \sum_i \max\{0, 1 - y_i f(x_i)\}$$

Meta-learning with differentiable convex optimization. Lee et al., CVPR 2019.



Sensitivity and perturbation analysis

Differentiable convex optimization layers. Agrawal*, Amos*, Barratt*, Boyd*, Diamond*, Kolter*, NeurIPS 2019.

Adjoint derivatives for optimization problems have been studied for decades
We have focused on uses for learning, but also widely used for **sensitivity analysis**

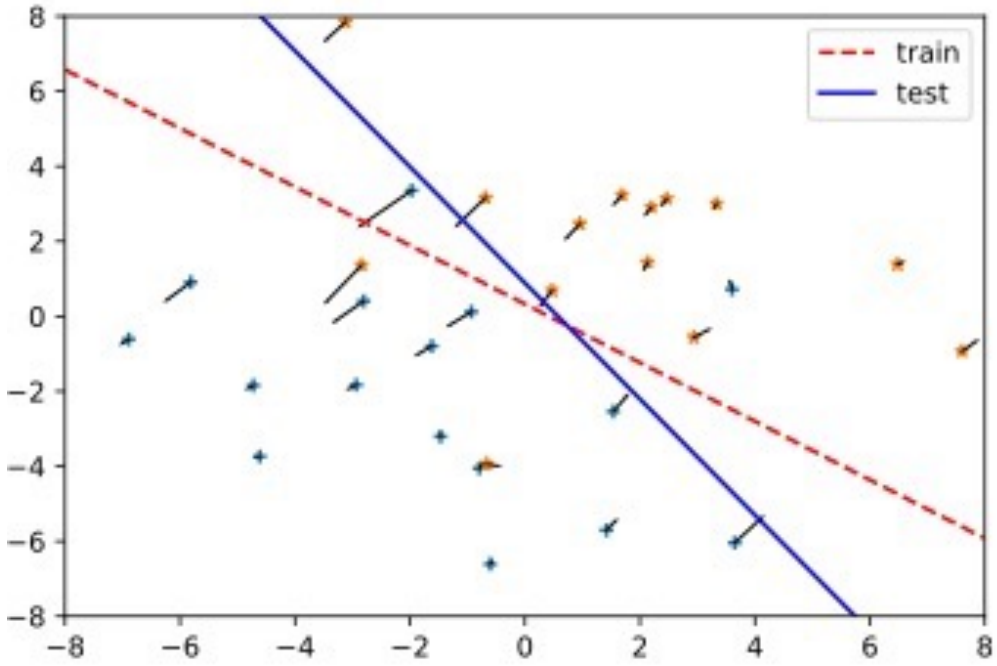
Logistic regression example
Find optimal decision boundary:

$$\theta^\star(\mathcal{D}) \in \operatorname*{argmax}_\theta \sum_i \log p_\theta(y_i \mid x_i)$$

$\theta^\star(\mathcal{D})$ is **just a function** of the data

Derivatives $\frac{\partial \theta^\star}{\partial x_i}$ provide **sensitivity** to the data points

- *Show how much the data impacts the decision boundary*



Optimal control drives systems

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

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

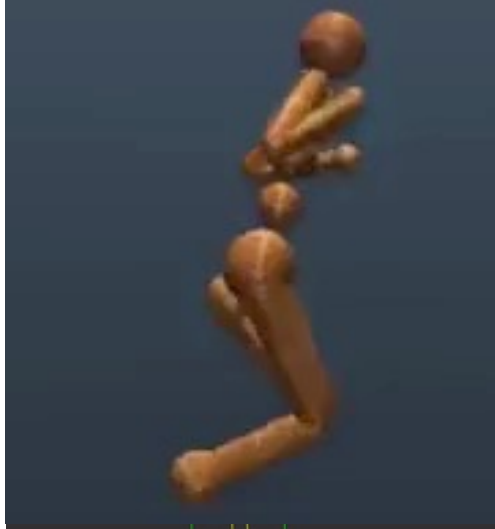
Full notation: $u^\star_{1:T}(x_{\text{init}}, f_\theta)$

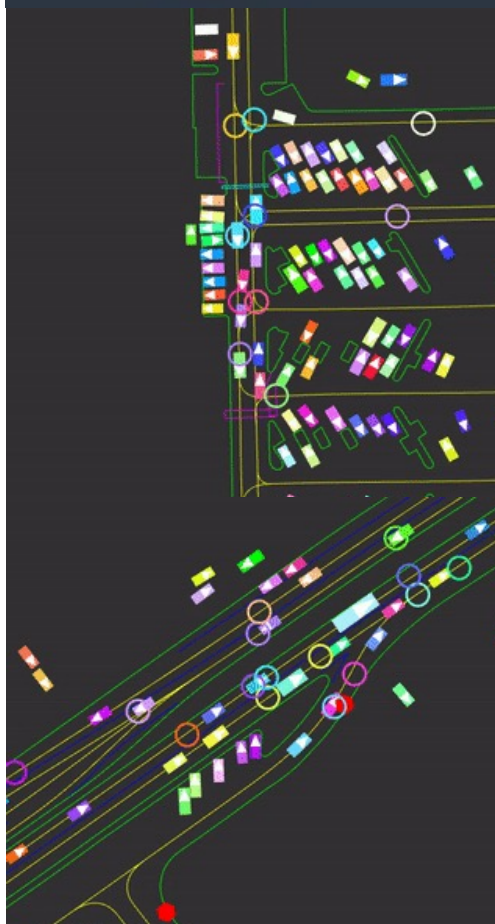
Widely deployed over the past century in aviation, robotics, vehicles, HVAC
Often for a **Markov decision process** but doesn't have to be

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

No learning necessary if we know the system — just pure optimization

On the model-based stochastic value gradient for continuous reinforcement. B. Amos et al., L4DC 2021
Nocturne: a driving benchmark for multi-agent learning. E. Vinitsky et al., NeurIPS Datasets and Benchmarks 2022





Differentiable control

Differentiable MPC for end-to-end planning and control. Amos et al., NeurIPS 2018. *The differentiable cross-entropy method.* Amos and Yarats, ICML 2020.

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

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

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

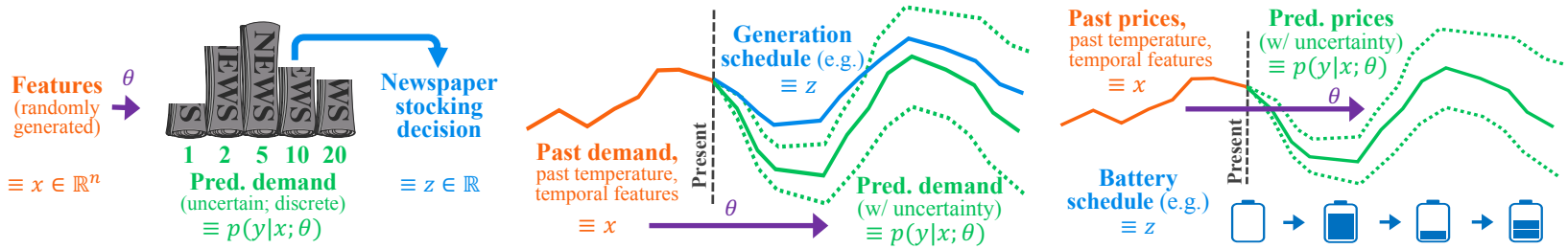
$\nabla_\theta \mathcal{J}(\pi_\theta)$

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

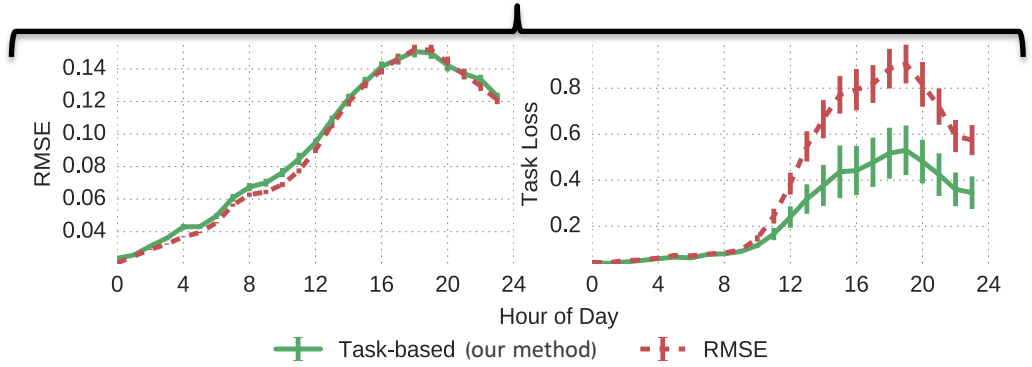
Optimization as a loss for downstream tasks

Task-based end-to-end model learning in stochastic optimization. Donti, Amos, and Kolter, NeurIPS 2017.

A model's predictions may be used in a **downstream optimization problem** (e.g., scheduling)
Challenge: models are inaccurate and **not aware of how errors impact the downstream problem**
Idea: use the **downstream optimizer as a loss** for the model using differentiable optimization



(a) Inventory stock problem (b) Load forecasting/generator scheduling (c) Price forecasting/battery arbitrage



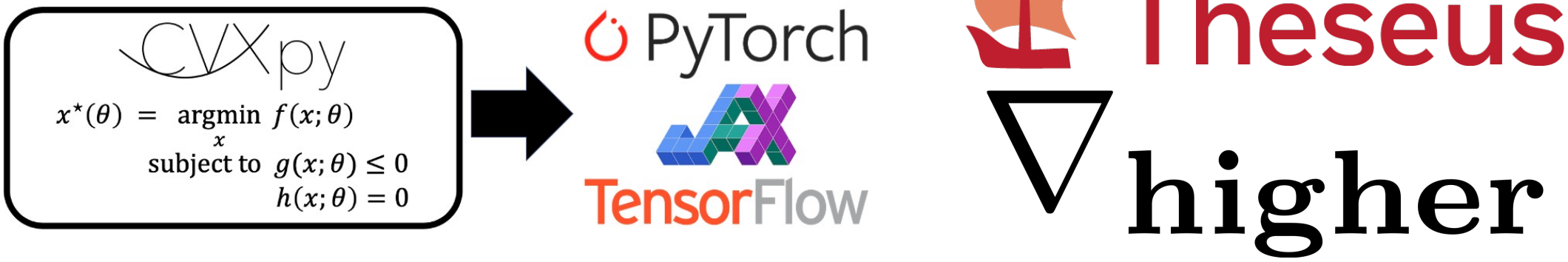
$$\underset{z \in \mathbb{R}^{24}}{\text{minimize}} \ \mathbf{E}_y \left[\sum_{i=1}^{24} \gamma_s [y_i - z_i]_+ + \gamma_e [z_i - y_i]_+ + \frac{1}{2} \|z_i - y_i\|^2 \right] \quad \text{subject to} \ |z_i - z_{i+1}| \leq c_{\text{ramp}}.$$

Implementing differentiable optimization

Manually: unroll or implicitly differentiate the optimality conditions (such as the KKT system)
My original differentiable QP implementation was **~1k lines of code** to make efficient

Use a software package to prototype such as:

- `cvxpylayers`: differentiable convex optimization in CVXPY [Agrawal*, Amos*, et al., NeurIPS 2019]
  Implements a differentiable QP in ~10 lines of code
- `Theseus`: differentiable non-linear least squares with a focus on robotics [Pineda et al., NeurIPS 2022]
- `higher`: differentiable unrolled optimizers [Grefenstette et al., 2019]

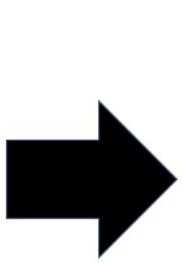


Differentiable CVXPY

Differentiable convex optimization layers. Agrawal*, Amos*, Barratt*, Boyd*, Diamond*, Kolter*, NeurIPS 2019.

CVXpy

$$x^\star(\theta) = \underset{x}{\text{argmin}}\ f(x;\theta)$$
$$\text{subject to}\ \ g(x;\theta) \leq 0$$
$$h(x;\theta) = 0$$



PyTorch

TensorFlow

(Officially part of CVXPY!)

$$z_{i+1} = \underset{z}{\text{argmin}}\ \frac{1}{2} z^\top Q(z_i) z + q(z_i)^\top z$$
$$\text{subject to}\ \ A(z_i) z = b(z_i)$$
$$G(z_i) z \leq h(z_i)$$

Parameters/Submodules : Q, q, A, b, G, h

Before: 1k lines of code, **now:**

```
import cvxpy as cp
from cvxpyth import CvxpyLayer
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])
```

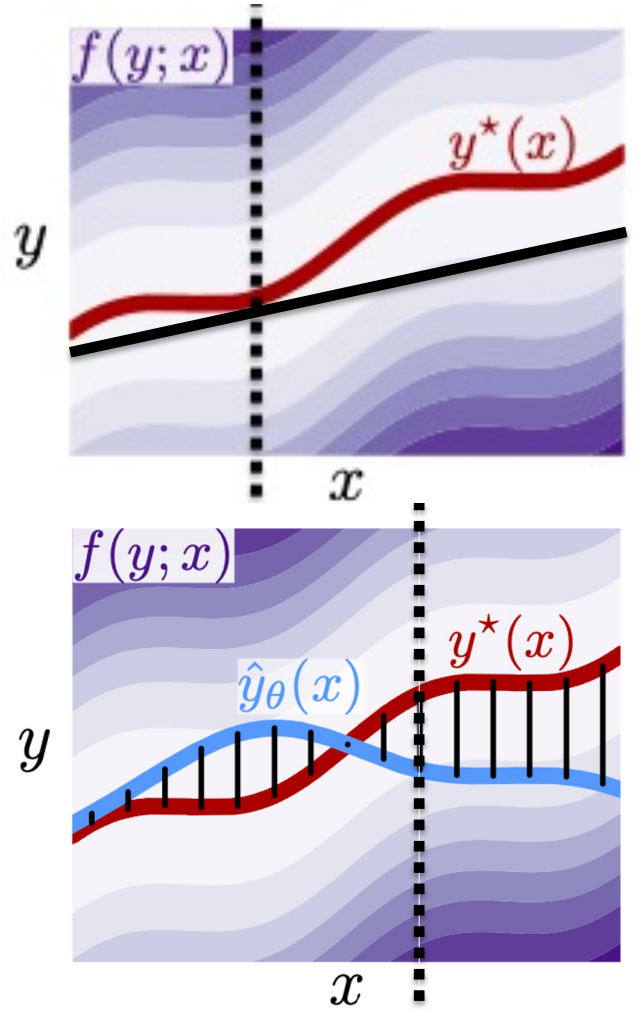
This talk

Differentiable optimization — $\frac{\partial}{\partial x} y^\star(x)$

Foundations
Differentiable convex optimization layers
Differentiable control
Optimization-based task losses

Amortized optimization — $\hat{y}_\theta(x) \approx y^\star(x)$

Foundations
RL as amortized optimization
Amortization via learning latent subspaces
Meta Optimal Transport
Amortizing convex conjugates



Amortized optimization and learning to optimize

Tutorial on amortized optimization for learning to optimize over continuous domains. Amos, Foundations and Trends in Machine Learning (to appear)

- **Difficulty:** optimization is computationally expensive
  - Sometimes **solving once** may be difficult
  - Exasperated when **repeatedly** solving, e.g., as part of a differentiable optimization layer
- **Insight: optimal solutions share structure** and don't live in isolation
- **Solution:** amortized optimization (or learning to optimize)
  - Use machine learning to uncover the shared structure
  - Create learning-augmented versions of classical optimization solvers
  - Far surpasses average or worst-case convergence rates

Amortized variational inference and VAEs

Origin of the term "amortization"

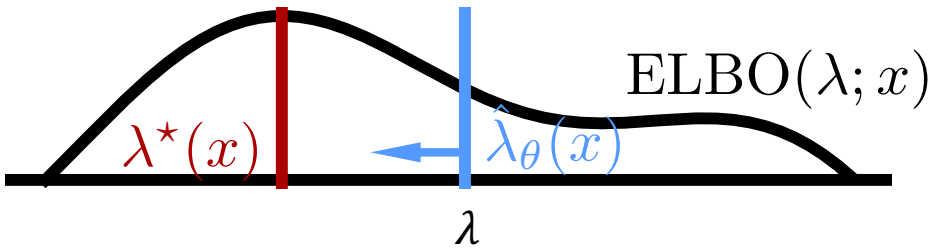
Given a VAE model $p(x) = \log \int_z p(x|z)p(x)$, **encoding** can be done by solving the optimization problem

$$\lambda^\star(x) = \text{argmax}_\lambda \, \text{ELBO}(\lambda; x) \quad \text{where} \quad \text{ELBO}(\lambda; x) \coloneqq \mathbb{E}_{q(z;\lambda)}[\log p(x|z)] - \text{D}_{\text{KL}}(q(x;\lambda)|p(z)).$$

VAEs amortized this (along with other amortized variational inference methods) and solve:

$$\underset{\theta}{\text{argmax}} \, \mathbb{E}_{x\sim p(x)} \text{ELBO}(\hat{\lambda}_\theta; x)$$

x_1 x_2 x_3



Amortization, optimal control, and RL

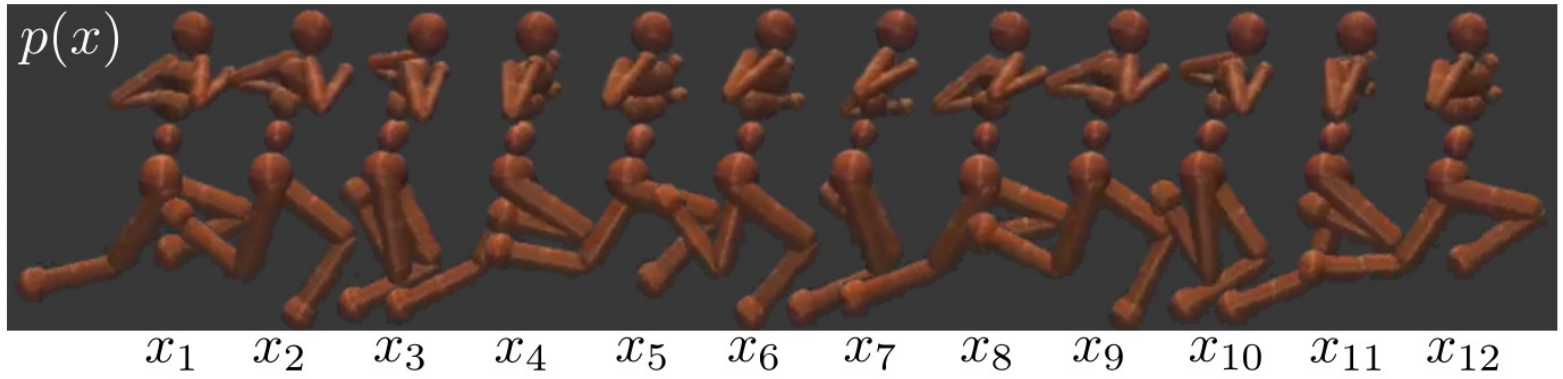
Tutorial on amortized optimization for learning to optimize over continuous domains. Amos, Foundations and Trends in Machine Learning (to appear)
On the model-based stochastic value gradient for continuous reinforcement learning. Amos et al., L4DC 2021.

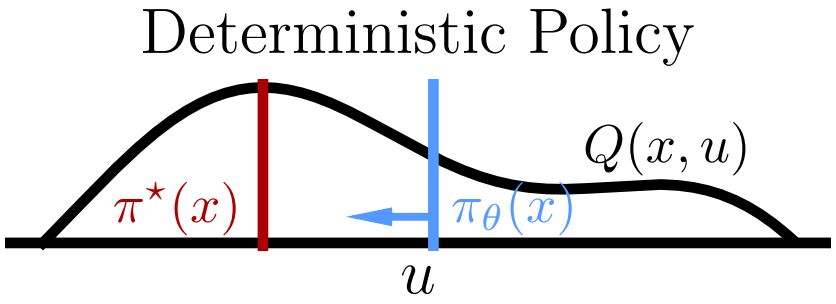
Given a value estimate $Q(x, u)$, **optimal control** finds the best action u for a given state x by solving

$$\pi^\star(x) \coloneqq \underset{u}{\operatorname{argmax}}\, Q(x, u)$$

Almost every reinforcement learning method amortizes this optimization problem!

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

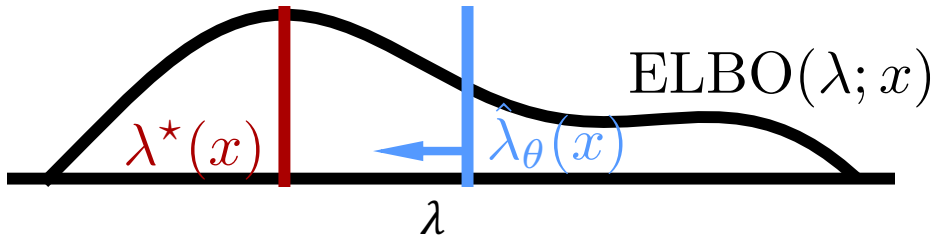




Amortization similarity between VAEs and RL

VAE posterior amortization

$$\underset{\theta}{\text{argmax}}\ \mathbb{E}_{p(x)}\ \text{ELBO}(\hat{\lambda}_\theta(x); x)$$

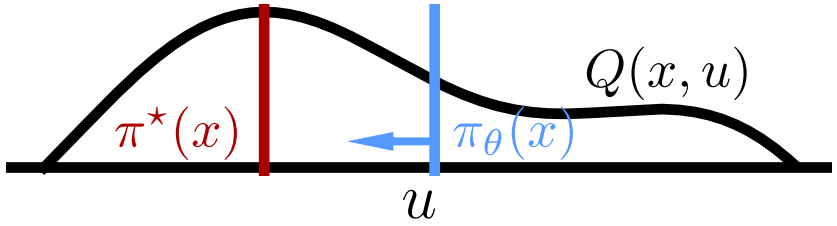


x: images from dataset

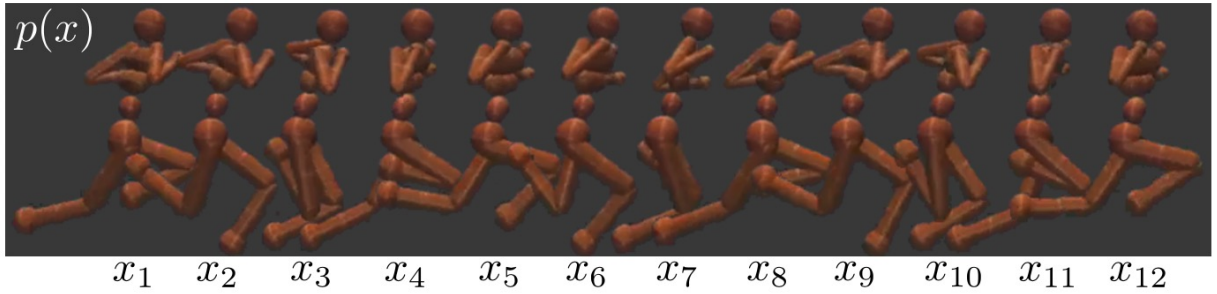
x_1 x_2 x_3

Value gradient amortization in RL

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



x: states from system



Other examples of amortized optimization

Tutorial on amortized optimization for learning to optimize over continuous domains. Amos, Foundations and Trends in Machine Learning (to appear)

My goal: characterize and **connect** applications **previously developed independently**

Variational inference	6.1	VAE	$-$ ELBO	variational posterior	data	full	$\mathcal{L}_{\text{obj}}$
		SAVAE/IVAE	|	|	|	semi	|
Sparse coding	6.2	PSD	reconstruction	sparse code	data	full	$\mathcal{L}_{\text{reg}}$
		LISTA	|	|	|	semi	|
Meta-learning	6.3	HyperNets	task loss	model parameters	tasks	full	$\mathcal{L}_{\text{obj}}$
		LM	|	|	|	semi	$\mathcal{L}_{\text{obj}}^{\text{RL}}$
		MAML	|	|	|	|	$\mathcal{L}_{\text{obj}}$
		Neural Potts	pseudo-likelihood	|	protein sequences	full	$\mathcal{L}_{\text{obj}}$
Fixed-points and convex optimization	6.4	NeuralFP	FP residual	FP iterates	FP contexts	semi	$\mathcal{L}_{\text{obj}}^{\Sigma}$
		HyperAA	|	|	|	|	$\mathcal{L}_{\text{reg}}^{\Sigma}$
		NeuralSCS	CP residual	CP iterates	CP parameters	|	$\mathcal{L}_{\text{obj}}^{\Sigma}$
		HyperDEQ	DEQ residual	DEQ iterates	DEQ parameters	|	$\mathcal{L}_{\text{reg}}^{\Sigma}$
		NeuralNMF	NMF residual	factorizations	input matrices	|	$\mathcal{L}_{\text{obj}}^{\Sigma}$
		RLQP	R_{RLQP}	QP iterates	QP parameters	|	$\mathcal{L}_{\text{obj}}^{\text{RL}}$
Optimal transport	6.5	AmorConj	c-transform obj	supp(α)	supp(β)	full	$\mathcal{L}_{\text{obj}}$
		$\mathcal{A}$-SW	max-sliced dist	slices Θ	mini-batches	|	$\mathcal{L}_{\text{obj}}$
		Meta OT	dual OT cost	optimal couplings	input measures	|	$\mathcal{L}_{\text{obj}}$
Reinforcement learning	6.6	BC/IL	$-Q$-value	controls	state space	full	$\mathcal{L}_{\text{reg}}$
		(D)DPG/TD3	|	|	|	|	$\mathcal{L}_{\text{obj}}$
		PILCO	|	|	|	|	$\mathcal{L}_{\text{obj}}$
		POPLIN	|	|	|	full or semi	$\mathcal{L}_{\text{reg}}$
		DCEM	|	|	|	semi	$\mathcal{L}_{\text{reg}}$
		IAPO	|	|	|	|	$\mathcal{L}_{\text{obj}}$
		SVG	$\text{D}_{\mathcal{Q}}$ or $-\mathcal{E}_Q$	control dists	|	full	$\mathcal{L}_{\text{obj}}$
		SAC	|	|	|	|	$\mathcal{L}_{\text{obj}}$
		GPS	|	|	|	|	$\mathcal{L}_{\text{KL}}$

Amortized optimization definitions

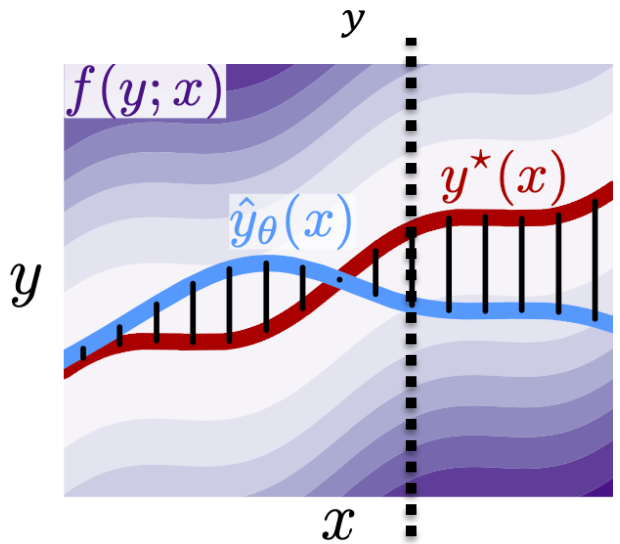
Tutorial on amortized optimization for learning to optimize over continuous domains. Amos, Foundations and Trends in Machine Learning (to appear)

Two key components:

1. **Amortization model** $\hat{y}_\theta(x)$ tries to approximate $y^\star(x)$
   Fully-amortized: A neural network mapping from x to the solution
   Semi-amortized: Allowed to query the objective f

2. **Loss** $\mathcal{L}$ measures how well $\hat{y}$ fits $y^\star$ and optimized with $\min_\theta \mathcal{L}(\hat{y}_\theta)$
   Regression: $\mathcal{L}_{\text{reg}}(\hat{y}_\theta) \coloneqq \mathbb{E}_{p(x)} \|\hat{y}_\theta(x) - y^\star(x)\|_2^2$
   Objective-based: $\mathcal{L}_{\text{obj}}(\hat{y}_\theta) \coloneqq \mathbb{E}_{p(x)} f(\hat{y}_\theta(x); x)$

$y^\star(x) \in \operatorname{argmin} f(y; x)$



Vertical slices are optimization problems

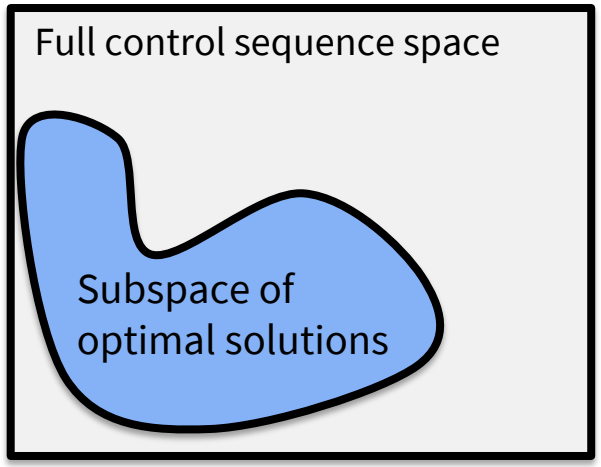
Amortization is widely deployed (although often not connected as amortization)
Amortized variational inference (VAEs)
Meta-learning (hypernetworks, MAML)
Reinforcement learning (policy learning for actor-critic methods, SAC)

Amortization via learning latent subspaces

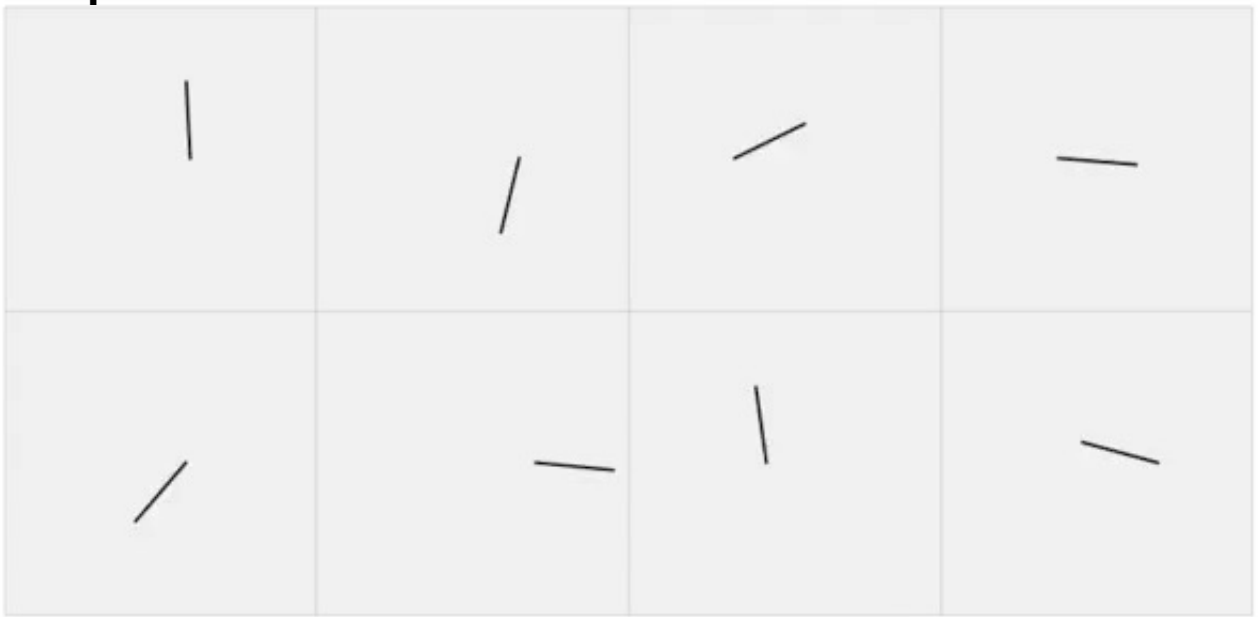
The differentiable cross-entropy method. Amos and Yarats, ICML 2020.

Amortize the problem by **learning a latent subspace** of optimal solutions
Only search over optimal solutions rather than the entire space

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



Cartpole videos

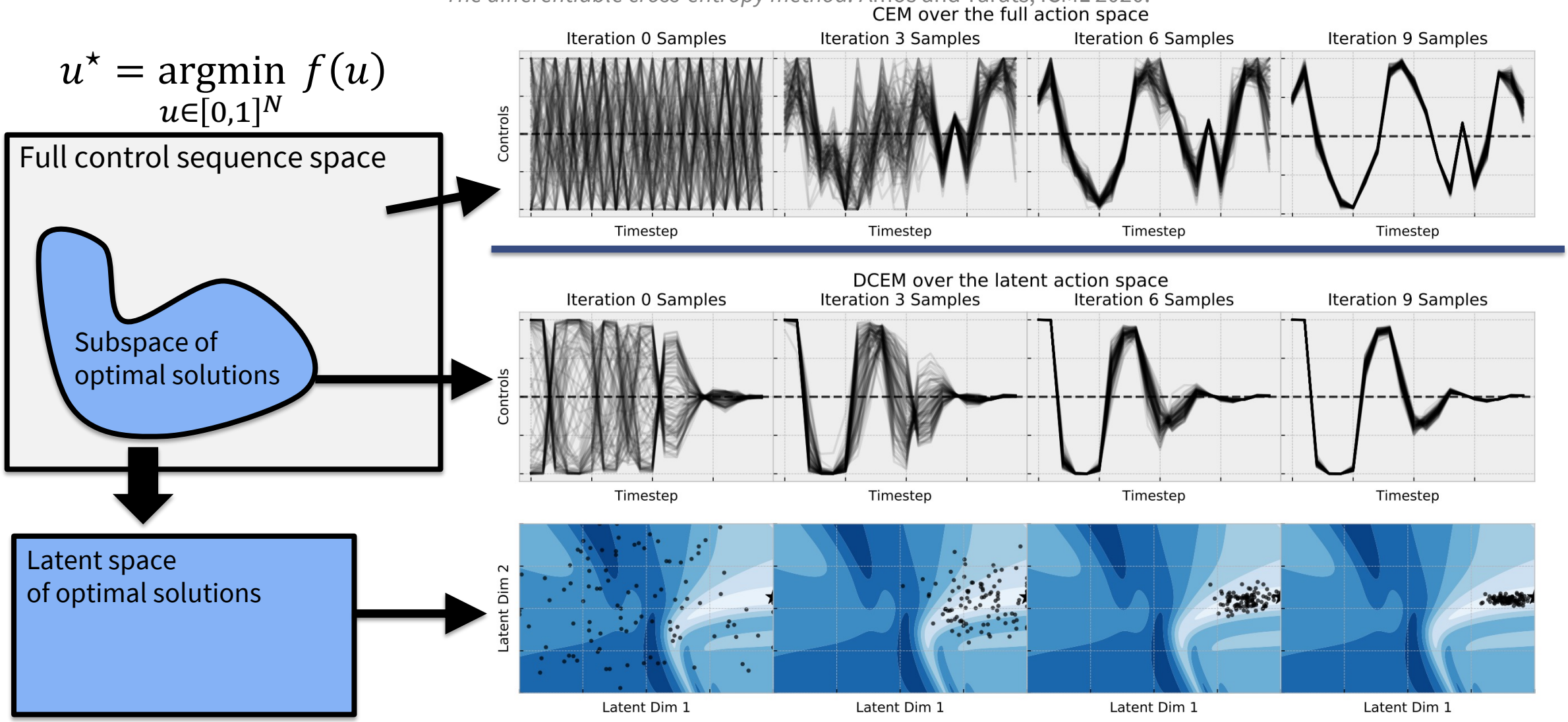


Optimal controls over time — force on the cartpole

Amortization via learning latent subspaces

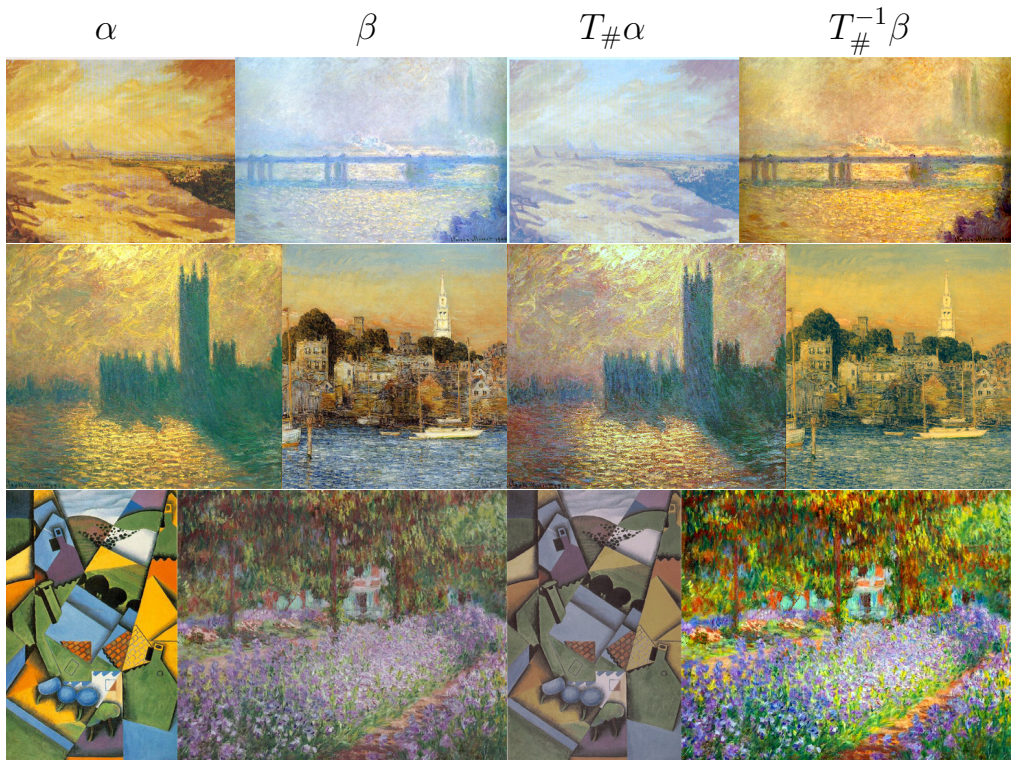
The differentiable cross-entropy method. Amos and Yarats, ICML 2020.

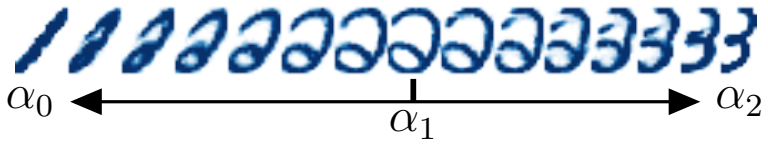
$$u^\star = \underset{u \in [0,1]^N}{\text{argmin}} \; f(u)$$



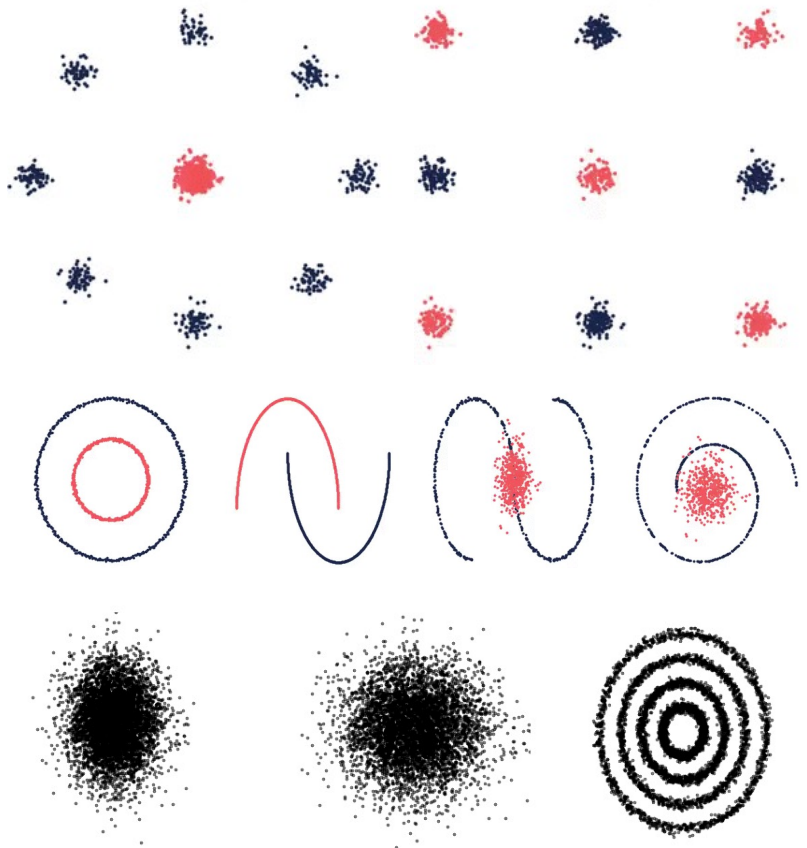
New work: amortization in optimal transport

Meta Optimal Transport. Amos et al., 2022





On amortizing convex conjugates for optimal transport. Amos, ICLR 2023



Closing thoughts

Optimization expresses **non-trivial reasoning operations**

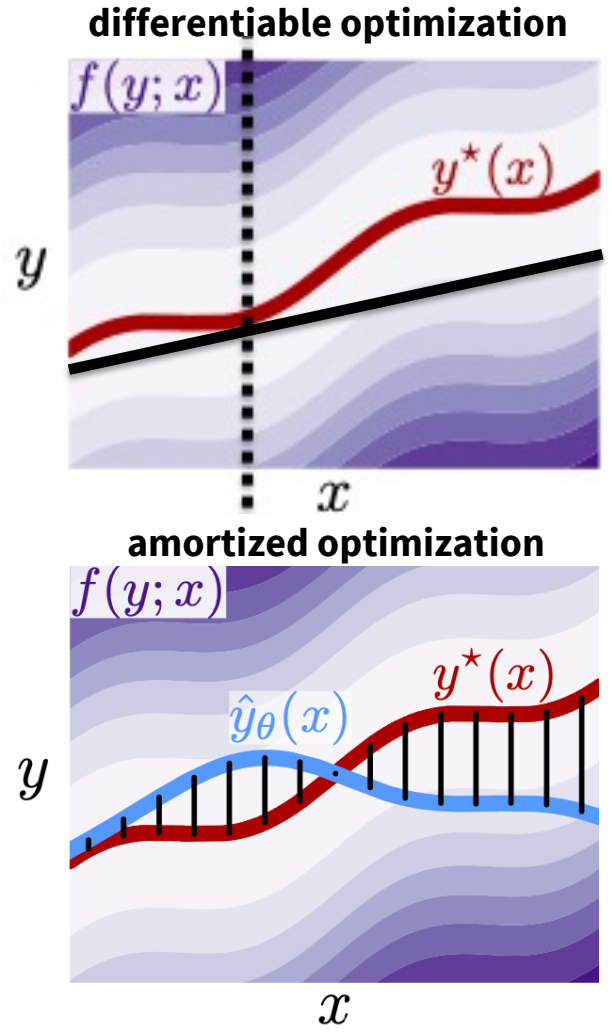
- Integrates nicely with machine learning by **seeing it as a function**

Differentiable optimization is a powerful primitive within larger systems

- Theoretical and engineering foundations are here
- Can be propagated through and learned, just like any layer
- Provides a perspective to analyze existing models and layers

Amortized optimization enables fast learning-augmented solvers

- Foundations are likewise here
- Provides a perspective on existing methods in RL, VI, OT



Future directions

Goal: build intelligent systems that **understand and interact** with the world
Why? To advance scientific and engineering discoveries

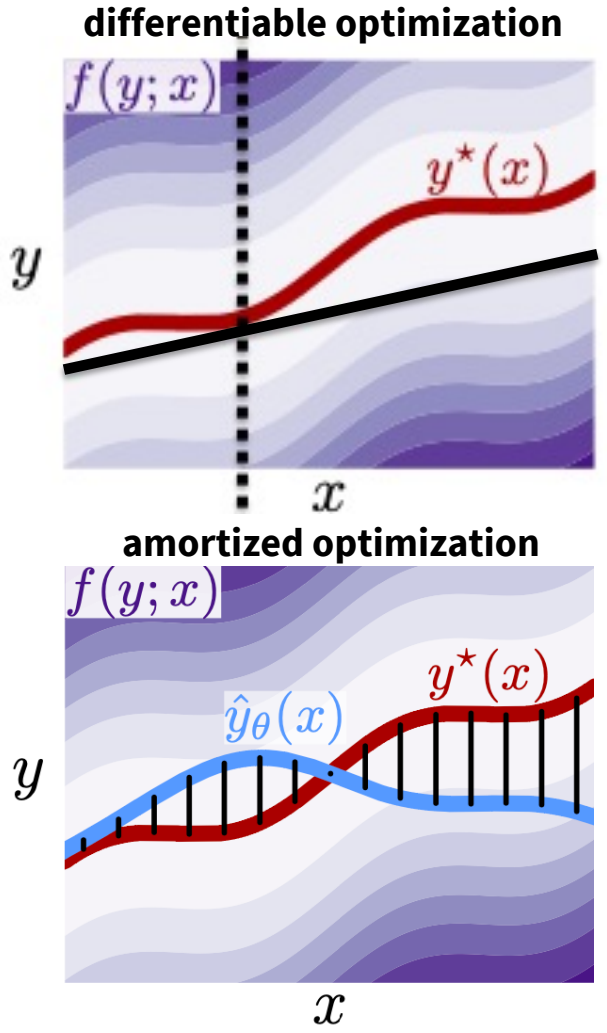
Advancing **optimization** and **machine learning foundations** is crucial

Differentiable optimization — $\frac{\partial}{\partial x} y^\star(x)$

- *Differentiable control: learning control-aware representations*
- *Numerics: "differentiation", ill-behaved functions, approximations*
- *Generalizing to non-convex, non-Euclidean, and/or discrete settings*

Amortized optimization — $\hat{y}_\theta(x) \approx y^\star(x)$

- *Numerics: amortizing with distributions and Markov kernels*
- *Building general learning-augmented optimizers*
- *Amortization models that use differentiable optimization*



Learning with differentiable and amortized optimization

Brandon Amos • Meta AI (FAIR) NYC

http://github.com/bamos/presentations

[ICML 2017] *Differentiable QPs: OptNet*
[NeurIPS 2017] *Differentiable Task-based Model Learning*
[NeurIPS 2018] *Differentiable MPC for End-to-end Planning and Control*
[NeurIPS 2019] *Differentiable Convex Optimization Layers*
[Ph.D. Thesis 2019] *Differentiable Optimization-Based Modeling for ML*
[arXiv 2019] *Differentiable Top-k and Multi-Label Projection*
[arXiv 2019] *Generalized Inner Loop Meta-Learning*
[ICML 2020] *Differentiable Cross-Entropy Method*
[ICML 2021] *Differentiable Combinatorial Optimization: CombOptNet*
[L4DC 2021] *On the model-based stochastic value gradient*
[NeurIPS 2022] *Theseus: Differentiable Nonlinear Optimization*
[arXiv 2022] *Meta Optimal Transport*
[ICLR 2023] *On amortizing convex conjugates for optimal transport*
[Foundations and Trends in ML, to appear] *Tutorial on amortized optimization*

Collaborators: Akshay Agrawal, Alexander Rives, Andrew Gordon Wilson, Anselm Paulus, Arnaud Fickinger, Artem Molchanov, Austin Wang, Byron Boots, Caroline Chen, Daniel DeTone, Denis Yarats, Douwe Kiela, Edward Grefenstette, Franziska Meier, Georg Martius, Giulia Luise, Hengyuan Hu, Ievgen Redko, Ivan Jimenez, Jacob Sacks, Jason Liu, Jing Dong, Joseph Ortiz, Joshua Meier, Kyunghyun Cho, Luis Pineda, Maurizio Monge, Michal Rolínek, Mustafa Mukadam, Nathan Lambert, Noam Brown, Omry Yadan, Paloma Sodhi, Phu Mon Htut, Priya Donti, Ricky Chen, Robert Verkuil, Roberto Calandra, Samuel Cohen, Samuel Stanton, Shane Barratt, Shobha Venkataraman, Soumith Chintala, Stephen Boyd, Steven Diamond, Stuart Anderson, Stuart Russell, Taosha Fan, Tom Sercu, Vít Musil, Yann LeCun, Zeming Lin, Zico Kolter