OptNet: End-to-End Differentiable Constrained Optimization

Brandon Amos
Carnegie Mellon University

Joint work with J. Zico Kolter, Priya Donti, Jacob Sacks,
Ivan Jimenez Rodriguez, and Byron Boots

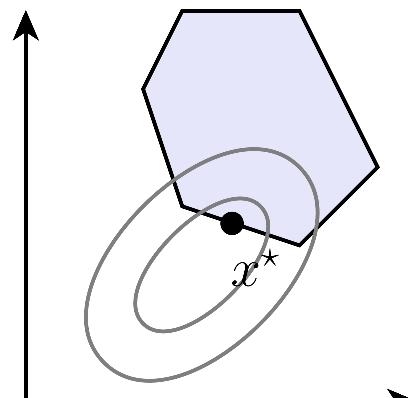
Where do today's ML systems break down?

A lot of data → Model → Predictions → Loss

Current Primitive Operations: Linear maps, convolutions, activation functions, random sampling, simple projections (e.g. onto the simplex or Birkhoff polytope)

OptNet: Optimization as a new primitive operation

- Consider optimization as another potential **layer**, to be **composed with others**
- Why? Optimization is an extremely powerful paradigm for decision-making.
  - Applications in **finance** (Markowitz portfolio optimization), **machine learning** (support vector machines), **control** (linear-quadratic model predictive control), **geometry** (projections onto polyhedra)

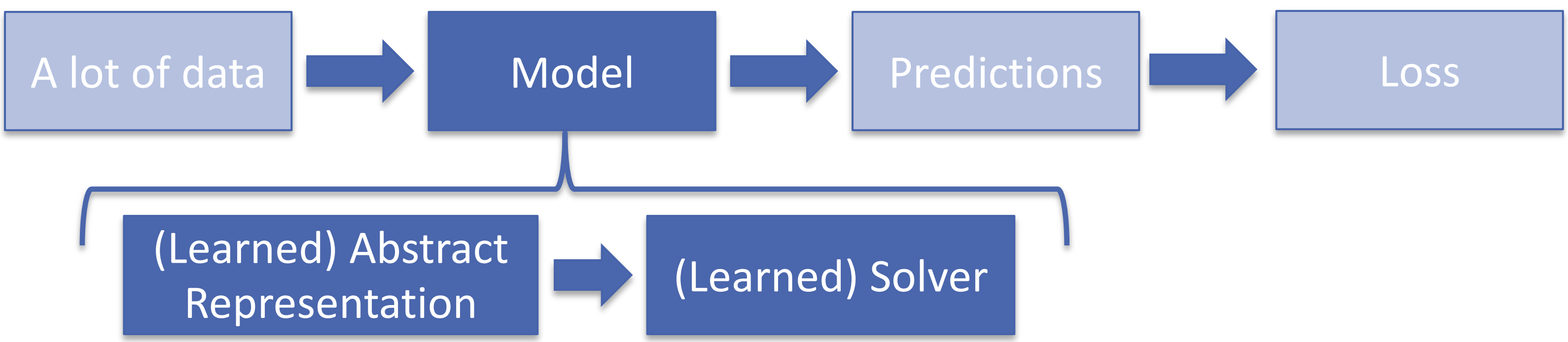


Note: we already use parameter optimization in the learning procedures, but we should also consider it as an operation for inference and control

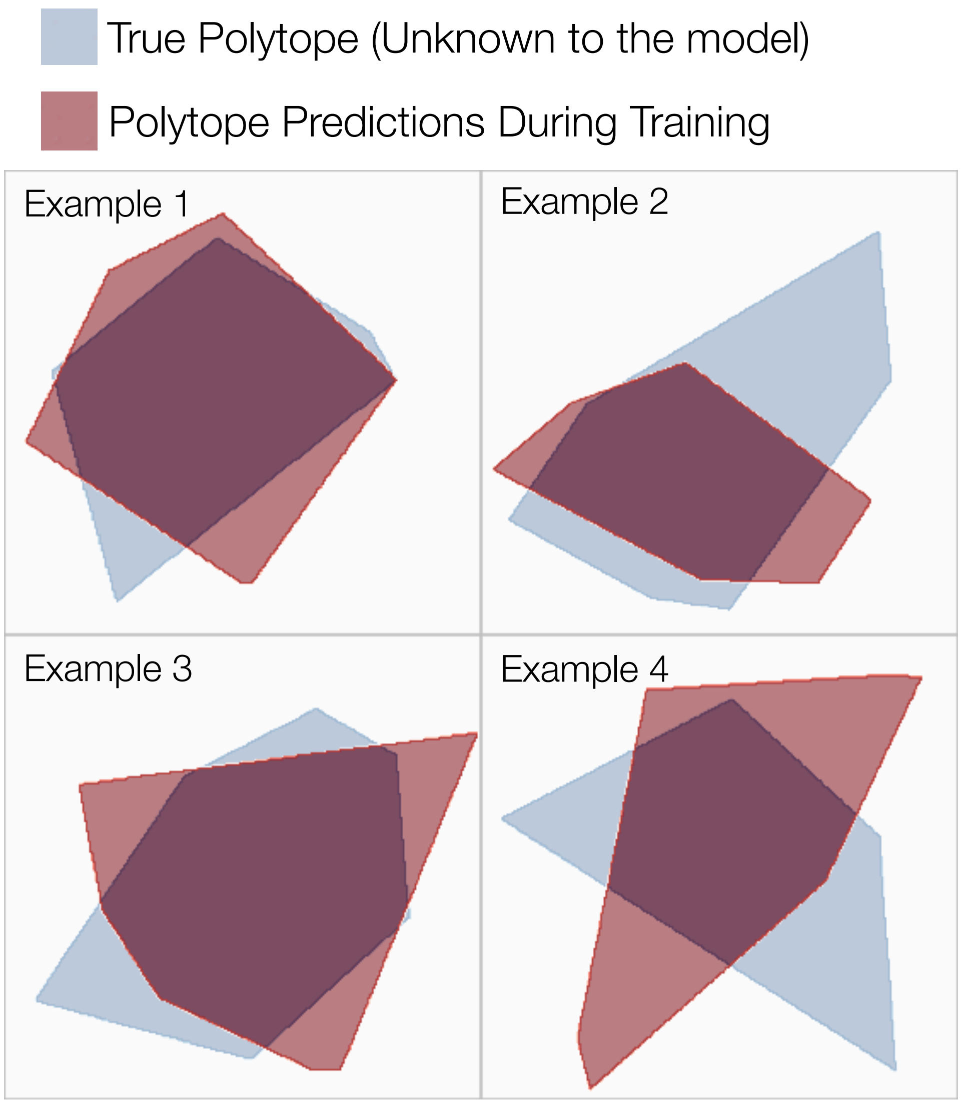
Why is optimization a useful primitive operation in learning systems?

We have **incomplete domain knowledge** about what we want to model

- Fill in parts of the optimization problem that we know
- Use data to **learn the parts that we don't**



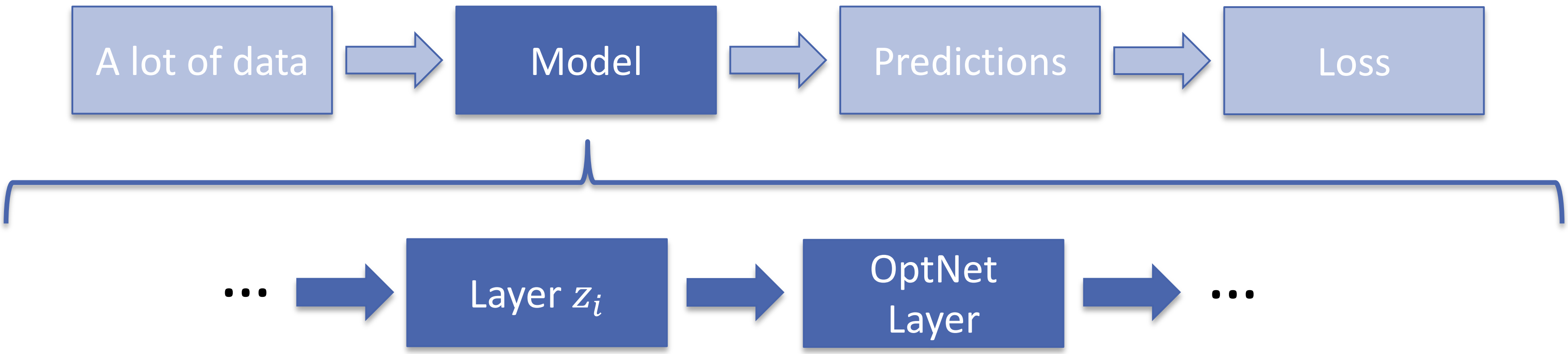
OptNet Application: Approximating Polytopes



This Talk

- The OptNet Layer
- Starting Simple: Learning Projections, Sudoku, and Denoising
- End-to-End Task-Based Learning for Stochastic Optimization
- End-to-End Model Predictive Control

The OptNet Layer



$$
\begin{aligned}
z_{i+1} = \quad & \underset{z}{\operatorname{argmin}} \; \frac{1}{2} z^T \boxed{Q(z_i)} z + q(z_i)^T z \\
\text{subject to} \quad & A(z_i) z = b(z_i) \\
& G(z_i) z \leq h(z_i)
\end{aligned}
$$

Learnable parameters: Q, q, A, b, G, h

The matrix $Q(z_i)$ depends on the previous layer z_i

Differentiating a convex argmin

Consider a convex optimization problem with inputs p and parameters θ:

$$\begin{aligned} x^\star = \operatorname*{argmin}_x \quad & f(x, p, \theta) \\ \text{subject to} \quad & g(x, p, \theta) \leq 0 \\ & h(x, p, \theta) = 0 \end{aligned}$$

From convex optimization theory, the Karush-Kuhn-Tucker conditions provide necessary and sufficient equations for optimality.

To obtain $\partial x^\star / \partial p$ and $\partial x^\star / \partial \theta$, implicitly differentiate the KKT conditions.

Implicitly differentiating the KKT conditions

Solve linear systems of the form:

$$\underbrace{\begin{bmatrix} Q & G^T & A^T \\ D(\lambda)G & D(Gx-h) & 0 \\ A & 0 & 0 \end{bmatrix}}_{\text{Generalized Jacobian of KKT conditions}} \underbrace{\begin{bmatrix} \partial z \\ \partial \lambda \\ \partial \nu \end{bmatrix}}_{\text{Desired gradients}} = \underbrace{\begin{bmatrix} -\partial Q z - \partial p - \partial G^T \lambda - \partial A^T \nu \\ -\lambda \circ (\partial G^T z - \partial h) \\ -\partial A z + \partial b \end{bmatrix}}_{\text{Gradients of optimization problem parameters}}$$

If done correctly, just requires a single solve to compute **all** gradients

- More details are in our OptNet paper (ICML 2017)

Efficient implementation

Optimization in every single pass of the network, even using highly optimized (but necessarily still general purpose) solvers, is ***slow***

Implemented our own primal-dual interior point algorithm for QPs, specialized for minibatch processing of multiple same-sized problems using batch GPU factorization, plus some additional tricks

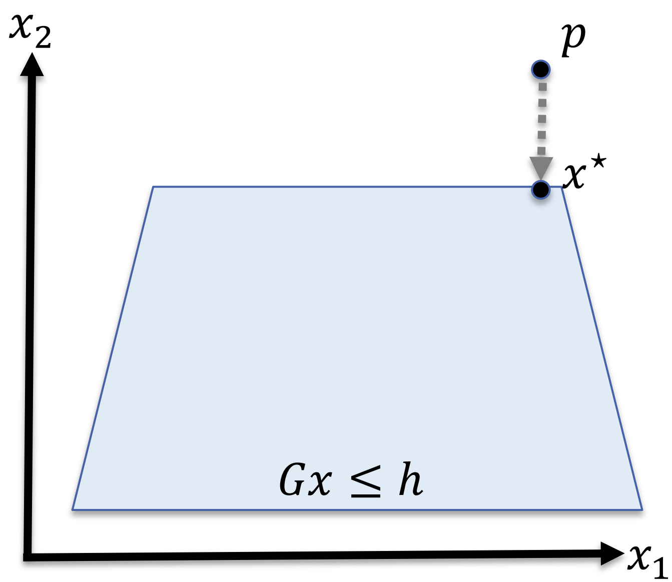
Very nice property: matrix solution needed for backprop is exactly the same as that used in interior point final inner solve, meaning we get backprop through the solver effectively “for free”

Open source implementation (for PyTorch) available at:
http://locuslab.github.io/qpth

This Talk

- The OptNet Layer
- **Starting Simple: Learning Projections, Sudoku, and Denoising**
- End-to-End Task-Based Learning for Stochastic Optimization
- End-to-End Model Predictive Control

Projection as a machine learning problem

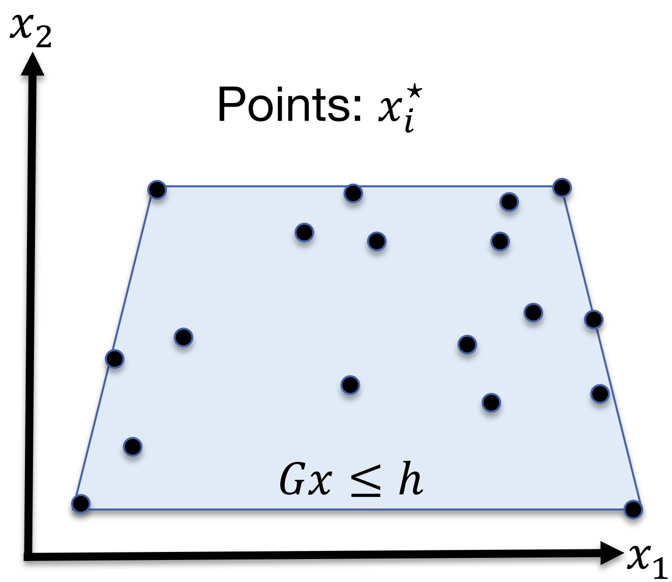


True Problem

$$x^\star = \underset{x}{\operatorname{argmin}} \ \ \operatorname{dist}(x, p)$$
$$\text{subject to} \ \ Gx \leq h$$

What if we are given example input/output pairs $(p_i, x_i^\star)$ and want to recover G and h?

One Approach: Projection with a convex hull



True Problem

$$x^\star = \underset{x}{\mathrm{argmin}} \;\; \mathrm{dist}(x, p) \quad \text{subject to} \;\; Gx \leq h$$

What if we are given example input/output pairs $(p_i, x_i^\star)$ and want to recover G and h?

The convex hull gives this, but is difficult to approximate in high dimensions and under noise.

The OptNet approach to learning projections

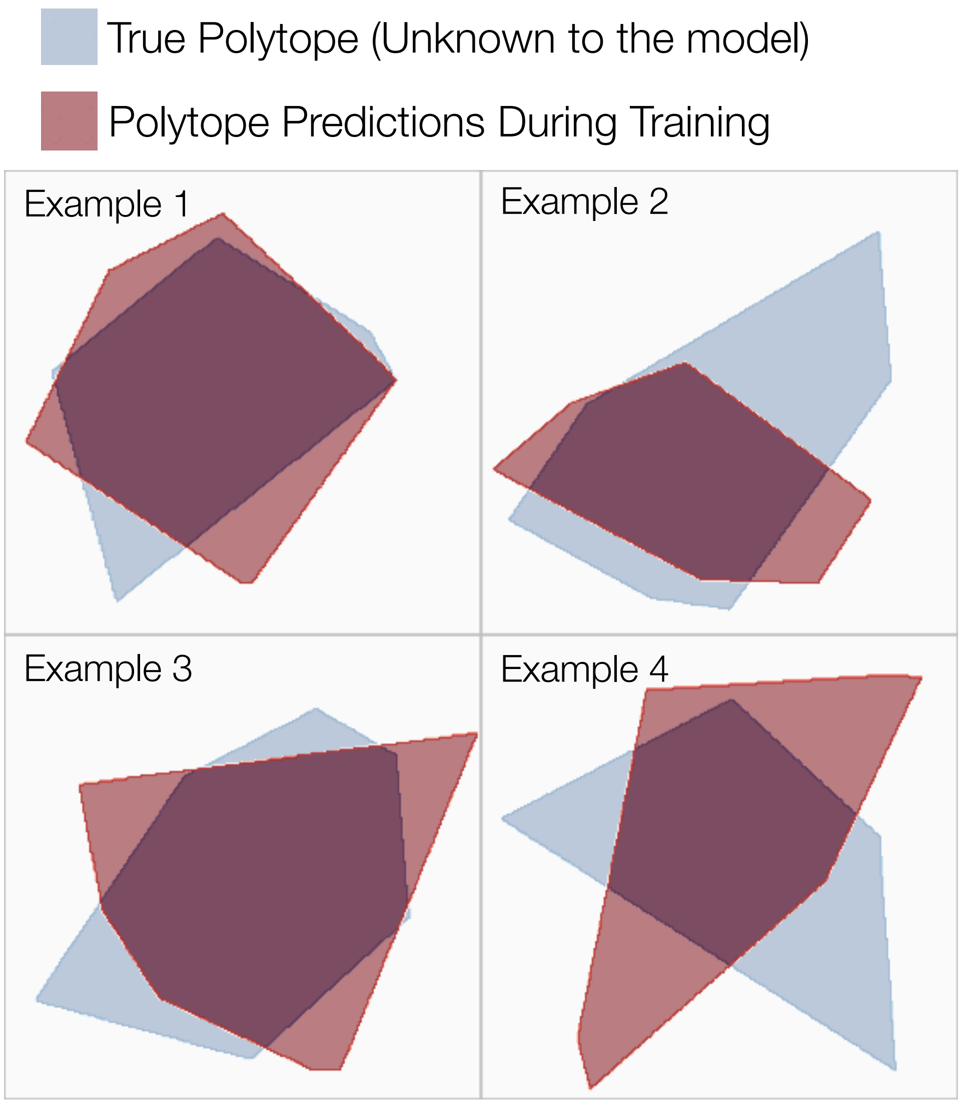
Model:

$$\begin{aligned} x^\star = \quad & \underset{x}{\operatorname{argmin}} \quad \text{dist}(x, p) \\ & \text{subject to} \quad Gx \le h \end{aligned}$$

Data: Example input/output pairs $(p_i, x_i^\star)$

Training: The output is just a function of p, G, and h. Randomly initialize a new polytope $\hat{G}$ and $\hat{h}$, define a loss function ℓ, and take gradient steps with $\partial \ell / \partial \hat{G}$ and $\partial \ell / \partial \hat{h}$.

OptNet Application: Approximating Polytopes



Application: Sudoku

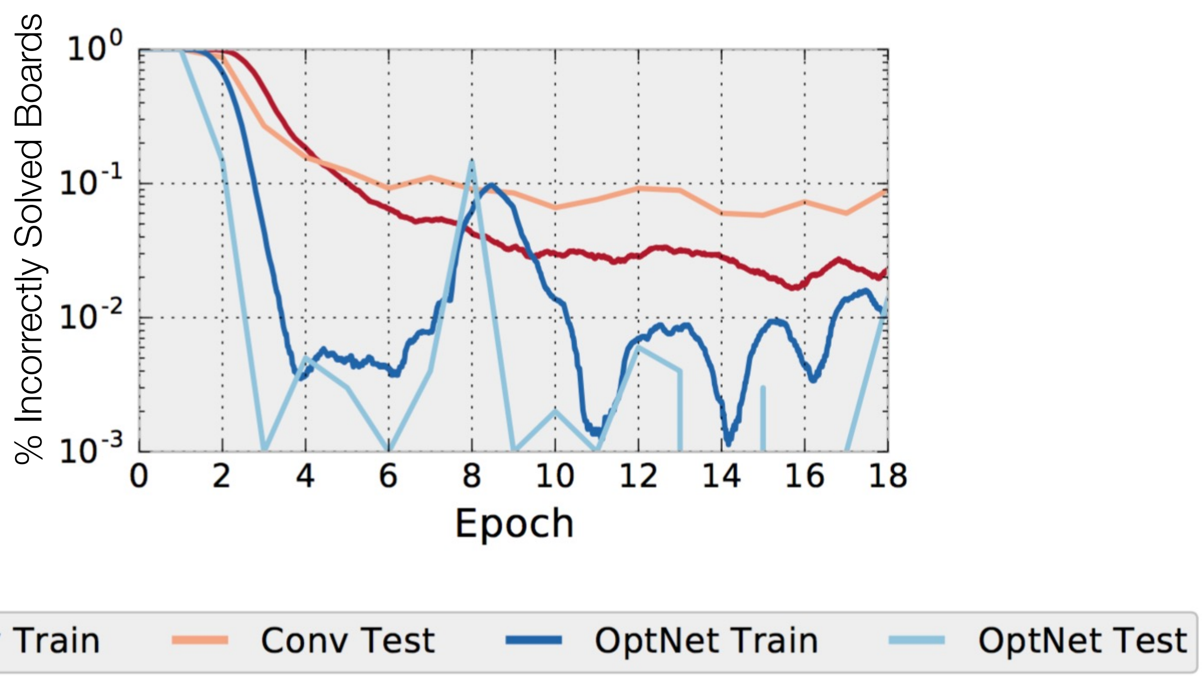
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Application: Sudoku

$$x^\star = \operatorname*{argmin}_x \; \mathrm{dist}(x, p)$$
$$\text{subject to } \; Ax = b$$

The OptNet layer exactly learns the mini-Sudoku constraints from data!
Baseline: A deep convolutional feed-forward network



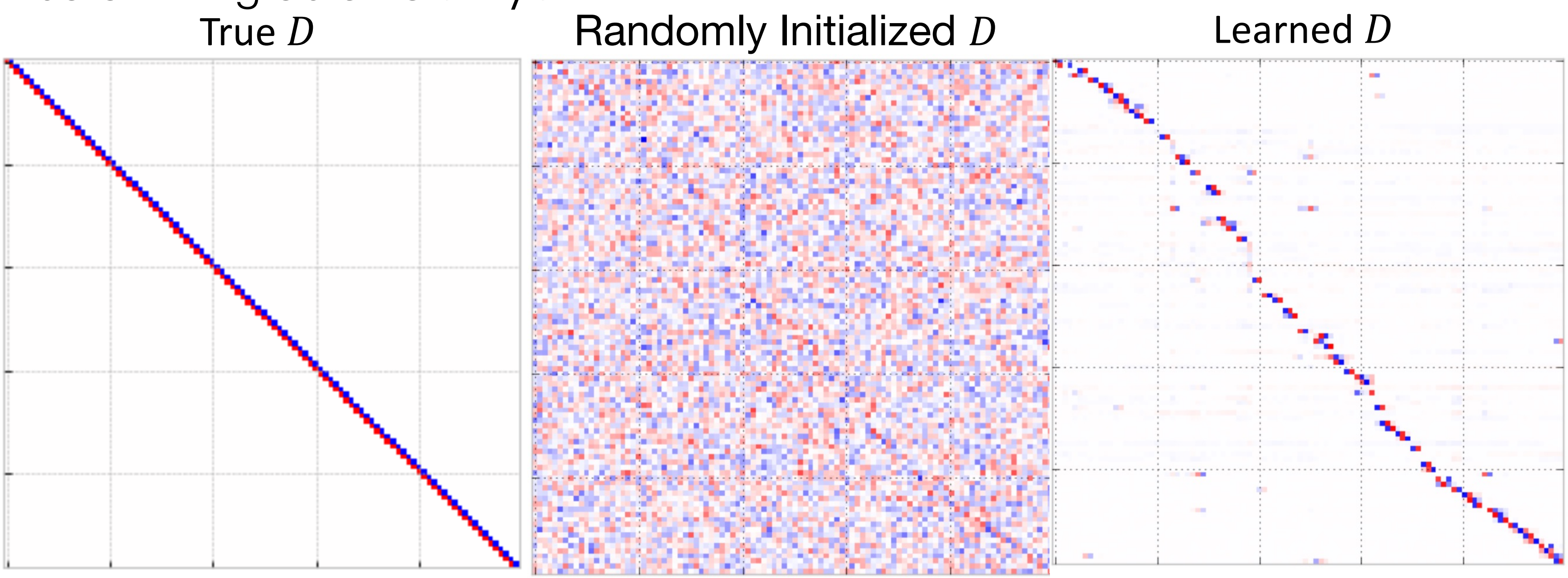
Application: 1D Signal Denoising

Task: Learn a model from data that maps from a noisy signal to a denoised signal.

Total Variation Denoising Approach: Solve the following optimization problem where D is the differencing operator.

$$z^\star = \underset{z}{\operatorname{argmin}} \frac{1}{2}||y - z||_2^2 + \lambda||Dz||_1$$

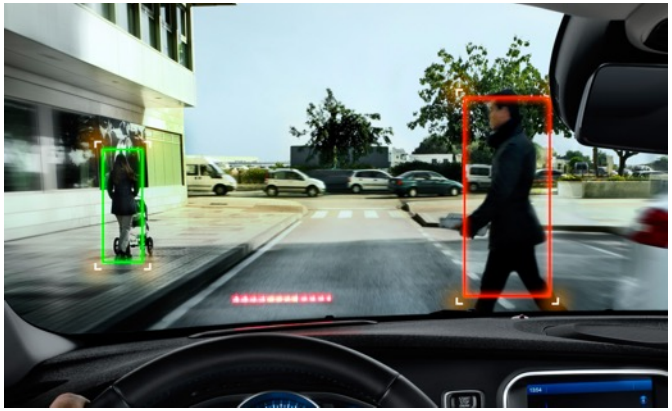
OptNet Application: Randomly initialize the differencing operator D and learn it from data with gradients $\partial z^\star / \partial D$



This Talk

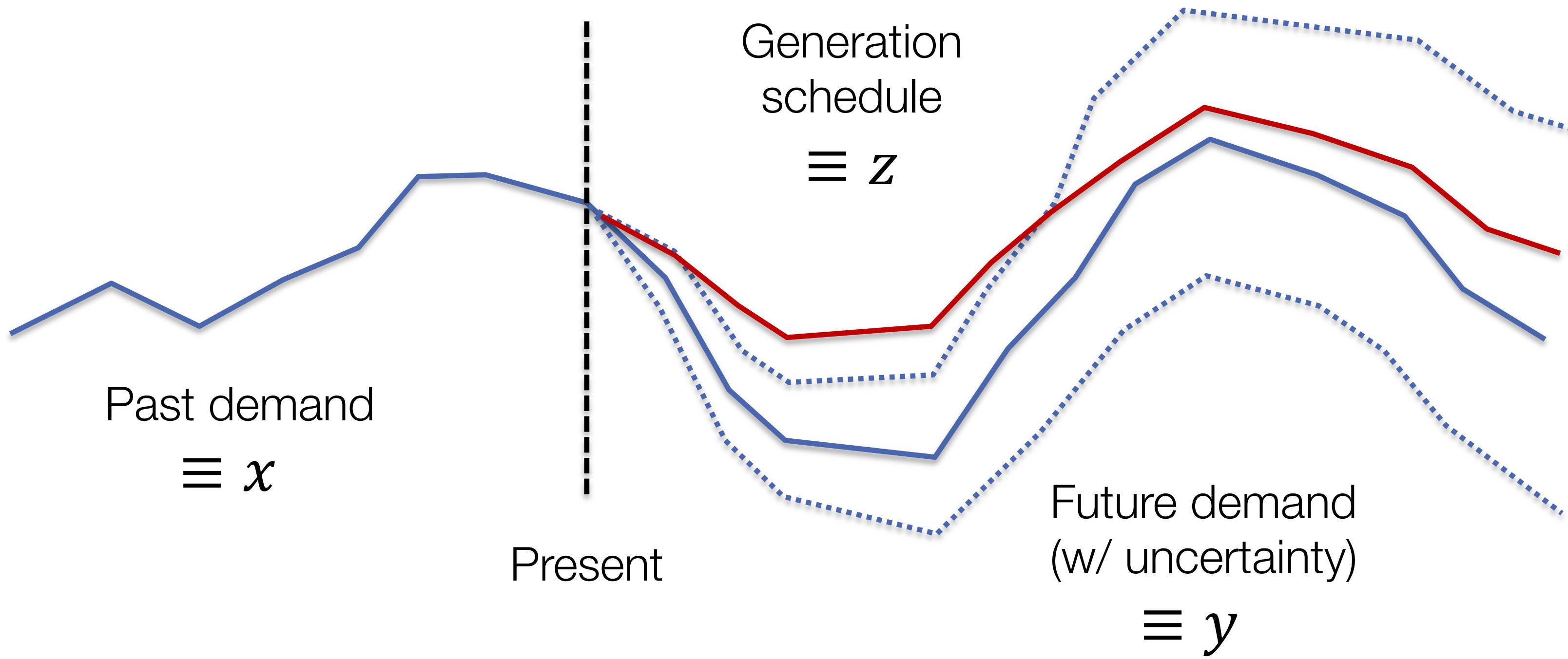
- The OptNet Layer
- Starting Simple: Learning Projections, Sudoku, and Denoising
- End-to-End Task-Based Learning for Stochastic Optimization
- End-to-End Model Predictive Control

The future of machine learning



Data → Machine learning algorithm → Decision making → Cost, e.g. $$

Example: electricity generation



Stochastic programming

Given some distribution over y, solve the optimization problem to find generation schedule z

$$\begin{aligned} \underset{z}{\text{minimize}} \quad & \mathbf{E}[f(y,z)] \\ \text{subject to} \quad & \mathbf{E}[g(y,z)] \le 0 \\ & h(z) = 0 \end{aligned}$$

I.e., “schedule generation to minimize expected cost under distribution”

where expectations are with respect to y

Crucial point: solving stochastic program requires a *model* of random variable y (need to draw *multiple* samples)

The whole (complicated) process

1. Pick form of model $p(y|x;\theta)$, and learn via maximum likelihood

E.g., previous and future (actual) demand

2. Given some *new* example (x', y'):

   A. Receive features x', form distribution $p(y|x';\theta)$

   B. Solve stochastic optimization problem

   $$\underset{z}{\text{minimize}}\ \mathbf{E}[f(y,z)]\ \text{s.t.}\ \mathbf{E}[g(y,z)] \le 0\ \ h(y,z) = 0$$

   call the resulting solution $z^\star(x';\theta)$

   C. Suffer cost $f\big(y', z^\star(x';\theta)\big)$

Something is wrong here…

We are learning the model based upon log likelihood $\log p\left(y^{(i)} \middle| x^{(i)}; \theta\right)$, but evaluating it using a task-based cost function $f\left(y', z^\star(x'; \theta)\right)$…

Unless the true underlying distribution is in the model class (never the case), these are two different and competing objectives

Our proposed alternative: **adjust the *model parameters* to optimize the actual performance of the closed-loop system**

Task-based end-to-end model learning

Basic idea: treat $z^\star(x;\theta)$ as a "black box policy", whose parameters happen to be the parameters of a prediction model

Given samples $x^{(i)}, y^{(i)}$, directly optimize model parameters to improve the performance of the policy

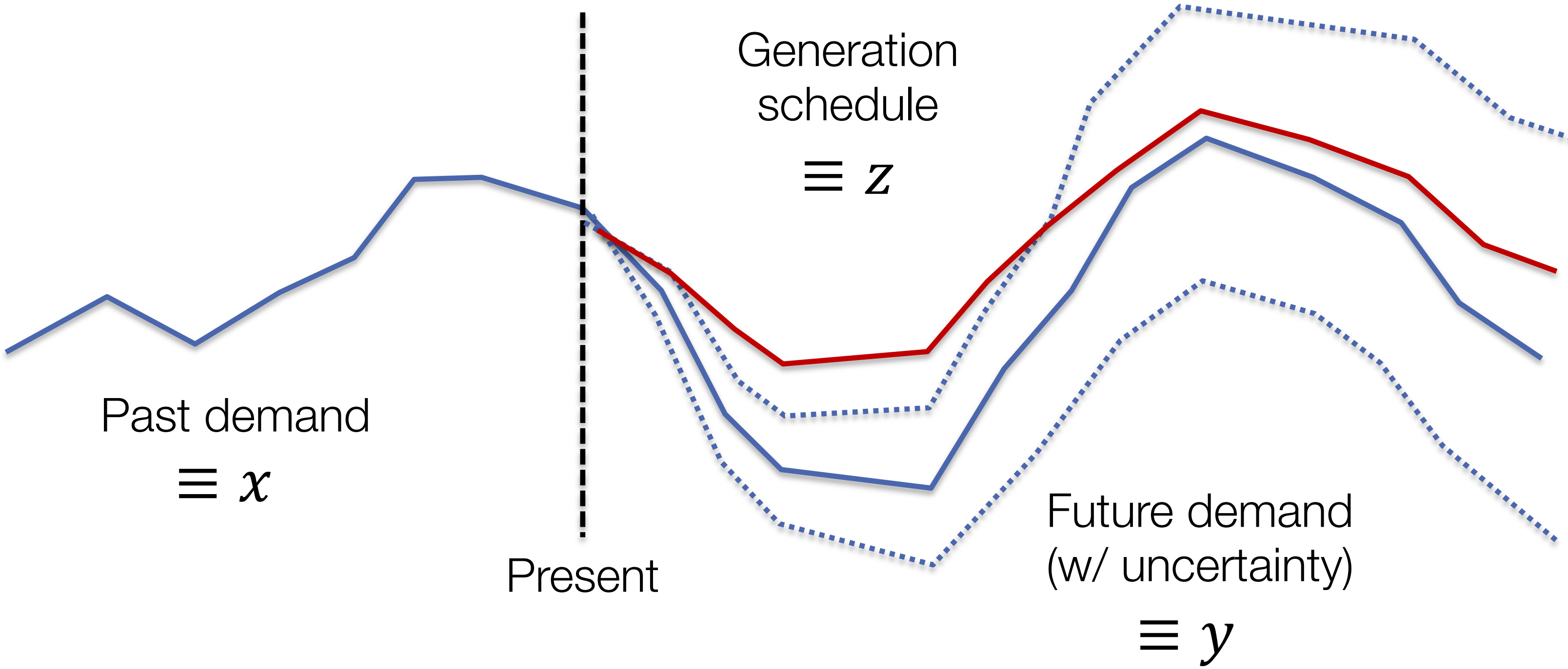
$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} f\left(y^{(i)}, z^\star(x^{(i)};\theta)\right)$$

Requires computing the Jacobian

$$\frac{\partial}{\partial\theta} f\left(y^{(i)}, z^\star(x^{(i)};\theta)\right) = \frac{\partial f\left(y^{(i)}, z^\star\right)}{\partial z^\star}\frac{\partial z^\star}{\partial \theta}$$

The Jacobian of the *solution* to the optimization problem

Task-based learning application: electricity generation



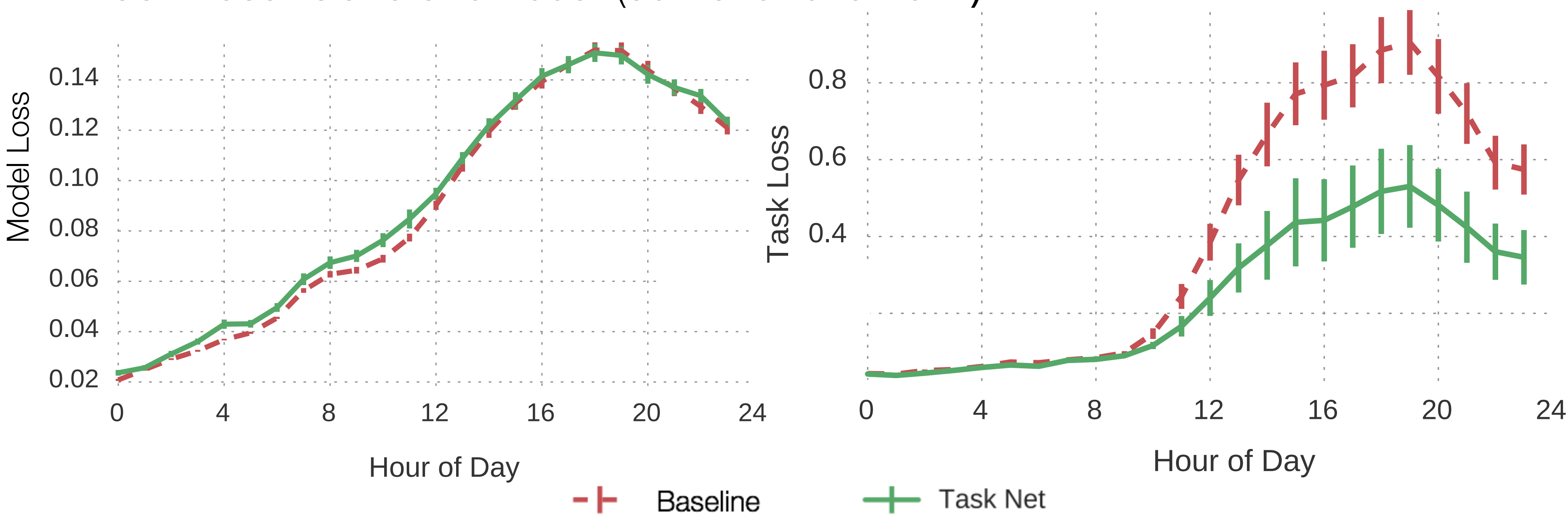
Model Loss: Prediction error of y into the future

Task Loss: Generation cost (some function of z)

Results: electricity generation

The task net incurs nearly the same model loss as the baseline, but learns to make errors in places that aren't as harmful to the task loss.

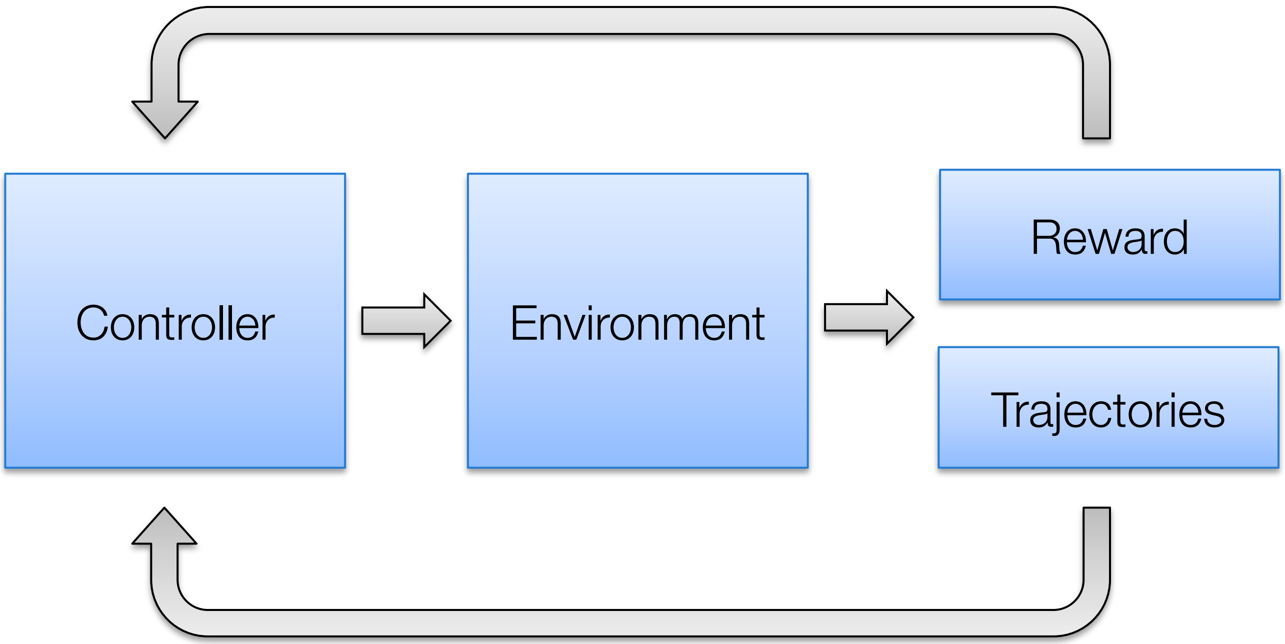
Model Loss: Prediction error of y into the future
Task Loss: Generation cost (some function of z)



This Talk

- The OptNet Layer
- Starting Simple: Learning Projections, Sudoku, and Denoising
- End-to-End Task-Based Learning for Stochastic Optimization
- End-to-End Model Predictive Control

In model-based RL



Related work on combining model-based and model-free RL

Recently there has been a lot of interest in model-based priors for model-free reinforcement learning:

- Among others: Dyna-Q (Sutton, 1990), GPS (Levine and Koltun, 2013), Imagination-Augmented Agents (Weber et al., 2017), Value Iteration Networks (Tamar et al., 2016), TreeQN (Farquhar et al., 2017)

These typically involve:

1. **Using an RNN:** Efficient but not as expressive and general as control
2. **Unrolling an LQR solver:** Expressive/general but inefficient

Our Approach: Model Predictive Control

Traditionally viewed as a pure **planning problem** given known (potentially non-convex) **cost** and **dynamics**:

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

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

Execute u_1 in the environment, observe the next observation, and repeat.

Cost and dynamics explicitly represented and learned.

Model Predictive Control with SQP

- The standard way of solving MPC is to use **sequential quadratic programming (SQP),** using LQR (linear quadratic regulator) in most cases
- **Form approximations** to the cost and dynamics around the current iterate
- Repeat until a **fixed point** is reached and **differentiate through it**

QP iterate i

…

$$\begin{aligned} \tau_{1:T}^{i} = \operatorname*{argmin}_{\tau_{1:T}} \ & \sum_t \tilde{C}_\theta^i(\tau_t) \\ \text{subject to} \ & x_1 = x_{init} \\ & x_{t+1} = \tilde{f}_\theta^i(\tau_t) \\ & \underline{u} \leq u \leq \overline{u} \end{aligned}$$

…

A Differentiable MPC Module

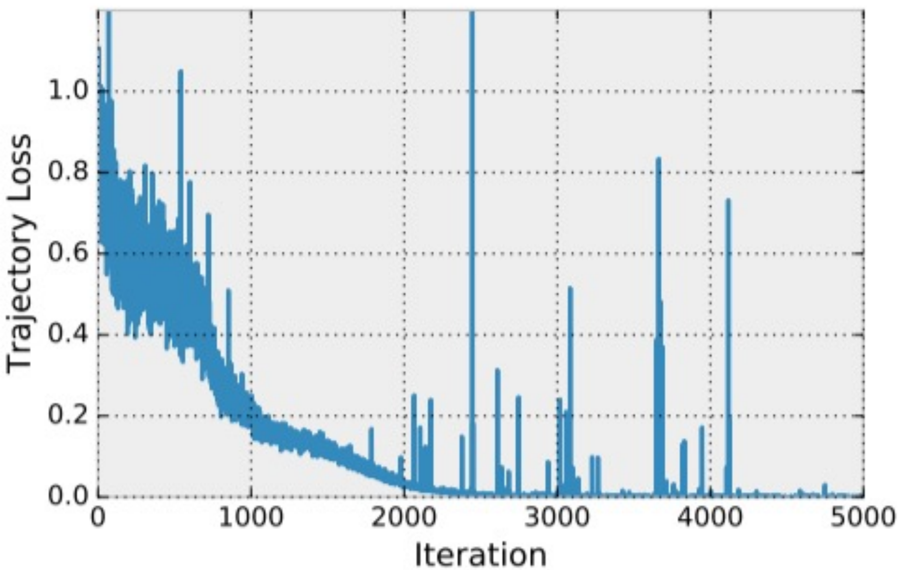
Solve MPC with SQP, differentiate through the fixed point with OptNet

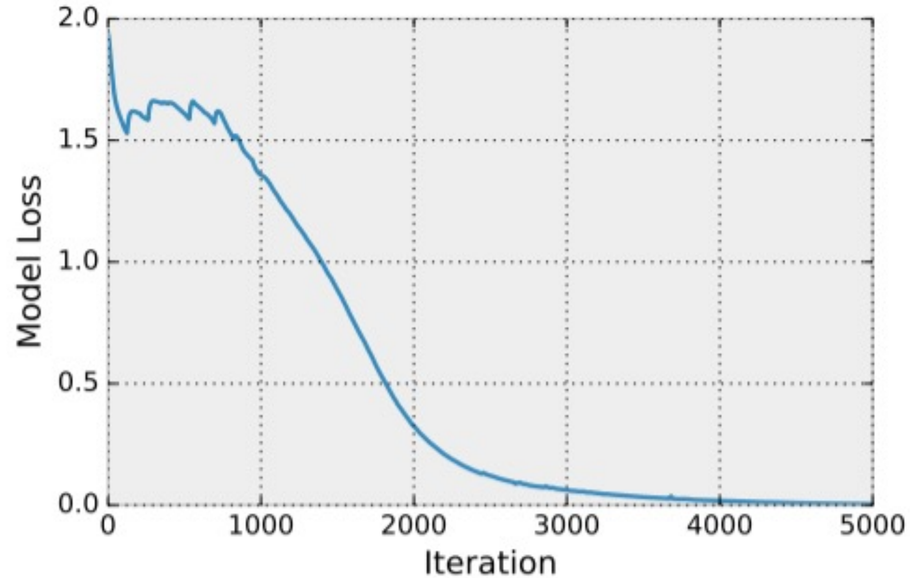
What can we do with this now?

Replace neural network policies in model-free algorithms with MPC policies, and also **replace the unrolled controllers** in other settings (hindsight plan, universal planning networks)

The cost can also be learned! No longer have to hard-code in a known value.

Imitation learning with a linear model





Some closing thoughts

(Exact) optimization is a powerful primitive to use within larger, interconnected systems

Such solvers can be propagated through and learned, just like any layer

Many applications of the technique

The general possibility of training complex end-to-end decision making systems is just getting started…

OptNet: End-to-End Differentiable Optimization

Brandon Amos | Carnegie Mellon University

bamos.github.io
brandondamos

Applications: Projections, Sudoku, Denoising, Task-based Learning, Model Predictive Control

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

Task-based End-to-end Model Learning
P. Donti, **B. Amos**, and J. Z. Kolter
NIPS 2017

https://locuslab.github.io/qpth/
https://github.com/locuslab/optnet
https://github.com/locuslab/e2e-model-learning