OptNet:

Differentiable Optimization as a Layer in Neural Networks

Brandon Amos and J. Zico Kolter
Carnegie Mellon University
School of Computer Science

ICML 2017

Big picture

What are the “atomic operations” or building blocks of modern AI systems?

View of the current situation: Matrix-vector products (dense or sparse/structured), (sub)differentiable non-linear functions, random sampling

This talk: We should consider (convex) optimization as another potential layer, to be composed with others

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

Optimization in deep learning

Recently there has been a lot of work in applying more generic optimization methods within deep learning architectures

Approach 1: *Unroll* an optimization procedure (like gradient descent) as a network itself (Domke, 2012; Goodfellow, 2013; Maclaurin et al., 2015; Belanger and McCallum, 2015; Andrychowicz et al., 2016; Metz et al., 2017; Gregor and LeCun, 2010)

Approach 2: Directly differentiate through the argmin (Bradley and Bagnell, 2009; Mairal et al., 2012; Gould et al., 2016; Johnson et al., 2016; Amos et al., 2016; Barron and Poole, 2016)

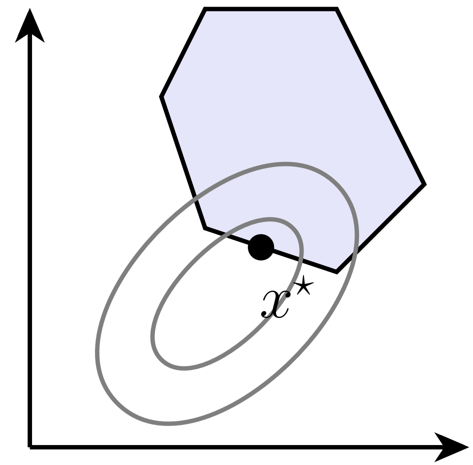
- We're going to use this approach, but consider a bit more general setting and efficient backpropagation algorithms

Optimization as a "primitive"

Optimization problems are an extremely powerful paradigm for decision-making

Example: quadratic program

$$
\begin{aligned}
\underset{x}{\text{minimize}} \quad & \frac{1}{2} x^T Q x + q^T x \\
\text{subject to} \quad & Ax = b \\
& Gx \le h
\end{aligned}
$$



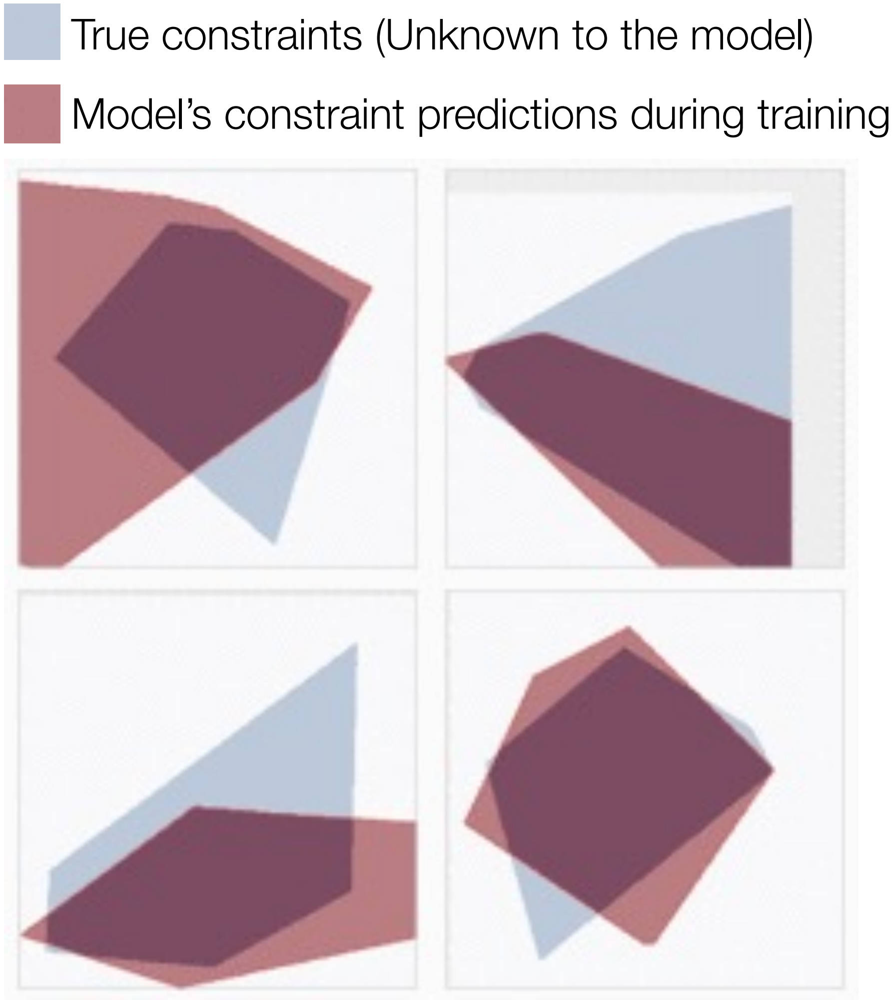
Applications in finance (Markowitz portfolio optimization), machine learning (support vector machines), control (linear-quadratic model predictive control), geometry (projections onto polyhedra)

Illustrative Example: Learning Hard Constraints

Given regression data (x, y) generated from a constrained optimization problem.

Idea: Randomly initialize hard constraints in an OptNet layer and learn them from data with gradients

$$\hat{y} = \underset{z}{\operatorname{argmin}} \ f(x, z)$$
$$\text{subject to } Gz \leq h$$



Talk Overview

1. Our contribution: OptNet layers
2. qpth: Our efficient and differentiable PyTorch QP solver
3. Experiments
    1. MNIST
    2. 1D Signal Denoising
    3. Mini-Sudoku

Our Contribution: The OptNet Approach

A network where the output of a single layer is the *solution* to a QP involving parameters defined by the previous layer z_i

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

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

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

Can capture much more expressive functions than a single traditional feedforward layer (polytope of QP has exponential number of points)

Continuous in z_i if parameters are all continuous functions, and $Q(z_i)$ strictly positive definite

Example OptNet Layer

General Definition:

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

Parameterization that is **always feasible**:

- Connect the previous layer only in the linear term $q(z_i) = z_i$
- Use a Cholesky so that $Q = LL^T + \epsilon$
- Pick some feasible point $z_0 \in \mathbb{R}$ and $s_0 > 0$ and let $b = Az_0$ and $h = Gz_0 + s_0$

Learnable parameters: $L, A, G, z_0, \text{and } s_0$

Differentiating through OptNet layers

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

How do we compute the Jacobians?

$$
\frac{\partial z_{i+1}}{\partial z_i} \quad \frac{\partial z_{i+1}}{\partial Q} \quad \frac{\partial z_{i+1}}{\partial q} \quad \frac{\partial z_{i+1}}{\partial A} \quad \frac{\partial z_{i+1}}{\partial b} \quad \frac{\partial z_{i+1}}{\partial G} \quad \frac{\partial z_{i+1}}{\partial h}
$$

We show how to compute these by using implicit differentiation of the KKT conditions with matrix differentials. (The details are in our paper)

Talk Overview

1. Our contribution: OptNet layers
2. qpth: Our efficient and differentiable PyTorch QP solver
3. Experiments
    1. MNIST
    2. 1D Signal Denoising
    3. Mini-Sudoku

qpth: Our efficient and differentiable PyTorch QP solver

OptNet formulation is slow compared to Linear+ReLU layers, even with highly optimized solvers

We 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

Nice property: We can backprop through the solver effectively "for free"

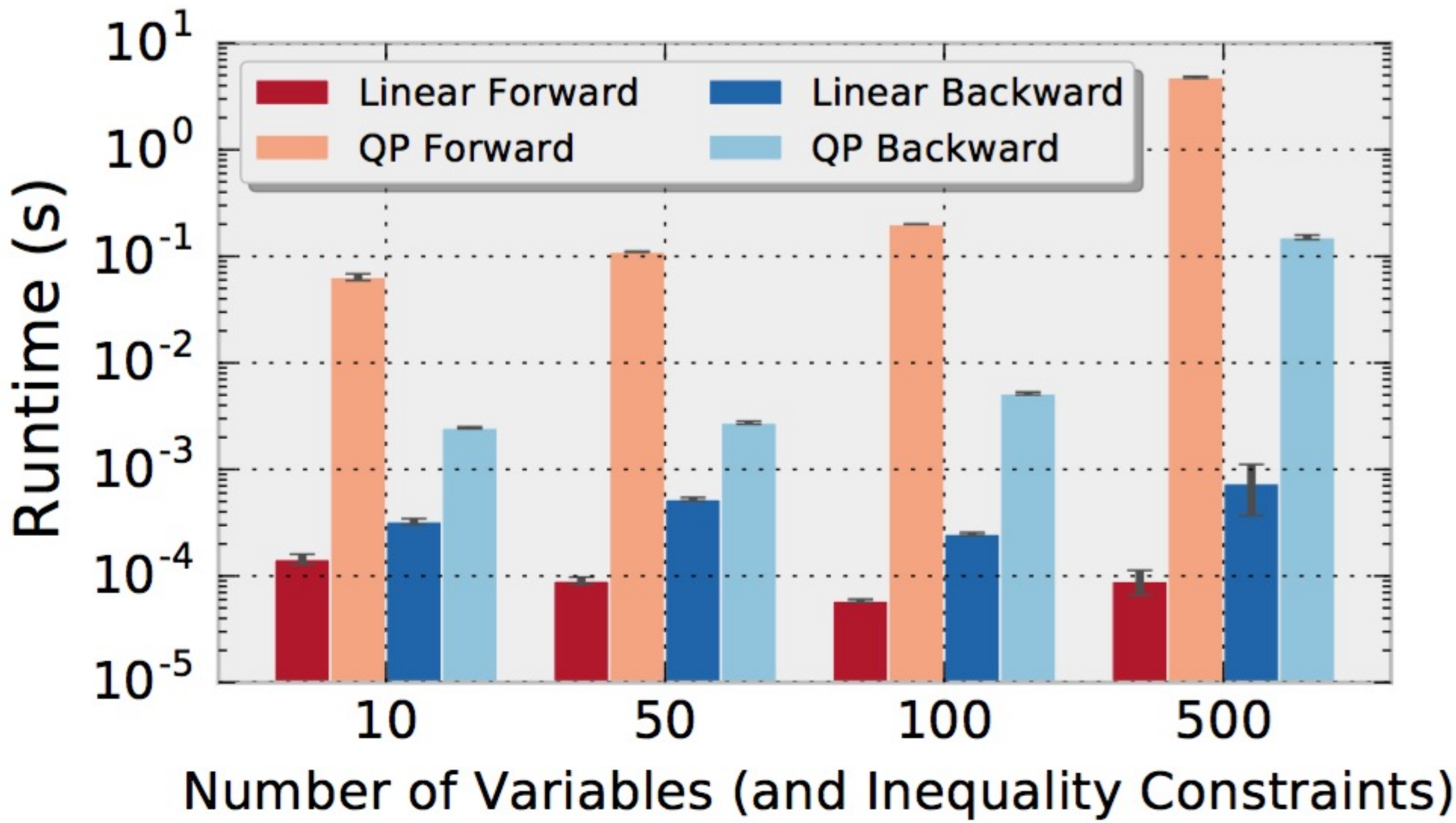
Our open source PyTorch library is available at http://locuslab.github.io/qpth

Add a differentiable QP OptNet layer to your PyTorch models with one line of code with our PyTorch `Function` after defining the parameters:

```
z = QPFunction()(Q, p, G, h, A, b)
```

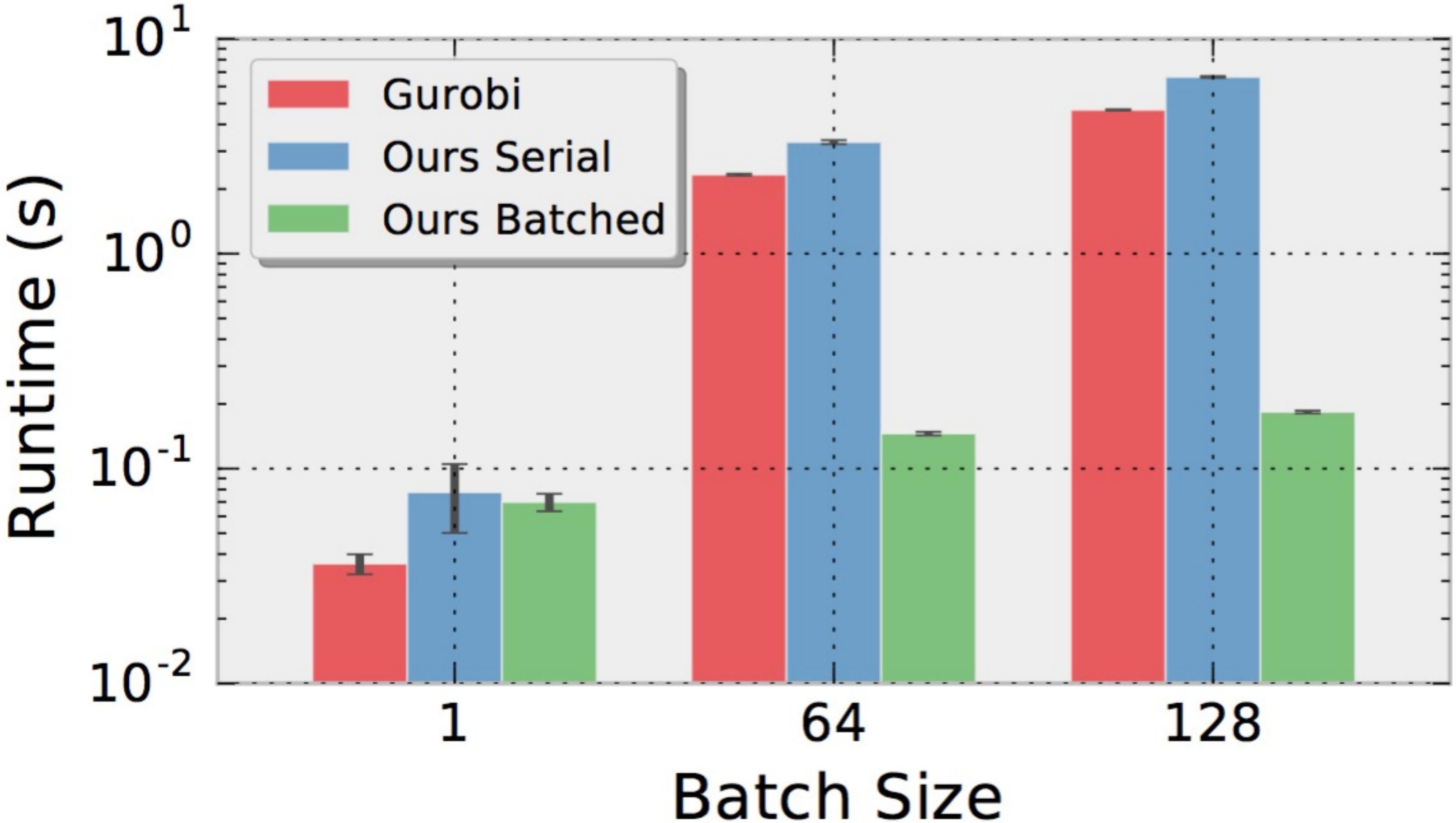
Timing Results: Comparison to a linear layer

OptNet layers are more expensive but still tractable



Timing Results: Comparison to Gurobi

Batched QP solvers are crucial for tractability

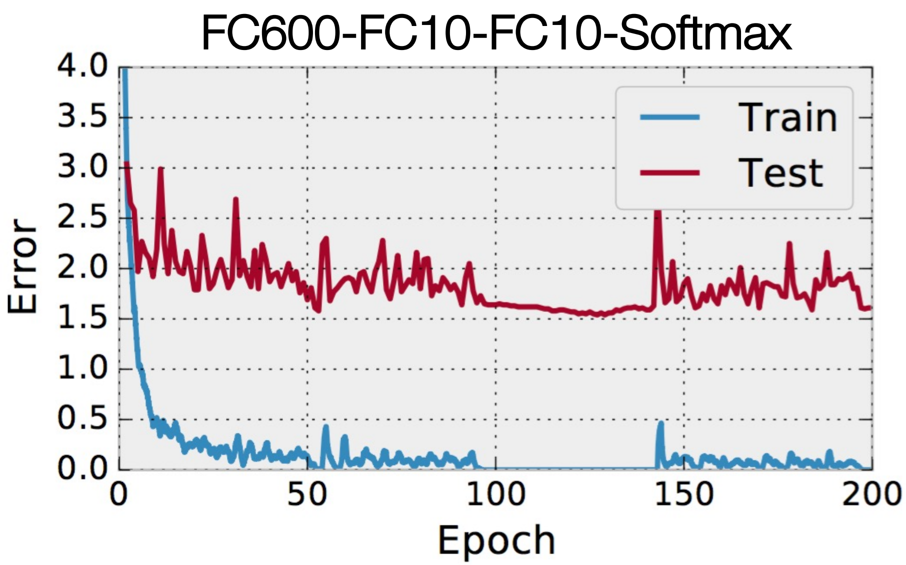


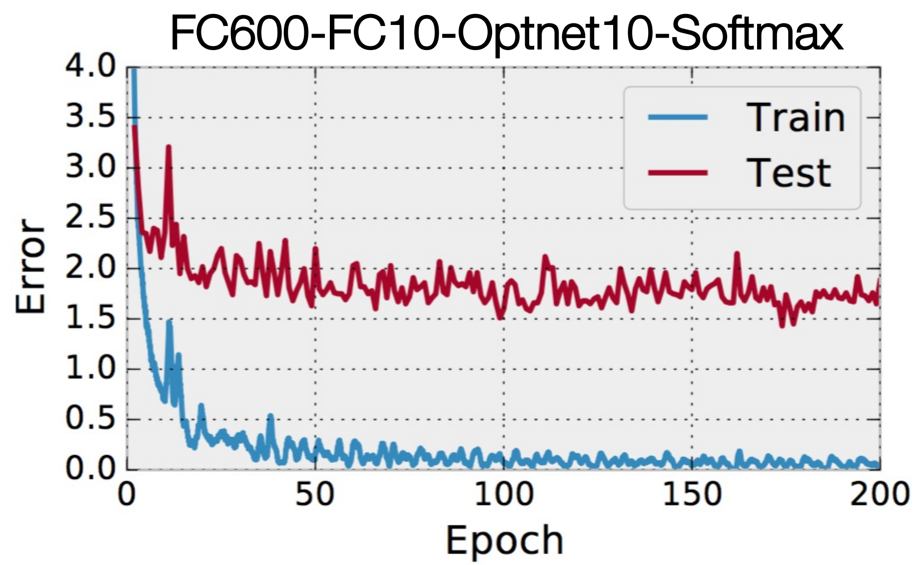
Talk Overview

1. Our contribution: OptNet layers
2. qpth: Our efficient and differentiable PyTorch QP solver
3. Experiments
    1. MNIST
    2. 1D Signal Denoising
    3. Mini-Sudoku

Results: MNIST

Only interesting as a sanity check and to show that an OptNet layer can be added as a layer without harming the training process.





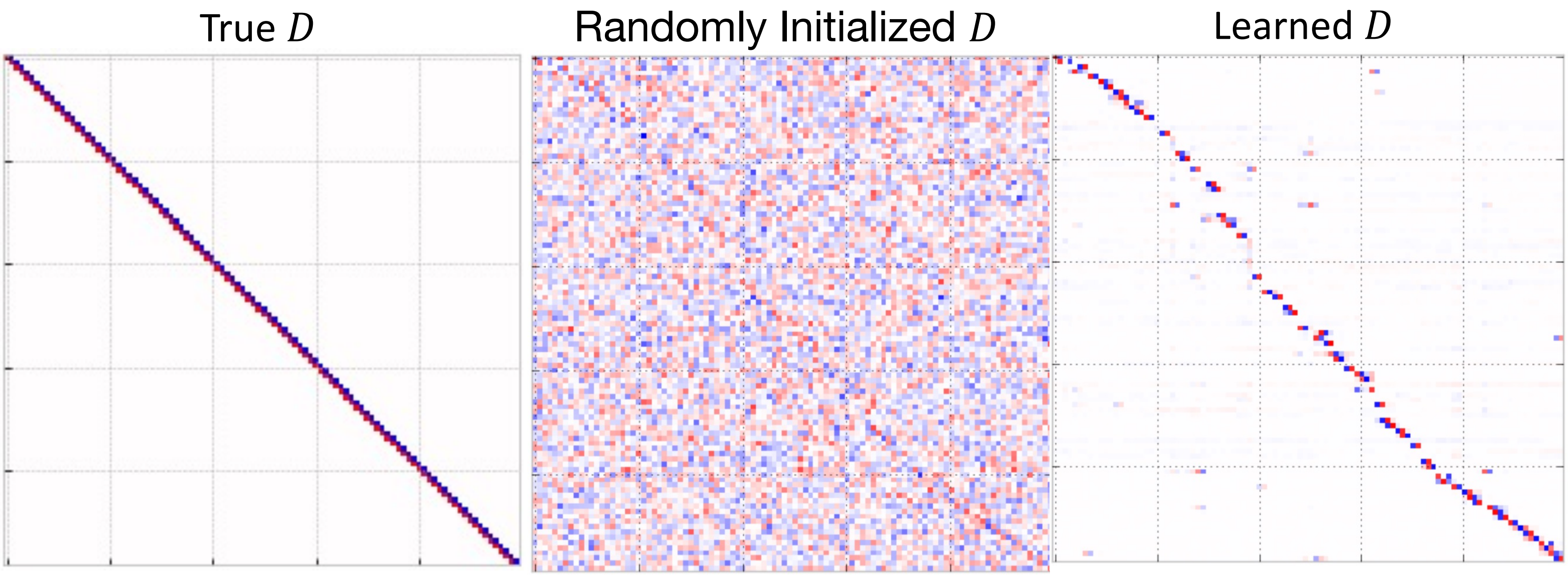
Results: 1D Signal Denoising

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.

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

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



Results: Mini-Sudoku

Sudoku can be posed as a constraint-satisfaction optimization problem

- Every row should contain the digits 1-4
- Every column should contain the digits 1-4
- Every partitioned sub-block should contain the digits 1-4

Task: Learn a model from data that maps from unsolved boards to solved boards.

Example input/output pair:

			3
1			
		4	
4			1

2	4	1	3
1	3	2	4
3	1	4	2
4	2	3	1

The OptNet Approach:

$$\begin{aligned} \operatorname*{argmin}_{z} \quad & \epsilon \|z\|_2^2 + p^T z \\ \text{subject to} \quad & Az = b \\ & z \geq 0 \end{aligned}$$

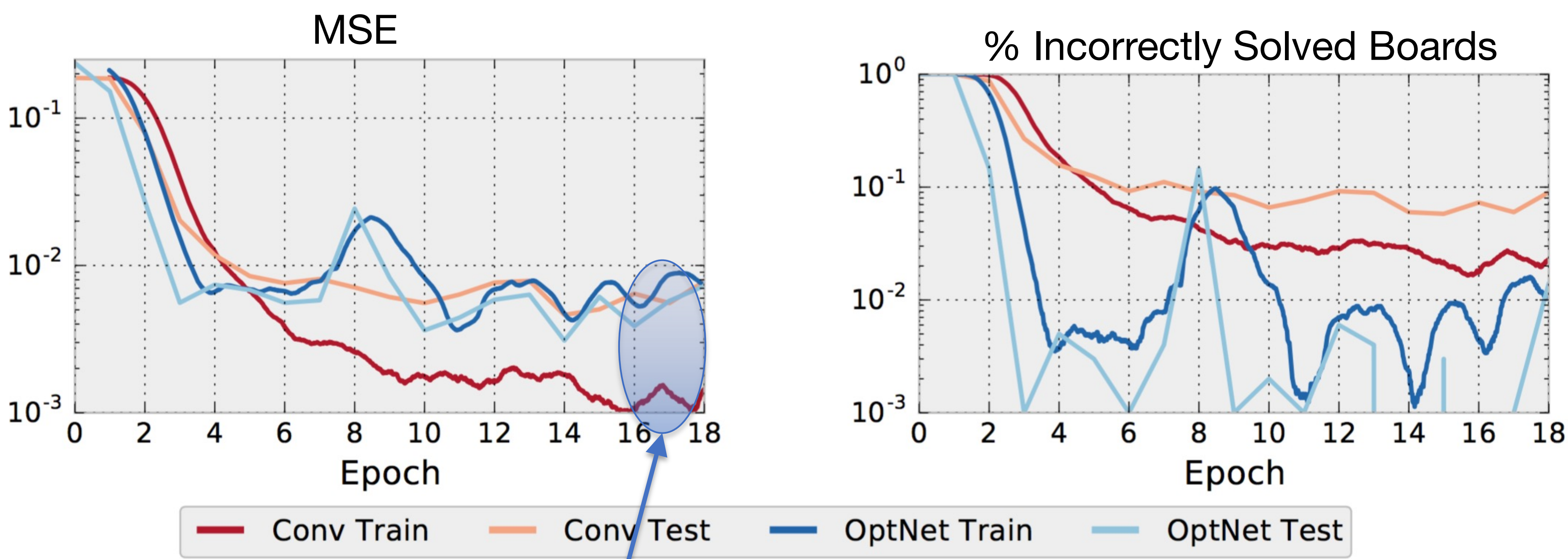
Arbitrary parameters

One-hot encoding of unsolved puzzle

Results: Mini-Sudoku

The OptNet layer exactly learns the mini-Sudoku constraints from data!

Baseline: A deep convolutional feed-forward network



Convolutional network: Significant train/test gap
OptNet: Small gap, generalizes well

OptNet:

Differentiable Optimization as a Layer in Neural Networks

Brandon Amos and J. Zico Kolter
Carnegie Mellon University
School of Computer Science

@brandondamos

@zicokolter

1. Our contribution: OptNet layers

2. qpth: Our efficient and differentiable PyTorch QP solver

3. Experiments
    1. MNIST
    2. 1D Signal Denoising
    3. Mini-Sudoku

The full PyTorch source code to reproduce all of our experiments is available online at https://github.com/locuslab/optnet

Our PyTorch QP solver is freely available online at https://locuslab.github.io/qpth