OptNet: Differentiable Optimization as a Layer in Neural Networks

A short report on [Amos and Kolter, 2017]

Nazarov Ivan

Skoltech

August 17, 2018

QP - OptNet

The output of the i-th layer is the solution of a quadratic problem

$$\begin{aligned} z_{i+1} = \arg\min_{z} \quad & \tfrac{1}{2} z^{\mathrm{T}} Q_i z + q_i^{\mathrm{T}} z \,, \\ \text{subject to} \quad & A_i z = b_i \,, G_i z \leq h_i \,, \end{aligned} \tag{QP-layer}$$

where Q_i, q_i, A_i, b_i, G_i and h_i are determined by upstream layers.

Limitations

- only small scale QP
- cubic complexity of solving the problem (forward pass)
- likely require much more tuning, since there are manifolds in the parameter space with no effect on the layer's output

High representational power

Convex optimization layers capture constraints and complex dependencies:

- e.g. projection onto a simplex

$$\begin{array}{ll} \underset{\xi \in \mathbb{R}^n}{\text{minimize}} & \frac{1}{2}\|x - \xi\|^2 \,, \\ \text{subject to} & 0 \leq \xi \,, \mathbf{1}^{\mathrm{T}} \xi = 1 \,. \end{array}$$

- e.g. represent a piecewise linear function $f : \mathbb{R}^d \to \mathbb{R}$

$$x \mapsto f(x) = \sum_{k=1}^{n} w_k \max\{A_i^{\mathrm{T}} x + b_i, 0\}\,, \tag{1}$$

with $w \in \{\pm 1\}^n$, $A \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$ as the y-part of the solution to

$$\begin{aligned} \underset{y \in \mathbb{R}, \xi \in \mathbb{R}^n}{\text{minimize}} \quad & \tfrac{1}{2}\|\xi\|^2 + \tfrac{1}{2}(y - w^{\mathrm{T}}\xi)^2\,, \\ \text{subject to} \quad & Ax + b \leq \xi\,, \end{aligned}$$

- additively stack to represent vector-valued piecewise function

Simpler Universal Approximator

A ReLU + linear network is sufficient to construct a universal function approximator, [Sonoda and Murata, 2017]

- ... may require exponentially wide ReLU layer, [Amos and Kolter, 2017]
- QP-layer trades off extra computational complexity for representational simplicity

Approaches to Optimization layers

- no general **analytic** solutions to constrained problems
- **Unrolling** first-order gradient based iterations with a barrier function requires substantially larger computational graph, more compute and memory (e.g. BPTT)
- **argmin differentiation** ...

Contributions

Authors use

- sensitivity analysis and implicit differentiation to differentiate through (QP-layer)
- develop a custom solver for multiple small QP in batch form
- make use of a clever factorization of a primal-dual interior point method to almost effortlessly backpropagate through (QP-layer)

General Convex Quadratic Problem

Linearly Constrained Quadratic Problem

$$\begin{aligned} \underset{z\in\mathbb{R}^n}{\text{minimize}} \quad & \tfrac{1}{2} z^{\mathrm{T}} Q z + q^{\mathrm{T}} z \,, \\ \text{subject to} \quad & Az = b \,, Gz \leq h \,, \end{aligned} \tag{LCQP}$$

with

- $Q \in \mathbb{R}^{n\times n}$ positive semidefinite matrix
- $q \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $G \in \mathbb{R}^{r\times n}$ and $h \in \mathbb{R}^r$

The KKT conditions

The Lagrangian for (LCQP)

$$\mathcal{L}(z;\nu,\lambda) = \tfrac{1}{2} z^{\mathrm{T}} Q z + q^{\mathrm{T}} z + \nu^{\mathrm{T}}(Az - b) + \lambda^{\mathrm{T}}(Gz - h)\,,$$

[Boyd and Vandenberghe, 2004, sec. 5.2.3]:

- for affine constraints feasibility implies strong duality
  $\Rightarrow$ saddle points of $\mathcal{L}$ solve (LCQP)

Saddle points of $\mathcal{L}$ satisfy the KKT conditions

- **Primal feasibility** $Az = b$ and $Gz \leq h$
- **Dual feasibility** $\lambda \geq 0$
- **FoC** $Qz + q + A^{\mathrm{T}}\nu + G^{\mathrm{T}}\lambda = 0$
- **Complementary slackness** $\operatorname{diag}(\lambda)(Gz - h) = 0$

Differentiating the solution

We need to differentiate the solution to (LCQP) w.r.t. the input

Strict convexity of (LCQP)
$\Rightarrow$ the saddle point (z, λ, ν) is unique and differentiable

Provided the saddle point has nice properties, we can use

- sensitivity analysis of the KKT conditions

$$\begin{aligned} Qz + q + A^{\mathrm{T}}\nu + G^{\mathrm{T}}\lambda &= 0\,, \\ Az - b &= 0\,, \\ \operatorname{diag}(\lambda)(Gz - h) &= 0\,, \end{aligned}$$

Sensitivity of the KKT

Taking matrix derivatives and collecting into a block matrix:

$$\underbrace{\begin{bmatrix} Q & A^{\mathrm{T}} & G^{\mathrm{T}} \\ \operatorname{diag}(\lambda)G & \operatorname{diag}(Gz-h) & 0 \\ A & 0 & 0 \end{bmatrix}}_{K} \begin{bmatrix} \partial z \\ \partial \lambda \\ \partial \nu \end{bmatrix} = \overbrace{\begin{bmatrix} -(\partial Q)z - \partial q - (\partial G)^{\mathrm{T}}\lambda - (\partial A)^{\mathrm{T}}\nu \\ \operatorname{diag}(\lambda)\partial h - \operatorname{diag}(\lambda)(\partial G)z \\ \partial b - (\partial A)z \end{bmatrix}}^{\chi}$$

The full differential

The full differential of the saddle point w.r.t. the parameters is

$$\begin{bmatrix} \partial z \\ \partial \lambda \\ \partial \nu \end{bmatrix} = K^{-1} \chi \,, \tag{2}$$

where

- K is the left-hand of the KKT sensitivity matrix
- χ is a differential form w.r.t. ∂Q, ∂b, etc.

$\Rightarrow$ Never compute this explicitly for efficient backprop

Backpropagation

Backpropagation uses chain rule with right-side products **only**

$$\nabla_\theta L(H(\theta)) = \underbrace{\left(\frac{\partial H(\theta)}{\partial \theta^{\mathrm{T}}}\right)^{\mathrm{T}}}_{J_\theta^H} \nabla_z L(z)\big|_{z=H(\theta)},$$

- $\nabla_z L$ is an $m \times 1$ downstream gradient
- J_θ^H is the Jacobian of the current layer $H\colon \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ w.r.t. a input / parameter θ

The full differential of $L(H(\theta))$ where H is (LCQP) is

$$\partial_\theta L(H(\theta)) = \chi^{\mathrm{T}} K^{-\mathrm{T}} \begin{pmatrix} \nabla_z L(z) \\ 0 \\ 0 \end{pmatrix} \Bigg|_{z=H(\theta)},$$

QP-Layer: forward & backward passes

Amounts to solving the (QP-layer)

- for batch size 1 any method / library shall do
- what about batched solving QP-layer in parallel?

The batched primal-dual interior point QP solver

- specialized solver for batches of quadratic problems (LCQP)
- LU-factorizes K as a by-product of iterative solution
  - quadratic instead of cubic complexity

Implemented in PyTorch

- `https://github.com/locuslab/qpth`

QP solver performance

- significantly slower then a fully connected layer
- faster than non-batched specialized existing solvers

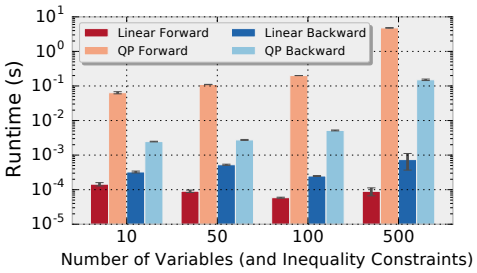


Figure 1: Performance of a linear layer and a QP layer. (Batch size 128)

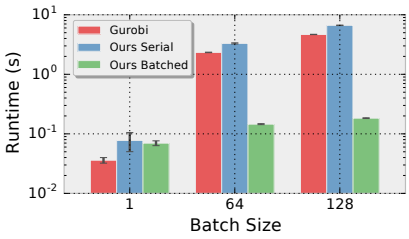


Figure 2: Performance of Gurobi and our QP solver.

Total variation signal denoising layer

Smooth some noisy signal y by solving the problem

$$\underset{z \in \mathbb{R}^n}{\text{minimize}} \quad \tfrac{1}{2}\|y - z\|^2 + \lambda \left\| Dz \right\|_1 ,$$

where D is the $(n-1) \times n$ first difference matrix.

The problem is equivalent to the following LCQP

- Equivalent to

$$\begin{aligned} \underset{z \in \mathbb{R}^n, \xi \in \mathbb{R}^{n-1}}{\text{minimize}} \quad & \tfrac{1}{2}\|y - z\|^2 + \lambda \mathbf{1}^{\mathrm{T}} \xi , \\ \text{subject to} \quad & \begin{pmatrix} D & -I \\ -D & -I \end{pmatrix} \begin{pmatrix} z \\ \xi \end{pmatrix} \leq 0 , \end{aligned}$$

Scenaria

- Total variation denoising with OptNet
- Learning with a fully-connected neural network (FC Net)
- Learning the differencing operator with OptNet from scratch (Pure OptNet)
- Seeding OptNet with the a noisy D (OptNet Tuned TV)

Method	Train MSE	Test MSE
Total Variation	16.3	16.5
FC Net	18.5	29.8
Pure OptNet	52.9	53.3
OptNet Tuned TV	13.8	**14.4**

Table 1: Denoising task error rates.

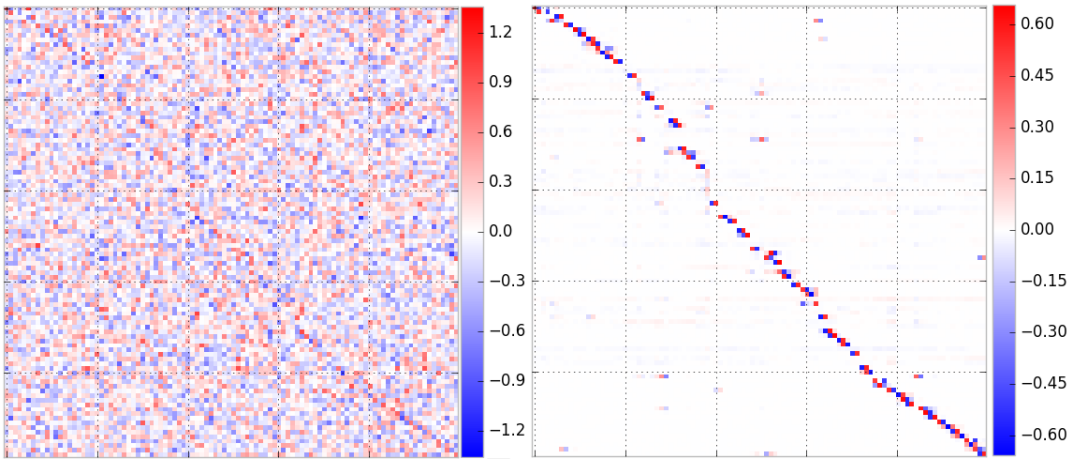


Figure 3: Initial and learned difference operators for denoising.

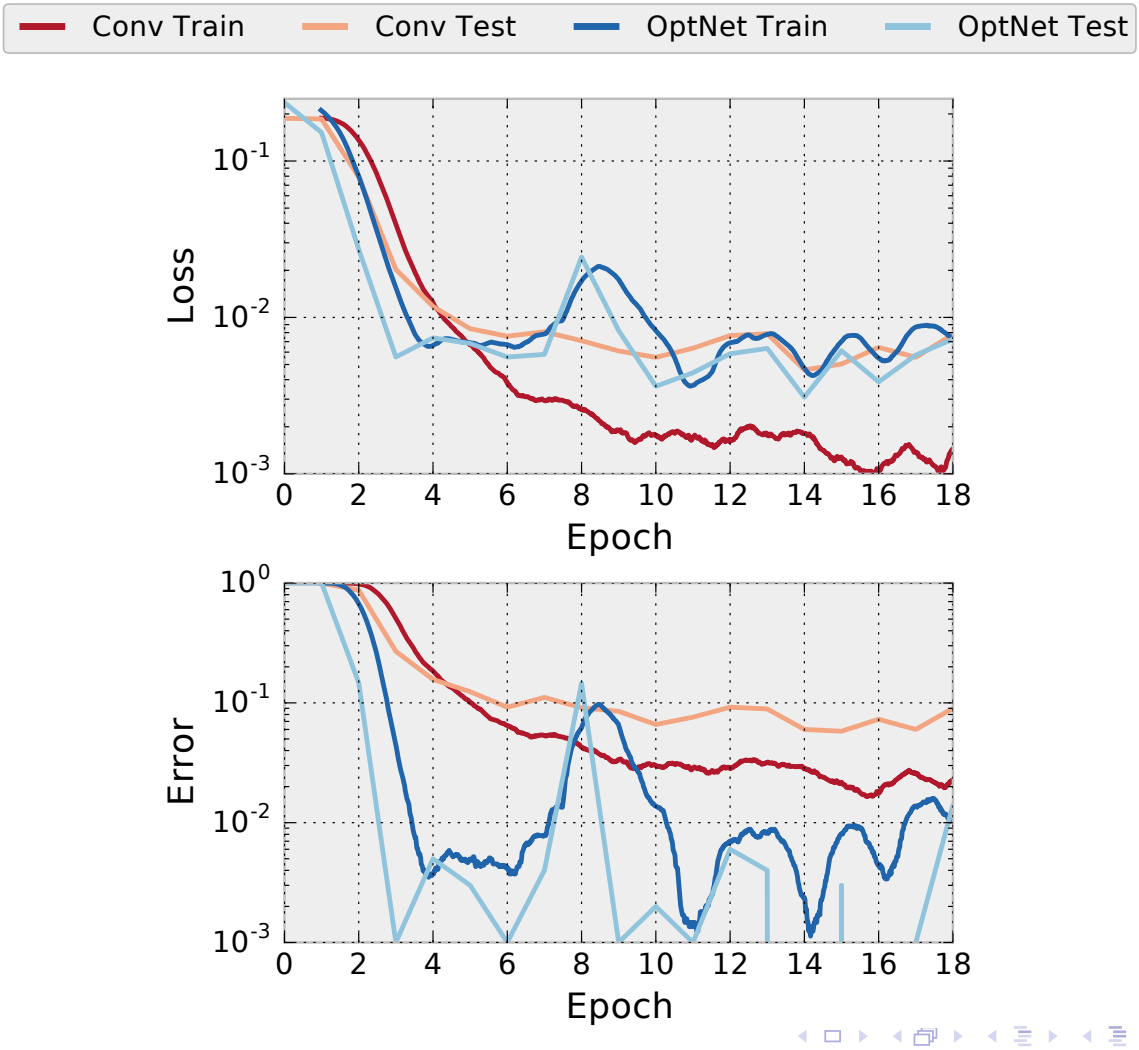
4×4 Sudoku

Sudoku is essentially a constraint satisfaction problem

Neural solver trained on pairs of unsolved and solved puzzles

- deep CNN: 10 conv layers with $512 \times 3 \times 3$ filters
- OptNet QP-layer with
  - positivity inequality constraints
  - arbitrary learnt constraint matrix $Ax = b$
  - $Q = 0.1I$ for strict convexity and feasible
  - q is the input one-hot encoding of the Sudoku problem

4×4 Sudoku



MNIST

Compare on MNIST

- `fc600-fc10-fc10-softmax`
- `fc600-fc10-qp10-softmax`
  - linear constraints determined by upstream layers
  - other parameters of the LCQP freely learnt (C in $Q = CC^{\mathrm{T}}$)

- **Failure** QP OptNet gives only marginal improvement

References


Amos, B. and Kolter, J. Z. (2017).
Optnet: Differentiable optimization as a layer in neural networks.
arXiv preprint arXiv:1703.00443.


Boyd, S. and Vandenberghe, L. (2004).
Convex Optimization.
Cambridge University Press, New York, NY, USA.


Mattingley, J. and Boyd, S. (2012).
Cvxgen: A code generator for embedded convex optimization.
Optimization and Engineering, 13(1):1–27.


Sonoda, S. and Murata, N. (2017).
Neural network with unbounded activation functions is universal approximator.
Applied and Computational Harmonic Analysis, 43(2):233 – 268.


Sra, S., Nowozin, S., and Wright, S. J. (2011).
Optimization for Machine Learning.
The MIT Press.