Report project of Deep Learning: Automatic differentiation of optimization solutions

Mohamed Fyras Telmini

January 28, 2024

Abstract

This report serves to explain the different results of papers [2] and [1]. [2] treats exact, constrained optimization as an individual layer within a deep learning architecture. It presents OptNet, a network architecture that integrates optimization problems (specifically in the form of quadratic programs) as individual layers in larger end-to-end trainable deep networks. These layers encode constraints and complex dependencies between the hidden states that traditional convolutional and fully-connected layers often cannot capture. [1] discusses the problem of convex cone programs and the methods to solve it. Namely via differentiation of the cone program. They also propose an efficient implementation of their approach, which can be used for automatic differentiation.

1 OptNet: Differentiable Optimization as a Layer in Neural Networks

The paper treats the following optimization problem as an individual layer within a neural network:

$$\begin{aligned} z_{i+1} = \arg\min_z \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z \\ \text{subject to } A(z_i)z = b(z_i), G(z_i)z \leq h(z_i) \end{aligned} \tag{1}$$

The output of the $i+1$-th layer is the solution to a constrained optimization problem based upon previous layers. The optimization variable is z, and $Q(z_i)$, $q(z_i)$, $A(z_i)$, $b(z_i)$, $G(z_i)$, and $h(z_i)$ are parameters of the optimization problem. These parameters can depend in any differentiable way on the previous layer z_i, and can eventually be optimized just like any other weights in a neural network. Also, this is exactly the forward pass of the network, as it only requires solving the optimization problem for z.
For the backwards pass, it requires the computation of the derivative of the solution to the problem with respect to its input parameters. This is done with using matrix differential calculus on the differentiation of the KKT conditions at a solution to the optimization problem.

1.1 Backwards pass calculation

The first step in the calculation of the various derivatives needed to satisfy the backwards pass of the network is to determine the Lagrangian of the problem (1). It's obtained directly as:

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

With ν and $\lambda \geq 0$ the dual variables that correspond to the equality and inequality constraint respectively. The Lagrangian allows the writing of the KKT optimum conditions as:

$$\begin{aligned} Qz^* + q + A^T\nu^* + G^T\lambda^* = 0 \\ Az^* - b = 0 \\ Diagonal(\lambda^*)(Gz^* - h) = 0 \end{aligned} \tag{3}$$

Where λ^*,ν^* and z^* are the optimal primal and dual variables. The differentials of (3) are expressed in matrix form as follows:

$$\begin{bmatrix} Q & G^T & A^T \\ Diagonal(\lambda^*)G & Diagonal(Gz^* - h) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} dz \\ d\lambda \\ d\nu \end{bmatrix} = - \begin{bmatrix} dQz^* + dq + dG^T\lambda^* + dA^T\nu^* \\ Diagonal(\lambda^*)dGz^* - Diagonal(\lambda^*)dh \\ dAz^* - db \end{bmatrix} \tag{4}$$

This formulation is sufficient to express the desired back-propagation vectors, by writing the identity:

$$\begin{bmatrix} d_z \\ d_\lambda \\ d_\nu \end{bmatrix} = - \begin{bmatrix} Q & G^T Diagonal(\lambda^*) & A^T \\ G & Diagonal(Gz^* - h) & 0 \\ A & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} (\frac{\partial l}{\partial z^*})^T \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

The computable back-propagation vectors are then:

$$\begin{aligned} \nabla_Q l &= \frac{1}{2}(d_z z^{*T} + z^* d_z^T) \\ \nabla_q l &= d_z \\ \nabla_A l &= d_\nu z^{*T} + \nu^* d_z^T \\ \nabla_b l &= -d_\nu \\ \nabla_G l &= Diagonal(\lambda^*) d_\lambda z^{*T} + \lambda^* d_z^T \\ \nabla_h l &= -Diagonal(\lambda^*) d_\lambda \end{aligned} \tag{6}$$

1.2 Batched quadratic program solver

The paper introduces another contribution, that is aimed to solve (6) by exploiting the parallelizations that take place in usual neural network architectures via GPU training. The authors implement a batched Primal-Dual inerior point solver. The theory behind the solver is not novel but the implementation that they propose is, since it allows for more efficient computations through parallelization. The solver does not directly solve (6), since its a quadratic program solver, it solves (1). The solutions for (6) are then obtained directly from that compuation, which allows for a very efficient backward-pass calculation. The authors explain that the only meaningful calculation bottleneck is the forward-pass, specifically during the KKT matrix factorization which computes in cubic time.

1.3 Theoretical properties of the network

The authors introduce 3 theorems aimed at demonstrating certain key properties of this network, which are:

1. OptNet layers are subdifferentiable everywhere and have a single unique element (the Jacobian) for all but a measure-zero set of points. Which means that the network converges to the correct and unique solution of the quadratic problem.

2. OptNet layers can approximate arbitrary elementwise piecewise-linear functions and can represent a ReLU layer using O(nk) parameters, where k is the number of linear regions. Which demonstrates that they can generalize common neural networks.

3. There are functions representable by an OptNet layer which cannot be represented exactly by a two-layer ReLU layer. Which shows that this architecture offers an advantage in representational power when compared to common neural networks.

Additionally, the authors note that while OptNet layers have several strong points, there are also potential drawbacks to the approach. One limitation is that solving optimization problems exactly, as is done in OptNet layers, has cubic complexity in the number of variables and/or constraints, which contrasts with the quadratic complexity of standard feed-forward layers. This means that the number of hidden variables in an OptNet layer cannot be too large for practical purposes. Another limitation is that there are many improvements to OptNet layers that are still possible, such as incorporating sparse matrix methods into the fast custom solver to exploit sparsity in input data for efficiency. Finally, while OptNet layers can be trained just like any neural network layer, they may require more tuning to work due to manifolds in the parameter space that have no effect on the resulting solution.

1.4 Experiments

The authors conducted various experiments aimed at showing the usefulness and potential of this approach. These experiments are as follows:

1. **Batch quadratic program (QP) solver performance:** This experiment compares the performance of a linear layer to a QP OptNet layer with varying batch sizes and input dimensions. The results show that while the OptNet layer is significantly slower than the linear layer, it is still tractable in many practical contexts. The experiment also compares the performance of Gurobi and CPLEX, two standard non-batch solvers, to the custom batched QP solver used in OptNet. The results show that, Although it underperforms on a single batch, the custom batched solver outperforms Gurobi and CPLEX for reasonable batch sizes.

2. **Total variation denoising:** This experiment studies how OptNet can be used to improve upon signal processing techniques that currently use convex optimization as a basis. Specifically, the goal is to denoise a noisy 1D signal given training data consisting of noisy and clean signals generated from the same distribution. The experiment compares four approaches: total variation denoising, learning with a fully-connected neural network, learning the differencing operator using an OptNet layer, and fine-tuning and improving the total variation solution using an OptNet layer. The results show that the fine-tuned OptNet approach is able to improve upon the total variation solution by 12

3. **MNIST**: This experiment illustrates that OptNet layers can be included in existing architectures and that gradients can be efficiently propagated through the layer. The experiment compares the performance of a fully-connected feedforward network with and without an OptNet layer on the MNIST dataset. This showed marginal (if any) improvements over the standard fully-connected networks. The authors

claim that this experiment mainly shows that the OptNet is able to be included in usual neural network architecture, and that it can backpropagate efficiently.

4. **Sudoku:** This experiment presents an illustrative example of the representational power of OptNet by using it to learn the game of Sudoku, concidered to be a constraint satisfaction problem. The experiment compares two approaches: a multilayer feedforward network and an OptNet network. The OptNet explicitely tries to capture the contrained optimization problem implicit in the Sodoku game. The results show that while the feedforward network is able to learn all of the necessary logic for the task, it ends up overfitting to the training data. In contrast, the OptNet network learns most of the correct hard constraints and is able to generalize much better to unseen examples.

2 Differentiating Through a Conic Program

In this article, the authors treat a second kind of optimization problems: conic optimization. Which is characterized by a linear objective function that's minimized over the intersection of a subspace and a convex cone. Formally, these problems are written in a Primal (P) and Dual (D) form as such:

$$\begin{aligned} &\text{(P)} \qquad &\min_{x} \quad & c^T x \\ && \text{subject to} \quad & Ax + s = b \\ &&& s \in K \\ &\text{(D)} \qquad &\min_{y} \quad & b^T y \\ && \text{subject to} \quad & A^T y + c = 0 \\ &&& y \in K^* \end{aligned} \tag{7}$$

$x \in R^n$ is the primal variable and $y \in R^m$ is the dual variable. $s \in R^m$ is the primal slack variable. $K \subseteq R^m$ is a nonempty, closed, convex cone with dual cone $K^* \subseteq R^m$. $A \in R^{m \times n}$, $b \in R^m$, and $c \in R^n$ are the problem data. The solution (x,y,s) is defined as:

$$Ax + s = b, \quad A^T y + c = 0, \quad s \in K, \quad y \in K^*, \quad s^T y = 0 \tag{8}$$

Which is directly related to a solution map S, which maps (A,b,c) to (x,y,s). S can be seen as a composition of 3 functions $\phi \circ s \circ Q$, where

- $Q : R^{m \times n} \times R^m \times R^n \to Q$ maps the problem data to the corresponding skew-symmetric matrix in Q,
- $s : Q \to R^N$ furnishes a solution of the homogeneous self-dual embedding, and
- $\phi : R^N \to R^n \times R^m \times R^m$ maps a solution of the homogeneous self-dual embedding to the solution of the primal-dual pair (1).

Where

$$Q = \begin{bmatrix} 0 & A^T & c \\ -A & 0 & b \\ -c^T & -b^T & 0 \end{bmatrix} \tag{9}$$

The point-wise derivative of S is obtained via the chain rule:

$$d_S(A, b, c) = d_\phi(z) d_s(Q) d_Q(A, b, c) \tag{10}$$

The authors then continue by expliciting forms of the various derivatives needed, through a usage of mathematical theorems that i struggeled to really understand.

2.1 Implementation

The authors detail how to form the derivative of a cone program as an abstract linear map. They discuss how to compute the derivative and its adjoint, which involves computing projections onto convex cones, solving a linear system, and exploiting sparsity. They suggest using LSQR to solve the linear system when it is large. The authors also provide a Python implementation of their method. I refrain from writing the various steps of this approach, as I'm not comfortable re-writing mathematics that i don't understand well.

3 Experiments

All experiments are reported on the notebook: DL-Work.ipynb

3.1 Fashion MNIST

This experiment reproduces the MNIST experiment in [2] using the Fashion-MNIST dataset instead, which is concidered to be a harder yet equivalent dataset to MNIST. The performance of the model with and without the OptNet layer is reported.

3.2 Por qué no los dos?

Since [1] lacks any extensive experiments, the idea that came to my mind was to implement the solver it provides into OptNet and see if there are any improvements in performance.

Bibliography

[1] Akshay Agrawal et al. "Differentiating Through a Conic Program". In: Apr. 2019.

[2] Brandon Amos and J. Zico Kolter. "OptNet: Differentiable Optimization as a Layer in Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, June 2017, pp. 136–145. URL: https://proceedings.mlr.press/v70/amos17a.html.