Differentiable Convex Optimization Layers

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J. Zico Kolter

Neural Information Processing Systems, 2019

Convex optimization layers

- A convex optimization problem can be viewed as a function mapping a parameter \( \theta \in \mathbb{R}^n \) to a solution \( x(\theta) \); this map is sometimes differentiable.
- Prior work has shown how to differentiate through convex cone programs.
- We show how to differentiate through high-level descriptions of convex optimization programs, specified in a domain-specific language for convex optimization.
- We implement our method in CVXPY, TensorFlow 2.0, and PyTorch.

Solution map

We represent a parametrized disciplined convex program as the composition \( R \circ C \):  
- The canonicalizer \( C \) converts a DCP-compliant program to the problem data for a convex cone program.
- The solver \( s \) solves a convex cone program.
- The retriever \( R \) retrieves a solution for the original program.

We mildly restrict DCP to ensure that \( C \) and \( R \) are affine. This means that we can differentiate through the DSL, without explicitly backpropagating through it.

Disciplined parametrized programming

We introduce disciplined parametrized programming (DPP). DPP programs have the form

\[
\begin{align*}
\text{minimize} & \quad f_0(x, \theta) \\
\text{subject to} & \quad f_i(x, \theta) \leq f_i(x, \theta), \quad i = 1, \ldots, m_1, \\
& \quad g_i(x, \theta) = \tilde{g}_i(x, \theta), \quad i = 1, \ldots, m_2,
\end{align*}
\]

- \( \theta \in \mathbb{R}^n \) is a parameter.
- \( f_i \) are convex, \( \tilde{f}_i \) are concave
- \( g_i \) and \( \tilde{g}_i \) are affine

DPP is a subset of DCP that does parameter-dependent analysis:
- parameters are treated as affine
- the product of a parameter-affine and parameter-free expression is affine

DPP guarantees that \( C \) and \( R \) are affine.

Differentiation

The adjoint of the derivative of a disciplined parametrized program is

\[
\begin{align*}
D^T S(\theta) &= D^T C(\theta) D^T S(A, b, c) D^T R(\tilde{z}^T).
\end{align*}
\]

Because \( C \) and \( R \) are affine, \( D^T C \) and \( D^T R \) are easy to compute. We use prior work to differentiate through \( s \) [1].

Experiments

Figure 1: Gradients (black lines) of the training set with respect to the Adam optimizer.

Figure 2: Per-iteration cost while learning an ADP policy for stochastic control.

References
