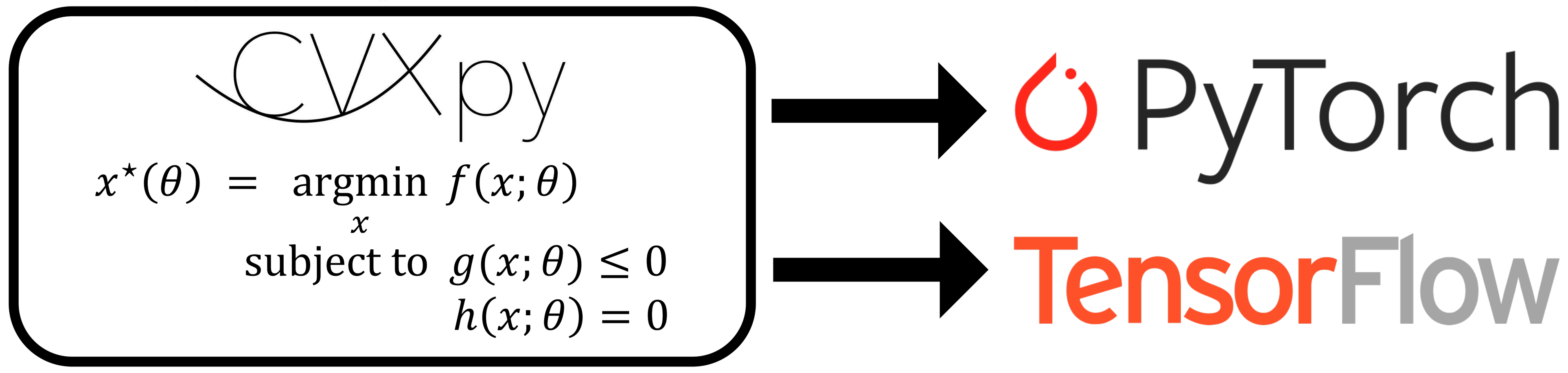
Differentiable Convex Optimization Layers

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

Neural Information Processing Systems, 2019

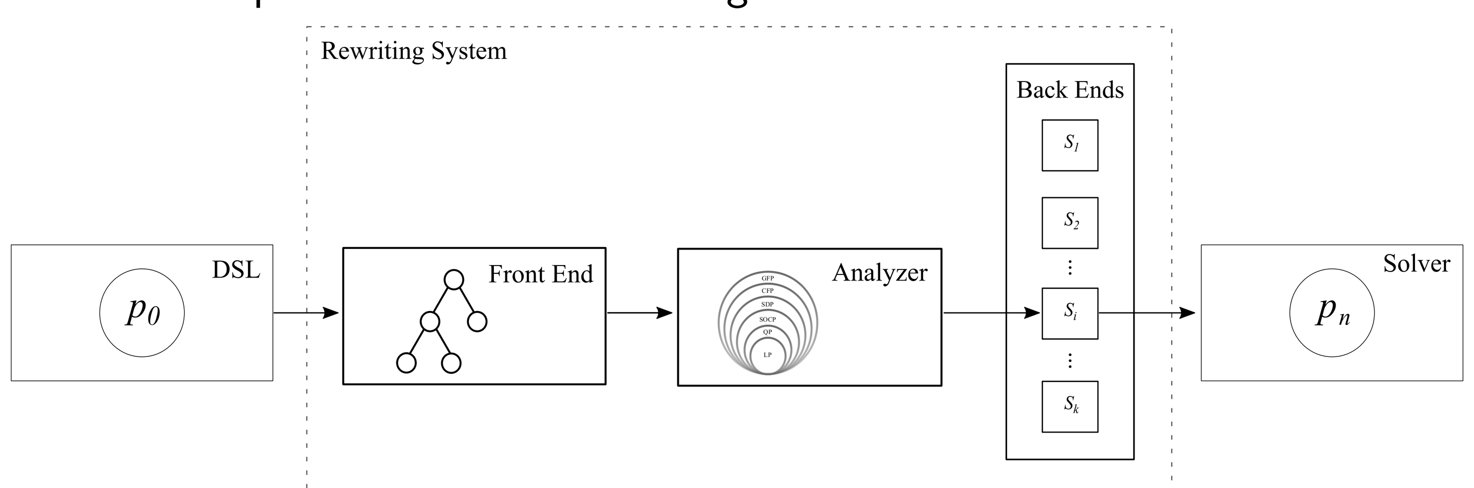


Convex optimization layers

- A convex optimization problem can be viewed as a function mapping a parameter $\theta \in \mathbf{R}^p$ to a solution $x^\star(\theta)$; 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.

Domain-specific languages (DSLs)

- DSLs for convex optimization make it easy to specify, solve convex problems
- Modern DSLs (CVXPY, CVXR, Convex.jl, CVX) based on disciplined convex programming (DCP) [2].
- DCP is a library of functions (atoms) with known curvature and monotonicity, and a composition rule for combining them.



Solution map

We represent a parametrized disciplined convex program as the composition $R \circ s \circ C$:

- 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 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

$$\begin{array}{ll} \text{minimize} & f_0(x,\theta) \\ \text{subject to} & f_i(x,\theta) \le \tilde{f}_i(x,\theta), \quad i = 1,\ldots,m_1, \\ & g_i(x,\theta) = \tilde{g}_i(x,\theta), \quad i = 1,\ldots,m_2, \end{array}$$

- $\theta \in \mathbf{R}^p$ 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

$$\mathsf{D}^T\mathcal{S}(\theta) = \mathsf{D}^T C(\theta)\mathsf{D}^T s(A,b,c)\mathsf{D}^T R(\tilde{x}^\star).$$

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

cvxpylayers

Available at www.github.com/cvxgrp/cvxpylayers.

Specify a problem using CVXPY 1.1:

```python
import cvxpy as cp

m, n = 20, 10
x = cp.Variable((n, 1))
F = cp.Parameter((m, n))
g = cp.Parameter((m, 1))
lambd = cp.Parameter((1, 1), nonneg=True)
objective_fn = cp.norm(F @ x - g) + lambd * cp.norm(x)
constraints = [x >= 0]
problem = cp.Problem(cp.Minimize(objective_fn), constraints)
assert problem.is_dpp()
```

Convert CVXPY problem to a PyTorch layer:

```python
from cvxpylayers.torch import CvxpyLayer

layer = CvxpyLayer(problem, parameters=[F, g, lambd], variables=[x])
```

Differentiate:

```python
import torch

F_t = torch.randn(m, n, requires_grad=True)
g_t = torch.randn(m, 1, requires_grad=True)
lambd_t = torch.rand(1, 1, requires_grad=True)
x_star, = layer(F_t, g_t, lambd_t)
x_star.sum().backward()
```

Experiments

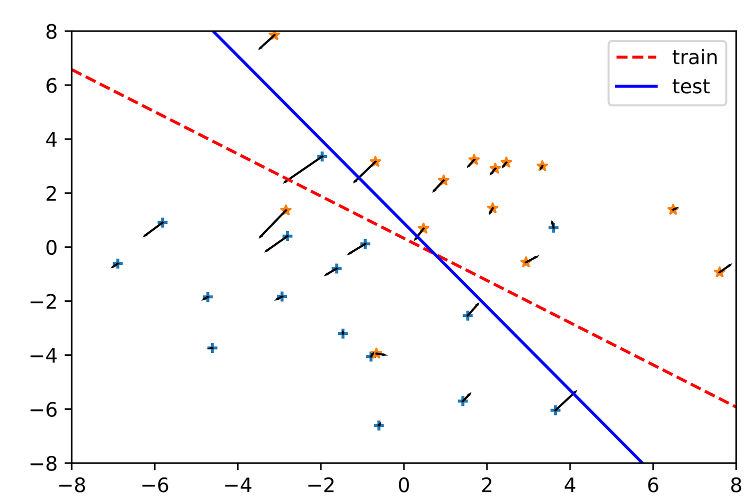


Figure 1: Gradients (black lines) of the logistic test loss with respect to the training data.

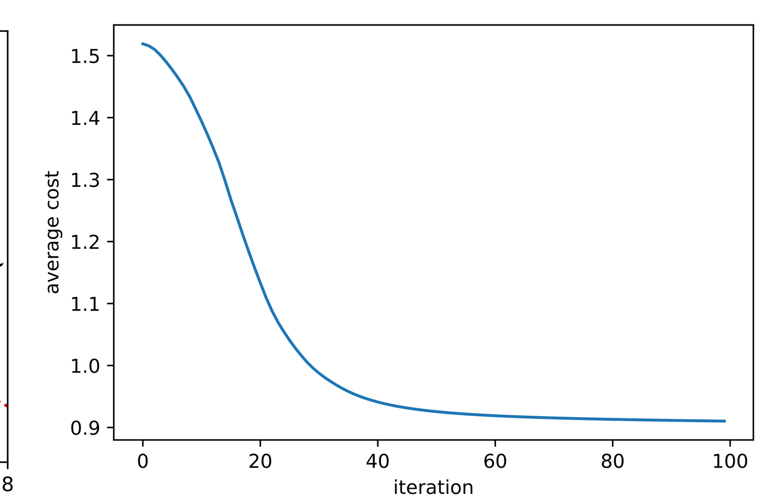


Figure 2: Per-iteration cost while learning an ADP policy for stochastic control.

References

[1] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. Moursi. Differentiating through a cone program. In: *Journal of Applied and Numerical Optimization* 1.2 (2019), pp. 107–115.

[2] M. Grant, S. Boyd, and Y. Ye. Disciplined convex programming. In: *Global optimization*. Springer, 2006, pp. 155–210.