Differentiable Convex Optimization Layers

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

Stanford University Carnegie Mellon University Facebook AI

Convex optimization

Convex optimization problems

$$
\begin{array}{ll}
\text{minimize} & f(x;\theta) \\
\text{subject to} & g(x;\theta) \leq 0 \\
& A(\theta)x = b(\theta)
\end{array}
$$

with variable $x \in \mathbf{R}^n$

- objective and inequality constraints $f_0, \ldots, f_m$ are convex

  i.e., graphs of f_i curve upward
- equality constraints are linear
- find a value for x that minimizes objective, while satisfying constraints

Why convex optimization?

- beautiful, fairly complete, and useful theory
- solution algorithms that work well, in theory and practice
- **many applications** in
  - machine learning, statistics
  - control
  - signal, image processing
  - networking
  - engineering design
  - finance

. . . and many more

How do you solve a convex optimization problem?

use someone else's ('standard') solver

- your problem *must* be written in a standard form
- analogous to writing machine code

write your own (custom) solver

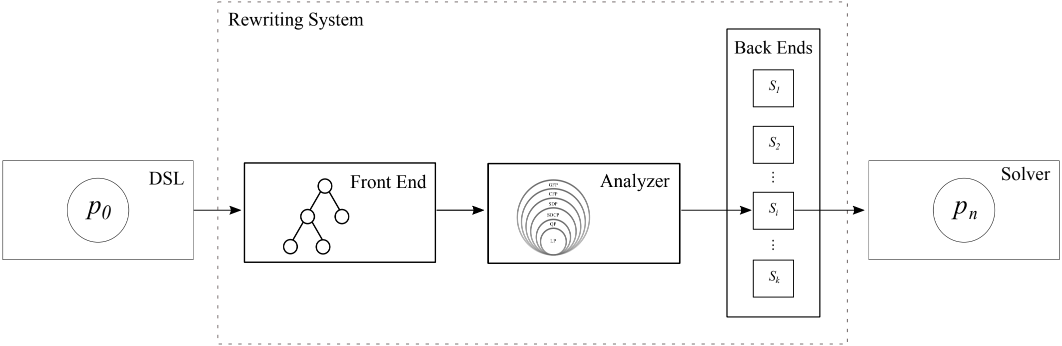
- lots of work, but can take advantage of special structure

use a domain-specific language

- transforms user-friendly format into solver-friendly standard form
- extends reach of problems solvable by standard solvers

Domain-specific languages (DSLs)

- DSLs make it easy to specify and solve convex problems
- Grammar and semantics based on a rule from convex analysis [GBY06]
- Examples: CVXPY, CVXR, Convex.jl, CVX



Example

CVXPY is a Python-embedded DSL [DB16; AVD+18]

```
import cvxpy as cp
import numpy as np

m, n = 30, 20
A = np.random.randn(m, n)
b = np.random.randn(m)

x = cp.Variable(n)
objective = cp.Minimize(cp.sum_squares(A @ x - b))
constraints = [0 <= x, x <= 1]
problem = cp.Problem(objective, constraints)
problem.solve()
```

Neural networks?

Differentiable programming

- Deep learning uses derivatives ("backpropagation") to train neural networks
- A special case of *differentiable programming*
  - Library of *parametrized* atomic functions
  - Each atomic function is differentiable
  - A differentiable program is a composition of atomic functions
  - Use chain rule to tune parameters to better achieve some goal

Differentiable programming

This talk: (analytically) differentiating through CVXPY

Why?

- encode prior knowledge (e.g., physics) into a differentiable program
- implement hard constraints or specialized operations in a neural network
- sensitivity analysis
- *learn* the structure of convex problems
    - learn to control a vehicle
    - model utility functions for agents
    - tune portfolio optimization policies

and more . . .

Differentiating through convex optimization problems

Parametrized convex optimization problems

A convex optimization problem with variable $x \in \mathbf{R}^n$ can be *parametrized* by numerical data $\theta \in \mathbf{R}^p$:

$$\begin{array}{ll} \text{minimize} & f_0(x;\theta) \\ \text{subject to} & f_i(x;\theta) \leq 0, \quad i = 1, \ldots, m \\ & A(\theta)x = b(\theta), \end{array}$$

(here, A and b are functions of θ).

Parametrized convex optimization problems

Toy example:

$$\text{minimize} \quad (x - 2\theta)^2,$$

with variable $x \in \mathbf{R}$ and parameter $\theta \in \mathbf{R}$.

- Each choice of θ induces a new optimization problem.

Parametrized convex optimization problems

Toy example, in CVXPY:

```
import cvxpy as cp

theta = cp.Parameter()
x = cp.Variable()
objective = (x - 2*theta)**2
problem = cp.Problem(cp.Minimize(objective))

# solve an instance of problem with theta == 3.0
theta.value = 3.0
problem.solve()
```

Solution mapping of a convex optimization problem

- A convex problem can be viewed as a map from parameters to solutions

$$
\begin{array}{rll}
x^\star(\theta) = & \text{argmin} & f_0(x;\theta) \\
& \text{subject to} & f_i(x;\theta) \leq 0, \quad i = 1, \dots, m \\
& & A(\theta)x = b(\theta)
\end{array}
$$

- $x^\star(\theta) = \{2\theta\}$ for the problem of minimizing $(x - 2\theta)^2$
- For most problems $x^\star(\theta)$ cannot be written down analytically

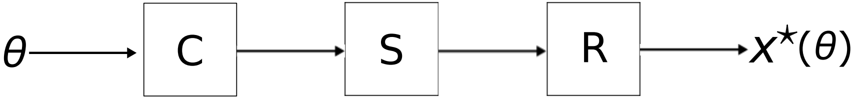
Derivative of the solution map of a convex problem

- When $x^\star(\theta)$ is single-valued, we can compute its derivative [ABB+19]
  - requires implicitly differentiating optimality conditions of a cone program
- For our toy example, $x^\star(\theta) = 2\theta$, $\frac{dx^\star}{d\theta}(\theta) = 2$
- Can efficiently differentiate through problems, even when solution is not analytical

Differentiating through CVXPY

Solution map of a parametrized CVXPY problem: $x^\star(\theta) = (R \circ S \circ C)(\theta)$

- Problem is *canonicalized* (C) to a standard form
- The canonicalized problem is *solved* (S)
- A solution for the original problem is *retrieved* (R)



Differentiating through CVXPY

We can efficiently differentiate through C, S, and R [AAB+19]

- in fact, we ensure C and R are affine
- we call this *affine-solver-affine* (ASA) form
- we introduce a grammar that ensures problem is in ASA form

Differentiating through CVXPY

Differentiate through the solver (S) by differentiating through a *cone program*

- every convex program can be written as a convex cone program
- solving a cone program equivalent to finding a 0 of a map $\mathcal{N}$
- a vector z can be used to construct a solution of a cone program if and only if $\mathcal{N}(z, Q) = 0$, where Q is an embedding of problem data
- if technical conditions are satisfied, the solution z is given by a function of Q, and we can compute its derivative
  - application of implicit function theorem
- details in [ABB$^+$19]

Differentiating through CVXPY

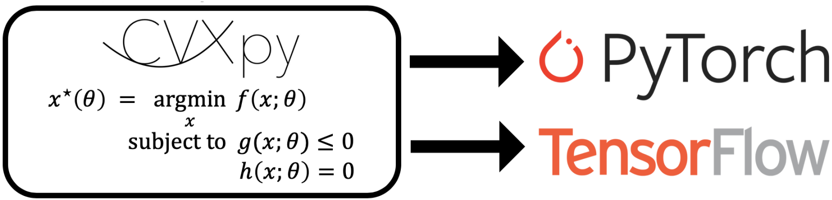
```
import cvxpy as cp

theta = cp.Parameter()
x = cp.Variable()
objective = (x - 2*theta)**2
problem = cp.Problem(cp.Minimize(objective))

theta.value = 3.0
problem.solve(requires_grad=True)
problem.backward()      # backpropagate through solution
print(theta.gradient)   # theta.gradient now equals 2.0
```

Exporting to PyTorch and TensorFlow

`cvxpylayers`: an open-source library for exporting CVXPY problems to PyTorch and TensorFlow



Exporting to PyTorch and TensorFlow

```
from cvxpylayers.torch import CvxpyLayer
import torch

# export to torch (or tensorflow) with just one line
layer = CvxpyLayer(problem, parameters=[theta], variables=[x])

theta_tch = torch.tensor(3.0, requires_grad=True)
soln = layer(theta_tch)[0]
soln.backward()
print(theta_tch.grad)
```

Examples

Learning convex-optimization control policies

Consider a stochastic control problem

$$
\begin{array}{ll}
\text{minimize} & \lim_{T\to\infty} \mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1} \|x_t\|_2^2 + \|\phi(x_t)\|_2^2\right] \\
\text{subject to} & x_{t+1} = Ax_t + B\phi(x_t) + \omega_t, \quad t = 0, 1, \ldots, \\
& \|\phi(x_t)\|_\infty \leq u^{\max}, \quad t = 0, 1, \ldots,
\end{array}
$$

- $x_t \in \mathbf{R}^n$ is the state, $\phi(x_t) \in \mathbf{R}^m$ the control, ω_t is random
- expectation is over ω_t and x_0
- optimization is over states x_t and policy $\phi : \mathbf{R}^n \to \mathbf{R}^m$
- this problem is computationally intractable

Learning convex-optimization control policies

- Common heuristic for stochastic control is approximate dynammic programming (ADP), which parametrizes ϕ
- ADP replaces minimization over functions ϕ with minimization over parameters
- Differentiable convex optimization layers can be used in an ADP method

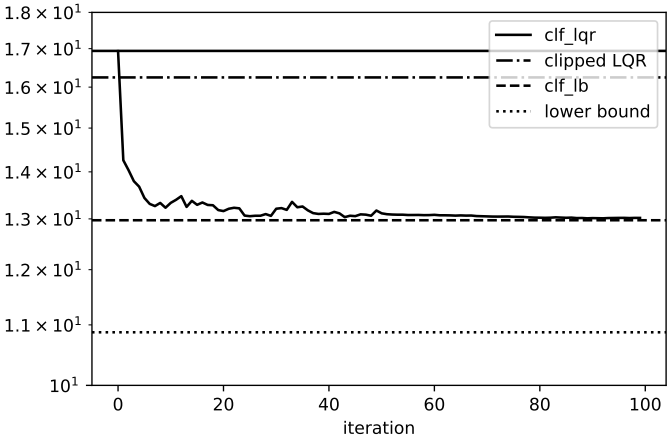
Learning convex-optimization control policies

- Take ϕ to be the solution of a convex optimization problem:

$$\begin{array}{rll} \phi(x_t) = & \underset{u}{\text{argmin}} & \|P^{1/2}(Ax_t + Bu)\|_2^2 + \|u\|_2^2 \\ & \text{subject to} & \|u\|_\infty \leq u^{\max} \end{array}$$

- Here, the parameter is $P^{1/2} \in \mathbf{R}^{n \times n}$
- Learning method:
  - simulate the system with the policy in the loop
  - approximate expected cost
  - update $P^{1/2}$ using gradient descent

Learning convex-optimization control policies



Learning convex-optimization control policies

- Many real-world applications, including
  - finance
  - vehicle control
  - supply-chain management
- See our paper: "Learning convex-optimization control policies"

Data poisoning attack

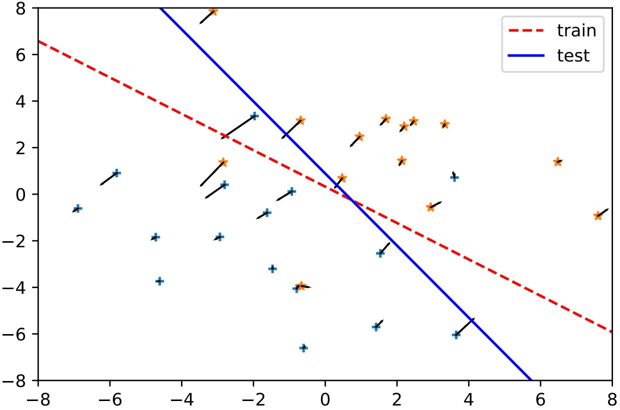
- given data (x_i, y_i), $i = 1, \ldots, m$
  - $x_i \in \mathbf{R}^n$ are feature vectors
  - $y_i \in \{0, 1\}$ are associated boolean outcomes
- linear classifier: $\hat{y} = \mathbf{1}[\beta^T x \geq 0]$
- find optimal weights $\beta^\star$ by minimizing

$$(1/m) \sum_i \mathcal{L}(\beta; x_i, y_i) + r(\beta)$$

  - $\mathcal{L}(\beta; x_i, y_i) = \log(1 + \exp(\beta^T x_i + b)) - y_i \beta^T x_i$ is the logistic loss
  - $r(\beta) = 0.1\|\beta\|_1 + 0.1\|\beta\|_2^2$ is elastic-net regularization
- adversary seeks to increase test loss $\mathcal{L}^{\text{test}}(\beta^\star)$ by (just barely) perturbing training data

Data poisoning attack

- $\beta^\star$ is a solution to a convex problem parametrized by x_i
- $\nabla_{x_i}\mathcal{L}^{\text{test}}(\beta^\star)$ gives direction x_i should be moved to achieve greatest increase in test loss



Data poisoning attack

```
X = cp.Parameter((m, n))
beta = cp.Variable(n)
log_likelihood = cp.sum(
    cp.multiply(Y, X @ beta) - cp.logistic(X @ beta)
)
r = 0.1*cp.norm(beta, 1) + 0.1*cp.norm(beta, 2)**2)
problem = cp.Problem(cp.Minimize(-log_likelihood/m + r))
fit_logreg = CvxpyLayer(problem, parameters=[X], variables=[beta])
beta_star = fit_logreg(X_train)[0]
test_loss = compute_loss(beta_star, X_test, Y_test)
test_loss.backward()
```

Summary

Our software makes it easy to differentiate through DSLs for convex optimization, letting you pair

- convex optimization
  - rich modelling capabilities
  - large number of applications
  - efficient solution algorithms
  - mature, high-level software libraries
- with machine learning

Software

https://github.com/cvxgrp/cvxpylayers

https://github.com/cvxgrp/cvxpy

References

[AAB+19] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*. 2019.

[ABB+19] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. Moursi. Differentiating through a cone program. *Journal of Applied and Numerical Optimization* 1.2 (2019), pp. 107–115.

[AVD+18] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision* 5.1 (2018), pp. 42–60.

[DB16] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research* 17.1 (2016), pp. 2909–2913.

[GBY06] M. Grant, S. Boyd, and Y. Ye. Disciplined convex programming. In *Global optimization*. Springer, 2006, pp. 155–210.

Additional examples

Tuning a Markowitz policy

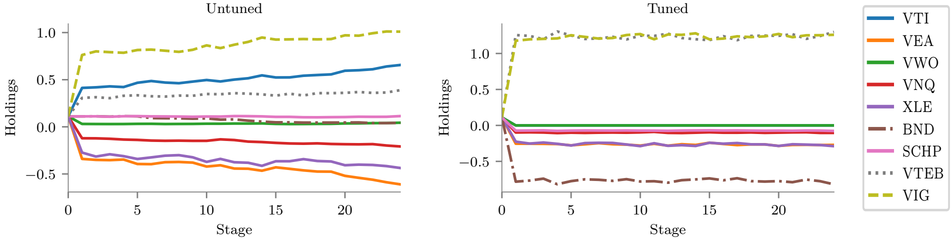


Figure: left: untuned; right: policy with tuned constraints, mean and covariance

Tracking a vehicle trajectory

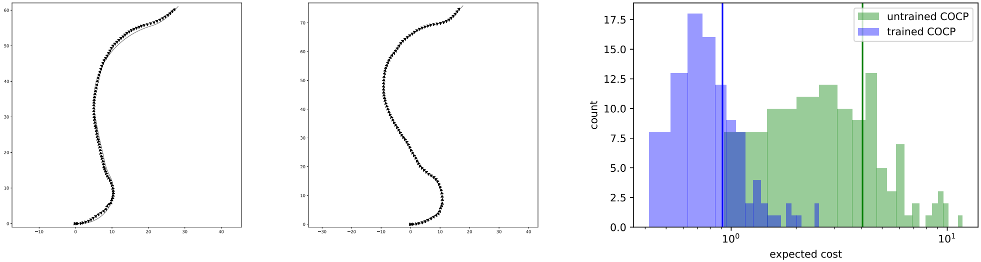


Figure: left: untrained path; middle: trained path: right: expected cost histogram.