Convex optimization layers

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

Akshay Agrawal
@akshaykagrawal
Convex optimization

minimize \( f(x; \theta) \)
subject to \( g(x; \theta) \leq 0 \)
\( A(\theta)x = b(\theta) \)

- \( x \in \mathbb{R}^n \) is the variable
- \( \theta \in \mathbb{R}^m \) is the parameter
- \( f \) and \( g \) are convex (curve upwards)
- find \( x \) that minimizes \( f \) while satisfying the constraints
Why convex optimization?

➢ Convex optimization problems can be solved quickly, reliably, and exactly
Why convex optimization?

➢ Convex optimization problems can be solved quickly, reliably, and exactly
➢ Software libraries like CVXPY make convex optimization easy
Why convex optimization?

➢ Convex optimization problems can be solved quickly, reliably, and exactly
➢ Software libraries like CVXPY make convex optimization easy

```python
import cvxpy as cp

x = cp.Variable(n)
objective = cp.Minimize(cp.sum_squares(A @ x - b) + cp.pnorm(x, p=1))
constraints = [0 <= x, x <= 1]
prob = cp.Problem(objective, constraints)
result = prob.solve()
```
Why convex optimization?

- Convex optimization problems can be solved quickly, reliably, and exactly
- Software libraries like CVXPY make convex optimization easy
- *Tons of applications*
Landing rockets
Designing airplanes
Until now ...

- Difficult to use convex optimization problems in TensorFlow pipelines
- Parameters $\theta$ were chosen and tuned by hand
CVXPY Layers

\[ x^*(\theta) = \arg\min_{\mathbf{x}} \quad f(x; \theta) \]
\[ \text{subject to} \quad g(x; \theta) \leq 0 \]
\[ A(\theta)x = b(\theta) \]
```python
import cvxpy as cp
import tensorflow as tf
from cvxpylayers.tensorflow import CvxpyLayer

n, m = 2, 3
x = cp.Variable(n)
A, b = cp.Parameter((m, n)), cp.Parameter(m)
constraints = [x >= 0]
objective = cp.Minimize(0.5 * cp.pnorm(A @ x - b, p=1))
problem = cp.Problem(objective, constraints)

cvxpylayer = CvxpyLayer(problem, parameters=[A, b], variables=[x])

A_tf = tf.Variable(tf.random.normal((m, n)))
b_tf = tf.Variable(tf.random.normal((m,)))
with tf.GradientTape() as tape:
    solution, = cvxpylayer(A_tf, b_tf)
    summed_solution = tf.math.reduce_sum(solution)
    gradA, gradb = tape.gradient(summed_solution, [A_tf, b_tf])
```
```python
import cvxpy as cp
import tensorflow as tf
from cvxpylayers.tensorflow import CvxpyLayer

n, m = 2, 3
x = cp.Variable(n)
A, b = cp.Parameter((m, n)), cp.Parameter(m)
constraints = [x >= 0]
objective = cp.Minimize(0.5 * cp.pnorm(A @ x - b, p=1))
problem = cp.Problem(objective, constraints)

cvxpylayer = CvxpyLayer(problem, parameters=[A, b], variables=[x])

A_tf = tf.Variable(tf.random.normal((m, n)))
b_tf = tf.Variable(tf.random.normal((m,)))
with tf.GradientTape() as tape:
solution, = cvxpylayer(A_tf, b_tf)
summed_solution = tf.math.reduce_sum(solution)
gradA, gradb = tape.gradient(summed_solution, [A_tf, b_tf])
```
```python
import cvxpy as cp
import tensorflow as tf
from cvxpylayers.tensorflow import CvxpyLayer

n, m = 2, 3
x = cp.Variable(n)
A, b = cp.Parameter((m, n)), cp.Parameter(m)
constraints = [x >= 0]
objective = cp.Minimize(0.5 * cp.pnorm(A @ x - b, p=1))
problem = cp.Problem(objective, constraints)

cvxpylayer = CvxpyLayer(problem, parameters=[A, b], variables=[x])

A_tf = tf.Variable(tf.random.normal((m, n)))
b_tf = tf.Variable(tf.random.normal((m,)))
with tf.GradientTape() as tape:
    solution, = cvxpylayer(A_tf, b_tf)
    summed_solution = tf.math.reduce_sum(solution)
    gradA, gradb = tape.gradient(summed_solution, [A_tf, b_tf])
```
```python
import cvxpy as cp
import tensorflow as tf
from cvxpylayers.tensorflow import CvxpyLayer

n, m = 2, 3
x = cp.Variable(n)
A, b = cp.Parameter((m, n)), cp.Parameter(m)
constraints = [x >= 0]
objective = cp.Minimize(0.5 * cp.pnorm(A @ x - b, p=1))
problem = cp.Problem(objective, constraints)

cvxpylayer = CvxpyLayer(problem, parameters=[A, b], variables=[x])

A_tf = tf.Variable(tf.random.normal((m, n)))
b_tf = tf.Variable(tf.random.normal((m,)))
with tf.GradientTape() as tape:
    solution, = cvxpylayer(A_tf, b_tf)
    summed_solution = tf.math.reduce_sum(solution)
    gradA, gradb = tape.gradient(summed_solution, [A_tf, b_tf])
```
Learning to control a car
Learning to control a car
Learning to control a car
For more ...

github.com/cvxgrp/cvxpylayers

NeurIPS paper

Learning control policies (L4DC)

with examples in

➢ controlling a car
➢ managing a supply chain
➢ allocating financial portfolios
Thank you!