



cvxpylayers

`cvxpylayers` is a Python library for constructing differentiable convex optimization layers in PyTorch, JAX, and TensorFlow using CVXPY. A convex optimization layer solves a parametrized convex optimization problem in the forward pass to produce a solution. It computes the derivative of the solution with respect to the parameters in the backward pass.

This library accompanies our NeurIPS 2019 paper on differentiable convex optimization layers. For an informal introduction to convex optimization layers, see our blog post.

Our package uses CVXPY for specifying parametrized convex optimization problems.

- Installation
- Usage
- Examples
- Contributing
- Projects using `cvxpylayers`
- License
- Citing

Installation

Use the package manager `pip` to install `cvxpylayers`.

```
1 pip install cvxpylayers
```

Our package includes convex optimization layers for PyTorch, JAX, and TensorFlow 2.0; the layers are functionally equivalent. You will need to install PyTorch, JAX, or TensorFlow separately, which can be done by following the instructions on their websites.

cvxpylayers has the following dependencies: * Python 3 * NumPy * CVXPY \geq 1.1.a4 * PyTorch \geq 1.0, JAX \geq 0.2.12, or TensorFlow \geq 2.0 * diffcp \geq 1.0.13

Usage

Below are usage examples of our PyTorch, JAX, and TensorFlow layers. Note that the parametrized convex optimization problems must be constructed in CVXPY, using DPP.

PyTorch

```
1 import cvxpy as cp
2 import torch
3 from cvxpylayers.torch import CvxpyLayer
4
5 n, m = 2, 3
6 x = cp.Variable(n)
7 A = cp.Parameter((m, n))
8 b = cp.Parameter(m)
9 constraints = [x >= 0]
10 objective = cp.Minimize(0.5 * cp.pnorm(A @ x - b, p=1))
11 problem = cp.Problem(objective, constraints)
12 assert problem.is_dpp()
13
14 cvxpylayer = CvxpyLayer(problem, parameters=[A, b], variables=[x])
15 A_tch = torch.randn(m, n, requires_grad=True)
16 b_tch = torch.randn(m, requires_grad=True)
17
18 # solve the problem
19 solution, = cvxpylayer(A_tch, b_tch)
20
21 # compute the gradient of the sum of the solution with respect to A, b
22 solution.sum().backward()
```

Note: CvxpyLayer cannot be traced with `torch.jit`.

JAX

```
1 import cvxpy as cp
2 import jax
3 from cvxpylayers.jax import CvxpyLayer
4
5 n, m = 2, 3
6 x = cp.Variable(n)
7 A = cp.Parameter((m, n))
8 b = cp.Parameter(m)
```

```

 9 constraints = [x >= 0]
10 objective = cp.Minimize(0.5 * cp.pnorm(A @ x - b, p=1))
11 problem = cp.Problem(objective, constraints)
12 assert problem.is_dpp()
13
14 cvxpylayer = CvxpyLayer(problem, parameters=[A, b], variables=[x])
15 key = jax.random.PRNGKey(0)
16 key, k1, k2 = jax.random.split(key, 3)
17 A_jax = jax.random.normal(k1, shape=(m, n))
18 b_jax = jax.random.normal(k2, shape=(m,))
19
20 solution, = cvxpylayer(A_jax, b_jax)
21
22 # compute the gradient of the summed solution with respect to A, b
23 dcvxpylayer = jax.grad(lambda A, b: sum(cvxpylayer(A, b)[0]), argnums
    =[0, 1])
24 gradA, gradb = dcvxpylayer(A_jax, b_jax)

```

Note: CvxpyLayer cannot be traced with the JAX `jit` or `vmap` operations.

TensorFlow 2

```

1 import cvxpy as cp
2 import tensorflow as tf
3 from cvxpylayers.tensorflow import CvxpyLayer
4
5 n, m = 2, 3
6 x = cp.Variable(n)
7 A = cp.Parameter((m, n))
8 b = cp.Parameter(m)
9 constraints = [x >= 0]
10 objective = cp.Minimize(0.5 * cp.pnorm(A @ x - b, p=1))
11 problem = cp.Problem(objective, constraints)
12 assert problem.is_dpp()
13
14 cvxpylayer = CvxpyLayer(problem, parameters=[A, b], variables=[x])
15 A_tf = tf.Variable(tf.random.normal((m, n)))
16 b_tf = tf.Variable(tf.random.normal((m,)))
17
18 with tf.GradientTape() as tape:
19     # solve the problem, setting the values of A, b to A_tf, b_tf
20     solution, = cvxpylayer(A_tf, b_tf)
21     summed_solution = tf.math.reduce_sum(solution)
22 # compute the gradient of the summed solution with respect to A, b
23 gradA, gradb = tape.gradient(summed_solution, [A_tf, b_tf])

```

Note: CvxpyLayer cannot be traced with `tf.function`.

Log-log convex programs

Starting with version 0.1.3, `cvxpylayers` can also differentiate through log-log convex programs (LL-CPs), which generalize geometric programs. Use the keyword argument `gp=True` when constructing a `CvxpyLayer` for an LLC. Below is a simple usage example

```
1 import cvxpy as cp
2 import torch
3 from cvxpylayers.torch import CvxpyLayer
4
5 x = cp.Variable(pos=True)
6 y = cp.Variable(pos=True)
7 z = cp.Variable(pos=True)
8
9 a = cp.Parameter(pos=True, value=2.)
10 b = cp.Parameter(pos=True, value=1.)
11 c = cp.Parameter(value=0.5)
12
13 objective_fn = 1/(x*y*z)
14 objective = cp.Minimize(objective_fn)
15 constraints = [a*(x*y + x*z + y*z) <= b, x >= y**c]
16 problem = cp.Problem(objective, constraints)
17 assert problem.is_dgp(dpp=True)
18
19 layer = CvxpyLayer(problem, parameters=[a, b, c],
20                   variables=[x, y, z], gp=True)
21 a_tch = torch.tensor(a.value, requires_grad=True)
22 b_tch = torch.tensor(b.value, requires_grad=True)
23 c_tch = torch.tensor(c.value, requires_grad=True)
24
25 x_star, y_star, z_star = layer(a_tch, b_tch, c_tch)
26 sum_of_solution = x_star + y_star + z_star
27 sum_of_solution.backward()
```

Solvers

At this time, we support two open-source solvers: SCS and ECOS. SCS can be used to solve any problem expressible in CVXPY; ECOS can be used to solve problems that don't use the positive semidefinite or exponential cone (this roughly means that if you have positive semidefinite matrices or use atoms like `cp.log`, ECOS cannot be used to solve your problem via `cvxpylayers`). By default, `cvxpylayers` uses SCS to solve the problem.

Using ECOS

First make sure that you have `cvxpylayers` and `diffcp` up to date, by running

```
1 pip install --upgrade cvxpylayers diffcp
```

Then, to use ECOS, you would pass the `solver_args` argument to the layer:

```
1 solution = layer(*params, solver_args={"solve_method": "ECOS"})
```

Passing arguments to the solvers

One can pass arguments to both SCS and ECOS by adding the argument as a key-value pair in the `solver_args` argument. For example, to increase the tolerance of SCS to $1e-8$ one would write:

```
1 layer(*parameters, solver_args={"eps": 1e-8})
```

If SCS is not converging, we highly recommend switching to ECOS (if possible), and if not, using the following arguments to SCS:

```
1 solver_args={"eps": 1e-8, "max_iters": 10000, "acceleration_lookback":  
0}
```

Examples

Our examples subdirectory contains simple applications of convex optimization layers in IPython notebooks.

Contributing

Pull requests are welcome. For major changes, please open an issue first to discuss what you would like to change.

Please make sure to update tests as appropriate.

Please lint the code with `flake8`.

```
1 pip install flake8 # if not already installed  
2 flake8
```

Running tests

cvxpylayers uses the `pytest` framework for running tests. To install `pytest`, run:

```
1 pip install pytest
```

Execute the tests from the main directory of this repository with:

```
1 pytest cvxpylayers/{torch,jax,tensorflow}
```

Projects using cvxpylayers

Below is a list of projects using cvxpylayers. If you have used cvxpylayers in a project, you're welcome to make a PR to add it to this list. * Learning Convex Optimization Control Policies * Learning Convex Optimization Models

License

cvxpylayers carries an Apache 2.0 license.

Citing

If you use cvxpylayers for research, please cite our accompanying NeurIPS paper:

```
1 @inproceedings{cvxpylayers2019,
2   author={Agrawal, A. and Amos, B. and Barratt, S. and Boyd, S. and
3     Diamond, S. and Kolter, Z.},
4   title={Differentiable Convex Optimization Layers},
5   booktitle={Advances in Neural Information Processing Systems},
6   year={2019},
7 }
```

If you use cvxpylayers to differentiate through a log-log convex program, please cite the accompanying paper:

```
1 @article{agrawal2020differentiating,
2   title={Differentiating through log-log convex programs},
3   author={Agrawal, Akshay and Boyd, Stephen},
4   journal={arXiv},
5   archivePrefix={arXiv},
6   eprint={2004.12553},
7   primaryClass={math.OA},
8   year={2020},
9 }
```