JuMP - Optimization problem differentiation

Akshay Sharma

March 29, 2020

Abstract

Differentiable optimization is a promising field of convex optimization and has many potential applications in game theory, control theory and machine learning. Unlike CVXPY, JuMP currently lacks the feature to differentiate solutions of disciplined convex optimization problems and render them as layers in machine learning libraries. I propose to develop this feature and make it accessible from the JuMP interface.

Contents

1 About Me

The Student

Name Akshay Sharma

E-Mail akshay.sharma.mat16@iitbhu.ac.in

Github https://github.com/AKS1996

Website http://www.imakshay.com

The Institute

University Indian Institute of Technology (BHU), Varanasi

Major Mathematics and Computing

Year Pre-final

Degree Dual Degree (Bachelors and Masters

My Background

I have been programming since last four years at college. I have been part of quite a good number of projects and I have developed a flair in tackling challenges.
In winter of 2019, I was a research intern at the Indian Statistical Institute working on improving PEGASOS algorithm [1] for large scale content-based URL classification. It involved digesting large textual data dumps (far prominent than all UCI repositories).
In summer of 2019, I interned at Adobe Systems. My project involved pattern mining of Adobe Photoshop user logs on Spark. I learned about the problems while handling big data, as there TBs of usage logs produced every day, and I had to handle and run usage statistics on four weeks' worth of data.
In summer of 2018, I completed the Google Summer of Code in ViSP - an open-source visual processing library. My project was developing a Java and an Android SDK for their C++ codebase. We used Native C++ libraries to maintain near-realtime detection time.
For more information on my projects, you can head over to my website.

My Motivation

Since past few years, I have developed an interest in optimization and applied research. Last year, I co-authored an article on finding globalized

markov equilibrium for discounted stochastic games [2]. For my Master's thesis, I'm working on developing a parallel asynchronous variant for Primal SVM solver for large scale classification on the Common Crawl (CC) dataset. I have used proprietary solvers to implement mathematical programs. I came to know about this project from a friend of mine, who is a part of Julia community now, and introduced me to the concept of Automatic Differentiated and JuMP modelling interface in Julia. This project is a great learning opportunity for me. Given my past project experience, I feel confident to work on this project.

2 The Project

In this section, I present a detailed version of proposed plan. This will include the deliverables, timeline and potential bottlenecks. I expect this structure to be enhanced under the guidance of a mentor.
I tried to keep mathematics to a minimum in proposal, but the project is appreciably theoretical in nature. I would like to mention that while the project description on the idea page refers differentiating optimal solution of convex optimization problems, I assume it was meant for disciplined parametrized optimization problems (DPP). Differentiation for a broader category of convex optimization problems is out of scope of a summer project.
The words *program*, *model* and *optimization problem* are used interchangeably.

2.1 Differentiation of disciplined convex programs

As discussed in the original article [3], solution of a DPP (a `ModelLike` object) can be viewed as a map $\mathcal{S}$ that maps parameters of the problem θ to its solutions x^* and we propose to find derivative (`Vector` object) of $\mathcal{S}$ with respect to θ (`Variable` object). The authors present the case of modelling the DPP as a conic optimization problem. Specifically, they show that $\mathcal{S}$ can be decomposed as $R \circ s \circ C$ where

- C represents a canonicalizer that maps parameters of DPP to cone problem data
- s is the solver for cone program
- R represents a retriever that maps solution of cone program to original DPP

The adjoint of the derivative of DPP can be expressed as

$$D^T\mathcal{S} = D^T C \circ D^T s \circ D^T R$$

In next sections I describe how to find above components.

2.2 Canonicalization

Canonicalization is the method of generating a cone program (another `MathOptInterface.ModelLike` object) from the original problem. The generated cone program can be readily modelled by `Convex.jl` (refer this blog for an example).
Canonicalization begins with expanding functions or *atoms* in their graph implementations. Convex.jl already has this support in order to support the underlying solvers. In the original article [3], this results in a canonicalization map or a *sparse matrix*. This map be constructed using the `get_conic_form` method.

2.3 Differentiating the cone solver

Obtaining the derivative of the cone program plays a central role in differentiating the original problem. The original article models it as a mapping ψ from cone problem data to its solution. It can be decomposed to $\phi \circ s \circ Q$ [5] where

- Q is a static matrix composed of cone program data
- s is an algorithm (an intended `function` in julia) to solve *homogeneous self-dual embedding* of the cone problem. The article describes an implicit differentiation procedure to obtain its derivative
- ϕ is a mapping from self-dual embedding to the cone program. In implementation, it will be a matrix composed of projections on dual of cone data

Hence the derivative and adjoint derivative of the cone program can be obtained as

$$D\psi = D\phi \circ Ds \circ DQ$$
$$D^T\psi = D^TQ \circ D^Ts \circ D^T\phi$$

Obtaining derivative of homogeneous self-dual embedding will be a major feat. As mentioned in [5], this is a three step process involving implicit differentiation to get a system of linear equations and solving them:

1. Computing derivative of solution map ψ with respect to perturbation in cone problem data
2. Computing ajdoint of the derivative with respect to perturbation in input variables
3. Integration into AD

First two steps require computing matrix inverses and computing projections on dual of the cone program data. Julia's `inv` function in Linear Algebra package will be suitable for finding inverse of small matrices (refer 3.1). Projections can be found using `LazySets.jl` [9].

The calculations from first two steps can be readily integrated in any AD system as third step. The original article provides implementation to compute derivative of solution map in backward pass. For instance, they use Tensorflow's `GradientTape` function for it. We can do the same using gradient function in `Zygote.jl`. This can be achieved if we stick to pure Julia solvers such as `Pajarito.jl` [6] which is capable of handling conic program.

2.4 Solution retrieval

The method R that maps solution of the cone program $\bar{x}^*$ to a solution of original problem x^* is essentially a linear map $R(\bar{x}^*) = x^*$ and can be obtained by slicing.

3 Road Map

In the wake of COVID-19, national colleges may plan to resume semester this summer. In that case, I will not be available for a week for exams. Apart from this I have no prior commitments.

3.1 Caveats

As I have already mentioned, the scope is project is pretty large. I can look at the following features if time permits:

- `inv` function will succumb in finding inverse of large matrices while implicit differentiation of Ds. In such cases, `PETSc.jl` [8] or `Krylov.jl` [7] might come handy.

- Another potential bottleneck is non-invertibility of matrices. As mentioned in the article, it can be approximated as solving a least squares program, preferably by LSQR method of `IteratedSolvers.jl` package
- I assume the program will be fed DPPs only. As `Convex.jl` emits warnings for non DCP compliant models, I will implement the same for DPP models (similar to `is_dpp` function in CVXPY).

3.2 Timeline

I expect to complete the following milestones in order.

3.2.1 Community Bonding Period

- Discuss the project with my mentor in further detail
- Get more acquainted with JuMP and `Zygote.jl`. Working knowledge of `MathOptInterface`

3.2.2 Coding Period

- First two weeks of June: Implement a function to obtain canonicalization map. Begin implementing a function to find derivative of solution map.
- Last two weeks of June: Implement a function to find adjoint of derivative of solution map. It will broadly use the same matrix inverse and projection methods.
- First two weeks of July: Using perturbation values to obtain derivative using `Zygote.jl`. This marks integration with AD
- Last two weeks of July: Implement a method to obtain retrieval map. Test overall model differentiation flow.
- First two weeks of August: Testing the differentiation on different programs(LPs, QPs, etc) and tweaking the software if necessary. Bench marking against CVXPY and providing tutorials.
- Last two weeks of August: One week for documenting everything with demonstrations and a buffer week for any unexpected delays.

3.2.3 Post GSoC

Automatic differentiation is a promising emerging branch of computational mathematics. Julia provides me with a great platform to hone my mathematical skills (symbolic math is love). There are many feats that are out of scope of a summer project, where JuMP interface can improve. I would be glad to contribute and enhance it.

References

[1] `PEGASOS`: primal estimated sub-gradient solver for SVM
https://link.springer.com/article/10.1007/s10107-010-0420-4

[2] Globalized robust Markov perfect equilibrium for discounted stochastic games and its application on intrusion detection in wireless sensor networks, *Japan Journal of Industrial and Applied Mathematics*
https://link.springer.com/article/10.1007/s13160-019-00397-9

[3] Differentiable Convex Optimization Layers, *NIPS*, 2019
http://web.stanford.edu/~boyd/papers/pdf/diff_cvxpy.pdf

[4] Zygote: source to source Automatic Differentiation in Julia
https://github.com/FluxML/Zygote.jl

[5] Differentiating through a cone program, *Journal of Applied and Numerical Optimization*, 2019
http://jano.biemdas.com/issues/JANO2019-2-2.pdf

[6] `Pajarito.jl`: https://github.com/JuliaOpt/Pajarito.jl

[7] `Krylov.jl`: https://github.com/JuliaSmoothOptimizers/Krylov.jl

[8] `PETSc.jl`: https://github.com/JuliaParallel/PETSc.jl

[9] `LazySets.jl`: https://juliareach.github.io/LazySets.jl