BPQP: A Differentiable Convex Optimization Framework for Efficient End-to-End Learning

Jianming Pan[1], Zeqi Ye[2],
Xiao Yang[3]*, Xu Yang[3], Weiqing Liu[3], Lewen Wang[3], Jiang Bian[3]

[1]University of California, Berkeley, [2]Nankai University,
[3]Microsoft Research Asia

Nov 1, 2024

Motivation

Output of the forward functions in NN layers can be determined by the solutions to optimization problems. Applications:

- Portfolio optimization with estimated returns and risk matrices;
- Autonomous driving with pre-identified traffic status;
- Electricity power allocation with predicted electricity demand.

Problem formulation

Definition (Differentiable Convex Optimization Layer)

Given the input $y \in \mathbb{R}^p$, output $z^* \in \mathbb{R}^d$, a differentiable convex optimization layer is defined as

$$z^*(y) = \arg\min_{z \in \mathbb{R}^d} f_y(z), \ s.t. \ \ h_y(z) = 0, \ g_y(z) \leq 0.$$

Given loss function $\mathcal{L}$, the derivative of $\mathcal{L}$ *w.r.t* y can be written as $\frac{\partial \mathcal{L}}{\partial y} = \frac{\partial \mathcal{L}}{\partial z^*} \frac{\partial z^*}{\partial y}$.

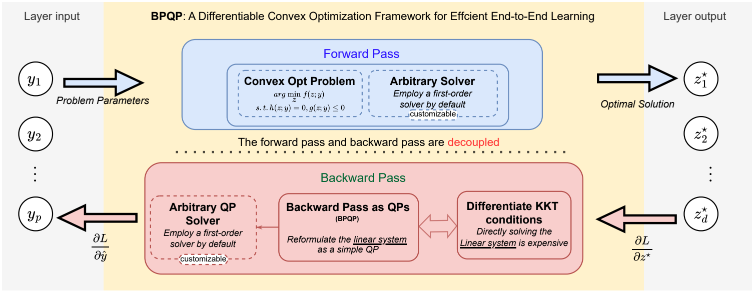
Related work and problems

Real-world scenarios involve different kind of convex problems, large-scale datasets and numerous constraints.

- Unrolling: unroll the iterations of the optimization process and use the decision variable from the final iteration as a proxy solution. (Additional unrolling cost)
- Implicit Methods: Implicit function theorem to KKT condition. (Relied on coupled optimizers/need to solve high-dim linear system/most on QP)
- Learn to optimize: Train a solver network to solve constrained problem. (Low efficiency and accuracy)

Framework

- Decoupled forward and backward pass;
- Reformulate backward pass as a simple QP.



Layer input
BPQP: A Differentiable Convex Optimization Framework for Effcient End-to-End Learning
Layer output
y_1
y_2
$\vdots$
y_p
Problem Parameters
Forward Pass
Convex Opt Problem
$\arg\min_z f(z;y)$
$s.t.\ h(z;y)=0, g(z;y)\le 0$
Arbitrary Solver
Employ a first-order solver by default
customizable
Optimal Solution
$z_1^\star$
$z_2^\star$
$\vdots$
$z_d^\star$
The forward pass and backward pass are decoupled
Backward Pass
Arbitrary QP Solver
Employ a first-order solver by default
customizable
Backward Pass as QPs
(BPQP)
Reformulate the linear system as a simple QP
Differentiate KKT conditions
Directly solving the Linear system is expensive
$\frac{\partial L}{\partial \hat{y}}$
$\frac{\partial L}{\partial z^\star}$

Strengths

Decoupled forward and backward pass:

- Choose different optimizers in the forward and backward pass (i.e. a suitable advanced optimizer for specific convex forward pass problem, and another optimizer which can solve the simple QP in the backward pass easily).
- Any choice with flexibility! Improvement with evolving solver technologies.

Reformulate backward pass as a simple QP:

- Instead of solving the high-dim KKT+IFT linear system, we can instead solve a equivalent QP problem (but only with several equality constraints).
- Much faster than solving the linear system, and other coupled/unrolling methods!

The linear system for gradient calculation

By [1], we can calculate the gradient $\frac{\partial z^*}{\partial y}$ by applying IFT to KKT system after the problem solved, then get $\nabla_y \mathcal{L} = \frac{\partial \mathcal{L}}{\partial z^*}\frac{\partial z^*}{\partial y}$:

$$\begin{bmatrix} P(z^\star, \nu^\star, \lambda^\star) & G(z^\star)^\top & A(z^\star)^\top \\ D(\lambda^\star)\, G(z^\star) & D(g(z^\star)) & 0 \\ A(z^\star) & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial z^\star}{\partial y} \\ \frac{\partial \lambda^\star}{\partial y} \\ \frac{\partial \nu^\star}{\partial y} \end{bmatrix} = - \begin{bmatrix} q(z^\star, \nu^\star, \lambda^\star) \\ c(z^\star, \lambda^\star) \\ b(z^\star) \end{bmatrix}$$

Where ν and λ are lagrangian multipliers, and
$P(z^\star, \nu^\star, \lambda^\star) = \nabla^2 f(z^\star) + \nabla^2 h(z^\star)\nu^\star + \nabla^2 g(z^\star)\lambda^\star$,
$A(z^\star) = \nabla h(z^\star)$ and $G(z^\star) = \nabla g(z^\star)$. Let
$q(z^\star, \nu^\star, \lambda^\star) = \partial(\nabla f(z^\star) + \nabla h(z^\star)\nu^\star + \nabla g(z^\star)\lambda^\star)/\partial y$,
$b(z^\star) = \partial h(z^\star)/\partial y$ and $c(z^\star, \lambda^\star) = \partial(D(\lambda^\star)g(z^\star))/\partial y$.

Reformulation: Main theorem

Theorem

Under regularity conditions,
$\nabla_y \mathcal{L} = [q(z^\star, \nu^\star, \lambda^\star), c(z^\star, \lambda^\star), b(z^\star)][\tilde{z}, \tilde{\lambda}, \tilde{\nu}]^\top$ *and* $\tilde{z}, \tilde{\lambda}, \tilde{\nu}$ *is the optimal solution of following QP:*

$$\underset{\tilde{z}}{\text{minimize}} \ \frac{1}{2}\tilde{z}^\top P' \tilde{z} + {q'}^\top \tilde{z} \quad s.t. \ A'\tilde{z} = b', \ G'_+ \tilde{z} = c'_+,$$

where all the matrices and vectors above are directly derived from formulating the KKT conditions and solving the forward pass problem.

We use OSQP[2] as our backward pass solver in experiments.

Why our method works?

Rather than solving the KKT + IFT linear system, we only need to solve a simple QP to obtain the gradients(equivalent). **Why compute gradients in this way is much faster?**

- The constraint in the new QP is $A'\tilde{z} = b'$, $G'_{+}\tilde{z} = c'_{+}$. After solving the forward pass, the number of active constraints is much less than the original problem, and we only take use of the equality form of these active constraints to form this new QP.
- Solvers evolve quickly. We now have decoupled forward pass and backward pass, which means that we can apply any solver to the forward convex problem (QP, LP, SOCP, etc.) and any QP solver to the backward problem. Highly potential!

Discussion: nonconvex forward pass

- BPQP allows for the derivation of gradients that preserve the KKT norm, as elaborated in our paper, under "General Gradients." , which means that when KKT norm is small, BPQP can derive a high quality gradient.
- Therefore, when a non-convex solver used in the forward pass successfully achieves a solution that is close to or even reaches a local or global minimum (small KKT norm), BPQP can still compute well-behaved gradients effectively.

Small Scale

Size Problem	Method	100×20	500×100
		Runtime (scale 1.0e-04)	
QP	BPQP	**35.1(±3.8)**	**571.6(±130.7)**
	Alt-Diff [3]	475.3(±402.2)	3044.7(±992.3)
	OptNet [1]	1030.2(±238.5)	1872.8(±1254.0)
	CVXPY [4]	1862.9(±240.5)	4716.2(±185.6)
LP	BPQP	**29.5(±6.7)**	**318.7(±106.4)**
	OptNet [1]	963.1(±230.0)	970.5(±252.5)
	CVXPY [4]	930.4(±150.3)	5046.1(±189.7)
SOCP	BPQP	**63.1(±1.6)**	**242.9(±2.7)**
	CVXPY[4]	105.0(±2.9)	334.3(±3.2)

Table: Repeat 200 times.

Large Scale

Size	**Method**	**3000×1000**	**5000×2000**
Problem		**Runtime (scale 1.0e-01)**	
QP	BPQP	**70.6(±4.7)**	**262.8(±17.0)**
	Alt-Diff [3]	282.4(±152.8)	1820.6(±59.4)

Table: Repeated 50 times

Method	**BPQP**	**CVXPY**	**OptNet**	**Alt-Diff**
Avg. CosSim.	**0.992(±0.09)**	0.924(±0.15)	0.989(±0.12)	0.985(±0.11)

Table: Gradient accuracy on QP problems.

Implementation

NEURAL INFORMATION PROCESSING SYSTEMS

- Experiments in real-world applications (Quant finance) can be found on our paper.
- We will release example implementations & applications in Qlib.

Qlib

Microsoft Research 微软亚洲研究院

References

[1] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[2] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[3] Haixiang Sun, Ye Shi, Jingya Wang, Hoang Duong Tuan, H Vincent Poor, and Dacheng Tao. Alternating differentiation for optimization layers. In *The Eleventh International Conference on Learning Representations*, 2022.

[4] Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17(1):2909–2913, 2016.