Towards Constituting Mathematical Structures for Learning to Optimize

Jialin Liu* (Alibaba US) Xiaohan Chen* (Alibaba US) Zhangyang Wang (UT Austin) Wotao Yin (Alibaba US) HanQin Cai (UCF)

Overview

A generic learning-to-optimize (L2O) approach parameterizes the iterative update rule and learns the update direction as a black-box network. While the black-box approach is widely applicable, the learned model can overfit and may not generalize well to out-of-distribution test sets.

We derive the basic mathematical conditions that successful update rules commonly satisfy. Consequently, we propose a novel L2O model with a mathematics-inspired structure that is broadly applicable and generalized well to out-of-distribution problems. [1]

arXiv *GitHub*

Introduction

What is *Learning to Optimize (L2O)*?
An example:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \tag{1}$$

To solve (1), one can iteratively update $\boldsymbol{x}$ using a parameterized model:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{d}(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k); \phi) \tag{2}$$

- In this equation, ϕ denotes the parameters to learn.
- The model $\boldsymbol{d}(\cdot,\cdot;\phi)$ can be a RNN [2] or MLP [3].

Find ϕ that optimizes its resulting trajectory $\{\boldsymbol{x}_k\}$:

$$\phi_* = \operatorname*{argmin}_{\phi} \mathbb{E}_{f\in\mathcal{F}} \sum_{k=1}^{K} f(\boldsymbol{x}_k) \tag{3}$$

on a set of optimization instances $\mathcal{F}$.
We hope the trained model $\boldsymbol{d}(\cdot,\cdot;\phi_*)$ results in fast convergence.

Recall the gradient descent:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) \tag{4}$$

Comparing the L2O model (2) and the conventional method (4):
- (2) covers (4).
- (2) has more tunable parameters.

The L2O model has the potential for enhanced performance.

References

[1] J. Liu, X. Chen, Z. Wang, W. Yin, and H. Cai, "Towards constituting mathematical structures for learning to optimize," in *ICML*, 2023.

[2] M. Andrychowicz, M. Denil, S. Gomez, *et al.*, "Learning to learn by gradient descent by gradient descent," *NeurIPS*, 2016.

[3] K. Li and J. Malik, "Learning to optimize," in *ICLR*, 2016.

[4] T. Chen, X. Chen, W. Chen, *et al.*, "Learning to optimize: A primer and a benchmark," *JMLR*, 2022.

Motivations and Main Results

An observation in the literature [4]:
- Trained models sometimes fail to converge, even causing divergence on *unseen instances*.

What's the reason?
- Neural networks are *universal approximators*: Given any continuous operator, there exists a NN that is arbitrarily close to it.
- Seeking a optimized NN (3) equals to searching over the following space

$$\{\boldsymbol{d}: \mathbb{R}^{2n} \to \mathbb{R}^n, \boldsymbol{d} \text{ is continuous}\}$$

Such a space is too large, including undesirable operators.

Illustrative example: $f(\boldsymbol{x}) = (1/2)\|\boldsymbol{x}\|^2$.
The following operators should obviously be excluded:
- Operator A: $\boldsymbol{d}(\boldsymbol{x},\boldsymbol{g}) = \boldsymbol{g} + \mathbf{1}$. The corresponding update rule yields:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \nabla f(\boldsymbol{x}_k) + \mathbf{1} = \mathbf{1}$$

where optimal solution $\boldsymbol{x}_* = \mathbf{0}$ is NOT a fixed point of the above scheme.
- Operator B: $\boldsymbol{d}(\boldsymbol{x},\boldsymbol{g}) = 10\boldsymbol{x}$. The update rule

$$\boldsymbol{x}_{k+1} = -9\boldsymbol{x}_k$$

diverges almost surely, although the optimal solution is a fixed point.

Can we exclude these "bad" operators?
- Impose assumptions on (2) and derive a structured new rule.

Theorem 1 (Informal) *For any convex and smooth f and any update rule yielding* (2)*, as long as the following two conditions hold,*
- If $\boldsymbol{x}_k$ is an optimal solution to f, then it holds that $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$.
- The sequence $\{\boldsymbol{x}_k\}$ must converge to one of the optimal solutions to f.
there exist $\mathbf{P}_k \in \mathbb{R}^{n\times n}$ and $\boldsymbol{b}_k \in \mathbb{R}^n$ satisfying

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathbf{P}_k \nabla f(\boldsymbol{x}_k) - \boldsymbol{b}_k, \tag{5}$$

with $\mathbf{P}_k$ is bounded and $\boldsymbol{b}_k \to \mathbf{0}$ as $k \to \infty$.

Some discussions:
- A meaningful operator serving as an optimization scheme is not free.
- Instead of using neural networks to suggest the update direction, one may using the output of NN to suggest $\mathbf{P}_k$ and $\boldsymbol{d}_k$.
- Ensuring boundedness: using a sigmoid function on the NN output.
- Ensuring $\boldsymbol{b}_k \to \mathbf{0}$ is challenging; we recommend fixing $\boldsymbol{b}_k = \mathbf{0}$.

Extensions: Such results can be extended to (Refer to [1] for details)

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + r(\boldsymbol{x})$$

where f is convex and smooth, r is convex and possibly nonsmooth. Equation (5) would be extended to

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \operatorname{prox}_{r,\mathbf{P}_k}\Big(\boldsymbol{y}_k - \mathbf{P}_k \nabla f(\boldsymbol{y}_k) - \boldsymbol{b}_{1,k}\Big),\\ \boldsymbol{y}_{k+1} &= \boldsymbol{x}_{k+1} + \mathbf{A}_k(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) + \boldsymbol{b}_{2,k}.\end{aligned} \tag{6}$$

- With fixing $\boldsymbol{b}_{1,k} = \boldsymbol{b}_{2,k} = \mathbf{0}$, (6) reduces to a generalized FISTA.
- Instead of learning the update rule, we recommend learning *preconditioner* $\mathbf{P}_k$ and *accelerator* $\mathbf{A}_k$. (L2O-PA)

Numerical Validation

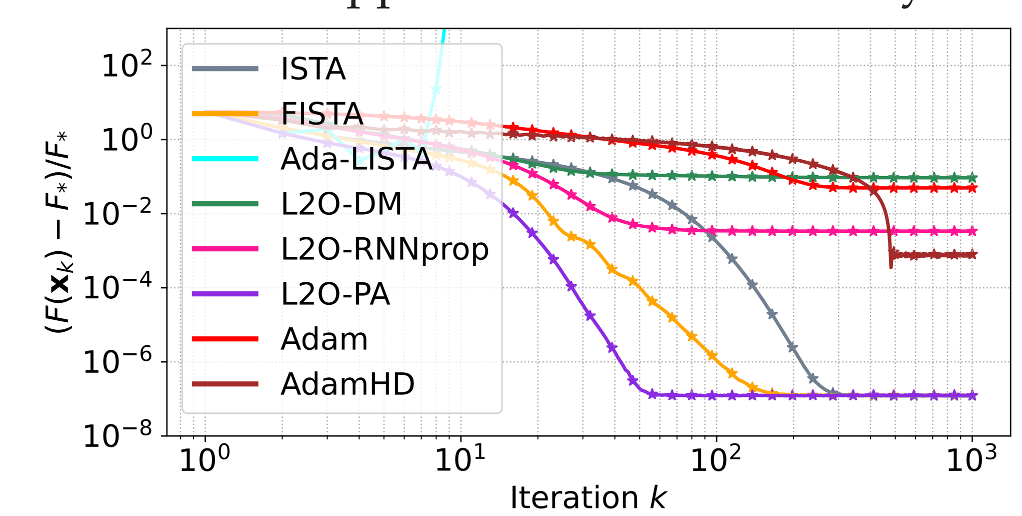
LSTM Parameterization. We choose diagonal $\mathbf{P}_k, \mathbf{A}_k$ over full matrices for efficiency. Similar to [2], we model $\boldsymbol{p}_k$ and $\boldsymbol{a}_k$ as the output of a coordinate-wise LSTM, which is parameterized by learnable parameters ϕ_{LSTM} and takes the current estimate $\boldsymbol{x}_k$ and the gradient $\nabla f(\boldsymbol{x}_k)$ as the input:

$$\begin{aligned}\boldsymbol{o}_k, \boldsymbol{h}_k &= \text{LSTM}\big(\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k), \boldsymbol{h}_{k-1}; \phi_{\text{LSTM}}\big),\\ \boldsymbol{p}_k, \boldsymbol{a}_k &= \text{MLP}(\boldsymbol{o}_k; \phi_{\text{MLP}}).\end{aligned} \tag{7}$$

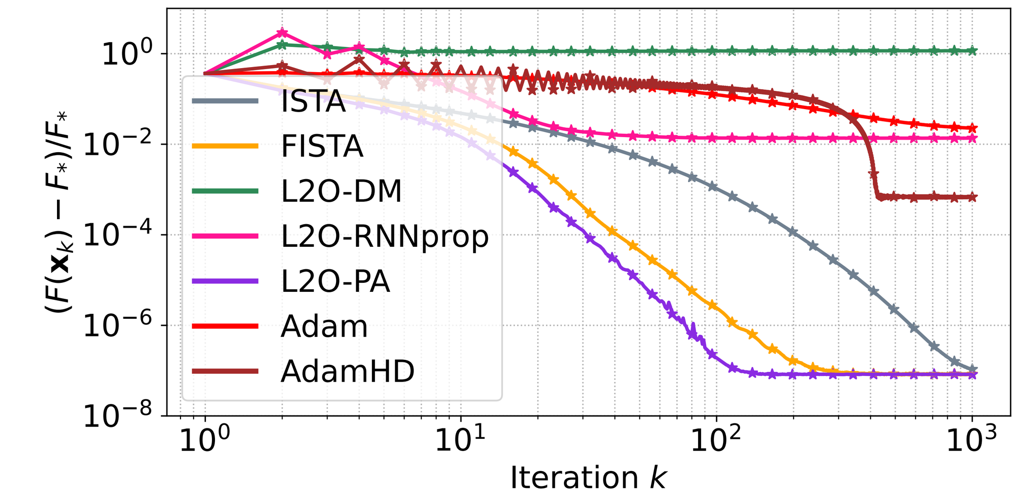
Here, $\boldsymbol{h}_k$ is the internal state maintained by the LSTM with $\boldsymbol{h}_0$ randomly sampled from Gaussian distribution.

Experiment Settings. We test our proposed model (6) with experiments on LASSO and logistic regression using both synthetic data and real data.
- For our method, we learn to predict the diagonal $\boldsymbol{p}_k$ and $\boldsymbol{a}_k$ with LSTM.
- For LASSO, we sample $\boldsymbol{A} \in \mathbb{R}^{250\times 500}$, $\boldsymbol{b} \in \mathbb{R}^{250}$ for the synthetic setting; $\boldsymbol{A} \in \mathbb{R}^{64\times 128}$, $\boldsymbol{b} \in \mathbb{R}^{64}$ extracted with 1,000 8×8 patches from BSD500.
- For logistic regression, we sample $\boldsymbol{A} \in \mathbb{R}^{1000\times 50}$ for the synthetic setting and use *Ionosphere* and *Spambase* datasets as real data.
- Models trained on synthetic data are applied to real data directly.

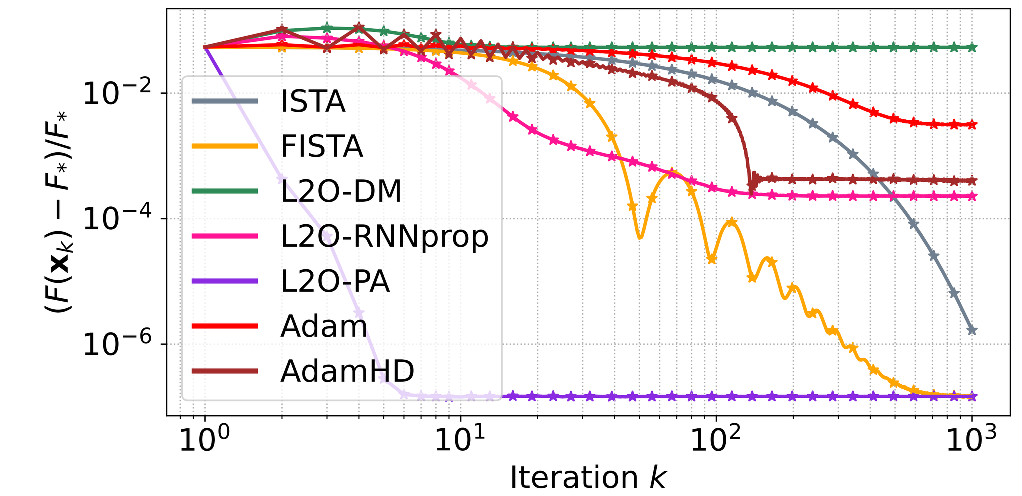
LASSO Synthetic



LASSO Real
Directly Transferred from Synthetic



Logistic Synthetic



Logistic Ionosphere
Directly Transferred from Synthetic