Learning to Optimize: A Primer and A Benchmark

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Zhangyang Wang[*]
{TIANLONG.CHEN,XIAOHAN.CHEN,WUYANG.CHEN,ATLASWANG}@UTEXAS.EDU
Department of Electrical and Computer and Engineering
The University of Texas at Austin, Austin, TX 78712, USA

Howard Heaton
HHEATON@UCLA.EDU
Department of Mathematics, University of California Los Angeles, Los Angeles, CA 90095, USA

Jialin Liu, Wotao Yin
{JIALIN.LIU,WOTAO.YIN}@ALIBABA-INC.COM
Alibaba US, Damo Academy, Decision Intelligence Lab, Bellevue, WA 98004, USA

Abstract

Learning to optimize (**L2O**) is an emerging approach that leverages machine learning to develop optimization methods, aiming at reducing the laborious iterations of hand engineering. It automates the design of an optimization method based on its performance on a set of training problems. This data-driven procedure generates methods that can efficiently solve problems similar to those in the training. In sharp contrast, the typical and traditional designs of optimization methods are theory-driven, so they obtain performance guarantees over the classes of problems specified by the theory. The difference makes L2O suitable for repeatedly solving a certain type of optimization problems over a specific distribution of data, while it typically fails on out-of-distribution problems. The practicality of L2O depends on the type of target optimization, the chosen architecture of the method to learn, and the training procedure. This new paradigm has motivated a community of researchers to explore L2O and report their findings.

This article is poised to be the first comprehensive survey and benchmark of L2O for continuous optimization. We set up taxonomies, categorize existing works and research directions, present insights, and identify open challenges. We also benchmarked many existing L2O approaches on a few but representative optimization problems. For reproducible research and fair benchmarking purposes, we released our software implementation and data in the package **Open-L2O** at `https://github.com/VITA-Group/Open-L2O`.

Keywords: Learning to Optimize, Meta Learning, Optimization, Algorithm Unrolling

1. Introduction

1.1 Background and Motivation

Classic optimization methods are typically hand-built by optimization experts based on theories and their experience. As a paradigm shift from this conventional design, *learning to optimize* (**L2O**) uses machine learning to improve an optimization method or even generate a completely new method. This paper provides a timely and up-to-date review of the rapid growing body of L2O results, with a focus on continuous optimization.

---

*. All authors are listed in alphabetic order. Correspondence shall be addressed to Z. Wang.

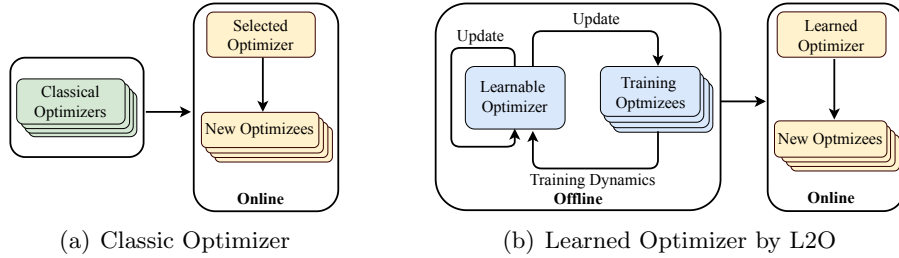(a) Classic Optimizer   (b) Learned Optimizer by L2O

Figure 1: (a) Classic optimizers are manually designed; they usually have few or no tuning parameters; (b) Learned optimizers are trained in an L2O framework over a set of similar optimizees (called a task distribution) and designed to solve unseen optimizees from the same distribution.

Classic optimization methods are built upon components that are basic methods — such as gradient descent, conjugate gradient, Newton steps, Simplex basis update, and stochastic sampling — in a theoretically justified manner. Most conventional optimization methods can be written in a few lines, and their performance is guaranteed by their theories. To solve an optimization problem in practice, we can select a method that supports the problem type at hand and expect the method to return a solution no worse than its guarantee.

L2O is an alternative paradigm that develops an optimization method by training, i.e., learning from its performance on sample problems. The method may lack a solid theoretical basis but improves its performance during the training process. The training process often occurs offline and is time consuming. However, the online application of the method is (aimed to be) time saving. When it comes to problems where the target solutions are difficult to obtain, such as nonconvex optimization and inverse-problem applications, the solution of a well-trained L2O method can have better qualities than those of classic methods. Let us call the optimization method (either hand-engineered or trained by L2O) the *optimizer* and call the optimization problem solvable by the method the *optimizee*. Figure 1 compares classic optimizers and L2O optimizers and illustrates how they are applied to optimizees (yellow boxes).

In many applications of optimization, the task is to perform a certain type of optimization over a specific distribution of data repeatedly. Each time, the input data that define the optimization are new but similar to the past. We say such an application has a narrow *task distribution*. Conventional optimizers may be tuned for a task distribution, but the underlying methods are design for a theory-specified class of optimization problems. We often describe a conventional optimizer by the formulation (and its math properties), not the task distribution. For example, we say an optimizer can minimize a *smooth-and-convex* objective function subject to *linear* constraints. In L2O, the training process shapes the optimizer according to both the formulation *and* the task distribution. When the distribution is concentrated, the learned optimizer can "overfit" to the tasks and may discover "short cuts" that classic optimizers do not take.

L2O aims to generate optimizers with the following strengths:

1. An optimizer learned by L2O is expected to complete a set of optimizees from the same task distribution at a much faster speed than classic methods. In particular, such an L2O method can run even faster than so-called optimal methods such as Nesterov's faster gradient descent (FGD) [3] on a set of problems well suited for these optimal methods.
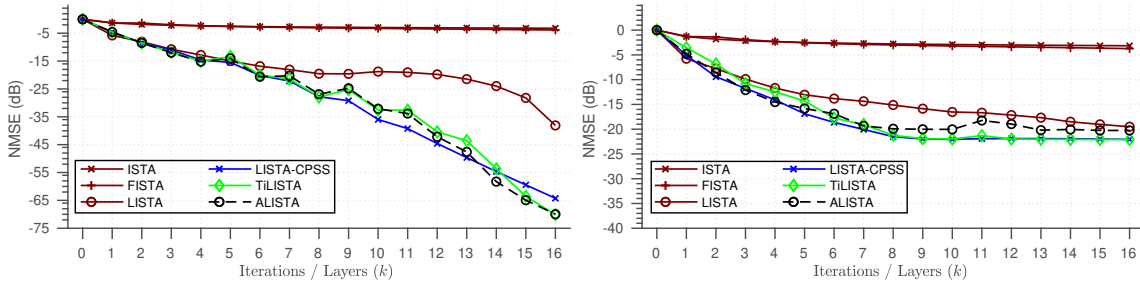
2

Figure 2: L2O for compressive sensing. In our prior work [1; 2], we successfully demonstrated that L2O optimizers (LISTA, LISTA-CPSS, TiLISTA and ALISTA) converge much faster than the two popular iterative solvers (ISTA, FISTA). Left: noiseless case. Right: noisy case (SNR = 20). X-axis is the number of iterations; Y-axis is normalized mean squared error (lower is better). *See more details from the image source: Figure 1 of [2].*

2. The learned optimizer may also return a higher-quality solution to a difficult task than classic methods given a similar amount of computing budget. For instance, an L2O method can recover a sparse signal that is much more faithful (meanwhile much faster) than LASSO[1].

Therefore, when it comes to a small set of optimization tasks, L2O presents a potential approach to break the limits of analytic methods.

L2O has started to demonstrate great potential in some areas of optimization and applications. Examples include convex ℓ_1-minimization [4], neural network training [5], black-box optimization [6] and combinatorial optimization [7]. For example, a class of L2O methods, in the form of algorithm unrolling (which we review below), has state-of-the-art results in compressed sensing and inverse problems. The L2O methods in [1; 2] converge much faster than the classic ISTA/FISTA optimizers (see Figure 2), using an order-of-magnitude fewer iterations on unseen optimizees from the same distribution to reach the same accuracy. Application areas benefiting from L2O methods include computer vision [8; 9; 10], medical imaging [11; 12], signal processing and communication [13; 14], policy learning [15], game theory [16; 17], computational biology [18; 19], and even software engineering [20]. For example, the applications of algorithm unrolling, a popular form of model-based L2O, has been specifically reviewed in Section 3.3. As another example, in the practical application of domain adaptation, model-free L2O has demonstrated the ability to adapt a pre-trained classifier to a new domain at less computational resource costs, achieving the same accuracy or generalization performance [21; 22].

1.2 Preliminaries

L2O starts with an architecture of the optimizer with free parameters to learn, so one must prepare a set of training samples that represent the task distribution, as well as choosing a training approach. The training process (Figure 1 (b) left) executes the approach, which iteratively applies the current optimizer to the sample optimizees and uses the observed performance to update the parameters. Over the time, the optimizer adapts to the training samples. This process trains an optimizer to take a series of actions on the optimizees, analogous of classic learning where we train ML models to make predictions or decisions.

---

1. In this example, the task is to recover a sparse signal, not necessarily solving the LASSO minimization.

We now formalize our notation. Consider an optimization problem $\min_{\mathbf{x}} f(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^d$. A classic optimizer often iteratively updates $\mathbf{x}$ based on a handcrafted rule. For example, the first-order gradient descent algorithm is to take an update at each iteration t based on the local landscape information at the instantaneous point $\mathbf{x}_t$: $\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha \nabla f(\mathbf{x}_t)$, where α is the step size.

L2O has lots of freedom to use the available information. The information $\mathbf{z}_t$ available at time t can include the existing iterates $\mathbf{x}_0, \ldots, \mathbf{x}_t$ as well as their gradients $\nabla f(x_0), \ldots, \nabla f(\mathbf{x}_t)$, etc. Then the L2O approach models an update rule by a function g of $\mathbf{z}_t$: $\mathbf{x}_{t+1} = \mathbf{x}_t - g(\mathbf{z}_t, \phi)$, where the mapping function g is parameterized by ϕ.

Finding an optimal update rule can be formulated mathematically as searching for a good ϕ over the parameter space of g. In order to find a desired ϕ associated with a fast optimizer, [5] proposed to minimize the weighted sum of the objective function $f(\mathbf{x}_t)$ over a time span (which is called the unrolling length) T given by:

$$\min_{\phi} \mathbb{E}_{f \in \mathcal{T}} \left[\sum_{t=1}^{T} w_t f(\mathbf{x}_t) \right], \quad \text{with} \quad \mathbf{x}_{t+1} = \mathbf{x}_t - g(\mathbf{z}_t, \phi), \ t = 1, \ldots, T-1, \qquad (1)$$

where $w_1, \ldots, w_T$ are the weights and f represents an optimizee from an ensemble $\mathcal{T}$ of optimizees that represent the target task distribution. Note that the parameter ϕ determines the objective value through determining the iterates $\mathbf{x}_t$. L2O solves the training problem eq. (1) for a desirable ϕ and the update rule $g(\mathbf{z}_t, \phi)$.

In practice, the choices of w_t vary by case and depend on empirical settings. For example, many L2O models for sparse coding unroll to a fixed length T for all optimizees, and then only minimize the step-T functional value [23; 24], i.e., $w_T = 1$ and $w_1 = \cdots = w_{T-1} = 0$. The work in [5], on the other hand, assigns nonzero w_t values and report more efficient Backpropagation Through Time (BPTT).

In L2O, one specifies the architecture of optimizer, which consists of both learnable components ϕ and fixed components. We address both of them in this paper. The update rule g is often parameterized by multi-layer neural networks or recurrent neural networks. In theory, neural networks are universal approximators, so L2O may discover completely new, optimal update rules without referring to any existing updates. Since this kind of L2O architecture does not need a model, we refer to it as *model-free* L2O.

The shortcomings of model-free L2O include: lacking convergence guarantees and requiring a high number of training samples. On tasks where classic operations — such as projection, normalization, and decaying step sizes — are critical to good performance, model-free L2O either cannot achieve good performance or require largely many training problems to discover classic operations from the scratch. To avoid these shortcomings, we consider incorporating the existing methods as base or starting points for learning, which reduce the search to fewer parameters and a smaller space of algorithms. We call this alternative approach *model-based* L2O.

1.3 Broader Contexts

When the tasks are general machine learning tasks, e.g., determining the parameters of a prediction model by minimizing its training loss, L2O overlaps with meta-learning [25; 26; 27]. It is worth noting that "meta-learning" in different communities has different meanings. Here, meta-learning refers to using a method to improve learning algorithm(s), also called "learning to learn" [5; 6]. Recent results in this line of research contributes a significant

portion of L2O development [28]. The goal of L2O captures two main aspects of meta learning: rapid learning within tasks, and transferable learning across many tasks from some same distribution. However, L2O is not entirely meta-learning, because it takes into account domain knowledge of optimization and applies to many non-learning optimization tasks such as solving inverse problems.

L2O shares many similarities with AutoML [29]. The term "AutoML" broadly refers to the automation of any step(s) in the ML lifecycle. Traditionally, AutoML research focuses on model selection, algorithm selection, and hyperparameter optimization. These methods accelerate the design iterations of many types of ML algorithms, such as random forests, gradient boosting, and neural networks. AutoML recently draws (back) the mainstream attention because of its significant success in enhancing the performance of deep learning [30]. Among the topics under AutoML, those most relevant to L2O are algorithm configuration [31] and hyperparameter optimization (HPO) [32]. Algorithm configuration determines a high-performing configuration of some algorithm across a given set of problem instances. HPO tunes a set of hyperparameters specifically for an ML algorithm, via Bayesian optimization [33] or classification-based optimization [34]. [35] has combined algorithm selection and hyperparameter optimization, also known as CASH. The main difference between these works and L2O lies in that L2O can discover *new optimization methods* from a parameterized algorithmic space of optimizers, rather than only *selecting* from or *tuning* a few existing optimizers. Yet, the boundary between HPO and L2O is often blurry since certain L2O methods [2; 36; 37] configure or predict hyperparameters for existing optimizers only.

L2O is closely related to the new frontier of *learning-augmented algorithms* [38]. Classic algorithms are designed with worst-case performance guarantees in mind and do not adapt to actual inputs. On the other hand, ML algorithms often achieve competitive performance by adapting to inputs, but their worst-case performance on (unseen) inputs degrades significantly. Learning-augmented algorithms combine the best of both worlds, using ML to improve the performance of classic algorithms, by adapting their behaviors to the input distribution. Examples include learning-augmented data structures [39; 40], streaming and sketching algorithms [41; 42], online algorithms [43], error-correcting codes [44; 45], scheduling algorithms [46], approximation algorithms [47], and safeguarded learned algorithms [24]. L2O can be counted as a subset of those learning-augmented algorithms. A comprehensive list of relevant materials can be found in [38].

Previous review articles and differences   There have been a number of review articles on meta learning [25; 26] and AutoML [29; 48; 49].

The work in [50] surveys algorithm unrolling. Work [51] reviews model-based deep learning. These articles have overlapping scopes with ours as algorithm unrolling is a main (but not the only) technique of model-based L2O. This article features a more comprehensive coverage including both model-based and model-free L2O approaches.

Lastly, let us mention recent works that leverage ML for combinatorial and discrete optimization [52; 53; 54; 55; 56]. [57] provides a comprehensive review on ML for combinatorial optimization.

1.4 Paper Scope and Organization

We draw distinctions between model-free and model based L2O approaches and review many L2O techniques. Emphases are given to recurrent network-based L2O methods, algorithm

unrolling, and plug-and-play. We discuss how to train them effectively and how their designs can benefit from the structures of the optimizees and classic optimization methods.

We benchmark existing model-based and model-based L2O approaches on a few representative optimization problems. We have released our software implementation and test cases as the **Open-L2O** package at (`https://github.com/VITA-Group/Open-L2O`), in the spirit of reproducible research and fair benchmarking.

The rest of the article is organized as follows. Section 2 and Section 3 review and categorize the existing works in model-free and model-based L2O, respectively. Section 4 describes the Open-L2O testbeds, our comparison experiments, and our result analyses. The article is concluded by Section 5 with a discussion on research gaps and future work.

2. Model-Free L2O Approaches

A model-free L2O approach in general aims to learn a parameterized update rule of optimization without taking the form of any analytic update (except the updates being iterative). The recent mainstream works in this vein [5; 6; 58; 59; 18] leverage recurrent neural networks (RNNs), most of which use the long short-term memory (LSTM) architecture. An LSTM is unrolled to perform iterative updates and trained to find short optimization trajectories. One set of parameters are shared across all the unrolled steps. At each step, the LSTM takes as input the optimizee's local states (such as zero-order and first-order information) and returns the next iterate.

Model-free L2O shares the same high-level workflow depicted in Figure 1 (b). It is divided into two stages: an offline *L2O training* stage that learns the optimizer with a set of optimizees sampled from the task distribution $\mathcal{T}$; then, an online *L2O testing* stage that applies the learned optimizer to new optimizees, assuming they are samples from the same task distribution. Table 1 compares a few recent, representative works in this area, from multiple lens including what optimizer architecture to select, what objectives and metrics that meta training and meta testing each use, and what input feature and other techniques are adopted. There are a great variety.

Table 1: Summary and comparison of representative model-free L2O methods.

Paper	Optimizer Architecture	Input Feature	Meta Training Objective	Additional Technique	Evaluation Metric
[5]	LSTM	Gradient	Meta Loss	Transform input gradient ∇ into $\log(\nabla)$ and $\text{sign}(\nabla)$	Training Loss
[6]	LSTM	Objective Value	Objective Value	N/A	Objective Value
[58]	LSTM	Gradient	Meta Loss	Random Scaling Combination with Convex Functions	Training Loss
[59]	Hierarchical RNNs	Scaled averaged gradients, relative log gradient magnitudes, relative log learning rate	Log Meta Loss	Gradient History Attention Nesterov Momentum	Training Loss
[60]	MLP	Gradient	Meta Loss	Unbiased Gradient Estimators	Training Loss Testing Loss
[61]	RNN Controller	Loss, Gradient	Meta Loss	Coordinate Groups	Training Loss
[62]	Searched Mathematical Rule by Primitive Functions	Scaled averaged gradients	Meta Loss	N/A	Testing Accuracy
[18]	Multiple LSTMs	Gradient, momentum, particle's velocity and attraction	Meta Loss and Entropy Regularizer	Sample- and Feature- Attention	Training Loss
[63]	RNN	Input Images, Input Gradient	Meta Loss	N/A	Standard and Robust Test Accuracies
[64]	LSTM	Input Gradient	Meta Loss	N/A	Training Loss and Robust Test Accuracy

2.1 LSTM Optimizer for Continuous Minimization: Basic Idea and Variants

As the optimization process can be regarded as a trajectory of iterative updates, RNNs are one natural option with a good inductive bias to learn the update rule. The first pioneering model-free L2O method [5] proposed to model the update rule implicitly by gradient descent. This first LSTM method for L2O has often been referred to as <u>L2O-DM</u> in literature[2].

At high-level, the learned optimizer (modeled by LSTM) is updated by the gradients from minimizing the loss induced by the optimizee, and optimizees are updated by the update rule predicted by the optimizer:

$$\mathcal{L}(\phi) = \mathbb{E}_{(\theta_0, f) \in \mathcal{T}} \left[\sum_{t=1}^{\mathrm{T}} w_t f(\theta_t) \right] \quad \text{where} \quad \begin{array}{c} \theta_{t+1} = \theta_t + g_t \\ \begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = m(\nabla_t, h_t, \phi) \end{array} . \tag{2}$$

In Eq. 2, f represents an optimizee (e.g. a neural network with its loss function), with θ_0 denoting the initialization of f. A set of optimizees and their initializations are sampled from the task distribution $\mathcal{T}$. ϕ is the parameters of the L2O optimizer. $\nabla_t = \nabla_\theta f(\theta_t)$ is the gradient of the objective function with respect to the optimizee's parameters. m is the LSTM optimizer, and g and h are update and hidden state produced by m, respectively. T is the maximal unrolling length for LTSM, often set due to the memory limitation; and w_t = 1 was used here for every t.

Two ad-hoc training techniques were utilized here. Firstly, each optimization variable (or called coordinate) of the optimizee's parameter θ_t shared the same LSTM weights, but with different hidden states, in order to reduce the memory overhead faced when scaling up the number of optimization coordinates; this trick has been inherited by many LSTM-type L2O works. Secondly, to stabilize the optimizer's training, the authors proposed to reduce the dynamic range of the optimizee's gradient magnitudes, by preprocessing ∇_t into $(\log(\nabla_t), \mathrm{sgn}(\nabla_t))$ as the input into the optimizer.

The authors of [5] conducted a few proof-of-concept studies on small-scale tasks such as MNIST classification, where they showed the learned optimizer m can converge faster than some stochastic gradient descent based optimizers such as SGD, RMSprop, and Adam. Despite this initial success, the scalability and generalizability of [5] remain underwhelming. Two specific questionable shortcomings are:

- **Generalizability of learned optimizer to unseen and potentially more complicated optimizees.** The L2O training set contains sampled optimizees from the target distribution. Just like typical machine learning models, we might expect a learned optimizer to both interpolate and extrapolate from the seen (meta-)training set, the latter being more challenging. Taking training deep networks for example, during meta-testing, we might expect the learned optimizer to generalize (extrapolate) to the training of deeper or wider networks beyond instances seen in the meta training set. This generalizability is demanded also due to the memory bottleneck during meta training. To update the L2O optimizer $m(\phi)$ using back-propagation, we need to keep the gradients and the computation graph of the optimizee in memory. Therefore, the memory bottleneck arises when the unroll length of the $m(\phi)$ becomes large, or the dimension of optimizee's parameter θ is high. Other types of generalization, such as training a network with unseen different activation functions or loss functions, are also found to be desirable yet challenging [5].

---

2. Methods that are underscored in this section are also later evaluated and benchmarked in Section. 4.

Moreover, applying the L2O optimizers to train more complex neural networks may meet more sophisticated loss landscapes, which cannot be well learned from observing simple training dynamics from small networks. For example, it is well known that training over-parameterized networks will exhibit sharply distinct behaviors at small and large learning rates, and the two regimes usually co-exist in practical training and are separated by a phase transition [65; 66; 67].

- **Generalizability of learned optimizer to longer training iterations.** Training larger models naturally takes more iterations, which calls for longer-term L2O modeling. An LSTM model can in principle characterize longer-term dependency by unfolding more time steps (i.e., longer unrolling length T). However, that often results in L2O training instability due to the optimization artifacts of LSTM such as gradient explosion or vanishing, in addition to the practical memory bottleneck. Therefore, most LSTM-based L2O methods [5; 6; 58; 59; 60; 18; 19; 68] are forced to limit their maximal unroll lengths and to truncate the unrolled optimization (e.g., up to 20 steps). As a result, the entire optimization trajectory is divided into consecutive shorter pieces, where each piece is optimized by applying a truncated LSTM.

  However, choosing the unrolling (or division) length faces a well-known dilemma [58]: on one hand, a short-truncated LSTM can result in premature termination of the iterative solution. Naively unfolding an LSTM-based L2O optimizer (trained with a small T) to more time stamps at meta-testing usually yields unsatisfactory results. The resulting "truncation bias" causes learned optimizers to exhibit instability and yield poor-quality solutions when applied to training optimizees.

The above research gaps motivate several lines of follow-up works, as summarized below:

Debiasing LSTM Truncation As explained above, selecting the unrolling length T reflects a fundamental dilemma in LSTM-type L2O models: a longer T causes optimization difficulty at L2O training, while a smaller T incurs larger "truncation bias" and hampers the generalization at meta-testing.

Many LSTM-type works focus on addressing this truncation bias. [59] drew the unroll length of their truncated optimization from a heavy tailed distribution. [60] proposed to replace LSTM with just MLP for the optimizer. The authors of [60] proposed to smooth the loss landscape by using two unbiased gradient estimators with dynamic re-weighting:

$$g_{\mathrm{rp}} = \frac{1}{S}\sum_{s=1}^{S}\nabla_\theta f(\theta + n_s), \quad n_1, \cdots, n_S \sim N(0, \sigma^2 I) \quad \text{i.i.d.} \tag{3}$$

$$g_{\mathrm{es}} = \frac{1}{S}\sum_{s=1}^{S}f(\tilde\theta_s)\nabla_\theta[\log(\mathbb{P}(\tilde\theta_s))], \quad \tilde\theta_1, \cdots, \tilde\theta_S \sim N(\theta, \sigma^2 I) \quad \text{i.i.d.} \tag{4}$$

$$g_{\mathrm{merged}} = \frac{g_{\mathrm{rp}}\sigma_{\mathrm{rp}}^{-2} + g_{\mathrm{es}}\sigma_{\mathrm{es}}^{-2}}{\sigma_{\mathrm{rp}}^{-2} + \sigma_{\mathrm{es}}^{-2}} \tag{5}$$

g_{rp} and g_{es} are graident estimated by "reparameterization trick" [69] and "evolutionary strategies" [70], and σ_{rp} and σ_{es} are empirical estimates of the variances of g_{rp} and g_{es}, respectively. In this way, they demonstrated L2O can train convolutional network networks faster in wall-clock time compared to tuned first-order methods, with reduced test losses. Their later work [71] also found that the meta-learned optimizer can train image classification models such that they are robust to unseen image corruptions.
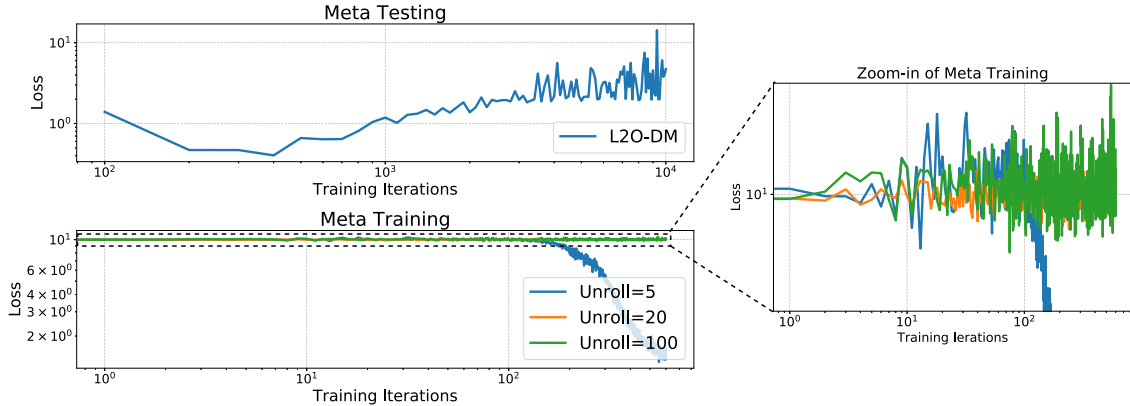
Figure 3: The dilemma of longer unrolling in both L2O testing and L2O training: (1) *generalization failure* at L2O testing: the upper figure shows the meta-testing loss of the vanilla L2O method [5] to quickly diverge, as we increase the number of steps at the meta-testing. This failure case was also observed in Figure 2 of [59] and in Figure 2 of [58]. (2) *optimization failure* at L2O training: the bottom figure shows the L2O training loss to also not decrease when it adopts longer unrolling lengths, due to the optimization artifacts of LSTM such as gradient explosion or vanishing.

Stronger LSTM Architectures Another direction explored is to introduce stronger RNN/LSTM architecture to L2O. Instead of using a single RNN layer, Wichrowska et al. [59] leveraged three RNN layers to learn the optimization in different scales. They organized three RNN layers in a hierachical fashion, here we simply denote them as "bottom RNN", "middle RNN", "upper RNN". The "bottom RNN" directly takes the scaled gradients from the optimizee as input. Given a specific training step, the "middle RNN" receives the average all hidden states from "bottom RNN" across the optimizee's parameter coordinates, and outputs a bias term to the "bottom RNN". Further, the "upper RNN" takes the averaged hidden states from "middle RNN" across a certain window of optimizee's training steps. By using smaller hidden states in "bottom RNN", this hierarchical design of L2O optimizer achieved lower memory and compute overhead, while achieving better generalization. We refer [59] as <u>L2O-Scale</u> in this article.

Improved L2O Training Techniques In order to improve the generalization of the learned optimizer to both longer unrolling (i.e. longer optimization iterations) and unseen functions, [58] proposed two training tricks. The first one, called *random scaling*, could be viewed as a special "data augmentation" for L2O: a coordinate-wise scale factor **c** was randomly generated at each iteration to scale the parameters of the optimizee during L2O training: $f_{\mathbf{c}}(\theta) = f(\mathbf{c}\theta)$. It was motivated by the observation that in many optimization problems such as the quadratic function $f(\theta) = \lambda\|\theta\|_2^2$, the ideal update rule should achieve the same minima under varying λ. The second trick was to add a convex term during L2O training, as inspired by the proximal algorithms [72]; and avoided large random updates when the L2O optimizer is under-trained. We refer [58] as <u>L2O-RNNprop</u> in this article.

Lately, the authors of [73] took a deeper dive into improved training techniques for L2O models. The authors first presented a progressive training scheme, which gradually increased the optimizer unroll length to mitigate the L2O dilemma of truncation bias (shorter unrolling) versus gradient explosion (longer unrolling). Furthermore, they presented an off-policy imitation learning approach to guide the L2O training, by forcing the L2O op-

timizer to mimic the update rules generated by analytic optimizers. The authors of [73] evaluated their improved training techniques with a variety of state-of-the-art L2O models [5; 58; 59], and achieved boosted performance (lower training losses of unseen optimizees) without changing the original L2O RNN architecture in each method. We refer [73] as L2O-enhanced in this article.

2.2 Other Common Implementations for Mode-Free L2O

While LSTM is so far the mainstream model, other optimizer models have also been explored. We describe two alternatives: *reinforcement learning* (RL), and *neural symbolics.*

[61] proposed to learn an RL policy π to predict the update rule, as the learned optimizer. The policy π samples the update steps from a Gaussian distribution. The mean and variance of the Gaussian distribution are learnable parameters of the L2O policy, updated by reinforcement learning. The observation of the policy is composed of the current value of objective function (i.e. loss), the recent gradients, and changes of the objective values up to H steps ($H = 25$ in their experiments). The policy receives the decrease of training loss as the reward. Logistic regression functions and two-layer neural nets were leveraged as the testbeds. Further on, [28] proposed to group coordinates under permutation invariance (e.g., weight matrix or a bias vector) into a coordinate group. This formulation reduced the computation cost of expensive optimization problems, making the proposed method extensible to wider neural networks. Generalizability of the learned policy was also studied by training the policy on shallow and narrow networks, and test on wider layers. [74] learned to update optimizer hyperparameters instead of model parameters, directly using novel features, actions, and a reward function to feed RL. They demonstrated promising scalability to large-scale real problems.

It is worthy to mention a special L2O work [62], that explored L2O from the neural symbolic prospective. The authors also leveraged an RL controller, but avoided directly modeling the update rules. Instead, they designed a search space of operands (gradient g, gradient's running exponential moving average $\hat{m}$, etc.), unary functions of input x (e^x, $\log(x)$, $\sqrt{|x|}$, etc.), and binary functions (mapping (x, y) to $x + y$, $x - y$, $x * y$, etc.). The RL controller was learned to select a sequence of elements from the search space, formulate a function to process the input, and output the update rule. Their searched best L2O optimizer (the RL controller) shows strong transferability, and improved training performance (lower training losses) on different tasks and architectures, including ImageNet classification and Google's neural machine translation system. Their idea was further developed in [75], to automatically discover complete machine learning algorithms from raw mathematical operations as building blocks, which concerns a more general problem than L2O.

2.3 More Optimization Tasks for Model-Free L2O

Black-box Optimization [6] pioneered to extend the LSTM L2O framework [5] to derivative-free or black-box function optimization. Due to the absence of the optimizee's gradients as input, the authors of [6] instead treated the optimizee's input-output pair as the observation of the optimizer, and formulated the optimization as an exploration-exploitation trade-off problem. They updated the optimizer's hidden state $\mathbf{h}_t$ by the observation from

the last step $(\mathbf{x}_{t-1}, y_{t-1})$ and then chose a new query point $\mathbf{x}_t$ to explore

$$\mathbf{h}_t, \mathbf{x}_t = \text{RNN}_\theta \left(\mathbf{h}_{t-1}, \mathbf{x}_{t-1}, y_{t-1} \right) \tag{6}$$

$$y_t \sim p \left(y \mid \mathbf{x}_t, \cdots, \mathbf{x}_1 \right), \tag{7}$$

where the function value y_t was incrementally sampled from a Gaussian Process (p) at each query point $\mathbf{x}_t$ in the experiments. During L2O training, it was assumed that the derivatives of function value y_t can be computed with respect to the input $\mathbf{x}_t$, which means the errors will be backpropagated for L2O training, but not needed at L2O testing time. [6] demonstrated that their learned RNN optimizers are competitive with state-of-the-art Bayesian optimization packages (Spearmint [76], SMAC [31], and TPE [77]).

Particle Swarm Optimization   Current L2O methods mostly learn in the space of continuous optimization algorithms that are point-based and uncertainty unaware. Inspired by population-based algorithms (e.g. swarm optimization), [18] estimated the posterior directly over the global optimum and used an uncertainty measure to help guide the learning process. The authors designed a novel architecture where a population of LSTMs jointly learned iterative update formula for a population of samples (or a swarm of particles). The model can take as input both point-based input features, such as gradient momentum; and population-based features, such as particle's velocity and attraction from swarm algorithms. To balance exploration and exploitation in search, the authors directly estimated the posterior over the optimum and included in the meta-loss function the differential entropy of the posterior. Furthermore, they learn feature- and sample-level importance reweighting (often called "attention" in deep learning) in the L2O model, for more interpretable learned optimization rules. Their empirical results over non-convex test functions and the protein-docking application demonstrated that this new L2O largely outperforms the off-the-shelf Particle Swarm Optimization (PSO) algorithm [78] and the vanilla LSTM-based L2O methods that are not uncertainty-aware [5]. We refer [18] as L2O-Swarm in this article.

Minimax Optimization   One more challenging testbed for model-free L2O is to solve continuous minimax optimization, that is of extensive practical interest [79; 80; 81], yet notoriously unstable and difficult. Three prior works [63; 82; 64] tried to plug in L2O into a specific application of minimax optimization called adversarial training [80]:

$$\min_\theta \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}) \sim D} \left\{ \max_{\boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x}, \epsilon)} \mathcal{L} \left(f \left(\boldsymbol{x}' \right), \boldsymbol{y} \right) \right\}, \tag{8}$$

where D is the empirical distribution of input data, and the inner maximization is defined as the worst-case loss within a small neighborhood $\mathbb{B}(\boldsymbol{x}, \epsilon)$. In both works, the L2O only predicted update directions for the inner maximization problem, while the outer minimization was still solved by classic optimizer. They empirically showed that L2O can improve the solution quality to the inner maximization optimization, hence also leading to a better minimax solution and a more robustly trained model.

A latest work [83] extended L2O to solve minimax optimization from end to end, for the first time. The authors proposed Twin L2O, consisting of two LSTMs for updating min and max variables, respectively. This decoupled design was shown by ablation experiments to facilitate learning, particularly when the min and max variables are highly non-symmetric. Several enhanced variants and training techniques were also discussed. The authored benchmarked their L2O algorithm on several relatively basic and low-dimensional test problems,

11

and on which L2O compared favorably against state-of-the-art minimax solvers. How to scale up L2O for fully solving minimax problems of practical interest, such as adversarial training or GANs, remained to be an open challenge.

Game Theory RL-based model-free L2O has very recently found interest from the game theory field. [16] proposed to train multi-agent systems (MAS) to achieve symmetric pure Nash equilibria. Such equilibria needs to satisfy certain constraints so that MAS are calibrated for practical use. The authors adopted a novel dual-RL-based algorithm to fit emergent behaviors of agents in a shared equilibrium to external targets. They used parameter sharing with decentralized execution, to train multiple agents using a single policy network, while each agent can be conditioned on agent-specific information. The methodology shared similarities to [18]. Another work [17] extended consensus optimization to the constrained case. They introduced a new framework for online learning in non zero-sum games, where the update rule's gradient and Hessian coefficients along a trajectory are learned by an RL policy, conditioned on the game signature. The authors mentioned that the same problem might potentially be solved by using the LSTM-based approach too [83].

Few-shot Learning Another application of the LSTM L2O framework [5] explores the application of model-free L2O in the small data regime. [84] first adopted a model-free L2O approach to learn a few-shot classifier, accessing only very few labeled examples per class at training. The authors of [84] proposed to learn an LSTM-based meta-learner to optimize each task-specific classifier using the cell states in the LSTM, inspired by their observation that the gradient descent update on the parameters in the classifier resembles the cell state update in an LSTM. [85] simplified [84] by constraining the optimization on the learner classifier to one step of gradient descent but with learnable initialization and step size.

3. Model-Based L2O Approaches

We now overview model-based L2O approaches, which mark the recent movement to fuse traditional model-based optimization algorithms with powerful deep learning architectures [50]. Rather than use general-purpose LSTMs, these methods model their iterative update rules through a learnable architecture that is inspired by analytic optimization algorithms. Often, these learned methods can approximate problem solutions with tens of iterations whereas their classic counterparts make require hundreds or thousands of iterations [2].

At a high level, model-based L2O can be viewed as a "semi-parameterized" option that takes advantage of both model-based structures/priors and data-driven learning capacity[3]. The growing popularity of this approach lies in its demonstrated effectiveness in developing compact, data-efficient, interpretable and high-performance architectures, when the underlying optimization model is assumed available or can be partially inferred. There is a large design space to flexibly balance between the two. Most model-based L2O methods take one of the two following mainstream approaches.

The first approach is known as **plug and play (PnP)**, and the key idea here is to *plug* a previously trained neural network (NN) into part of the update for an optimization algorithm (i.e., in place of an analytic expression), and then *play* by immediately applying the modified algorithm to problems of interest (without additional training). We illustrate this with the common alternating direction method of multipliers (ADMM) formulation of

---

3. We slightly abused the term of *semi-parametric model* from statistical learning, to represent different meanings: here it refers to blending pre-defined structures/priors into black-box learning models.

PnP (e.g., see [86]). Here the underlying task of the original optimization algorithm is to minimize the sum of two functions f and g, using successive proximal operations. The PnP formulation replaces the proximal for g with an operator H_θ to obtain the iteration:

$$x^{k+1} = H_\theta(y^k - u^k) \tag{9a}$$

$$y^{k+1} = \text{prox}_{\alpha f}(x^{k+1} + u^k) \tag{9b}$$

$$u^{k+1} = u^k + x^{k+1} - y^{k+1}. \tag{9c}$$

Before inserting H_θ into the iteration above, the parameters θ are learned independently as the solution to a training problem, i.e.,

$$\theta \in \arg\min_{\tilde\theta} \mathcal{L}(\tilde\theta). \tag{10}$$

The loss function $\mathcal{L}$ is designed by an independent goal (e.g., to learn a natural image denoising operator). For example, one might model an image recovery problem by using total variation (TV) as a regularizer [87]. In the optimization scheme chosen (e.g., ADMM) to recover the image, one step of the process could be to perform a proximal operation with TV, which effectively acts as a denoiser. The PnP framework proposes replacement of the TV proximal with an existing denoiser (e.g., a neural network [88] or BM3D [89]).

The second approach is known as **algorithm unrolling**, whichs unrolls a truncated optimization algorithm into the structure of a neural network [50]. Updates take the form

$$x^{k+1} = T(x^k; \theta^k), \quad k = 0, 1, 2, \cdots, K. \tag{11}$$

Upon establishing this form, the parameters are learned by an end-to-end approach:

$$\min_{\{\theta^k\}_{k=0}^{K-1}} \mathcal{L}\big(x^K(\{\theta^k\}_{k=0}^{K-1})\big), \tag{12}$$

where $\mathcal{L}$ is the loss function we use in training. We emphasize the distinction that the parameters in unrolled schemes are trained end-to-end using the iterate x^K as a function of each θ^k whereas training occurs separately for PnP. We will introduce how to design $\mathcal{L}$ later in this section. Below we discuss each of these approaches, their typical features, and which approach is suitable for various applications.

	Opt	IP	PnP
Tunable Optimization Model		✓	✓
Tunable Update Formulas	✓	✓	✓
Training Loss tied to Model	✓		
Training Loss measure $u^\star$ error		✓	
Convergence guarantee to ideal $u^\star$	S	S	
Interpretable Updates	S	S	✓

Table 2: Comparison of properties for three types of model-based L2O methods: unrolled objective-based (Opt), inverse problem (IP), and Plug and Play (PnP) methods. Here ✓ and S mean that the corresponding property *always* and *sometimes* holds, respectively.
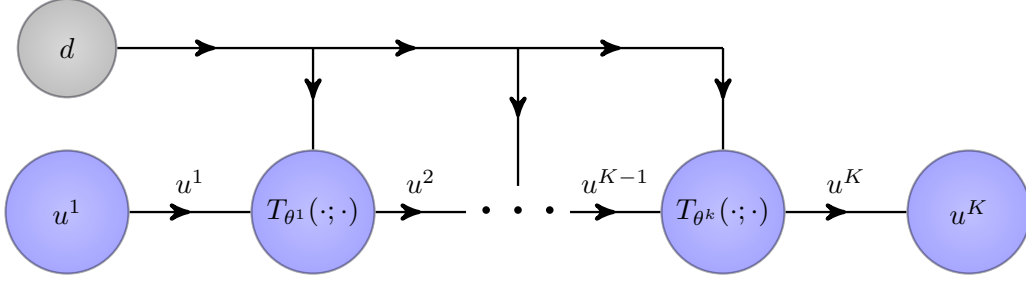
Figure 4: A common approach in various L2O schemes is to form feed forward networks by unrolling an iterative algorithm, truncated to K iterations, and tuning parameters at each layer/iteration k. This generalizes the update formula $u^{k+1} = T(u^k; d)$ to include a dependence on weights θ^k, denoted by a subscript.

3.1 Plug and Play

PnP methods date back to 2013 when they were first proposed for ADMM [86]. The authors replaced one of the proximal operators in ADMM with a denoiser (e.g., K-SVD [90], BM3D [89], and non-local means [91]) and showed empirical performance surpassing the original ADMM algorithm. Several other PnP schemes were also introduced and their empirical success was shown in the literature [92; 93; 94; 95; 96; 97; 98; 99; 100; 101; 102; 103; 104; 105; 106; 107; 108]. The theoretical guarantees of convergence of PnP-ADMM was first provided in [109] under the assumption that the derivative of the denoiser was doubly stochastic. Then [110] proved the convergence of PnP-ADMM with the use of a more realistic "bounded denoisers" assumption. Some other PnP schemes were analyzed under different assumptions [111; 112; 113; 114; 115; 116; 117; 118; 119; 120; 121].

PnP was initially not connected to L2O, until some concurrent works [122; 123; 124] introduced the concept of "learning operators" into a PnP framework. Instead of using a manual-designed denoiser, they modeled the proximal operator as a deep neural network and learned it from data. The empirical performance of such an approach largely exceeded the prior PnP works. From an L2O perspective, the learned methods were able to improve performance by either accelerating execution of a subroutine in the optimization algorithm or providing a "better" solution than previously obtained.

Besides learning the operators in a PnP framework, one can also learn a functional as the regularizer. [125; 126] modeled the statistical distribution of nature images as a Denoising Autoencoder (DAE) and constructed the optimization objective by maximizing the likelihood. The DAE is differentiable; thus, the inverse problem can be solved with gradient-based optimization algorithms. A parameterized discriminator function defined in the Wasserstein distance between two distributions was used by [127] as a learned a functional that discriminates between a ground-truth and fake images. The authors treated this learned functional as a regularizer and used it in a gradient descent algorithm for image inverse problems.

Learning is not only able to help find the denoiser/regularizer in PnP, but is also able to be used in a more meta manner to help find good parameters in PnP. [128] learned a policy network to tune the parameters in Plug-and-Play with reinforcement learning. By using the learned policy, the guided optimization can reach comparable results to the ones using oracle parameters tuned via the inaccessible ground truth.

14

The marriage of PnP with L2O also provides theoretical blessings. While a manual-designed regularizer may not guarantee the convergence of PnP to the desired solution, a learning-based method provides the flexibility to meet the condition of convergence. [88] studied the convergence of some PnP frameworks using fixed-point iterations, showing guaranteed convergence under a certain Lipschitz condition on the denoisers. They then proposed a normalization method for training deep learning-based denoisers to satisfy the proposed Lipschitz condition. Similarly, [129] also provided an approach for building convergent PnP algorithms using monotone operator theory and constraining the Lipschitz constant of the denoiser during training. This fixed point framework was extended by [130] to the RED-PRO framework, which also showed convergence to global solutions. Independently of RED-PRO, [131] proposed a method within the RED-PRO framework for learning projection operators onto compact manifolds of true data. Utilizing assumptions about the manifold and sufficient representation by the sampled data, the authors proved their approach to constructing a PnP operator can provably approximate the projection (in probability) onto a low dimensional manifold of true data.

3.2 Algorithm Unrolling

Herein we overview L2O methods comprised of unrolling iterative optimization algorithms. We start by emphasizing there are two distinct goals of unrolled L2O methods: either to minimize an objective, or to recover a signal. The distinction is that the aim for inverse problems is to tune the parameters so that they minimize the reconstruction accuracy of some "true" signal rather than find a minimizer of the model's objective function. The practicality for objective minimization is to speed up convergence. Keeping these categorizations in mind will help provide the reader intuitive lenses for looking at different approaches. See Table 2 for a comparison of common qualities in these methods and PnP.

Below we provide a comprehensive and organized review of existing works, along multiple dimensions: what problems they solve, what algorithms they unroll, what goals they pursue (objective minimization or signal recovery), and to what extent they freely parameterize their learnable update rule.

3.2.1 Different target problems

We broadly categorize the main problems tackled by model-based L2O in four categories: probabilistic graph models, sparse and low rank regression, differential equations, and quadratic optimization.

- **Sparse and low rank regression**. The most investigated problems in the literature of unrolling is probably the sparse regression, inspired by the first seminal work [4], which unrolls the Iterative Shrinkage Thresholding Algorithm (ISTA) or its block coordinate variant as a recurrent neural network to solve LASSO for fast sparse coding. The unrolling philosophy is also used in low rank regression as it shares some common nature with sparse regression.

    - **LASSO:** Most of unrolling works following [4] also solve LASSO-type optimization problems for sparse coding [132; 133; 134; 135; 23; 136; 137; 138; 139]. Beyond naive LASSO, A natural extension is to apply the same *unrolling and truncating* methodology to solve *group LASSO* [140], i.e. finding solutions with struc-

tured sparsity constraints. [141; 142] extended solving (convolutional) LASSO in a multi-layer setting, and an adapted version of ISTA was unrolled in [143].

- **Analysis model:** Different from the sparse prior in LASSO (also known as the *Synthesis* formulation), which assumes the signal is the linear combinations of a small number of atoms in a dictionary, the sparse analysis model assumes the existence of a forward transformation that sparsifies the signal. [144] applied unrolling to regression problems with total variation regularizations, which follows the analysis formulation. Problems with analysis sparse priors were also tackled using unrolling in [145] in a bi-level optimization context.

- **Other sparse/anti-sparse regression problems:** [146] and [147] unroll the iterative hard thresholding (IHT) algorithm that solves ℓ_0 *minimization* problem instead of LASSO. Unrolling for ℓ_∞ minimization was also considered by [148], leading to so-called anti-sparse representation learning [149].

- **Low Rank Regression:** [150; 151; 152; 153] all extend the unrolling idea to low-rank matrix factorization. Specifically, [152] proposes tailored pursuit architectures for both robust PCA and non-negative matrix factorization with specialized algorithms inspired by the non-convex optimization techniques to alleviate the expensive SVD involved in each iteration.

- **Probabilistic Graphical Model**. The unrolling method can also be adopted to solve probabilistic graphical models. For example, [150] interpret conventional networks as mean-field inference in Markov random fields and obtains new architectures by modeling the belief propagation algorithm to solve the Markov random field problems. [154] formulate Conditional Random Fields with Gaussian pairwise potentials and mean-field approximate inference as recurrent neural networks.

- **Differential equations**. Another line of works [155; 156; 157; 158] unroll the evolution in *Partial Differential Equation* (PDE) systems. While PDEs are commonly derived based on empirical observations. those recent advances offer new opportunities for data-driven discovery of (time-dependent) PDEs from observed dynamic data. One can train feed-forward or recurrent neural networks to approximate PDEs, with applications such as fluid simulation [159]. Loosely related is also significant work on connecting neural networks with *Ordinary Differential Equation* (ODE) systems.

- **Quadratic optimization**. Some recent works investigate the unrolling method in quadratic optimization problems [37; 160], which are easier to solve compared to the problems studied above. The focus here is more on the theoretical analysis of the convergence, and/or the interplay between the unrolled algorithm and the resultant deep model's property (e.g., generalization and stability).

In some scenarios, we are not directly interested in the output of the unrolled model but use it for downstream tasks, e.g. clustering [161] and classification [146; 23; 138]. That will often lead to task-driven joint optimization and the end task output becomes the focus of evaluation, in place of the original unrolling algorithm's output fidelity.

Inverse Problems   In many cases, however, optimization problems with manually designed objective functions only provide approximations to the original signals that we are really interested. This is often due to inexact prior knowledge about the original signals. For example, sparsity and total variation regularizations only partially reflect the complexity of

natural images (approximated sparsity and smoothness) which are hardly true in real-world applications and do not depict the exact characteristics of natural images. Therefore, many works solve the *inverse problem* directly, striving to recover the original signals that we are really interested in. We return to this task in a later subsection.

Constrained Optimization Problems   Most target problems mentioned above in this subsection are unconstrained optimization problems. Some exceptions such as the sparsity-constrained and low-rank regression problems, e.g., LASSO, have equivalent unconstrained formulation under proper regularizations. Some others have easy-to-implement forms of projection onto the constraint sets, including the $\ell_{0/\infty}$-constrained regression [146; 147; 148], and non-negative matrix factorization with non-negative constraints [152].

There have been a few efforts directly tackling more general constrained optimization problems using unrolling. [162] unrolled Frank-Wolfe algorithm to solve the structured regression with general ℓ_p-norm constraint ($p \geq 1$), and proposed a novel closed-form nonlinear pooling unit parameterized by p for the projection. [163] unrolled Frank-Wolfe algorithm for least square problems with affine, non-negative and ℓ_p-norm constraints. It was also the first to apply the unrolled network to financial data processing. [10; 164] investigated image restoration with various hard constraints and unrolled proximal interior point algorithms while incorporating the constraints using a logarithmic barrier.

3.2.2 DIFFERENT ALGORITHMS UNROLLED

For the bulk of iterative optimization algorithms that have been unrolled, we classify them into three categories: forward backward splitting, primal-dual methods, and *others*. The first two categories consist entirely of first-order algorithms, which is due to their low computational complexity and the fact their resultant L2O methods are often more reliable to train. We also emphasize that although various works discussed below may use the same underlying algorithm as the base, they can vary greatly in their performance due to choices regarding what parameters are learned and what safeguard precautions are used to ensure convergence (discussed further in subsequent sections). We also emphasize that the majority of these algorithms revolve around obtaining some form of sparsity/low rank solution.

We begin with forward backward splitting (FBS). The simplest learned scheme is *Gradient Descent*, which is where the the gradient descent operation is applied (i.e., the forward operator) and the backward operator is simply the identity. This class of methods is studied in many works (e.g., see [135; 100; 165; 166; 160; 167; 168; 169; 170; 155; 127; 171; 139; 172; 173]). However, the most popular focus in the literature is on problems with sparsity constraints, which are usually modeled by ℓ_1/ℓ_0-minimization. The former, also known as LASSO, can be solved by the *iterative shrinkage-thresholding algorithm* (ISTA) [174] and its variants. This has yielded great interest in L2O schemes, as evidenced by the fact the works [4; 161; 140; 8; 9; 1; 2; 175; 136; 176; 177; 137; 178; 179], among others, provide various ways to unroll the original ISTA algorithm. Additionally, [133; 134; 180; 178] unroll a Nesterov accelerated ISTA known as *FISTA* (Fast ISTA) [181]. Continuing in the vein of sparsity, [182; 13; 183] unroll another algorithm, called *approximate message passing* (AMP), which introduces Onsager correction terms that whitens the noise in intermediate signals while [184; 176; 175] unroll *Orthogonal AMP* [185], an extension to the original AMP. For the ℓ_0-minimization problems, the *iterative hard-thresholding* (IHT) algorithm, which replaces the soft-thresholding function in ISTA with hard-thresholding, is unrolled

in [146; 147]. Switching gears, for problems that involve minimize an objective that is the sum of several terms, the incremental proximal gradient method has also been unrolled [186]. Further generalization of FBS is given by an abstract collection of flexible iterative modularization algorithms (FIMAs) [187] that perform updates using the composition of two learned operators (provided the composition monotonically decreases an energy). A caveat of this particular approach is that the learned operator is also composed with a classic proximal-gradient operation.

The next category of unrolled L2O algorithms consists of primal-dual schemes. These typically include variations of *primal-dual hybrid gradient* (PDHG) and the *alternating direction method of multipliers* (ADMM) (e.g., see [132; 188; 123; 168; 189; 138; 190]). Another variation used is the *Condat-Vu primal-dual hybrid gradient* [191]. As above, many of these methods also focus on leveraging the sparse structure of data.

The remaining group of miscellaneous methods take various forms. These include another first-order algorithm, *Half Quadratic Splitting* (HQS), for image restoration [124; 101]. Using a substitution and soft constraint, the HQS approach solves a problem using a model that approximates variational problems (VPs). Beyond first-order algorithms, unrolling is also applied to *second-order (Newton or Quasi-Newton) methods* [192; 193], *Differential Equations* [156; 194; 195] and *Interior Point* method [10; 164], and *Frank-Wolfe* algorithm [162; 163]. *Conjugate Gradient* was proposed [196] to help yield a more accurate enforcement of the data-consistency constraint at each iteration than comparable proximal-gradient methods. Lastly, [197] propose an unfolded version of a greedy pursuit algorithm, i.e., *orthogonal matching pursuit* (OMP), that directly targets at the original combinatorial sparse selection problem. Their methods called Learned Greedy Method (LGM) can accommodate a dynamic number of unfolded layers, and a stopping mechanism based on representation error, both adapted to the input. Besides, there are also L2O schemes that appear to be based entirely on heuristic combinations of classic optimization methods (e.g., see [198]).

3.2.3 Objective-Based v.s. Inverse Problems

Objective Based. The simplest L2O unrolling scheme is objective based. Training is applied here to yield rapid convergence for a particular distribution of data. The training loss can take various forms, including minimizing the expectation of an objective function [136], the objective's gradient norm, the distance to the optimal solution, or the fixed point residual of the algorithm [24] (e.g., if T is the update operator, then $\|x - T(x)\|$ is the fixed point residual). Examples of objectives that have been extensively studied in the literature are presented in Section. 3.2.1.

In addition, *safeguarding* can be used in this situation, for guiding learned updates to ensure convergence [199; 24]. This can be accomplished in multiple ways. A typical approach is to generate a tentative update using an L2O scheme and then check whether the tentative update satisfies some form of descent inequality (e.g., yields a lesser energy value or fixed point residual). If descent is obtained, then the tentative update is used; otherwise, a classic optimization update is used. These safeguarded schemes provide the benefit of reducing computational costs via L2O machinery while maintaining theoretical convergence guarantees.

Inverse Problems Several L2O methods attempt to solve inverse problems (IPs) (e.g., see [200; 198; 191; 201; 173; 202; 187; 171]). Here the task is to reconstruct a signal $x^\star$ from indirect noisy measurements. The measurements are typically expressed by $d \in \mathbb{R}^m$ and are related to a forward operator $A : \mathbb{R}^n \to \mathbb{R}^m$ by

$$d = A(x^\star) + \varepsilon, \tag{13}$$

where ε is noise. A few repeated themes arise the L2O IP literature. The typical process is to i) set up a variational problem (VP) as a surrogate model, ii) choose a parameterized optimization algorithm that can solve[4] (VP), and iii) perform supervised learning to identify the optimal parameter settings. We expound upon these themes and their nuances below.

First, by creating a variational problem, one assumes $u^\star$ approximately solves

$$\min_{x \in \mathbb{R}^n} \ell\left(d, A(x)\right) + J(x), \tag{VP}$$

where $\ell : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a fidelity term that encourages the estimate u to be consistent with the measurements d and $J : \mathbb{R}^n \to \mathbb{R}$ is a regularizer. Learning comes into play since $u^\star$ is usually *not* the solution to (VP), but instead some "close" variant. Thus, upon choosing the form of ℓ and J, one includes tunable parameters in the model and unrolls a parameterized optimization scheme for (typically) a fixed number K of iterations. Algorithm updates can be parameterized beyond their classic form (e.g., replace a fixed matrix with a matrix that has tunable entries as done in [4]). In most cases, the training loss takes the form of minimizing the expected value of the square of the Euclidean distance between the output estimate u^K and the true signal $u^\star$, i.e., the parameters Θ are trained to solve

$$\min_{\Theta} \mathbb{E}_{d \sim \mathcal{D}}\left[\|x^K(\Theta, d) - x_d^\star\|^2\right]. \tag{14}$$

The primary alternative tranining loss for L2O is to use estimates of the Wasserstein-1 distance between the distribution of reconstructed signals and the distributio of true signals (e.g., see [127]), which yields unsupervised training.

3.2.4 LEARNED PARAMETER ROLES

A key aspect of unrolled L2O methods is to determine how to parameterize each update. This subsection discusses some of the common considerations and roles for these parameters.

Learning parameters in the iterations First, the direct method of parameterization is to convert scalars/vectors/matrix/filters used in iterative algorithms into learnable parameters and learn them through data-driven training process. In this type of parameterization, learning can overcome the need to hand choose hyperparameters. For example, LISTA [4] unrolls ISTA, which usually has update formulation

$$x^{k+1} = \eta_{\lambda/L}\left(x^k - \frac{1}{L}A^T(Ax^k - d)\right), \tag{15}$$

where λ is the coefficient before the ℓ_1 regularization in LASSO, L is the largest eigenvalue of $A^T A$, and $\eta_\theta(\cdot)$ is the coordinate-wise soft-thresholding function parameterized with threshold θ[5]. Then LISTA parameterizes ISTA as a recurrent formulation

$$x^{k+1} = \eta_\theta\left(W_e d + S x^k\right), \tag{16}$$

---

4. We mean to say that, for some choice of parameters, the algorithms solves (VP).
5. The soft-thresholding function takes the formula as $\eta_\theta(z) = \text{sign}(z) \cdot \max(0, |z| - \theta)$

where taking $W_e \equiv A^T/L$, $S \equiv I - A^T A/L$ and $\theta \equiv \lambda/L$ reduces (16) to ISTA. A large amount of unrolling works follows the same methodology, but differ from each other in specific perspectives during the parameterization process, e.g. drop the recurrent structure in [4] and use feed-forward modeling by untying the learnable parameters in different layers and update them independently [150; 182]. This untying formulation can enlarge the capacity of the unrolled model [150] but can also cause the training instability due to the overwhelming parameter space and the difficulty of theoretical analysis on the convergence and other properties of the unrolled model.

To this end, there has been a line of efforts that strive to reduce the number of trainable parameters by relating and coupling different parts of parameterization [147; 1] and analytically generate parameters that satisfy certain constraints [2]. ALISTA [2] learns thresholding and step size parameters, reducing the number of parameters significantly – to two scalars per layer, which stabilizes the data-driven process while achieving state-of-the-art recovery performance at the time with theoretical convergence guarantee.

Besides, recent trends also started to apply model-free L2O methods, to predict hyperparameters in classic iterative optimization algorithms. This is seen as the fusion among model-free L2O (since the algorithmic techniques are LSTM- or RL-based), model-specific L2O (as the update rule is eventually based on the classic iterates), and more traditional hyperparameter optimization. Examples include learning an adaptive learning rate schedule for training deep networks with stochastic graduate descent [36; 203; 204]; coordinating specifically for layer-wise training [205] or domain adaptation speeds [22]; predicting the update combination weights and decay rates in Adam [206]; or estimating training sample importance [204; 207], among others.

Deep priors learned from data Another popular parameterization in unrolling is to use a deep model to learn a data-dependent prior on the interested signals to replace hand-designed priors such as sparsity and total variation, which are not accurate in real-world applications and thus introduce bias. However, previous works have various ways to use the "learned prior". For example in [123; 183; 100; 124], people used data-driven training to learn a proximal or a projection operator that is iteratively used in the unrolled algorithm. The learned operator takes recovered signals contaminated by noises and artifacts as inputs and outputs a refined estimation. Sometimes the learned operator is found to fit a denoiser. The main difference of these learned prior methods from Plug-and-Play methods is that the operator is usually learned in a end-to-end way, making it overfitting to the current task or data distribution and not be able to be plugged into other algorithms. [208; 209] supposed that the relevant vectors lie near the range of a generative model, and hence used generative adversarial networks (GANs) as a learning-based prior for image compressive sensing. [127; 171] perceive the prior as a loss function that can distinguish between coarsely recovered signals without considering any priors, and real on-domain signals with good quality. The prior as a loss function is adversarially trained to output high losses for coarse recoveries and low losses for real signals. Then we use the (sub-)gradient generative by the learned network via back-propagation in the unrolled algorithms e.g. gradient descent.

Others Besides the above two major ways of parameterization, there are also works that learn black-box agents that directly generate next-step estimations given the current iterate and historical information [167; 192; 170]. For instance, [179] learns an LSTM network that generates the step sizes and threshold parameters within each layer in ALISTA, instead of training them using back-propagation.

3.3 Applications

Unrolled L2O schemes have found many applications. Below we identify several of these and note that there is some overlap among the three approaches discussed in this section.

- **Image Restoration and Reconstruction.** The model-based L2O methods, including both Plug-and-Play and unrolling methods, are widely used for various tasks in image restoration, enhancement, and reconstruction. Popular application examples include denoising [165; 170; 124; 155; 210; 2; 189; 127; 211], deblurring [165; 124; 122; 212; 10; 211; 171; 104; 213], super-resolution [123; 135; 170; 124; 155; 212; 104], inpainting [123; 170; 210; 178], and compressive sensing [123; 183; 1; 165; 134; 214; 9; 176; 211], JPEG artifacts reduction [8; 155; 215], demosaicking [122], dehazing [101] and deraining [216]. Note that not all those works identically stick to the algorithm's original architecture; instead many only follow loosely the idea, and replace various components with convolutions or other modern deep learning building blocks.

- **Medical and Biological Imaging.** We specifically separate Medical and Biology Imaging applications from the previous Image Restoration part because the former has its own special scenario and challenges. Imaging techniques in medical such as MRI and CT require accurate reconstructions of images with as few measurements as possible that result in minimal discomfort or side-effect. It is also challenging to extend the model-based methods in natural images to properly deal with the complex-valued inputs [188]. Other work that applies model-based L2O to medical and biology imaging includes [168; 217; 167; 165; 127; 218; 171; 137; 219].

- **Wireless Communication.** Tasks in wireless communication systems can also be solved by unrolling methods, e.g. resource management [220; 175; 221], channel estimation [184], signal detection [184] and LDPC coding [172].For example, MIMO detection, which can be formulated as a sparse recovery problem, was shown to benefit from L2O methods based on "deep unfolding" of an iterative algorithm added with trainable parameters [14], such as LISTA. The model-based L2O approaches have already exhibited superior robustness and stability to low signal-to-noise (SNR), channel correlation, modulation symbol and MIMO configuration mismatches [184]. We refer the readers to a seminal survey about unfolding methods in communication systems [14].

- **Seismic Imaging.** Another important application of model-based L2O is seismic imaging [222; 223; 224; 225]. Most of them adopt a more plug-and-play manner to learn CNN-based projectors that are first trained using data and then integrated into classic iterative updates, due to the desirable emphasis on the physical modeling.

- **Miscellaneous Applications:** such as clustering [161; 226; 227] and classification [146; 23; 138], phase retrieval [228], RNA second structure prediction [19], speech recognition and source separation [150; 229], remote sensing [230], smart grid [231], graph recovery [232], and photometric stereo estimation [147].

3.4 Theoretical Efforts

Although successful empirical results show significant potential of L2O, limited theory exists due to black-box training pertaining to such learned optimizers. That important gap often

makes the broader usability of L2O questionable. We note that, the specific optimization problem and algorithm structure in model-based L2O often offer more opportunities for their theoretical analysis, by bridging us to the wealth of classic optimization tools, compared to model-free L2O. Therefore, model-based L2O in a few specific problems has been the main focus of existing theoretical efforts so far.

To explain why and how model-based L2O methods outperform traditional optimization algorithms, there are several essential questions at the heart of model-based L2O:

- (Capacity). Given the model of L2O, do there exist parameters in the model that makes L2O provably outperform traditional optimization algorithms, over the task distribution (i.e., "in distribution")? Is there any "safeguard" mechanism available on L2O to ensure them at least as good as traditional algorithms even on examples out of the task distribution (i.e., "out of distribution" or OoD)?

- (Trainability). Given the existence of such parameters, what training method should we use to obtain those parameters? Do guarantees exist that the training method converges to the ideal parameters?

- (Generalization). Does the trained model generalize, say, to testing instances from the same source of training instances (i.e., "interpolation")? Can the trained models "extrapolate", e.g., on testing instances more complicated than any training one?

- (Interpretability). How can we explain what the L2O models have learned?

We list brief answers to the four questions here. Capacity and interpretability have been partially solved on signal/image-processing related problems by some recent works, as to be detailed in the subsections below. Generalization gains increasing attention recently and some works provide bounds of generalization gap of some specific L2O models. Lastly, to the best of our knowledge, there has been very limited theoretical work on the trainability of L2O, due to the high nonconvexity of the training objectives (see Section 3.2.3). The main exception here is the recent work [37] discussing the local minima and gradient explosion/vanishing in L2O training for quadratic minimization. This was also partially addressed by [131] where the authors used sorting activation functions [233] and Lipschitz networks to encourage gradient norm preservation.

3.4.1 Capacity

To our best knowledge, [220] is the first effort on theories of L2O. It approximates a traditional method WMMSE (weighted minimum mean square error) with a fully-connected neural network and proves that the output of the neural network can be arbitrarily close to the result of WMMSE as long as its number of layers and units is large enough. In other words, this work estimates the approximation capacity of the neural network.

Approximation capacity is a generic notion in machine learning. Specifically for L2O, **convergence** can be used to describe the model capacity: *Do there exist parameters $\{\theta^k\}_k$ in (11) that make $\{x^k\}$ converge better than classic optimization algorithms as $k \to \infty$?* The work [1] adopts this way to describe the convergence of L2O on the sparse recovery problem and give a convergence rate of LISTA which is better than that of classic algorithms ISTA/FISTA. [2; 234; 178; 136; 235; 236] improve the theoretical result of LISTA by designing delicate models, in another word, designing operator T in (11). [189] analyzes the convergence rate of differentiable linearized ADMM.

Instead of studying parameters θ^k, another approach to establish convergence is to propose mathematical conditions on the operators (for example, operator T in (11) and operator H in (9a)) in the models. These mathematical conditions should not only guarantee the convergence but also can be satisfied practically. [88] proposes a continuity condition that can be satisfied by specifically designed training method. [237] assumes smoothness of the operator that is satisfied by choosing smooth activation functions in a neural network (e.g., the sigmoid function). [171] assumes convexity of the regularizer in their math proof and proposes to parameterize the regularizer by a convex neural network. [131] proves the convergence under assumptions about the manifold and sufficient representation by the sampled data, which are usually satisfied in practice.

While the above efforts focus on studying the convergence and acceleration effect of L2O over the target task distribution, a parallel and same important topic is to bound or characterize the convergence of L2O under OoD inputs: how much can the L2O convergence degrade when applied to optimizees deviating from the task distribution? Seemingly daunting at the first glance, that goal may be fulfilled by L2O with a safeguard mechanism, that can provide a way to establish convergence independent of the parameters and data. In this sense, the capacity of the original L2O models can be considered as enlarged as the convergence is attained on more OoD optimizees. The most common approach is to i) compute a tentative L2O update using any method under the sun, ii) check if the tentative update yields a reduction in the value of an energy, and iii) accept the L2O update if the energy is less than some relative bound (e.g., the energy at the current iterate). If the tentative L2O update is rejected, then a fallback scheme is to apply the update of a classic optimization algorithm. Because the energy is monotonically decreasing, it can be shown that the overall algorithm converges to a minimizer of the energy. The energy is typically defined to be the objective function in the variational problem (VP) or its differential [212; 238; 187; 199]. An alternative approach is define the energy to measure the residual between updates (e.g., [24]). That is, if T is the update operator for a classic algorithm, then at a tentative update u^k we check if the residual $\|u^k - T(u^k)\|$ is less than some relevant bound.

3.4.2 Interpretability

Interpretability is significant to a learning model, now even more than ever. Some efforts have been made on the interpretability of L2O, mainly about linking or reducing their behaviors to those of some analytic, better understood optimization algorithms. [147] studies unrolled iterative hard-thresholding (IHT) and points out that unrolled IHT adopts better dictionary and a wider range of RIP condition than IHT. [133] demonstrates that the mechanism of LISTA is related to a specific matrix factorization of the Gram matrix of the dictionary. [135] explains the success of LISTA with a tradeoff between convergence speed and reconstruction accuracy. [2] shows that the weights in LISTA has low coherence with the dictionary and proposes an analytic approach to calculate the weights. [232] analyzes alternating minimization algorithm quantitatively on a graph recovery problem and reveals the analogy between the learned L2O model and solving a sequence of adaptive convex programs iteratively. [166] points out the analogy between deep-unfolded gradient descent and gradient descent with Chebyshev step-size and shows that the learned step size of deep-unfolded gradient descent can be qualitatively reproduced by Chebyshev step-size.

3.4.3 Generalization

Generalization is an important topic for L2O, just like for any other ML domain. Recent work [239; 74] described a "generalization-first" perspective for L2O. The relationship between generalization gap and the number of training instances provides us an estimate on how many samples we should use. In the recent literature, [160], [240] and [241] studied the Rademacher complexity, an upper bound of generalization gap. [160] estimates the Rademacher complexity of gradient descent and Nesterov's accelerated gradients on a parameterized quadratic optimization problem; [240] estimates the Rademacher complexity of a deep thresholding network on sparse linear representation problem. [241] proposes a reweighted RNN for the signal reconstruction problem and provides its Rademacher complexity. As its definition suggests, generalization measures how the trained L2O models perform on unseen samples from the same distribution seen in training, but not on samples deviating from that distribution. Please refer to the previous discussion in Section. 3.4.1 on how methods such as Safeguard can mitigate the challenges of OoD optimizees.

Stability is highly related with generalization [160], some L2O works analyze the stability of their methods. For example, [201] estimates Lipschitz constants of their method to provide stability analysis results both with respect to input signals and network parameters. This yields a growth condition inequality. Another measure of stability is to ensure stability of the model as the weighting of the regularizer term in (VP) tends to zero [173; 202]. A third approach is that of Φ-regularization used with null space networks [242], which learns a regularizer based on the null space of A. Each of these methods yields different insights about the stability of L2O schemes for inverse problems.

Similar to the bias-variance trade-off in ML, L2O models are also subject to the trade-off between the capacity and the generalization. [160] provided insights on such trade-off in an unrolled model by proving a generalization bound as a function of model depth, and related to the properties of the algorithm unrolled.

4. The Open-L2O Benchmark

The community has made diverse attempts to explore L2O and generated a rich literature for solving different optimization tasks on different data or problems, as well as different software implementations on various platforms. Each L2O method has its own training recipe and hyperparameters. Because a common benchmark for the field has not yet been established, comparisons among different L2O methods are inconsistent and sometimes unfair. In this section, we present our efforts toward creating a comprehensive benchmark that enables fair comparisons. To the best of our knowledge, this is the first time of such an attempt.

Testbed problems: We choose some popular and representative test problems that have been used in the existing L2O literature: (i) convex sparse optimization, including both sparse inverse problems and LASSO minimization; (ii) minimizing the nonconvex Rastrigin function, and (iii) training neural networks (NNs), which is a more challenging nonconvex minimization problem.

Task distributions: Inspired by both research practice and real-world demands, we define task distributions in the following problem-specific way: (i) for sparse optimization, the optimizees during training and testing are optimization problems with the same objective function and decision variables but *different data*; (ii) in the Rastrigin-function test, during both training and testing, the optimizees have different decision variables and use

random initializations; (iii) in the NN training test, the training optimizees use the same network architecture and dataset but random initializations; however, testing samples optimizees from a *different distribution*, that is, the L2O optimizer is applied to train a network of an unseen architecture and on a different dataset.

Compared Methods and Evaluation Settings: For each problem, we choose applicable approaches that include both model-free and/or model-based ones, implement them in the TensorFlow framework, ensure identical training/testing data, and evaluate them in the same but problem-specific metrics. After presenting the results, we draw observations from these benchmark experiments.

Our datasets and software are available as the **Open-L2O** package at: `https://github.com/VITA-Group/Open-L2O`. We hope that Open-L2O will foster reproducible research, fair benchmarking, and coordinated development efforts in the L2O field.

4.1 Test 1: Convex Sparse Optimization

4.1.1 Learning to perform sparse optimization and sparse-signal recovery

Problem definition   The sparse recovery problem has been widely studied in the model-based L2O literature [4; 1; 2]. The task is to recover a sparse vector from its noisy linear measurements:

$$b_q = Ax_q^* + \varepsilon_q, \tag{17}$$

where $x_q^* \in \mathbb{R}^n$ is a sparse vector, $\varepsilon_q \in \mathbb{R}^m$ is an additive noise, and q indexes an optimizee instance. While $x_q^*, b_q, \varepsilon_q$ change for each q, the measurement matrix $A \in \mathbb{R}^{m \times n}$ is fixed across all training and testing. In practice, there is a matrix A associated with each sensor or sensing procedure.

Data generation   We generate $51,200$ samples as the training set and $1,024$ pairs as the validation and testing sets, following the i.i.d. sampling procedure in [1]. We sample sparse vectors x_q^* with components drawn i.i.d. drawn from the distribution $\text{Ber}(0.1) \cdot N(0,1)$, yielding an average sparsity of $\sim 10\%$. We run numerical experiments in four settings:

- **Noiseless**. We take $(m,n) = (256, 512)$ and sample A with $A_{ij} \sim N(0, 1/m)$ and then normalize its columns to have the unit ℓ_2 norm. The noise ε is always zero.

- **Noisy**. The same as above except for Gaussian measurement noises ϵ_q with a signal-to-noise ratio (SNR) of 20dB.

- **Coherent**. A Gaussian random matrix is highly incoherent, making sparse recovery relatively easy. To increase the challenge, we compute a dictionary $D \in \mathbb{R}^{256 \times 512}$ from 400 natural images in the BSD500 dataset [243] using the block proximal gradient method [244] and, then, use it as the measurement matrix A, which has a high coherence. Other settings remain unchanged from the first setting above.

- **Larger scale**. We scale up the noiseless case to $(m,n) = (512, 1024)$; the other settings stay the same.

Model and training settings   All our model-based L2O approaches take measurements b_q as input and return estimates $\hat{x}_q \approx x_q^*$. They are trained to minimize the mean squared error $\mathbb{E}_{q \sim Q} \|\hat{x}_q - x_q^*\|_2^2$. We adopt a progressive training scheme following [13; 1; 234]. We

use a batch size of 128 for training and a learning rate of 5×10^{-4}. Other hyperparameters follow the default suggestions in their original papers. After training, the learned models are evaluated on the test set in NMSE (Normalized Mean Squared Error) in the decibel (dB) unit:

$$\text{NMSE}_{\text{dB}}(\hat{x}_q, x_q^*) = 10 \log_{10} \left(\|\hat{x}_q - x_q^*\|^2 / \|x_q^*\|^2 \right).$$

We compare the following model-based L2O approaches, all of which are unrolled to 16 iterations: (1) a feed-forward version **LISTA** as proposed in [4], with untied weights across layers; (2) **LISTA-CP**, a variant of LISTA with weight coupling [147; 1]; (3) **LISTA-CPSS**, a variant of LISTA with weight coupling and support selection techniques [1; 2]; (4) **ALISTA** [2], the variant of LISTA-CPSS with minimal learnable parameters; (5) **LAMP** [13], an L2O network unrolled from the AMP algorithm; (6) **LFISTA** [245], an L2O network unrolled from the FISTA algorithm; (7) **GLISTA** [234], an improved LISTA model by adding gating mechanisms.

Results   Figure 5 summarizes the comparisons; its four subfigures present the NMSEs of sparse recovery under noiseless, noisy, coherent dictionary, and large scale settings, respectively. We make the following observations :

- In the noiseless setting, ALISTA yields the best final NMSE, and the support selection technique in LISTA-CPSS contributes to good performance. LAMP and GLISTA achieve slightly worse NMSEs than LISTA-CPSS but are much better than LISTA. Surprisingly, LFISTA fails to outperform LISTA, which we attribute to its heavier parameterization in [245] than other models and consequently harder training.

- In the noisy setting of 20dB SNR, the performance of LAMP degrades severely while LISTA-CPSS and ALISTA still perform robustly. GLISTA outperforms all the others in the setting.

- In setting of a coherent dictionary, ALISTA suffers significantly from the coherence while the other methods that learn the weight matrices from data cope this issue better. In particular, GLISTA has the best performance thanks to the gate mechanisms.

- When it comes to a larger scale setting $(m, n) = (512, 1024)$, LISTA, LISTA-CP, LFISTA and GLISTA have performance degradation due to the higher parameterization burden. In comparison, ALISTA becomes the best among all since it has the fewest parameters to learn: only the layer-wise thresholds and step-sizes. Therefore, ALISTA is least impacted by the problem scale and produces nearly the same performance as it does the smaller setting of $(m, n) = (256, 512)$.

4.1.2 Learning to minimize the Lasso model

Problem definition   Instead of sparse recovery, which aims to recover the original sparse vectors, our goal here is to minimize the LASSO objective even though its solution is often different from the true sparse vector:

$$x_q^{\text{Lasso}} = \arg\min_x f_q(x), \quad \text{where } f_q(x) = \frac{1}{2}\|Ax - b_q\|_2^2 + \lambda\|x\|_1, \tag{18}$$

where $A \in \mathbb{R}^{m \times n}$ is a known, fixed, and normalized dictionary matrix, whose elements are sampled i.i.d. from a Gaussian distribution. An optimizee instance with index q is
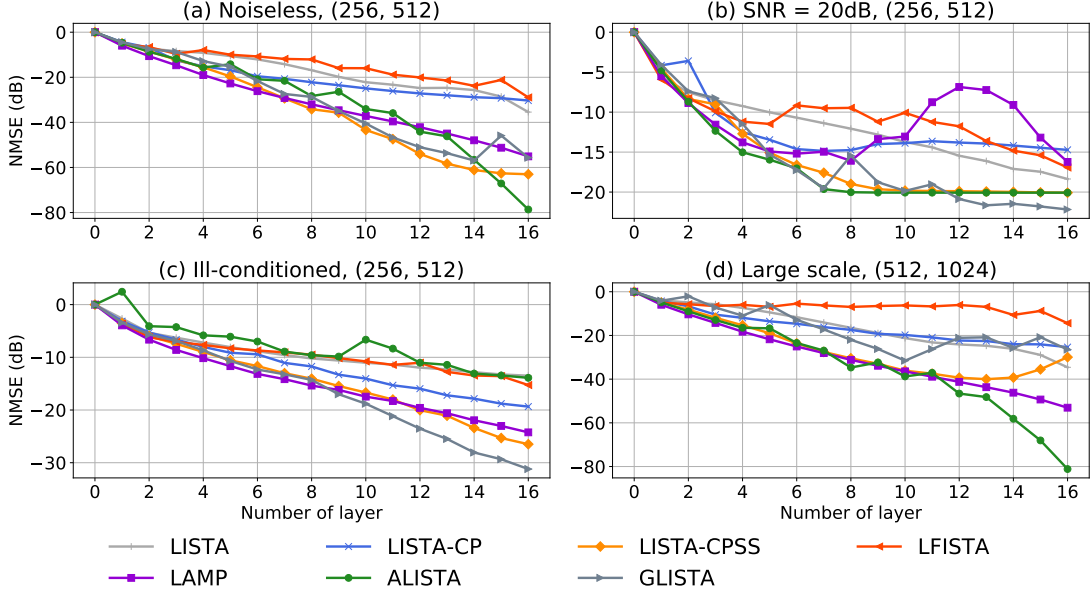
Figure 5: Results of sparse recovery in four different settings: (a) noiseless with $(m, n) = (256, 512)$; (b) additive Gaussian measurement noises of SNR=20dB and $(m, n) = (256, 512)$; (c) coherent dictionary with $(m, n) = (256, 512)$; (d) larger scale with $(m, n) = (512, 1024)$. The x-axis counts the layers and the y-axis is the NMSE of the recovery.

characterized by a sparse vector x_q^* and $b_q \in \mathbb{R}^{m \times 1}$ is the observed measurement under A from x_q^* following the same linear generation model in the previous subsection. λ is a hyperparameter usually selected manually or by cross-validation, and is chosen to be 0.005 by default in all our experiments.

Uniquely, we compare **both model-based and model-free** L2O methods in this section. For the former, we can adopt similar algorithm unrolling recipes as in the previous subsection. For the latter, we treat the problem as generic minimization and apply LSTM-based L2O methods. To our best knowledge, this is the first comparison between the two distinct L2O mainstreams. We hope the results provide quantitative understanding how much we can gain from incorporating problem-specific structures (when available) into L2O.

Data generation We run the experiments with $(m, n) = (5, 10)$, as well as $(m, n) = (25, 50)$. We did not go larger due to the high memory cost of LSTM-based model-free L2O methods. We sample 12,800 pairs of x_q^* and b_q for training and 1,280 pairs for validation and testing. The samples are noiseless. During the testing, we set 1,000 iterations for both model-free L2O methods and classic optimizers. We run model-based L2O ones with a smaller fixed number of iterations (layers), which are standard for them.

For each sample b_q, we let f_q^* denote the optimal Lasso objective value, $f_q(x^{\mathrm{Lasso}})$. The optimization performance is measured with a modified relative loss:

$$R_{f,\mathcal{Q}}(x) = \frac{\mathbb{E}_{q \sim \mathcal{Q}}[f_q(x) - f_q^*]}{\mathbb{E}_{q \sim \mathcal{Q}}[f_q^*]}, \tag{19}$$

where the optimal solution is generated by 2,000 iterations of FISTA and the expectations are taken over the entire testing set.
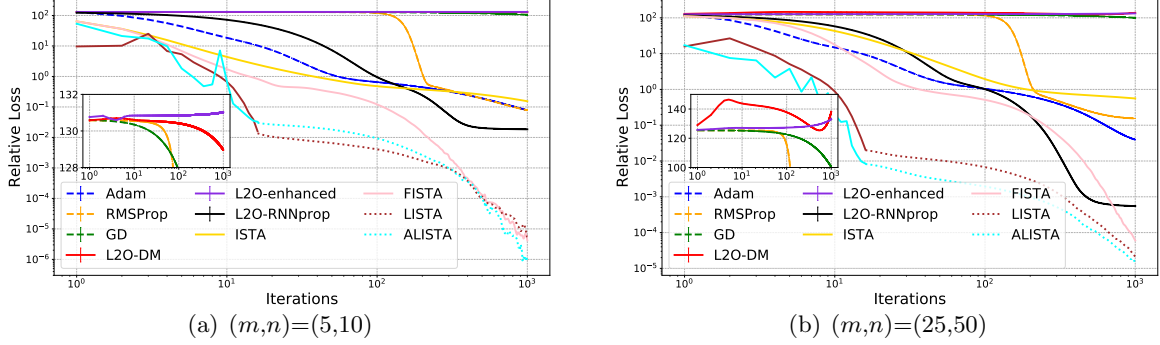
(a) (m,n)=(5,10) (b) (m,n)=(25,50)

Figure 6: Evaluation comparisons among analytic, model-based L2O, and model-free L2O optimizers on Lasso. y-axis represents the modified relative loss (19), and x-axis denotes the number of iterations, both in the logarithmic scale.

We run four categories of methods to solve the LASSO optimization problem:

- Three sub-gradient descent algorithms: GD, ADAM and RMSProp, which serve as "natural baselines" with neither learning nor problem-specific structure. All these algorithms use a step size of 10^{-3}. We verified that changing step sizes did not notably alter their performance. (Even though GD and its generalizations do not handle nonsmooth objectives without smoothing or using proximal operators, they are popular optimizers that people try on almost everything, so we choose to test them anyway.)

- Two problem-specific analytic optimizers: ISTA [174], a forward backward splitting algorithm using a soft-thresholding backward operator and its Nesterov-accelerated version, FISTA [181]. For both methods, we use a step size of $1/L$, where L is the largest eigenvalue of $A^T A$, and a threshold of λ/L for the soft-thresholding function.

- Two model-based L2O models: vanilla LISTA [4] and ALISTA [2], both unrolled to 16 layers. We follow the same training setting in the last subsection except that we replace the training loss from the mean squared error (w.r.t. the true solution) with the LASSO loss. Since other compared methods take far more iterations, we also tried to expand LISTA/ALISTA to more iterations during testing by appending FISTA iterations.

- Three model-free L2O methods: L2O-DM [5], L2O-RNNprop [58] and L2O-enhanced optimizers [73] (Section 2.1). All L2O optimizers are trained with the Lasso objective function as the reward for 100 epochs and 1,000 iterations per epoch. The reported evaluation is the average performance over 10 random starting points.

Results From Figure 6, we make the following observations:

- At the small problem size $(5, 10)$, both ISTA and FISTA converge fast in tens of iterations with FISTA being slightly ahead. Both model-based L2O models, ISTA and ALISTA of 16 iterations (layers), converge to solutions of precision compared to what FISTA can achieve after hundreds of iterations and better than what ISTA

28

can do at 1,000 iterations. The advantage of ISTA/ALISTA can sustain beyond 16 iterations using FISTA updates.

- In comparison, at the small problem size $(5, 10)$, model-free L2O methods exhibit far worse performance. L2O-RNNprop has slower convergence than ISTA/FISTA and only produces reasonably good solutions after 1,000 iterations – though still much better than analytic optimizers Adam and RMSProp. L2O-DM, L2O-enhanced, and vanilla GD completely fail to decrease the objective value.

- At the larger problem size $(25, 50)$, LISTA and ALISTA still converge to high-precision solutions with only 16 iterations with sustained advantages from the FISTA extension. Interestingly, ISTA now becomes much slower than FISTA. All the sub-gradient descent and model-free L2O methods remain to perform poorly; only L2O-RNNprop can converge faster than ISTA and comparable to FISTA, though reaching lower precision.

- Our experiments clearly demonstrate the dominant advantage of incorporating problem-specific structures to the optimizers when it comes to both analytic and learned optimizers.

4.2 Test 2: Minimization of non-convex function Rastrigin

We now turn to non-convex minimization. One popular non-convex test function is called the Rastrigin function:

$$f(\boldsymbol{x}) = \frac{1}{2} \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} \alpha \cos\left(2\pi x_i\right) + \alpha n, \tag{20}$$

where $\alpha = 10$. We consider a broad family of similar functions $f_q(\boldsymbol{x})$ that generalizes Rastrigin function:

$$f_q(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}_q \boldsymbol{x} - \boldsymbol{b}_q\|_2^2 - \alpha \boldsymbol{c}_q \cos(2\pi \boldsymbol{x}) + \alpha n, \tag{21}$$

where $\boldsymbol{A}_q \in \mathbb{R}^{n \times n}$, $\boldsymbol{b}_q \in \mathbb{R}^{n \times 1}$ and $\boldsymbol{c}_q \in \mathbb{R}^{n \times 1}$ are parameters whose elements are sampled i.i.d. from $\mathcal{N}(0, 1)$. Obviously, the function (20) is a special case in this family with $\boldsymbol{A} = \boldsymbol{I}, \boldsymbol{b} = \{0, 0, \ldots, 0\}^{\mathrm{T}}, \boldsymbol{c} = \{1, 1, \ldots, 1\}^{\mathrm{T}}$. For training, we sample $1,280$ triplets of $\{A_q, b_q, c_q\}$ om two problem scales: $n = 2$ and $n = 10$. For evaluation, we sample another 128 combinations of A_q, b_q and c_q and report the average performance over 10 random starting points. The number of steps for evaluation is $1,000$.

Two groups of methods are compared: (1) Four traditional gradient descent algorithms, including ADAM with a 10^{-1} step size, RMSProp with a 3×10^{-1} step size, GD with the line-searched step size started from 10^{-1}, and NAG (Nesterov Accelerated Gradient) with the line-searched step size started from 10^{-1}. All other hyperparamters are tuned by careful grid search. (2) Five model-free L2O methods, including L2O-DM [5], L2O-enhanced [73], L2O-Scale [59], L2O-RNNprop [58] (Section 2.1), and L2O-Swarm [18] (Section 2.3). All L2O optimizers are trained for 100 epochs and 1000 iterations per epoch. At the testing stage, we evaluate and report the logarithmic loss of unseen functions from the same family, which are plotted in Figure 7.
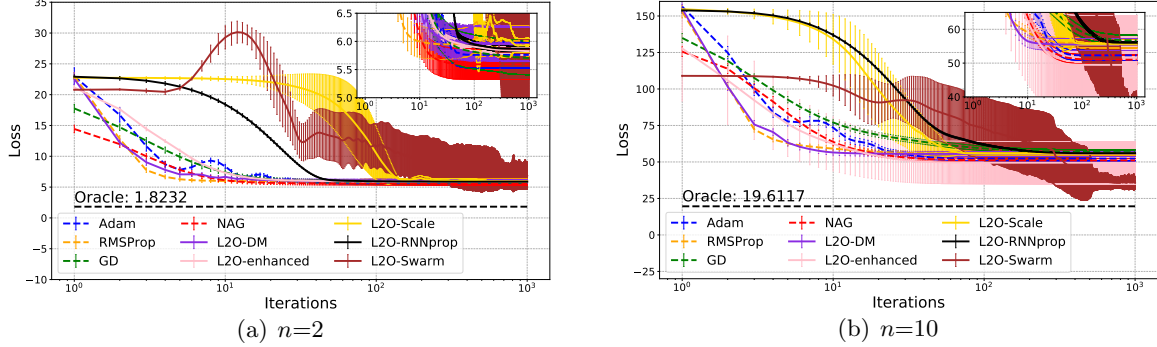
(a) n=2   (b) n=10

Figure 7: Evaluation and comparison across model-free L2O and analytic optimizers on the generalized family of Rastrigin functions. y-axis represents the average loss of function values, and x-axis denotes the number of iterations in the logarithm scale.

Results From Figure 7 we draw the following observations:

- ADAM and RMSProp converge slightly more quickly than GD and NAG with line search, especially in the early stage, for both $n = 2$ and $n = 10$. All analytic optimizers converge to local minima of similar qualities for $n = 2$, and NAG finds a slightly better solution for $n = 10$.

- L2O-DM, L2O-enhanced, L2O-RNNProp, and L2O-Scale perform similarly to analytic optimizers regarding both solution quality and convergence speed, showing no obvious advantage over analytic optimizers. For $n = 10$, L2O-enhanced finds a solution of better quality than the other two. But the three model-free L2Os have larger error bars at $n = 10$, which indicates model instability.

- Although not converging faster than others, L2O-Swarm locates higher quality solutions (lower losses) for both n values, especially $n = 10$. Since L2O-Swarm is the only method that leverages a population of LSTM-based L2O "particles", it explores a larger landscape than the other methods, so it is not surprising that its higher search cost leads to better solutions.

- The oracle objectives (dashed black lines) in Figure 7 are generated by the Run-and-Inspect method [246], which can provably find a global minimum if the objective function can be decomposed into a smooth strongly convex function (e.g., a quadratic function) plus a restricted function (e.g., sinusoidal oscillations). For $n = 2$, almost all methods reach a similar loss lied in $[5.5, 6.0]$ in the end. For $n = 10$, L2O-Swarm performs significantly better than other methods in terms of the achieved final loss, though it converges more slowly than most of the other approaches.

4.3 Test 3: Neural Network Training

Our last test is training multi-layer neural networks (NNs), one of the most common tasks of L2O since its beginning. This has been the playground for model-free L2O methods. There are few problem-specific structures to explore. Common optimizers are stochastic gradient descent and their variants. We hope this test to address two questions:

- Can model-free L2O optimizers outperform analytic ones on neural network training? If so, how much is the margin?

- Can model-free L2O optimizers generalize to unseen network architectures and data?

To fairly compare different methods and answer these questions, we train L2O optimizers on the same neural network used in [5]: a simple Multi-layer Perceptron (MLP) with one 20-dimension hidden layer and the sigmoid activation function, trained on the MNIST dataset to minimize the cross-entropy loss. Therefore, the task distribution becomes optimizing the same MLP model with different random initializations.

We probe the (out of distribution) generalizability of the learned optimizers by testing them on two unseen optimization tasks, following the practice in [5; 73]:

1. Train another MLP with one 20-dimension hidden layer, but using the ReLU activation function, on the MNIST dataset.

2. Train another ConvNet on the MNIST dataset, consisting two convolution layers, two max pooling layers, and one last fully connected layer. The first convolution layer uses 16 3×3 filters with stride 2. The second convolution layer uses 32 5×5 filters with stride 2. The max pooling layers have size 2×2 with stride 2.

We compare four model-free L2O optimizers: L2O-DM [5], L2O-enhanced [73], L2O-Scale [59], and L2O-RNNprop [58] (Section 2.1), all following the hyperparameters suggested in their original papers. SGD, Adam, and RMSProp are analytical optimizers served as the baselines. All L2O optimizers are trained with the single model from 10,000 different random initializations drawn from $\mathcal{N}(0, 01)$. On each optimizee, now corresponding to a random initialization, the optimizers run for 100 iterations. The training uses a batch size of 128. During each run of testing, we evaluate learned optimizers on an unseen testing optimizee for $10,000$ steps, which is much more than the training iteration number. We then report the training loss of the optimizee. We perform 10 independent runs and report the error bars.

It is worth mentioning that current L2O-optimizers can hardly scale to training large neural networks with over 1×10^6 parameters. This is because to update the L2O optimizer using back-propagation, we need to keep the gradients during unrolling and the computation graph of the optimizee in memory, therefore, consuming too much memory. This remains an open challenge.

Results   From Figure 8, we make the following observations:

- The relative performance of analytic optimizers is consistent: both Adam and RMSProp achieve similar training losses on both ReLU-MLP and ConvNets, and better than SGD. However, the learned optimizers exhibit different levels of transferability. L2O-DM completely diverges on ConvNet, but works on ReLU-MLP when the number of iterations is small, showing poor generalization.

- L2O-Scale and L2O-RNNprop can generalize to ReLU-MLP as they achieve similar or lower training losses than analytic optimizer), but they cannot effectively optimize ConvNet. L2O-RNNprop only works for a small number of iterations less than 4,000. L2O-Scale generalizes better to larger iteration numbers, thanks to its random scaling during training.
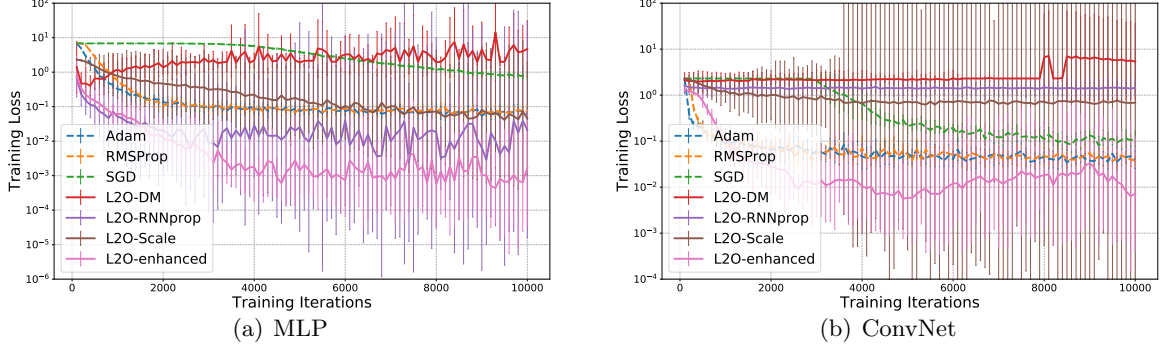
(a) MLP

(b) ConvNet

Figure 8: Evaluation and comparison across model-free L2O and analytic optimizers on neural network training. y-axis represents the average training loss values, and x-axis denotes the number of iterations. Each curve is the average of ten runs, and error bars are plotted.

- L2O-enhanced, which adopts increasing unroll length and off-line imitation strategy, achieves the best performance on both unseen networks after extended (optimizee) training iterations.

- The issue of lacking stability during testing exists for all L2O optimizers. On both ReLU-MLP and ConvNet, all L2O optimizers suffer from larger loss variances as the training iteration grows, causing the errors to fluctuate more severely that those of the analytic optimizers.

4.4 Take-home messages

★ When appropriate model or problem structures are available to be leveraged, model-based L2O methods clearly outperform model-free ones and analytic optimizers. On optimizees sampled from the same task distribution, model-based L2O optimizers have solid, consistent, and reliable performance.

★ Although in many cases we can also observe performance gains of model-free L2O approaches over analytic ones, the gains are not consistent. The benefits of current model-free L2O approaches come to questions when they are applied to larger optimizees or applied with more iterations during testing, let alone to "out-of-distribution" optimizees. We find no clear winner in existing model-free L2O approaches that can consistently outperforms others across most tests.

5. Concluding Remarks

This article provided the first "panoramic" review of the state-of-the-arts in the emerging L2O field, accompanied with the first-of-its-kind benchmark. The article reveals that, despite its promise, the L2O research is still in its infancy, facing many open challenges as well as research opportunities from practice to theory.

On the theory side, Section 3.4 has listed many open theoretical questions for us to understand why and how model-based L2O methods outperform traditional optimizers. Besides those, for model-free L2O methods, the theoretical foundation has been scarce if any

at all. For example, although the training of L2O optimizers is often successful empirically, there has been almost no theoretical result established for the convergence performance of such a L2O training process, putting the general feasibility of obtaining L2O models in question. Also, for both model-based and mode-free L2O approaches, the generalization or adaption guarantees of trained L2O to optimizees out of the task distribution are under-studied yet highly demanded.

On the practical side, the scalability of model-free L2O methods is perhaps the biggest hurdle for them to become more practical. That includes scaling up to both larger and more complicated models, and to more iterations during meta-testing. For model-specific L2O, the current empirical success is still limited to a few special instances in inverse problems and sparse optimization, and relying on case-by-case modeling. An exploration towards broader applications, and perhaps building a more general framework, is demanded.

Furthermore, there is no absolute border between mode-based and model-free L2O methods, and the spectrum between the two extremes can hint many new research opportunities. [50] suggested a good perspective that unrolled networks (as one example of model-based L2O) is an intermediate state between generic networks and analytic algorithms, and might be more data-efficient to learn. The view was supported by [247], which further advocated that the unrolled model might be considered as a robust starting point for subsequently data-driven model search. We also believe that end-to-end learning approaches can be improved with current continuous optimization algorithms to benefit from the theoretical guarantees and state-of-the-art algorithms already available.

So, to conclude this article, let us quote Sir Winston Churchill: *"Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning."* Although most approaches we discussed in this paper are still at an exploratory level of deployment, and are apparently not yet ready as general-purpose or commercial solvers, we are strongly confident that machine learning has just began to feed the classic optimization field, and the blowout of L2O research progress has yet to start.

Appendix A. List of Abbreviations

Abbreviation	Full Name	Description
Adam	Adaptive Moment Estimation	A first-order gradient-based optimization algorithm of stochastic objective functions, based on adaptive estimates of lower-order moments.
ADMM	Alternating Direction Method of Multipliers	An optimization algorithm that solves convex problems by breaking them into smaller pieces, each of which are then easier to handle.
AutoML	Automated Machine Learning	The process of automating the process of applying machine learning to real-world problems.
CASH	Combined Algorithm Selection and Hyperparameter Optimization	A model selection strategy that consider the choosing of data preparation, learning algorithm, and algorithm hyperparameters as one large global optimization problem.
GAN	Generative Adversarial Network	A class of machine learning frameworks [79], where two neural networks contest with each other in a zero-sum game, to learn to generate new data with the same statistics as a given training set.
HPO	Hyperparameter Optimization	The problem of choosing a set of optimal hyperparameters for a machine learning algorithm.
L2O	Learning to Optimize	Learnable optimizers to predict update rules (for optimizees) fit from data.
LASSO	Least Absolute Shrinkage and Selection Operator	A linear regression method that uses shrinkage to encourages simple, sparse models.
LSTM	Long-short Term Memory	A variant of artificial recurrent neural network (RNN) architecture, typically used to process sequence data.
MLP	Multi-layer Perceptron	A class of feedforward artificial neural network (ANN) composed of multiple layers of perceptrons (with threshold activation).
PGD	Projected Gradient Descent	Projected gradient descent minimizes a function subject to a constraint. At each step we move in the direction of the negative gradient, and then "project" onto the feasible set.
ReLU	Rectified Linear Unit	An activation function used in neural networks, defined as $f(x) = \max(x, 0)$.
RL	Reinforcement Learning	RL is an area of machine learning regarding how to make intelligent agents take actions in an environment to maximize the cumulative reward.
RMSProp	Root Mean Square Propagation	RMSprop gradient descent maintains a moving (discounted) average of the square of gradients, and then divide the gradient by the root of this average.

References

[1] Xiaohan Chen, Jialin Liu, Zhangyang Wang, and Wotao Yin. Theoretical linear convergence of unfolded ista and its practical weights and thresholds. In *Advances in Neural Information Processing Systems*, pages 9061–9071, 2018.

[2] Jialin Liu, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. ALISTA: Analytic weights are as good as learned weights in LISTA. In *International Conference on Learning Representations*, 2019.

[3] Yurii Nesterov. Smooth minimization of non-smooth functions. *Mathematical programming*, 103(1):127–152, 2005.

[4] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406, 2010.

[5] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

[6] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*, pages 748–756, 2017.

[7] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30:6348–6358, 2017.

[8] Zhangyang Wang, Ding Liu, Shiyu Chang, Qing Ling, Yingzhen Yang, and Thomas S Huang. D3: Deep dual-domain based fast restoration of jpeg-compressed images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2764–2772, 2016.

[9] Jian Zhang and Bernard Ghanem. Ista-net: Interpretable optimization-inspired deep network for image compressive sensing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1828–1837, 2018.

[10] M.-C. Corbineau, C. Bertocchi, E. Chouzenoux, M. Prato, and J.-C. Pesquet. Learned image deblurring by unfolding a proximal interior point algorithm. In *2019 IEEE International Conference on Image Processing (ICIP)*, page 4664–4668. IEEE, Sep 2019. ISBN 978-1-5386-6249-6. doi: 10.1109/ICIP.2019.8803438.

[11] Dong Liang, Jing Cheng, Ziwen Ke, and Leslie Ying. Deep magnetic resonance image reconstruction: Inverse problems meet neural networks. *IEEE Signal Processing Magazine*, 37(1):141–151, 2020.

[12] Tianwei Yin, Zihui Wu, He Sun, Adrian V Dalca, Yisong Yue, and Katherine L Bouman. End-to-end sequential sampling and reconstruction for mr imaging. *arXiv preprint arXiv:2105.06460*, 2021.

[13] Mark Borgerding, Philip Schniter, and Sundeep Rangan. Amp-inspired deep networks for sparse linear inverse problems. *IEEE Transactions on Signal Processing*, 65(16): 4293–4308, Aug 2017. ISSN 1941-0476. doi: 10.1109/TSP.2017.2708040.

[14] Alexios Balatsoukas-Stimming and Christoph Studer. Deep unfolding for communications systems: A survey and some new directions. In *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 266–271. IEEE, 2019.

[15] Joseph Marino, Alexandre Piché, Alessandro Davide Ialongo, and Yisong Yue. Iterative amortized policy optimization. *arXiv preprint arXiv:2010.10670*, 2020.

[16] Nelson Vadori, Sumitra Ganesh, Prashant Reddy, and Manuela Veloso. Calibration of shared equilibria in general sum partially observable markov games. *Advances in Neural Information Processing Systems*, 33, 2020.

[17] Nelson Vadori, Rahul Savani, Thomas Spooner, and Sumitra Ganesh. Consensus multiplicative weights update: Learning to learn using projector-based game signatures. *arXiv preprint arXiv:2106.02615*, 2021.

[18] Yue Cao, Tianlong Chen, Zhangyang Wang, and Yang Shen. Learning to optimize in swarms. In *Advances in Neural Information Processing Systems*, pages 15018–15028, 2019.

[19] Xinshi Chen, Yu Li, Ramzan Umarov, Xin Gao, and Le Song. Rna secondary structure prediction by learning unrolled algorithms. In *International Conference on Learning Representations*, 2019.

[20] Amritanshu Agrawal, Tim Menzies, Leandro L Minku, Markus Wagner, and Zhe Yu. Better software analytics via "duo": Data mining algorithms using/used-by optimizers. *Empirical Software Engineering*, 25(3):2099–2136, 2020.

[21] Chaojian Li, Tianlong Chen, Haoran You, Zhangyang Wang, and Yingyan Lin. Halo: Hardware-aware learning to optimize. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2020.

[22] Wuyang Chen, Zhiding Yu, Zhangyang Wang, and Anima Anandkumar. Automated synthetic-to-real generalization. *International Conference on Machine Learning (ICML)*, 2020.

[23] Joey Tianyi Zhou, Kai Di, Jiawei Du, Xi Peng, Hao Yang, Sinno Jialin Pan, Ivor W Tsang, Yong Liu, Zheng Qin, and Rick Siow Mong Goh. Sc2net: Sparse lstms for sparse coding. In *AAAI*, pages 4588–4595, 2018.

[24] Howard Heaton, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. Safeguarded learned convex optimization. *arXiv preprint arXiv:2003.01880*, 2020.

[25] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

[26] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

[27] Xiaohan Chen, Siyu Tang, Krikamol Muandet, et al. Mate: Plugging in model awareness to task embedding for meta learning. In *Neural Information Processing Systems 2020*, 2020.

[28] Ke Li and Jitendra Malik. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

[29] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Hu Yi-Qi, Li Yu-Feng, Tu Wei-Wei, Yang Qiang, and Yu Yang. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

[30] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[31] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[32] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, Cham, 2019.

[33] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial Intelligence and Statistics*, pages 528–536. PMLR, 2017.

[34] Yang Yu, Hong Qian, and Yi-Qi Hu. Derivative-free optimization via classification. In *AAAI*, volume 16, pages 2286–2292, 2016.

[35] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

[36] Zhen Xu, Andrew M Dai, Jonas Kemp, and Luke Metz. Learning an adaptive learning rate schedule. *arXiv preprint arXiv:1909.09712*, 2019.

[37] Xiang Wang, Shuai Yuan, Chenwei Wu, and Rong Ge. Guarantees for tuning the step size using a learning-to-learn approach. *arXiv preprint arXiv:2006.16495*, 2020.

[38] Mit eecs 6.890 learning-augmented algorithms. https://www.eecs.mit.edu/academics-admissions/academic-information/subject-updates-spring-2019/6890. Spring 2019.

[39] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, 2018.

[40] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, pages 464–473, 2018.

[41] Tanqiu Jiang, Yi Li, Honghao Lin, Yisong Ruan, and David P Woodruff. Learning-augmented data stream algorithms. In *International Conference on Learning Representations*, 2019.

[42] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

[43] Dylan J Foster, Alexander Rakhlin, and Karthik Sridharan. Online learning: Sufficient statistics and the burkholder method. In *Conference On Learning Theory*, pages 3028–3064, 2018.

[44] Hyeji Kim, Yihan Jiang, Ranvir Rana, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. Communication algorithms via deep learning. In *6th International Conference on Learning Representations, ICLR 2018*, 2018.

[45] Hyeji Kim, Yihan Jiang, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. Deepcode: Feedback codes via deep learning. *IEEE Journal on Selected Areas in Information Theory*, 1(1):194–206, 2020.

[46] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[47] Piotr Indyk, Ali Vakilian, and Yang Yuan. Learning-based low-rank approximations. In *Advances in Neural Information Processing Systems*, pages 7402–7412, 2019.

[48] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[49] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[50] Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *arXiv preprint arXiv:1912.10557*, 2019.

[51] Nir Shlezinger, Jay Whang, Yonina C Eldar, and Alexandros G Dimakis. Model-based deep learning. *arXiv preprint arXiv:2012.08405*, 2020.

[52] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pages 539–548, 2018.

[53] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning Combinatorial Optimization Algorithms over Graphs. *arXiv:1704.01665*, February 2018.

[54] Dimitris Bertsimas and Bartolomeo Stellato. Online mixed-integer optimization in milliseconds. *arXiv preprint arXiv:1907.02206*, 2019.

[55] Dimitris Bertsimas and Bartolomeo Stellato. The voice of optimization. *Machine Learning*, 110(2):249–277, 2021.

[56] Abhishek Cauligi, Preston Culbertson, Bartolomeo Stellato, Dimitris Bertsimas, Mac Schwager, and Marco Pavone. Learning mixed-integer convex optimization strategies for robot planning and control. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1698–1705. IEEE, 2020.

[57] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv:1811.06128*, 2018.

[58] Kaifeng Lv, Shunhua Jiang, and Jian Li. Learning gradient descent: Better generalization and longer horizons. *arXiv preprint arXiv:1703.03633*, 2017.

[59] Olga Wichrowska, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize, 2017.

[60] Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565, 2019.

[61] Ke Li and Jitendra Malik. Learning to optimize. In *International Conference on Learning Representations (ICLR)*, 2017.

[62] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 459–468, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/bello17a.html.

[63] Haoming Jiang, Zhehui Chen, Yuyang Shi, Bo Dai, and Tuo Zhao. Learning to defense by learning to attack, 2018.

[64] Yuanhao Xiong and Cho-Jui Hsieh. Improved adversarial training via learned optimizer, 2020.

[65] Guillaume Leclerc and Aleksander Madry. The two regimes of deep network training. *arXiv preprint arXiv:2002.10376*, 2020.

[66] Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*, 2020.

[67] Jingfeng Wu, Difan Zou, Vladimir Braverman, and Quanquan Gu. Direction matters: On the implicit regularization effect of stochastic gradient descent with moderate learning rate. *arXiv preprint arXiv:2011.02538*, 2020.

[68] Xinshi Chen, Hanjun Dai, Yu Li, Xin Gao, and Le Song. Learning to stop while learning to predict. *arXiv preprint arXiv:2006.05082*, 2020.

[69] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[70] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3381–3387. IEEE, 2008.

[71] Luke Metz, Niru Maheswaranathan, Jonathon Shlens, Jascha Sohl-Dickstein, and Ekin D Cubuk. Using learned optimizers to make models robust to input noise. *arXiv preprint arXiv:1906.03367*, 2019.

[72] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):127–239, 2014.

[73] Tianlong Chen, Weiyi Zhang, Jingyang Zhou, Shiyu Chang, Sijia Liu, Lisa Amini, and Zhangyang Wang. Training stronger baselines for learning to optimize. *arXiv preprint arXiv:2010.09089*, 2020.

[74] Diogo Almeida, Clemens Winter, Jie Tang, and Wojciech Zaremba. A generalizable approach to learning optimizers. *arXiv preprint arXiv:2106.00958*, 2021.

[75] Esteban Real, Chen Liang, David So, and Quoc Le. Automl-zero: evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*, pages 8007–8019. PMLR, 2020.

[76] Jasper Snoek, Kevin Swersky, Rich Zemel, and Ryan Adams. Input warping for bayesian optimization of non-stationary functions. In *International Conference on Machine Learning*, pages 1674–1682, 2014.

[77] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24:2546–2554, 2011.

[78] Iain H Moal and Paul A Bates. Swarmdock and the use of normal modes in protein-protein docking. *International journal of molecular sciences*, 11(10):3623–3648, 2010.

[79] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[80] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[81] Zhenyu Wu, Zhangyang Wang, Zhaowen Wang, and Hailin Jin. Towards privacy-preserving visual recognition via adversarial training: A pilot study. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 606–624, 2018.

[82] Yangjun Ruan, Yuanhao Xiong, Sashank Reddi, Sanjiv Kumar, and Cho-Jui Hsieh. Learning to learn by zeroth-order oracle. *arXiv preprint arXiv:1910.09464*, 2019.

[83] Jiayi Shen, Xiaohan Chen, Howard Heaton, Tianlong Chen, Jialin Liu, Wotao Yin, and Zhangyang Wang. Learning a minimax optimizer: A pilot study. In *International Conference on Learning Representations (ICLR)*, 2021.

[84] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations (ICLR)*, 2016.

[85] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[86] Singanallur V Venkatakrishnan, Charles A Bouman, and Brendt Wohlberg. Plug-and-play priors for model based reconstruction. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 945–948. IEEE, 2013.

[87] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

[88] Ernest Ryu, Jialin Liu, Sicheng Wang, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. Plug-and-play methods provably converge with properly trained denoisers. In *International Conference on Machine Learning (ICML)*, pages 5546–5557, 2019.

[89] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on image processing*, 16(8):2080–2095, 2007.

[90] Michal Aharon, Michael Elad, and Alfred Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322, 2006.

[91] Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 60–65. IEEE, 2005.

[92] Felix Heide, Markus Steinberger, Yun-Ta Tsai, Mushfiqur Rouf, Dawid Pajak, Dikpal Reddy, Orazio Gallo, Jing Liu, Wolfgang Heidrich, Karen Egiazarian, et al. Flexisp: A flexible camera image processing framework. *ACM Transactions on Graphics (TOG)*, 33(6):1–13, 2014.

[93] Christopher A Metzler, Arian Maleki, and Richard G Baraniuk. Bm3d-amp: A new image recovery algorithm based on bm3d denoising. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 3116–3120. IEEE, 2015.

[94] Arie Rond, Raja Giryes, and Michael Elad. Poisson inverse problems by the plug-and-play scheme. *Journal of Visual Communication and Image Representation*, 41:96–108, 2016.

[95] Alon Brifman, Yaniv Romano, and Michael Elad. Turning a denoiser into a super-resolver using plug and play priors. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 1404–1408. IEEE, 2016.

[96] Xiran Wang and Stanley H Chan. Parameter-free plug-and-play admm for image restoration. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1323–1327. IEEE, 2017.

[97] Shunsuke Ono. Primal-dual plug-and-play image restoration. *IEEE Signal Processing Letters*, 24(8):1108–1112, 2017.

[98] Ulugbek S Kamilov, Hassan Mansour, and Brendt Wohlberg. A plug-and-play priors approach for solving nonlinear imaging inverse problems. *IEEE Signal Processing Letters*, 24(12):1872–1876, 2017.

[99] Ji He, Yan Yang, Yongbo Wang, Dong Zeng, Zhaoying Bian, Hao Zhang, Jian Sun, Zongben Xu, and Jianhua Ma. Optimizing a parameterized plug-and-play admm for iterative low-dose ct reconstruction. *IEEE transactions on medical imaging*, 38(2):371–382, 2018.

[100] Harshit Gupta, Kyong Hwan Jin, Ha Q Nguyen, Michael T McCann, and Michael Unser. Cnn-based projected gradient descent for consistent ct image reconstruction. *IEEE transactions on medical imaging*, 37(6):1440–1453, 2018.

[101] Dong Yang and Jian Sun. Proximal dehaze-net: A prior learning-based deep network for single image dehazing. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 702–717, 2018.

[102] Dong Hye Ye, Somesh Srivastava, Jean-Baptiste Thibault, Ken Sauer, and Charles Bouman. Deep residual learning for model-based iterative ct reconstruction using plug-and-play framework. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6668–6672. IEEE, 2018.

[103] Qihui Lyu, Dan Ruan, John Hoffman, Ryan Neph, Michael McNitt-Gray, and Ke Sheng. Iterative reconstruction for low dose ct using plug-and-play alternating direction method of multipliers (admm) framework. In *Medical Imaging 2019: Image Processing*, volume 10949, page 1094906. International Society for Optics and Photonics, 2019.

[104] Kai Zhang, Wangmeng Zuo, and Lei Zhang. Deep plug-and-play super-resolution for arbitrary blur kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1671–1681, 2019.

[105] Xin Yuan, Yang Liu, Jinli Suo, and Qionghai Dai. Plug-and-play algorithms for large-scale snapshot compressive imaging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1447–1457, 2020.

[106] Rizwan Ahmad, Charles A Bouman, Gregery T Buzzard, Stanley Chan, Sizhuo Liu, Edward T Reehorst, and Philip Schniter. Plug-and-play methods for magnetic resonance imaging: Using denoisers for image recovery. *IEEE Signal Processing Magazine*, 37(1):105–116, 2020.

[107] Gary Mataev, Peyman Milanfar, and Michael Elad. Deepred: Deep image prior powered by red. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[108] Guangxiao Song, Yu Sun, Jiaming Liu, Zhijie Wang, and Ulugbek S Kamilov. A new recurrent plug-and-play prior based on the multiple self-similarity network. *IEEE Signal Processing Letters*, 27:451–455, 2020.

[109] Suhas Sreehari, S Venkat Venkatakrishnan, Brendt Wohlberg, Gregery T Buzzard, Lawrence F Drummy, Jeffrey P Simmons, and Charles A Bouman. Plug-and-play priors for bright field electron tomography and sparse interpolation. *IEEE Transactions on Computational Imaging*, 2(4):408–423, 2016.

[110] Stanley H Chan, Xiran Wang, and Omar A Elgendy. Plug-and-play admm for image restoration: Fixed-point convergence and applications. *IEEE Transactions on Computational Imaging*, 3(1):84–98, 2016.

[111] Afonso M Teodoro, José M Bioucas-Dias, and Mário AT Figueiredo. Scene-adapted plug-and-play algorithm with convergence guarantees. In *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2017.

[112] Afonso M Teodoro, José M Bioucas-Dias, and Mário AT Figueiredo. Image restoration and reconstruction using targeted plug-and-play priors. *IEEE Transactions on Computational Imaging*, 5(4):675–686, 2019.

[113] Gregery T Buzzard, Stanley H Chan, Suhas Sreehari, and Charles A Bouman. Plug-and-play unplugged: Optimization-free reconstruction using consensus equilibrium. *SIAM Journal on Imaging Sciences*, 11(3):2001–2020, 2018.

[114] Weisheng Dong, Peiyao Wang, Wotao Yin, Guangming Shi, Fangfang Wu, and Xiaotong Lu. Denoising prior driven deep neural network for image restoration. *IEEE transactions on pattern analysis and machine intelligence*, 41(10):2305–2318, 2018.

[115] Tom Tirer and Raja Giryes. Image restoration by iterative denoising and backward projections. *IEEE Transactions on Image Processing*, 28(3):1220–1234, 2018.

[116] Stanley H Chan. Performance analysis of plug-and-play admm: A graph signal processing perspective. *IEEE Transactions on Computational Imaging*, 5(2):274–286, 2019.

[117] Yu Sun, Brendt Wohlberg, and Ulugbek S Kamilov. An online plug-and-play algorithm for regularized image reconstruction. *IEEE Transactions on Computational Imaging*, 5(3):395–408, 2019.

[118] Ruturaj G Gavaskar and Kunal N Chaudhury. Plug-and-play ista converges with kernel denoisers. *IEEE Signal Processing Letters*, 27:610–614, 2020.

[119] Xiaojian Xu, Yu Sun, Jiaming Liu, Brendt Wohlberg, and Ulugbek S Kamilov. Provable convergence of plug-and-play priors with mmse denoisers. *IEEE Signal Processing Letters*, 27:1280–1284, 2020.

[120] Yu Sun, Zihui Wu, Brendt Wohlberg, and Ulugbek S Kamilov. Scalable plug-and-play admm with convergence guarantees. *arXiv preprint arXiv:2006.03224*, 2020.

[121] Yu Sun, Jiaming Liu, Yiran Sun, Brendt Wohlberg, and Ulugbek S Kamilov. Async-red: A provably convergent asynchronous block parallel stochastic method using deep denoising priors. *arXiv preprint arXiv:2010.01446*, 2020.

[122] Tim Meinhardt, Michael Moller, Caner Hazirbas, and Daniel Cremers. Learning proximal operators: Using denoising networks for regularizing inverse imaging problems. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1781–1790, 2017.

[123] JH Rick Chang, Chun-Liang Li, Barnabas Poczos, BVK Vijaya Kumar, and Aswin C Sankaranarayanan. One network to solve them all–solving linear inverse problems using deep projection models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5888–5897, 2017.

[124] Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. Learning deep cnn denoiser prior for image restoration. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3929–3938, 2017.

[125] Siavash Arjomand Bigdeli, Matthias Zwicker, Paolo Favaro, and Meiguang Jin. Deep mean-shift priors for image restoration. In *Advances in Neural Information Processing Systems*, pages 763–772, 2017.

[126] Siavash Arjomand Bigdeli. and Matthias Zwicker. Image restoration using autoencoding priors. In *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP*, pages 33–44. INSTICC, SciTePress, 2018. ISBN 978-989-758-290-5. doi: 10.5220/0006532100330044.

[127] Sebastian Lunz, Ozan Öktem, and Carola-Bibiane Schönlieb. Adversarial regularizers in inverse problems. In *Advances in Neural Information Processing Systems*, pages 8507–8516, 2018.

[128] Kaixuan Wei, Angelica Aviles-Rivero, Jingwei Liang, Ying Fu, Carola-Bibiane Schnlieb, and Hua Huang. Tuning-free plug-and-play proximal algorithm for inverse imaging problems. *arXiv preprint arXiv:2002.09611*, 2020.

[129] Matthieu Terris, Audrey Repetti, Jean-Christophe Pesquet, and Yves Wiaux. Enhanced convergent pnp algorithms for image restoration. In *IEEE ICIP 2021 Conference Proceedings*. IEEE, 2021.

[130] Regev Cohen, Michael Elad, and Peyman Milanfar. Regularization by denoising via fixed-point projection (red-pro). *arXiv preprint arXiv:2008.00226*, 2020.

[131] Howard Heaton, Samy Wu Fung, Alex Tong Lin, Stanley Osher, and Wotao Yin. Projecting to manifolds via unsupervised learning. *arXiv preprint arXiv:2008.02200*, 2020.

[132] Pablo Sprechmann, Roee Litman, Tal Ben Yakar, Alexander M Bronstein, and Guillermo Sapiro. Supervised sparse analysis and synthesis operators. *Advances in Neural Information Processing Systems*, 26:908–916, 2013.

[133] Thomas Moreau and Joan Bruna. Understanding the learned iterative soft thresholding algorithm with matrix factorization. In *International Conference on Learning Representations (ICLR)*, 2017. URL http://arxiv.org/abs/1706.01338.

[134] Dimitris Perdios, Adrien Besson, Philippe Rossinelli, and Jean-Philippe Thiran. Learning the weight matrix for sparsity averaging in compressive imaging. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3056–3060. IEEE, 2017.

[135] Raja Giryes, Yonina C. Eldar, Alex M. Bronstein, and Guillermo Sapiro. Tradeoffs between convergence speed and reconstruction accuracy in inverse problems. *IEEE Transactions on Signal Processing*, 66(7):1676–1690, Apr 2018. ISSN 1941-0476. doi: 10.1109/TSP.2018.2791945.

[136] Pierre Ablin, Thomas Moreau, Mathurin Massias, and Alexandre Gramfort. Learning step sizes for unfolded sparse coding. In *Advances in Neural Information Processing Systems*, pages 13100–13110, 2019.

[137] Satoshi Hara, Weichih Chen, Takashi Washio, Tetsuichi Wazawa, and Takeharu Nagai. Spod-net: Fast recovery of microscopic images using learned ista. In *Asian Conference on Machine Learning*, page 694–709, Oct 2019.

[138] Benjamin Cowen, Apoorva Nandini Saridena, and Anna Choromanska. Lsalsa: accelerated source separation via learned sparse coding. *Machine Learning*, 108(8-9): 1307–1327, 2019.

[139] Shanshan Wu, Alexandros G. Dimakis, Sujay Sanghavi, Felix X. Yu, Daniel Holtmann-Rice, Dmitry Storcheus, Afshin Rostamizadeh, and Sanjiv Kumar. Learning a compressed sensing measurement matrix via gradient unrolling. *arXiv:1806.10175 [cs, math, stat]*, Jul 2019. URL `http://arxiv.org/abs/1806.10175`. arXiv: 1806.10175.

[140] Pablo Sprechmann, Alex Bronstein, and Guillermo Sapiro. Learning efficient structured sparse models. In *International Conference on Machine Learning (ICML)*, pages 615–622, 2012.

[141] Vardan Papyan, Yaniv Romano, and Michael Elad. Convolutional neural networks analyzed via convolutional sparse coding. *The Journal of Machine Learning Research*, 18(1):2887–2938, 2017.

[142] Aviad Aberdam, Jeremias Sulam, and Michael Elad. Multi-layer sparse coding: The holistic way. *SIAM Journal on Mathematics of Data Science*, 1(1):46–77, 2019.

[143] Jeremias Sulam, Aviad Aberdam, Amir Beck, and Michael Elad. On multi-layer basis pursuit, efficient algorithms and convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 42(8):1968–1980, 2019.

[144] Hamza Cherkaoui, Jeremias Sulam, and Thomas Moreau. Learning to solve tv regularised problems with unrolled algorithms. *Advances in Neural Information Processing Systems*, 33, 2020.

[145] Benoît Malézieux, Thomas Moreau, and Matthieu Kowalski. Dictionary and prior learning with unrolled algorithms for unsupervised inverse problems. *arXiv preprint arXiv:2106.06338*, 2021.

[146] Zhangyang Wang, Qing Ling, and Thomas S Huang. Learning deep $\ell\,0$ encoders. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[147] Bo Xin, Yizhou Wang, Wen Gao, David Wipf, and Baoyuan Wang. Maximal sparsity with deep networks? In *Advances in Neural Information Processing Systems*, pages 4340–4348, 2016.

[148] Zhangyang Wang, Yingzhen Yang, Shiyu Chang, Qing Ling, and Thomas S. Huang. Learning a Deep l_∞ Encoder for Hashing. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 2174–2180. AAAI Press, 2016.

[149] Christoph Studer, Tom Goldstein, Wotao Yin, and Richard G Baraniuk. Democratic representations. *arXiv preprint arXiv:1401.3420*, 2014.

[150] John R. Hershey, Jonathan Le Roux, and Felix Weninger. Deep Unfolding: Model-Based Inspiration of Novel Deep Architectures. *arXiv:1409.2574*, 2014.

[151] P. Sprechmann, A. M. Bronstein, and G. Sapiro. Supervised non-euclidean sparse nmf via bilevel optimization with applications to speech enhancement. In *2014 4th Joint Workshop on Hands-free Speech Communication and Microphone Arrays (HSCMA)*, pages 11–15, 2014.

[152] P. Sprechmann, A. M. Bronstein, and G. Sapiro. Learning Efficient Sparse and Low Rank Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37 (9):1821–1833, September 2015.

[153] Tal Ben Yakar, Roee Litman, Pablo Sprechmann, Alexander M Bronstein, and Guillermo Sapiro. Bilevel sparse models for polyphonic music transcription. In *ISMIR*, pages 65–70, 2013.

[154] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1529–1537, 2015.

[155] Yunjin Chen and Thomas Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1256–1272, Jun 2017. ISSN 0162-8828, 2160-9292. doi: 10.1109/TPAMI.2016.2596743.

[156] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. In *International Conference on Machine Learning (ICML)*, pages 3208–3216, 2018.

[157] Zichao Long, Yiping Lu, and Bin Dong. PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, December 2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2019.108925.

[158] Daniel Greenfeld, Meirav Galun, Ron Kimmel, Irad Yavneh, and Ronen Basri. Learning to Optimize Multigrid PDE Solvers. *arXiv:1902.10248 [cs, math]*, August 2019. URL http://arxiv.org/abs/1902.10248. arXiv: 1902.10248.

[159] Steffen Wiewel, Moritz Becher, and Nils Thuerey. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer Graphics Forum*, volume 38, pages 71–82. Wiley Online Library, 2019.

[160] Xinshi Chen, Yufei Zhang, Christoph Reisinger, and Le Song. Understanding deep architectures with reasoning layer. *arXiv:2006.13401 [cs, stat]*, Jun 2020. URL http://arxiv.org/abs/2006.13401. arXiv: 2006.13401.

[161] Zhangyang Wang, Shiyu Chang, Jiayu Zhou, Meng Wang, and Thomas S Huang. Learning a task-specific deep architecture for clustering. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 369–377. SIAM, 2016.

[162] Dong Liu, Ke Sun, Zhangyang Wang, Runsheng Liu, and Zheng-Jun Zha. Frank-wolfe network: An interpretable deep structure for non-sparse coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[163] Ruben Pauwels, Evaggelia Tsiligianni, and Nikos Deligiannis. Hcgm-net: A deep unfolding network for financial index tracking. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3910–3914. IEEE, 2021.

[164] Carla Bertocchi, Emilie Chouzenoux, Marie-Caroline Corbineau, Jean-Christophe Pesquet, and Marco Prato. Deep unfolding of a proximal interior point method for image restoration. *arXiv:1812.04276 [cs, math]*, Jan 2020. URL `http://arxiv.org/abs/1812.04276`. arXiv: 1812.04276.

[165] Steven Diamond, Vincent Sitzmann, Felix Heide, and Gordon Wetzstein. Unrolled optimization with deep priors. *arXiv:1705.08041 [cs]*, Dec 2018. URL `http://arxiv.org/abs/1705.08041`. arXiv: 1705.08041.

[166] Satoshi Takabe and Tadashi Wadayama. Theoretical interpretation of learned step size in deep-unfolded gradient descent. *arXiv:2001.05142 [cs, math, stat]*, Jan 2020. URL `http://arxiv.org/abs/2001.05142`. arXiv: 2001.05142.

[167] Jonas Adler and Ozan Öktem. Solving ill-posed inverse problems using iterative deep neural networks. *Inverse Problems*, 33(12):124007, 2017.

[168] Jonas Adler and Ozan Öktem. Learned primal-dual reconstruction. *IEEE transactions on medical imaging*, 37(6):1322–1332, 2018.

[169] Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326, 2012.

[170] Patrick Putzky and Max Welling. Recurrent inference machines for solving inverse problems. *arXiv:1706.04008 [cs]*, Jun 2017. URL `http://arxiv.org/abs/1706.04008`. arXiv: 1706.04008.

[171] Subhadip Mukherjee, Sören Dittmer, Zakhar Shumaylov, Sebastian Lunz, Ozan Öktem, and Carola-Bibiane Schönlieb. Learned convex regularizers for inverse problems. *arXiv preprint arXiv:2008.02839*, 2020.

[172] Tadashi Wadayama and Satoshi Takabe. Deep learning-aided trainable projected gradient decoding for ldpc codes. *arXiv:1901.04630 [cs, math]*, Jan 2019. URL `http://arxiv.org/abs/1901.04630`. arXiv: 1901.04630.

[173] Andreas Kofler, Markus Haltmeier, Tobias Schaeffter, Marc Kachelrieß, Marc Dewey, Christian Wald, and Christoph Kolbitsch. Neural networks-based regularization for large-scale medical image reconstruction. *Physics in Medicine & Biology*, 65(13): 135003, 2020.

[174] Thomas Blumensath and Mike E Davies. Iterative thresholding for sparse approximations. *Journal of Fourier analysis and Applications*, 14(5-6):629–654, 2008.

[175] Satoshi Takabe, Tadashi Wadayama, and Yonina C Eldar. Complex trainable ista for linear and nonlinear inverse problems. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5020–5024, 2020.

[176] Daisuke Ito, Satoshi Takabe, and Tadashi Wadayama. Trainable ista for sparse signal recovery. *IEEE Transactions on Signal Processing*, 67(12):3113–3125, Jun 2019. ISSN 1053-587X, 1941-0476. doi: 10.1109/TSP.2019.2912879.

[177] Mengcheng Yao, Jian Dang, Zaichen Zhang, and Liang Wu. Sure-tista: A signal recovery network for compressed sensing. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, page 3832–3836. IEEE, May 2019. ISBN 978-1-4799-8131-1. doi: 10.1109/ICASSP.2019.8683182. URL https://ieeexplore.ieee.org/document/8683182/.

[178] Aviad Aberdam, Alona Golts, and Michael Elad. Ada-lista: Learned solvers adaptive to varying models. *arXiv:2001.08456 [cs, stat]*, Feb 2020. URL http://arxiv.org/abs/2001.08456. arXiv: 2001.08456.

[179] Freya Behrens, Jonathan Sauder, and Peter Jung. Neurally augmented alista. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=q_S44KLQ_Aa.

[180] Bahareh Tolooshams, Sourav Dey, and Demba Ba. Scalable convolutional dictionary learning with constrained recurrent sparse auto-encoders. In *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2018.

[181] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[182] Mark Borgerding and Philip Schniter. Onsager-corrected deep learning for sparse linear inverse problems. In *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, page 227–231, Dec 2016. doi: 10.1109/GlobalSIP.2016.7905837.

[183] Chris Metzler, Ali Mousavi, and Richard Baraniuk. Learned d-amp: Principled neural network based compressive image recovery. In *Advances in Neural Information Processing Systems*, pages 1772–1783, 2017.

[184] Hengtao He, Chao-Kai Wen, Shi Jin, and Geoffrey Ye Li. Model-driven deep learning for mimo detection. *IEEE Transactions on Signal Processing*, 68:1702–1715, 2020.

[185] Junjie Ma and Li Ping. Orthogonal amp. *IEEE Access*, 5:2020–2033, 2017.

[186] Erich Kobler, Teresa Klatzer, Kerstin Hammernik, and Thomas Pock. Variational networks: connecting variational methods and deep learning. In *German conference on pattern recognition*, pages 281–293. Springer, 2017.

[187] Risheng Liu, Shichao Cheng, Yi He, Xin Fan, Zhouchen Lin, and Zhongxuan Luo. On the convergence of learning-based iterative methods for nonconvex inverse problems. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[188] Jian Sun, Huibin Li, Zongben Xu, et al. Deep admm-net for compressive sensing mri. In *Advances in neural information processing systems*, pages 10–18, 2016.

[189] Xingyu Xie, Jianlong Wu, Guangcan Liu, Zhisheng Zhong, and Zhouchen Lin. Differentiable linearized admm. In *International Conference on Machine Learning*, pages 6902–6911, 2019.

[190] Jing Cheng, Haifeng Wang, Leslie Ying, and Dong Liang. Model learning: Primal dual networks for fast mr imaging. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 21–29. Springer, 2019.

[191] Mingyuan Jiu and Nelly Pustelnik. A deep primal-dual proximal network for image restoration. *arXiv preprint arXiv:2007.00959*, 2020.

[192] Xuehan Xiong and Fernando De la Torre. Supervised descent method and its applications to face alignment. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, page 532–539. IEEE, Jun 2013. ISBN 978-0-7695-4989-7. doi: 10.1109/CVPR.2013.75. URL http://ieeexplore.ieee.org/document/6618919/.

[193] Maojia Li, Jialin Liu, and Wotao Yin. Learning to combine quasi-newton methods.

[194] Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399: 108925, Dec 2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2019.108925.

[195] Xiaoshuai Zhang, Yiping Lu, Jiaying Liu, and Bin Dong. Dynamically unfolding recurrent restorer: A moving endpoint control method for image restoration. *arXiv:1805.07709 [cs]*, Oct 2018. URL http://arxiv.org/abs/1805.07709. arXiv: 1805.07709.

[196] Hemant K Aggarwal, Merry P Mani, and Mathews Jacob. Modl: Model-based deep learning architecture for inverse problems. *IEEE transactions on medical imaging*, 38 (2):394–405, 2018.

[197] Rajaei Khatib, Dror Simon, and Michael Elad. Learned greedy method (lgm): A novel neural architecture for sparse coding and beyond. *arXiv preprint arXiv:2010.07069*, 2020.

[198] Yixing Huang, Alexander Preuhs, Michael Manhart, Guenter Lauritsch, and Andreas Maier. Data consistent ct reconstruction from insufficient data with learned prior images. *arXiv preprint arXiv:2005.10034*, 2020.

[199] Michael Moeller, Thomas Mollenhoff, and Daniel Cremers. Controlling neural networks via energy dissipation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3256–3265, 2019.

[200] Kerstin Hammernik, Tobias Würfl, Thomas Pock, and Andreas Maier. A deep learning architecture for limited-angle computed tomography reconstruction. In *Bildverarbeitung für die Medizin 2017*, pages 92–97. Springer, 2017.

[201] Erich Kobler, Alexander Effland, Karl Kunisch, and Thomas Pock. Total deep variation: A stable regularizer for inverse problems. *arXiv preprint arXiv:2006.08789*, 2020.

[202] Housen Li, Johannes Schwab, Stephan Antholzer, and Markus Haltmeier. Nett: Solving inverse problems with deep neural networks. *Inverse Problems*, 2020.

[203] Lucas N Egidio, Anders Hansson, and Bo Wahlberg. Learning the step-size policy for the limited-memory broyden-fletcher-goldfarb-shanno algorithm. *arXiv preprint arXiv:2010.01311*, 2020.

[204] Jun Shu, Yanwen Zhu, Qian Zhao, Deyu Meng, and Zongben Xu. Meta-lr-schedule-net: Learned lr schedules that scale and generalize. *arXiv preprint arXiv:2007.14546*, 2020.

[205] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2127–2135, 2020.

[206] Shipeng Wang, Jian Sun, and Zongben Xu. Hyperadam: A learnable task-adaptive adam for network training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5297–5304, 2019.

[207] Xuxi Chen, Wuyang Chen, Tianlong Chen, Ye Yuan, Chen Gong, Kewei Chen, and Zhangyang Wang. Self-pu: Self boosted and calibrated positive-unlabeled training. *International Conference on Machine Learning (ICML)*, 2020.

[208] Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G Dimakis. Compressed sensing using generative models. In *International Conference on Machine Learning*, pages 537–546, 2017.

[209] Dave Van Veen, Ajil Jalal, Mahdi Soltanolkotabi, Eric Price, Sriram Vishwanath, and Alexandros G Dimakis. Compressed sensing with deep image prior and learned regularization. *arXiv preprint arXiv:1806.06438*, 2018.

[210] Hillel Sreter and Raja Giryes. Learned convolutional sparse coding. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2191–2195. IEEE, 2018.

[211] Morteza Mardani, Qingyun Sun, Vardan Papyan, Shreyas Vasanawala, John Pauly, and David Donoho. Degrees of freedom analysis of unrolled neural networks. *arXiv preprint arXiv:1906.03742*, 2019.

[212] Risheng Liu, Shichao Cheng, Long Ma, Xin Fan, Zhongxuan Luo, et al. A bridging framework for model optimization and deep propagation. *Advances in Neural Information Processing Systems*, 31:4318–4327, 2018.

[213] Yuelong Li, Mohammad Tofighi, Junyi Geng, Vishal Monga, and Yonina C Eldar. Efficient and interpretable deep blind image deblurring via algorithm unrolling. *IEEE Transactions on Computational Imaging*, 6:666–681, 2020.

[214] Morteza Mardani, Qingyun Sun, David Donoho, Vardan Papyan, Hatef Monajemi, Shreyas Vasanawala, and John Pauly. Neural proximal gradient descent for compressive imaging. *Advances in Neural Information Processing Systems*, 31:9573–9583, 2018.

[215] Xueyang Fu, Zheng-Jun Zha, Feng Wu, Xinghao Ding, and John Paisley. Jpeg artifacts reduction via deep convolutional sparse coding. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2501–2510, 2019.

[216] Hong Wang, Qi Xie, Qian Zhao, and Deyu Meng. A model-driven deep neural network for single image rain removal. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3103–3112, 2020.

[217] Jonathan I Tamir, X Yu Stella, and Michael Lustig. Unsupervised deep basis pursuit: Learning reconstruction without ground-truth data. In *Proc. Intl. Soc. Mag. Reson. Med*, volume 27, page 0660, 2019.

[218] Gili Dardikman-Yoffe and Yonina C Eldar. Learned sparcom: Unfolded deep super-resolution microscopy. *arXiv preprint arXiv:2004.09270*, 2020.

[219] Oren Solomon, Regev Cohen, Yi Zhang, Yi Yang, Qiong He, Jianwen Luo, Ruud JG van Sloun, and Yonina C Eldar. Deep unfolded robust pca with application to clutter suppression in ultrasound. *IEEE transactions on medical imaging*, 39(4):1051–1063, 2019.

[220] Haoran Sun, Xiangyi Chen, Qingjiang Shi, Mingyi Hong, Xiao Fu, and Nikos D Sidiropoulos. Learning to optimize: Training deep neural networks for wireless resource management. In *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–6. IEEE, 2017.

[221] Arindam Chowdhury, Gunjan Verma, Chirag Rao, Ananthram Swami, and Santiago Segarra. Unfolding wmmse using graph neural networks for efficient power allocation. *arXiv preprint arXiv:2009.10812*, 2020.

[222] Wenlong Wang, Fangshu Yang, and Jianwei Ma. Velocity model building with a modified fully convolutional network. In *SEG Technical Program Expanded Abstracts 2018*, pages 2086–2090. Society of Exploration Geophysicists, 2018.

[223] Fangshu Yang and Jianwei Ma. Deep-learning inversion: A next-generation seismic velocity model building method. *Geophysics*, 84(4):R583–R599, 2019.

[224] Fangshu Yang, Thanh-an Pham, Harshit Gupta, Michael Unser, and Jianwei Ma. Deep-learning projector for optical diffraction tomography. *Optics Express*, 28(3):3905–3921, 2020.

[225] Hao Zhang, Xiuyan Yang, and Jianwei Ma. Can learning from natural image denoising be used for seismic data interpolation? *Geophysics*, 85(4):WA115–WA136, 2020.

[226] Xi Peng, Shijie Xiao, Jiashi Feng, Wei-Yun Yau, and Zhang Yi. Deep subspace clustering with sparsity prior. In *IJCAI*, pages 1925–1931, 2016.

[227] Xi Peng, Ivor W Tsang, Joey Tianyi Zhou, and Hongyuan Zhu. k-meansnet: When k-means meets differentiable programming. *arXiv preprint arXiv:1808.07292*, 2018.

[228] Çağatay Işıl, Figen S Oktem, and Aykut Koç. Deep iterative reconstruction for phase retrieval. *Applied Optics*, 58(20):5422–5431, 2019.

[229] Zhong-Qiu Wang, Jonathan Le Roux, DeLiang Wang, and John R Hershey. End-to-end speech separation with unfolded iterative phase reconstruction. *arXiv preprint arXiv:1804.10204*, 2018.

[230] Suhas Lohit, Dehong Liu, Hassan Mansour, and Petros T Boufounos. Unrolled projected gradient descent for multi-spectral image fusion. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7725–7729. IEEE, 2019.

[231] Liang Zhang, Gang Wang, and Georgios B Giannakis. Real-time power system state estimation and forecasting via deep unrolled neural networks. *IEEE Transactions on Signal Processing*, 67(15):4069–4077, 2019.

[232] Harsh Shrivastava, Xinshi Chen, Binghong Chen, Guanghui Lan, Srinivas Aluru, Han Liu, and Le Song. Glad: Learning sparse graph recovery. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkxpMTEtPB.

[233] Cem Anil, James Lucas, and Roger Grosse. Sorting out lipschitz function approximation. In *International Conference on Machine Learning*, pages 291–301. PMLR, 2019.

[234] Kailun Wu, Yiwen Guo, Ziang Li, and Changshui Zhang. Sparse coding with gated learned ista. In *International Conference on Learning Representations*, 2019.

[235] Chengzhu Yang, Yuantao Gu, Badong Chen, Hongbing Ma, and Hing Cheung So. Learning proximal operator methods for nonconvex sparse recovery with theoretical guarantee. *IEEE Transactions on Signal Processing*, 2020.

[236] John Zarka, Louis Thiry, Tomás Angles, and Stéphane Mallat. Deep network classification by scattering and homotopy dictionary learning. *arXiv preprint arXiv:1910.03561*, 2019.

[237] Yunmei Chen, Hongcheng Liu, Xiaojing Ye, and Qingchao Zhang. Learnable descent algorithm for nonsmooth nonconvex image reconstruction. *arXiv preprint arXiv:2007.11245*, 2020.

[238] Risheng Liu, Long Ma, Yiyang Wang, and Lei Zhang. Learning converged propagations with deep prior ensemble for image enhancement. *IEEE Transactions on Image Processing*, 28(3):1528–1543, 2018.

[239] Luke Metz, Niru Maheswaranathan, C Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv preprint arXiv:2009.11243*, 2020.

[240] Arash Behboodi, Holger Rauhut, and Ekkehard Schnoor. Generalization bounds for deep thresholding networks. *arXiv preprint arXiv:2010.15658*, 2020.

[241] Huynh Van Luong, Boris Joukovsky, and Nikos Deligiannis. Interpretable deep recurrent neural networks via unfolding reweighted ℓ_1-ℓ_1 minimization: Architecture design and generalization analysis. *arXiv preprint arXiv:2003.08334*, 2020.

[242] Johannes Schwab, Stephan Antholzer, and Markus Haltmeier. Deep null space learning for inverse problems: convergence analysis and rates. *Inverse Problems*, 35(2): 025008, 2019.

[243] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 416–423. IEEE, 2001.

[244] Yangyang Xu and Wotao Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on imaging sciences*, 6(3):1758–1789, 2013.

[245] Thomas Moreau and Joan Bruna. Understanding Trainable Sparse Coding with Matrix Factorization. 2017.

[246] Yifan Chen, Yuejiao Sun, and Wotao Yin. Run-and-inspect method for nonconvex optimization and global optimality bounds for r-local minimizers. *Mathematical Programming*, 176(1):39–67, 2019.

[247] Tianjian Meng, Xiaohan Chen, Yifan Jiang, and Zhangyang Wang. A design space study for lista and beyond. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=GMgHyUPrXa.