SYMBOLIC LEARNING TO OPTIMIZE: TOWARDS INTERPRETABILITY AND SCALABILITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent studies on *Learning to Optimize* (**L2O**) suggest a promising path to automating and accelerating the optimization procedure for complicated tasks. Existing L2O models parameterize optimization rules by neural networks, and learn those *numerical* rules via meta-training. However, they face two common pitfalls: (1) *scalability*: the numerical rules represented by neural networks create extra memory overhead for applying L2O models, and limits their applicability to optimizing larger tasks; (2) *interpretability*: it is unclear what each L2O model has learned in its black-box optimization rule, nor is it straightforward to compare different L2O models in an explainable way. To avoid both pitfalls, this paper proves the concept that we can "kill two birds by one stone", by introducing the powerful tool of *symbolic regression* to L2O. In this paper, we establish a holistic symbolic representation and analysis framework for L2O, which yields a series of insights for learnable optimizers. Leveraging our findings, we further propose a lightweight L2O model that can be meta-trained on large-scale problems and outperformed human-designed and tuned optimizers. Our work is set to supply a brand-new perspective to L2O research, and our codes will be fully released upon acceptance.

1 INTRODUCTION

Efficient and scalable optimization algorithms (a.k.a., *optimizers*) are the cornerstone of almost all computational fields. In many practical applications, we will repeatedly perform a certain type of *optimization tasks* over a specific distribution of data. Each time, the inputs that define the optimization problem are new but similar to previous related tasks. We say such an application has a narrow *task distribution*. Conventional optimizers are theory-driven, so they obtain performance guarantees over the classes of problems specified by the theory. Instead of manually designing and tuning task-specific optimizers, one may be naturally interested to pursue more general-purpose and data-driven optimization frameworks.

Learning to optimize (**L2O**) (Almeida et al., 2021; Andrychowicz et al., 2016; Chen et al., 2017; Li & Malik, 2017a; Chen et al., 2021) is an emerging paradigm that automatically develops an optimizer by learning from its performance on a set of optimization tasks. This data-driven procedure generates optimizers that efficiently solve problems similar to those in the training tasks. Although often lacking a solid theoretical basis, the training process shapes the *learned optimizer* according to problems of interest. When the task distribution is concentrated, the learned optimizer can specifically fit the tasks by discovering "shortcuts" that classic optimizers do not take (Li & Malik, 2017a;b). L2O has demonstrated many practical benefits including faster convergence and/or better solution quality (Chen et al., 2021), even saving clock time (Li et al., 2020b) and hardware energy (Li et al., 2020a).

With promising progress being made, the questions persist: *how far is L2O from becoming principled science? To what extent can we trust L2O as being mature for real applications?* As a fast-growing new field, fascinating open challenges remain concerning L2O, among which two particular barriers arise our attention in this work:

- **Interpretability.** The learned optimizers are often *hard to interpret*. Traditional algorithm design follows analytic principles and admits "white box" symbolic forms, which are clear to explain, analyze, troubleshoot, and prove. In contrast, existing L2O approaches fit update

models using sophisticated data-driven predictors such as neural networks. As a result, it is unclear what rules have been learned in those "black boxes", prohibiting their further explainable analysis and comparison.

- **Scalability.** The learned optimizers can be *hard to scale*. L2O methods use learnable predictors to fit update rules, which incur extra overhead in memory that grows proportionally with the amount of optimization task variables. Moreover, the training of a learned optimizer often involves backpropagating through a full unrolled sequence for the parameter updates in the optimization tasks, and the memory cost grows fast with the unroll length. The scale of that growth results in complexities that current L2O methods cannot handle.

Our proposed solution, centered at addressing the two research gaps, is a brand-new **symbolic representation and interpretation framework for L2O**. While benefiting from large modeling capacity, L2O update rules fit by numerical predictors are inexplainable and memory-heavy. In contrast, the symbolic representations of traditional optimizers enjoy low memory overhead and clear explainability. We aim to integrate the strengths of symbolic forms into L2O. We propose to convert a numerical L2O predictor into a symbolic form that preserves the same problem-specific performance and is still tunable. Those symbolic rules are light, scalable, and interpretable. They can even be made tunable and subject to further meta testing, by a lightweight trainable re-parameterization. Based on them, we will further develop a set of L2O interpretability metrics.

Our main **technical innovations** are summarized below:

- We build the first *symbolic learning to optimize* (**symbolic L2O**) framework that is generally applicable to existing L2O approaches. That include defining the optimizer symbolic space and the fitting approach to convert a numeric L2O to its symbolic counterpart.
- Under the symbolic L2O framework, we establish a unified scheme for L2O interpretability. Our synergistic analysis in the symbolic domain uncovers a number of new insights that are otherwise difficult to observe, including what "shortcuts" L2O methods essentially learn and how they differs across various methods.
- The lightweight symbolic representation also allows L2O to scale better, and we introduce a lightweight trainable re-parameterization to enable further fine-tuning of symbolic rules. For the first time, we obtain a symbolic L2O method that can be meta-trained on ResNet-50 level (23.5 million parameters) optimizee within 30 GB GPU memory, and achieved the state-of-the-art (SOTA) performance when applied to training larger (ResNet-152, 58.2 million parameters) or very different deep models (MobileNet v2 (Sandler et al., 2018)) on various datasets, outperforming existing manually designed and tuned optimizers.

2 Preliminary and Related Work

2.1 Notation and Preliminaries

Without loss of generality, let us consider an optimization problem $\min_{\mathbf{x}} f(\mathbf{x})$ where $\mathbf{x} \in R^d$. A classic optimizer usually iteratively updates $\mathbf{x}$ based on a handcrafted rule. For example, the first-order gradient descent algorithm takes an update at iteration t based on the local gradient 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 much greater room to find flexible update rules. We define $\mathbf{z}_t$ as the optimization historical information available at time t, such as including the current/past iterates $\mathbf{x}_0, \ldots, \mathbf{x}_t$, and/or their gradients $\nabla f(x_0), \ldots, \nabla f(\mathbf{x}_t)$, etc. L2O models an update rule by a *predictor* function g of $\mathbf{z}_t$: $\mathbf{x}_{t+1} = \mathbf{x}_t - g(\mathbf{z}_t, \phi)$, where the mapping of 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. Practically, g is often a neural network (NN). Since NNs are universal approximators (Hornik et al., 1989), L2O has the potential to discover completely new update rules without relying on existing rules. In order to find a desired ϕ associated with a fast optimizer, (Andrychowicz et al., 2016) proposed to minimize the weighted sum of the objective function $f(\mathbf{x}_t)$ over a time span T:

$$\min_{\phi} \mathbb{E}_{f \in \mathcal{T}} \left[\sum_{t=0}^{T-1} w_t f(\mathbf{x}_t) \right], \quad \text{with} \quad \mathbf{x}_{t+1} = \mathbf{x}_t - g(\mathbf{z}_t, \phi),\ t = 0, \ldots, T-1, \tag{1}$$

where $w_0, \ldots, w_{T-1}$ are the weights whose choices depend on empirical settings. T is also called the unrolling length. f represents a sample optimization task from an ensemble $\mathcal{T}$ that represent the target task distribution. Note that ϕ determines the objective value through determining the iterates $\mathbf{x}_t$. L2O solves the problem (1) for a desirable ϕ and correspondingly the update rule $g(\mathbf{z}_t, \phi)$.

A typical L2O workflow is divided into two stages (Chen et al., 2021): a *meta-training* stage that learns the optimizer with a set of similar optimization tasks from the task distribution; and a *meta-testing* stage that applies the learned optimizer to new unseen optimization tasks. The meta-training process often occurs offline and is time consuming. However, the online application of the method at meta-testing is (aimed to be) time saving.

2.2 Related Works

L2O lies at the intersection of ML and optimization research. It overlaps with meta-learning (Vilalta & Drissi, 2002; Hospedales et al., 2021), which contributes a significant portion of L2O development (Andrychowicz et al., 2016; Li & Malik, 2017a). L2O captures two main aspects of meta learning: rapid learning within each task, and transferable learning across many similar tasks. Using data-driven *predictors* to fit update rules, recent L2O approaches have shown success for a wide variety of problems, from convex optimization (Gregor & LeCun, 2010) and nonconvex optimization (Andrychowicz et al., 2016), to black-box optimization (Chen et al., 2017) and combinatorial optimization (Khalil et al., 2017). Application areas benefiting the most from L2O methods include signal processing and communication (Borgerding et al., 2017; Balatsoukas-Stimming & Studer, 2019), image processing (Zhang & Ghanem, 2018; Corbineau et al., 2019), medical imaging (Liang et al., 2020; Yin et al., 2021) and computational biology (Cao et al., 2019; Chen et al., 2019).

The pioneering L2O work (Andrychowicz et al., 2016) leverages a long short-term memory (LSTM) as the coordinate-wise predictor, which is fed with the optimizee gradients and outputs the optimizee parameter updates. (Chen et al., 2017) takes the optimizee's objective value history as the input state of a reinforcement learning agent, which outputs the updates as actions. To enhance L2O generalization, (Lv et al., 2017) proposes random scaling and convex function regularizers tricks. (Wichrowska et al., 2017) introduces a hierarchical RNN to capture the relationship across the optimizee parameters and trains it via meta-learning on the ensemble of small representative problems.

Training LSTM-based L2O models is however notoriously difficult, that is rooted in the LSTM-style unrolling during L2O meta-training (Tallec & Ollivier, 2017; Metz et al., 2018; Pascanu et al., 2013; Parmas et al., 2019). A practical optimizer may need to take thousands or more of iterations. However, since naively unrolling LSTM to this full length is impractical for both memory cost and trainability by gradient propagation. Most LSTM-based L2O methods (Andrychowicz et al., 2016; Chen et al., 2017; Lv et al., 2017; Wichrowska et al., 2017; Metz et al., 2018; Cao et al., 2019) take advantage of truncating the unrolled optimization, yet at the price of the so-called "truncation bias" (Lv et al., 2017) that hampers their learned optimizer's generalization to unseen tasks (Chen et al., 2020).

Most L2O approaches are "black boxes" with neither theoretical backup nor clear explanation on what they learn. A handful of efforts have been made on "demystifying" L2O, mainly about linking or reducing their behaviors to those of some analytic, better understood optimization algorithms. The best-known success was on interpreting the learned sparse coding algorithm (Gregor & LeCun, 2010), as related to a specific matrix factorization of the dictionary's Gram matrix (Moreau & Bruna, 2017), to the projected gradient descent trade-off between convergence speed and reconstruction accuracy (Giryes et al., 2018), or simply reduced to the original iterative algorithm with data-driven hyperparameter estimation (Liu et al., 2019). Other examples include analyzing the learned alternating minimization on graph recovery (Shrivastava et al., 2020), and an analogy between deep-unfolded gradient descent and Chebyshev step-sizes (Takabe & Wadayama, 2020). However, all existing interpretation works are highly restricted to some specific problem or algorithm involved.

3 Technical Approach

3.1 Motivation and Challenges

If we take a unified mathematical perspective, the execution of an optimizer can be represented either through a *symbolic* rule, or by *numeric* computation. Current L2O approaches parameterize their

update rules g by *numerical predictors* that can be learned from data via meta-training. They often choose sophisticated predictive models such as LTSMs (Andrychowicz et al., 2016; Lv et al., 2017; Chen et al., 2020; Wichrowska et al., 2017), that predict the next update based on the current and historical optimization variables.

Despite the blessing of large learning capacity, the NN-based update rules are *uninterpretable* "black boxes". It is never well understood what rules existing L2O models have learned; nor is it easy to compare different L2O models and assess which has discovered more advanced rules. Moreover, numerical predictors, especially RNN-based ones, limit L2O scalability through severe *memory bottlenecks*. Their meta-training involves backpropagating through a full unrolled sequence of length T, for the parameter updates in their optimization task. The resultant memory cost is no less than simultaneously storing T copies of the task model. At meta-testing time, one has to store RNN's own parameters ϕ and hidden states, which are still far from negligible; especially the total size of hidden states often amounts to the task's parameter size (Andrychowicz et al., 2016; Wichrowska et al., 2017). Hence, existing L2O methods are difficult to scale to longer unrolling and larger tasks.

Those important gaps urge us to look back at symbolic update rules. Almost all traditional optimizers can be seen as formula-based symbolic methods within our framework, such as stochastic gradient descent (SGD), Adam, RMSprop, and so on. Their forms hinge on manual discovery and expert crafting. A symbolic optimizer has little memory overhead, and can be easily interpreted or transferred like an equation or a computer program (Bello et al., 2017; Runarsson & Jonsson, 2000; Orchard & Wang, 2016; Bengio et al., 1994). A handful of L2O methods tried to search for good symbolic rules from scratch, using evolutionary or reinforcement learning (Bello et al., 2017; Real et al., 2020; Runarsson & Jonsson, 2000; Orchard & Wang, 2016; Bengio et al., 1994). Unfortunately, those direct search methods quickly become inefficient when the search space of symbols becomes large. The open question remains as : *how to strike the balance between the tractability of finding an update rule, and the effectiveness, memory efficiency and interpretability of the found rule?*

In what follows, we will unify symbolic and numerical optimizers with a synergistic representation and analysis framework. In brief, we stick to numerical predictors at meta-training for its modeling flexibility and ease of optimization. We then convert numerical predictors to symbolic rules, which preserve almost the same effectiveness yet have negligible memory overhead and clear explainability. Facilitated by the new symbolic representations, we will develop a new suite of L2O interpretability metrics, and can meanwhile scale up L2O to much larger optimization tasks.

3.2 A Symbolic Distillation Framework for L2O

We will first demonstrate how to convert the update predictor g: $\mathbf{x}_{t+1} = \mathbf{x}_t - g(\mathbf{z}_t, \phi)$ into *symbolic* forms that preserve the same effectiveness. Contrasting with numerical forms, symbolic rules bear two advantages: (i) being **white-box** functions and enabling **interpretability** (see Section 3.3); and (ii) being much **lighter**, e.g., unlike RNNs that have to carry their own weights and hidden states, which removes one memory bottleneck for **scaling up** to large optimization tasks (see Section 3.4).

To achieve this goal, we leverage a classical tool of symbolic regression (SR) (Cranmer, 2020), a type of regression analysis that searches the space of mathematical expressions to find an equation that best fits a dataset, both in terms of accuracy and simplicity. Different from conventional regression techniques that optimize the parameters for a pre-specified model structure, SR infers both model structures and parameters from data. Popular algorithms rely on genetic programming to evolve from scratch (Runarsson & Jonsson, 2000; Gustafson et al., 2005; Orchard & Wang, 2016). However, SR algorithms are slow on large problems and rely on many heuristics, if evolving from scratch (Runarsson & Jonsson, 2000; Gustafson et al., 2005; Orchard & Wang, 2016). They will become highly inefficient if further being entangled with the rule search, like existing methods (Bello et al., 2017; Runarsson & Jonsson, 2000; Orchard & Wang, 2016).

Inspired by the idea of "knowledge distillation" (Gou et al., 2021), we propose a *symbolic distillation* approach that **decouples** the rule search step (in the numeric domain) and the subsequent representation step (to the symbolic domain). We will first learn a numerical predictor g using existing L2O methods (e.g., RNN-based), and then apply SR to fit a symbolic form that approximates the input-output relationship captured by g. In this way, the former step can stay with the more efficient gradient-based search in a continuously parameterized space.

Specifically, we will generate an off-line database $\mathcal{D}$ of $(input, out)$ pairs, then to train SR to minimize a fitting loss on $\mathcal{D}$. Each pair is obtained from running the learned g on an optimization task: $input$ is a sequence of $\mathbf{z}_t$ generated during optimization within the fixed unrolling length T; and $output$ is the corresponding sequence of the RNN teacher $g(\mathbf{z}_t, \phi)$. The training procedure of SR is to iteratively mutate a population of equation candidates, and filter out the better performing ones to come up with better equations. The final output of SR is a population of best equations composed of variables from $input$ that best fit $output$ of its RNN teacher $g(\mathbf{z}_t, \phi)$ on the target task distribution, using operaters coming from the pre-defined search space.

3.3 Concretizing the Symbolic Distillation Framework

To make symbolic distillation for L2O executable, two open questions have to be defined: **(i) "what to fit"**, i.e., what symbolic operators we use as building blocks to compose the final equation; and **(ii) "fit on what"**, i.e., what information we consider as SR input operands. These questions are open-ended and problem-dependent, and can be rather inconsistent among different L2O methods. To make our approach more generalizable, we choose relatively straightforward options for both.

For the first question, the selection of symbolic operators to use, we adopt the following set: $\{+, -, \times, /, (\cdot)^2, \sqrt{\cdot}, \exp(\cdot), x^y, \tanh, \text{arcsinh}, \sinh, \text{relu}, \text{erfc}\}$. The first eight operators are adopted since they are the basic building blocks of several traditional optimizers such as Adam/Momentum, and the rests are adopted since they are either the nonlinear activation functions used by typical L2O numerical predictors, or convey certain (upate) thresholding effects which often expected and empirically used in existing deep learning optimization (Sun, 2019). For a few operators whose input ranges are constrained to non-negative only, we extend them to be valid for any real-valued input (e.g., $\sqrt{x}$ is extended to $\text{sign}(x) * \sqrt{|x|}$, and x^y is extended to $\text{sign}(x) * |x|^y$, etc).

For the second question, we use the exact input variables of the numerical L2O (e.g., coordinate wise gradients) as a default option throughout this paper. To enable further flexibility for practitioners, we also enable to expand the input operand set, to provide additional optimization-related features and let the SR procedure decide which to use or not. On a high level, we model the obtained symbolic equation as a Finite Impulse Response (FIR) filter, and set a maximum horizon T of the input variables. In the following parts, if without further explanation, T is set to 20 (the latest 20 records of each input variables are visible to symbolic regression module). As another practical level consideration, in order to appropriately capture the non-linear dynamics of certain operators, the magnitude of input variables should better be re-scaled. In this paper, we re-scale all input variables so that they have the variances of 1.

We conduct a proof-of-concept experiment to distill a learned L2O rule using LSTM-based L2O (Andrychowicz et al., 2016). The input sequence is $\mathbf{z}_t = \{\nabla f(\mathbf{x}_t,)\nabla f(\mathbf{x}_{t-1}), ..., \nabla f(\mathbf{x}_{t-T+1})\}$, i.e., the current and past gradients within an unrolling length T = 20. The L2O meta-training was performed on multiple randomly-initialized LeNets on the MNIST dataset. To fit SR, we adopt a generic mathematical operator space defined in (Bello et al., 2017), and a classical genetic programming-based SR approach (Koza, 1994) that iteratively filters and randomly mutated equation candidates. The SR algorithm can score the equation complexity level as a function of the operator number, operator type diversity, and input variable number used (Cranmer et al., 2020). Hence, we can apply a range of complexity levels, and obtain a group of SR fitting rules from simple to complicated. One example of our distilled symbolic rules is displayed as (c_i, i = 0,..., 19, are scalars, omitted for simplicity):

$$\Delta x_t = \underbrace{0.013 e^{-0.835 \text{asinh}(\text{t})}}_{\text{Time-decaying step size}} \left(\text{erfc} \left(\sinh\left(\nabla f(\mathbf{x}_t)\right) + \underbrace{\sum_{i=0}^{19} c_i \tanh\left(\nabla f(\mathbf{x}_{t-i})\right)}_{\text{"momentum"}} \right) - 1 \right) \tag{2}$$

3.4 Interpretability from Symbolic L2O Representations

For general ML models, the interpretability is referred to as *the ability of the model to explain or to present in understandable terms to a human* (Doshi-Velez & Kim, 2017). When it specifically

comes to an L2O model, its interpretability will be uniquely linked to the immense wealth of domain knowledge in optimization, from rigorous guarantees to empirical observations (Nocedal & Wright, 2006). We hence define **L2O interpretability** as *how well a learned optimizer's behavior can align with the optimization domain knowledge, or be understandable to the optimization practitioners*.

In fact, one may easily notice that the symbolic forms of L2O possess certain levels of "baked-in" interpretability. For example, the update rule (2) immediately supplies a few interesting observations: L2O discovers a momentum-like historic weight averaging mechanism, a time-decaying step size, as well as several normalization-like or (nonlinear) gradient clipping operations such as $\mathrm{erfc}, \sinh$ and $\tanh$. All these are human-understandable as they represent well-accepted practices in deep learning optimization (Sun, 2019).

Our target is to establish a more principled and generally applicable L2O interpretability framework. The new framework will be dedicated to capturing L2O characteristics, quantifying their behaviors, and meaningfully comparing different learned optimizers. We propose two novel metrics that dissect L2O along two dimensions: (i) *Temporal Perception Field* (TPF), i.e., how long an input sequence $\mathbf{z}_t$ (past optimization trajectory) the learned optimizer "effectively" takes advantage of; and (ii) *Mapping Complexity* (MC), i.e., how complicated a predictor model g needs to "effectively" be.

The estimations of TPF and MC are both facilitated in the symbolic domain. For TPF, take Eqn. (2) for example again: the predicted update at time t involves a weighted (nonlinear) summation of past gradients $\nabla f(\mathbf{x}_{t-i})), i = 0, \cdots, T$ (set as 20). c_i can be seen as importance indicators for $\nabla f(\mathbf{x}_{t-i})$, and a preliminary idea to define TPF is simply the weighted mean of historical lengths:

$$\text{Temporal Perception Field} = \sum_{i=0}^{T-1} \frac{|c_i|}{\sum_{i=0}^{T-1} |c_i|} \cdot i \tag{3}$$

One challenge will be to estimate TPF when more complicated symbolic forms arise, e.g., the update rule has more heterogeneous compositions such as $a_i \left(\nabla f(\mathbf{x}_{t-i})\right)^{p_i} + b_i \left(\nabla f(\mathbf{x}_{t-i})\right)^{q_i} + c_i \tanh\left(\nabla f(\mathbf{x}_{t-i})\right)$. In such situations, more generic designs of TPF will have to be investigated and validated such as $\sum_{i=0}^{T-1} \frac{|a_i|+|b_i|+|c_i|}{\sum_{i=0}^{T-1} |a_i|+|b_i|+|c_i|} \cdot i$: we leave as future work.

For MC, the SR approach in (Cranmer et al., 2020) has defined a complexity score for any resultant equation, that is calculated based on the total number and types of operators as well as input variables used for fitting it. Based on that tool, we set a threshold on the maximally allowable difference between the original numerical predictor and its symbolically distilled form (over some $(input, out)$ validation set), and choose the minimum-complexity symbolic equation that falls under this threshold: its complexity score is used to indicate the intrinsic mapping complexity.

The above two definitions are in no way unique nor perfect: they are intended as our pilot study effort to quantify the understanding of L2O behaviors. We will extend the above ideas to comprehensively examining the existing L2O approaches. We aim at observing the relationship between the two metrics, and their empirical connections to L2O's convergence speed, stability, as well as transferrability under task distribution shifts. Extensive results will be reported in Section 4, from which we conclude a few hypotheses, including: (i) an L2O with larger TPF will converge faster and more stably on instances from the same task distribution, since it exploits more global optimization trajectory structure for this class of problems; and (ii) an L2O with smaller MC will be more transferable under distribution shifts, if meta-tuning (adaptation) is performed, owing to the predictor model's larger flexibility. We defer more details later in section 4.2.

From the general taxonomy of interpretable deep learning (Li et al., 2021), our approach belongs to the category of using *surrogate models*, i.e., fitting a simple and interpretable model (symbolic equation) from the original complicated one (numerical predictor), and using this surrogate model's explanations to explain the original predictions. Our new interpretability metrics and analyses will be the first-of-its-kind for L2O. The findings will provide troubleshooting tools for existing L2O methods, and insights for designing new L2O methods.

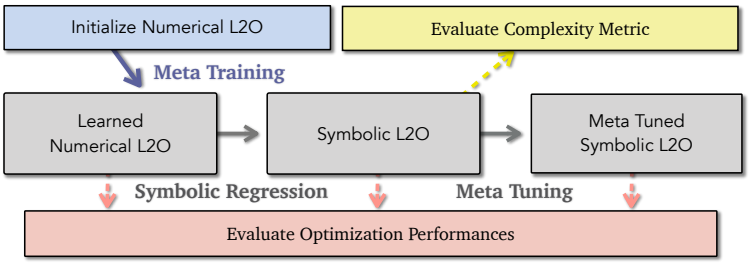

Figure 1: The proposed symbolic regression workflow for L2O models: The **meta training** step enjoys the ease of optimization in the neural network function representation space, and come up with a numerical L2O model; the **symbolic regression** step distill a light weight surrogate symbolic equation from the numerical L2O model; the **meta tuning** step make the distilled symbolic equation again amendable for re-parameterization.

3.5 Scalable L2O Tuning with Symbolic Representations

Symbolic L2O forms can not only be used as "frozen" update rules at testing, but also be tweaked to provide light-weight meta-tuning ability. Taking Eqn. (2) for example again, we can keep the equation form but set all c_i to be trainable coefficients, and continue to fit them on new data, using differentiable meta-training or other hyperparameter optimization (HPO) methods (Feurer & Hutter, 2019). This "tunability" is useful for *adapting* a learned optimizer to a shifted task distribution, such as to *larger optimization tasks* since the optimizer overhead now is much lighter compared to the original numerical predictor. The overall procedure of *training numerical L2O* $\rightarrow$ *distill symbolic rule* $\rightarrow$ *meta tuning on new task* are illustrated in Fig. 1.

4 Interpretability And Scalability Evaluations For Symbolic L2O

In this section, we conduct a series of experiments for the proposed symbolic distillation approach for L2O. We first verify the ability of symbolic regression to recover underlying optimization rules through sanity checks in section 4.1, then leverage the interpretability of symbolic L2O to quantize several properties of the numerical L2O models in section 4.2, finally, we evaluate the scalability of the symbolic L2O in section 4.3.

4.1 Sanity Checks For Symbolic Regression On Optimization Tasks

With a handful of adaptation approaches discussed in section 3.2 (operator selection and modification, injection of inductive bias with processed features, variable magnitude re-scaling, etc.), the symbolic regression are adequately armd to be capable to recover the underlying rule of the optimizers. A sanity check for regressing known optimizers are presented in table 1. In the table, the gradient data are sampled from training a ResNet-50 on Cifar10 dataset, and the most recent 20 timesteps are provided for symbolic regression. The hyperparameters for symbolic regressions are tuned (more details for symbolic regression hyperparameters are put in Appendix A. As can be seen in the results, the symbolic regression almost faithfully reproduce the known equations.

Table 1: Sanity checks for SR's ability to retrieve known equations. "Seconds" corresponds to the computation time accomplished via a 2.6 GHz Intel Core i7 CPU with 16 GB 2400 MHz DDR4 Memory.

Target Equation	Recovered Equation	R^2-score	Seconds
$-0.01 * g_t$	$-0.01 * g_t$	1.0	49
$\sum_{i=0}^{10}(1-0.1i)g_{t-i}$	$g_t + g_{t-1} + 0.81g_{t-2} + 0.72g_{t-3} + 0.56 * g_{t-4} + 0.44g_{t-5} + 0.44 * g_{t-6} + 0.30 * g_{t-7} + 0.20g_{t-8}$	0.94	126
$g_t^2 + g_{t-1} + 2g_{t-2} + e^{g_{t-4}}$	$0.997g_t^2 + 2\tanh(0.47g_{t-1} + g_{t-2} - 0.101g_{t-9}) + 1.0007 + g_{t-4}$	0.98	256
Momentum(0.6)	$g_t + 0.6g_{t-1} + 0.35g_{t-2} + 0.22g_{t-3} + 0.12g_{t-4} + 0.05\text{asinh}(g_{t-5} + g_{t-6})$	0.97	90
Adam(0.9)	$(\sum_{i=0}^{19} a_i g_{t-i} + b_i \text{asinh}(g_{t-i}))/\sqrt{\sum_{i=0}^{19} c_i g_t^2}$ (Though some noisy $\text{asinh}()$ appears, overall R^2-score remains high)	0.89	879

4.2 INTERPRETABILITY OF TRADITIONAL OPTIMIZERS AND LEARNED OPTIMIZERS

In this section, we align the learned optimizer's behavior with the optimization domain knowledge using the two quantization metrics defined in section 3.4, the Temporal Perception Field (TPF) and Mapping Complexity (MC). The results are presented in Table 2 and 3. To diversify the evaluations, in Table 2 and 3, we consider 3 different problems (optimization tasks) and 6 different numerical L2O backbones. The three problems are:

$$\mathcal{P}_1 : \min_{\boldsymbol{x}} : ||\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}||_2^2 + 0.5\boldsymbol{c}^{\mathrm{T}} \cos(\boldsymbol{x}) \qquad \mathcal{P}_2/\mathcal{P}_3 : \min_{\boldsymbol{x}} \text{CrossEntropy}\left(\text{labels}, f(\text{datas}; \boldsymbol{x})\right) \quad (4)$$

The first problem $\mathcal{P}_1$ is a non-convex generalized Rastrigin function series used in Cao et al. (2019), where $\boldsymbol{A} \in \mathbb{R}^{10\times 10}, \boldsymbol{x} \in \mathbb{R}^{10}$, elements in $\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}$ are i.i.d. and sampled from $\mathcal{N}(0, 1)$. In $\mathcal{P}_2/\mathcal{P}_3$, the the datas and labels come from the MNIST dataset, and $\boldsymbol{x}$ is the parameters of an MLP with Relu activations and has hidden layers (50,20) for $\mathcal{P}_2$ and (50,20,20,12) for $\mathcal{P}_3$.

The six numerical L2O backbones are referred to as DM (Andrychowicz et al., 2016), RP (Lv et al., 2017), DM+CL+IL (imitation learning and curriculum learning (Chen et al., 2020) applied to DM (Andrychowicz et al., 2016)), RP+CL+IL (CL and IL (Chen et al., 2020) applied to RP (Lv et al., 2017)), RP_s (RP parameterized with fewer coefficients: its input projection reduced from 20 dimensional to 6 dimensional, and two layer LSTM reduced to one layer with half number of coefficients), and RP_si (enriched the input feature set of RP_s by adding a new set of Adam-type gradient feature).

As observed from Table 3, the CL/IL generally helps to improve both TPF and MC. When aligned from their actual optimization performances, DM + CL + IL is better than DM, which linked to our interpretability that larger TPF will help converge faster. On the other hand, RP models generally have better performance better performance and is faster in adaptation, which links to that L2O models with smaller TPF and larger MC will be more transferable.

Table 2: The comparison of input feature set, mapping function, temporal perception field and mapping complexity across several optimizers.

Optimizers	SGD	$mom(\beta)$	$Adam(\beta_1, \beta_2)$	DM	RP
Input Features Set	g_t	g_t	$\hat{m}_t$	g_t	$\hat{m}_t, \hat{g}_t$
Mapping Function	$\psi(g_t) = -\alpha g_t$	$m_t = \beta m_{t-1} + (1-\beta) g_t$ $\psi(g_t) = -\alpha m_t$ $\psi(g_t) = -\alpha(1-\beta)\sum_{i=0}^{\infty} \beta^i g_{t-i}$	$\psi(\hat{m}_t) = -\alpha \hat{m}_t$ $\hat{m}_t$ defined in Eq. **??**.	$LSTM(g_t)$	$LSTM(\hat{m}_t, \hat{g}_t)$
Temporal Perception Field	0	$\frac{\beta}{1-\beta}$	0	See Table. 3	See Table. 3
Mapping Complexity	1	$\approx \frac{1}{2}\mathrm{i}_0^2$, where $\beta^{\mathrm{i}_0} = 0.05$	1 (when viewing $\hat{m}_t$ as input)	See Table. 3	See Table. 3

Table 3: The Temporal Perception Field and Mapping Complexity of several learned L2O models. The values are averaged across three different problems $\mathcal{P}_1/\mathcal{P}_2/\mathcal{P}_3$. The tuples indicate the corresponding models have more than one type of inputs.

Numerical L2O Models	DM	DM+CL+IL	RP	RP+CL+IL	RP_s	RP_si
Temporal Perception Field	25.2	27.4	(5.3, 2.4)	(5.7, 3.1)	(5.5, 2.2)	(4.6, 2.0, 4.1)
Mapping Complexity	182.9	220.6	104.7	121.4	80.3	67.0

4.3 SCALABILITY EVALUATIONS

In this section, we evaluate the scalability of symbolic L2O on large scale optimizees. Due to the memory burden of LSTM based L2O models, meta-training on large scale problems are considered as challenging tasks for LSTM based L2O model (Chen et al., 2020; Bello et al., 2017), as most L2O model work with small scale optimizee that is difficult to upscale to ResNet-50-level problems (Chen et al., 2020; Bello et al., 2017; Andrychowicz et al., 2016; Lv et al., 2017). However, the simple form and the hidden-states-free property of symbolic L2O enable it to be fast in inferencing and has little memory overhead for training, which brings better potential to scale up to these environments.

Optimizers	**momentum=0.5**			**momentum=0.9**		
	lr=0.001	lr=0.01	lr=0.1	lr=0.001	lr=0.01	lr=0.1
SGD (no cosine lr decay)	94.53	93.70	94.10	95.20	94.82	95.31
SGD (with cosine lr decay)	93.25	94.37	94.87	92.12	93.57	94.03
Symbolic L2O	**95.40**					

Table 4: Comparison of the meta-tuned symbolic rule and the traditional optimizers. Evaluation results are shown for optimizing a ResNet152 on Cifar10.

In the following, we show through experiments that meta-tuning a symbolic L2O on large scale neural network optimizees is not only feasible, but also leads to better performance than hand-designed and tuned optimizers.

For our evaluations, we picked three large scale optimizees: ResNet-50, ResNet-152 and MobileNetV2, and studied two different datasets on these optimizees: the Cifar-100 and the Cifar-10. For the selection of the teacher L2O model, we select the best performing one, the RP_si model. We first meta-pre-train this RP_si teacher model on $\mathcal{P}_1$ to get an initialization. We then distill it into the symbolic equation, and meta-fine-tune the distilled symbolic equation on the **ResNet-50 on Cifar-10**, which is a large enough task so as to forbid a dozon of previous L2O models to train on normal size single GPU. Thanks to the light-weight property, the meta-training process for our symbolic L2O only takes less than 30GB GPU memory over ResNet-50. Also, thanks to the initialization from pre-trained RP_si, it only takes one pass for the 200-epochs' meta training then the symbolic L2O is ready for evaluations.

The evaluation results are plotted in Fig.2 and Table 4. In the evaluations, we include the following hand designed optimizers as baselines for comparison: SGD optimizer with learning rate selected from (0.1, 0.01, 0.001), momentum selected from (0.5, 0.9, 0.99), with and without the nestrov momentum, with and without cosine learning rate annealing, and the Adam optimizer with learning rate selected from (0.01, 0.001). In the legend of Fig. 2, default values are: learning rate = 0.1, momentum = 0.9, without using nestrov momentum, without using cosine learning rate annealing. The hyperparameters different from the default values are plotted in the figure. The LSTM based requires significantly more memory to apply to large scale problems, hence they are not included in this section.

As can be observed in the figure, our proposed symbolic L2O model outperformed most human-engineered optimizers not only in convergence rate, but also in final accuracies.

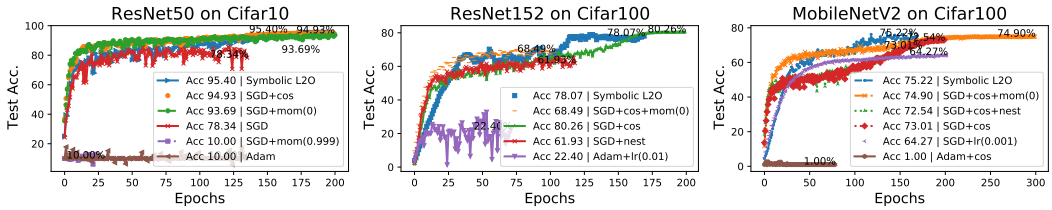

Figure 2: The performance comparison of the meta-fine-tuned symbolic L2O, and several traditional optimizers as baselines. The Symbolic L2O achieved the best performance even with the laboriously tuned traditional optimizers.

5 Conclusion

In this work, we proposed a principled learning and analysis framework for symbolic L2O. Our symbolic framework enabled better interpretability and scalability of L2O models, by first numerically learn a neural network based L2O model that enables a free room of optimization, then distill it into symbolic equation that links the immense wealth of optimization domain knowledge. This work revealed multiple insights that we believe are important for the L2O society, paving the way to make the L2O both more interpretable and boost the SOTA performance and scalability of L2O.

6 REPRODUCIBILITY CHECKLIST

To ensure reproducibility, we use the Machine Learning Reproducibility Checklist v2.0, Apr. 7 2020 (Pineau et al., 2021). An earlier version of this checklist (v1.2) was used for NeurIPS 2019 (Pineau et al., 2021).

- For all **models** and **algorithms** presented,
  - **A clear description of the mathematical settings, algorithm, and/or model.** We clearly describe all of the settings, formulations, and algorithms in Section 4.2.
  - **A clear explanation of any assumptions.** Our assumptions are justified in 4.2.
  - **An analysis of the complexity (time, space, sample size) of any algorithm.** We do not make the analysis.
- For any **theoretical claim**,
  - **A clear statement of the claim.** We do not make theoretical claims.
  - **A complete proof of the claim.** We do not make theoretical claims.
- For all **datasets** used, check if you include:
  - **The relevant statistics, such as number of examples.** We use widely adopted public datasets in Section 4.3.
  - **The details of train/validation/test splits.** We use the default splits as provided by the datasets.
  - **An explanation of any data that were excluded, and all pre-processing step.** We did not exclude any data or perform any pre-processing.
  - **A link to a downloadable version of the dataset or simulation environment.** We do not use specific environment that require additional labor to install.
  - **For new data collected,a complete description of the data collection process, such as instructions to annotators and methods for quality control.** We do not collect or release new datasets.
- For all shared **code** related to this work, check if you include:
  - **Specification of dependencies.** We give instructions in the supplemented codes.
  - **Training code.** The training code is available in the supplementary.
  - **Evaluation code.** The evaluation code is available in the supplementary.
  - **(Pre-)trained model(s).** We do not release any pre-trained models. The code to run all experiments in our work can be found in the supplementary.
  - **README file includes table of results accompanied by precise command to run to produce those results.** We include a README in the code.
- For all reported **experimental results**, check if you include:
  - **The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results.** We provide all details of the hyper-parameter tuning in Section 4.2.
  - **The exact number of training and evaluation runs.** Ten independent repetitions are conducted for each experiments.
  - **A clear definition of the specific measure or statistics used to report results.** We use the classification accuracy on test-set, which is defined in Section 4.
  - **A description of results with central tendency (e.g. mean) & variation (e.g. error bars).** We report accuracy and loss values for all experiments.
  - **The average runtime for each result, or estimated energy cost.** We do not report the running time or energy cost.
  - **A description of the computing infrastructure used.** All detailed descriptions are presented in the supplementary materials.

REFERENCES

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

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*, pp. 3981–3989, 2016.

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)*, pp. 266–271. IEEE, 2019.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. *arXiv preprint arXiv:1709.07417*, 2017.

Samy Bengio, Yoshua Bengio, and Jocelyn Cloutier. Use of genetic programming for the search of a new learning rule for neural networks. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pp. 324–327. IEEE, 1994.

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.

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

Tianlong Chen, Weiyi Zhang, Zhou Jingyang, Shiyu Chang, Sijia Liu, Lisa Amini, and Zhangyang Wang. Training stronger baselines for learning to optimize. *Advances in Neural Information Processing Systems*, 33, 2020.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828*, 2021.

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.

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*, pp. 748–756. PMLR, 2017.

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)*, pp. 4664–4668. IEEE, Sep 2019. ISBN 978-1-5386-6249-6. doi: 10.1109/ICIP.2019.8803438.

Miles Cranmer. Pysr: Fast & parallelized symbolic regression in python/julia, September 2020. URL `https://doi.org/10.5281/zenodo.4052869`.

Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *NeurIPS 2020*, 2020.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

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

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.

Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

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

Steven Gustafson, Edmund K Burke, and Natalio Krasnogor. On improving genetic programming for symbolic regression. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pp. 912–919. IEEE, 2005.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Timothy M Hospedales, Antreas Antoniou, Paul Micaelli, and Amos J Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

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.

John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

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 2020a.

Ke Li and Jitendra Malik. Learning to optimize. *International Conference on Learning Representations (ICLR)*, 2017a.

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

Maojia Li, Jialin Liu, and Wotao Yin. Learning to combine quasi-newton methods. *NeruIPS workshop on on Optimization for Machine Learning*, 2020b.

Xuhong Li, Haoyi Xiong, Xingjian Li, Xuanyu Wu, Xiao Zhang, Ji Liu, Jiang Bian, and Dejing Dou. Interpretable deep learning: Interpretation, interpretability, trustworthiness, and beyond. *arXiv preprint arXiv:2103.10689*, 2021.

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.

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.

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

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, C Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. *arXiv preprint arXiv:1810.10180*, 2018.

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`.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

Jeff Orchard and Lin Wang. The evolution of a generalized neural learning rule. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 4688–4694. IEEE, 2016.

Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. Pipps: Flexible model-based policy search robust to the curse of chaos. *arXiv preprint arXiv:1902.01240*, 2019.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318, 2013.

Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Lariviere, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research. *Journal of Machine Learning Research*, 22:1–20, 2021.

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

Thomas Philip Runarsson and Magnus Thor Jonsson. Evolution and design of distributed learning rules. In *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. Proceedings of the First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (Cat. No. 00*, pp. 59–63. IEEE, 2000.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

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.

Ruoyu Sun. Optimization for deep learning: theory and algorithms. *arXiv preprint arXiv:1912.08957*, 2019.

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.

Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017.

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

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International Conference on Machine Learning*, pp. 3751–3760. PMLR, 2017.

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.

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*, pp. 1828–1837, 2018.

A SYMBOLIC REGRESSION HYPERPARAMETERS

There are several hyperparameters for the symbolic regression procedure that could potentially influence the performance. We list them as follows: