Differentiable Convex Optimization for Meta-Learning

Tanmay Gautam, Samuel Pfrommer, Somayeh Sojoudi

Abstract— **Conventional optimization methods in machine learning and controls rely heavily on first-order update rules. Selecting the right method and hyperparameters for a particular task often involves trial-and-error or practitioner intuition, motivating the field of meta-learning. We generalize a broad family of preexisting update rules by proposing a meta-learning framework in which the inner loop optimization step involves solving a differentiable convex optimization (DCO). We illustrate the theoretical appeal of this approach by showing that it enables one-step optimization of a family of linear least squares problems, given that the meta-learner has sufficient exposure to similar tasks. Various instantiations of the DCO update rule are compared to conventional optimizers on a range of illustrative experimental settings.**

I. Introduction

First-order optimization methods underpin a wide range of modern control and machine learning (ML) techniques. The field of deep learning, including domains such as computer vision [1], [2], natural language processing [3], deep reinforcement learning [4], and robotics [5], has yielded revolutionary results when trained with variants of gradient descent such as stochastic gradient descent (SGD) [6] and Adam [7]. Algorithms like projected and conditional gradient descent extend the class of first-order methods to accommodate problems with constraints such as matrix completion, or training well-posed implicit deep models [8], [9].

While this proliferation of methods has facilitated rapid advances across the control and ML communities, designing update rules tailored to specific problems still remains a challenge. This challenge is exacerbated by the fact that different domains are tasked with solving distinct problem types. The deep learning community is, for instance, tasked with solving high-dimensional non-convex problems whereas the optimal control community often deals with constrained convex problems where the constraints encode restrictions on the state space and system dynamics. Moreover, even different problem instances within a particular problem class may require significantly varying update rules. As an example, within deep learning, effective hyperparameter (e.g. learning rate) selection for algorithms such as Adam and SGD is highly dependent on the underlying model that is to be trained.

A. Contributions

This work proposes a new data-driven approach for optimization algorithm design based on differentiable convex optimization (DCO). This approach enables the use of previous optimization experience to propose update rules that efficiently solve new optimization tasks sampled from the same underlying problem class. We start by introducing the notion of DCO as a means to parameterize optimizers within the meta-learning framework. We then propose an efficient instantiation of meta-training that can be leveraged by the DCO optimizer to learn appropriate meta-parameters. To illustrate the generality of the DCO meta-learning framework, we then formulate concrete differentiable quadratic optimizations to solve unconstrained optimization problems, namely, DCO Gradient (DCOG), DCO Momentum (DCOM) and DCO General Descent (DCOGD). These DCO instantiations are generalizations of existing first-order update rules, which in turn demonstrates that existing methods can be thought of special cases of the DCO meta-learning framework.

DCO also provides sufficient structure conducive to rigorous theoretical analysis for the meta-learning problem. We establish convergence guarantees for the DCOGD optimizer to the optimal first-order update rule that leads to one step convergence when considering a family of linear least squares (LS) problems. Finally, we illustrate the potential of our proposed DCO optimizer instantiations by comparing convergence speed with popular existing first-order methods on proof-of-concept regression and system identification tasks.

B. Related Works

1) Meta-Learning: Deep learning has been shown to be particularly performant in scenarios where there is an abundance of training data and computational resources [1], [10], [3], [11]. This, however, excludes many important applications where there is an inherent lack of data or where computation is very expensive. Meta-learning attempts to address this issue by gaining learning process experience on similarly structured tasks [12]. This learning-to-learn paradigm is aligned with the human and animal learning process which tends to improve with greater experience. Moreover, by making the learning process more efficient meta-learning targets the aforementioned issues of data and compute scarcity.

Meta-learning methods can be categorized into three broad classes. In [13], authors provide a unifying formulation that encapsulates a wide class of existing meta-learning approaches.

Optimizer-focused methods aim to improve the underlying optimizer in the inner loop used to solve the tasks at hand by meta-learning optimizer initialization or hyperparameters. Within few-shot learning, Model Agnostic Meta Learning (MAML) and its variants use prior learning experience to meta-learn a model/policy initialization that requires just a

All authors are with the Department of Electrical Engineering and Computer Sciences, University of California Berkeley, Berkeley, CA, 94720. tgautam23@berkeley.edu; sam.pfrommer@berkeley.edu*; sojoudi@berkeley.edu

few inner gradient steps to adapt to a new task [14], [15]. Other works aim to meta-learn optimizer hyperparameters. In [16], [17], authors attempt to identify optimal learning rate scheduling strategies. Another strategy within this category is to directly learn a parameterization of the optimizer. Due to the sequential structure of inner loop parameter updates, recurrent architectures have been considered in this space [18], [19]. The inner loop optimization has also been viewed as a sequential decision-making problem and consequently optimizers have also been characterized as policies within an RL setting [20].

Black-box methods represent the inner loop via a forward-pass of a single model. The learning process of the inner loop is captured by the activation layers of the underlying model. The inner loop learning can be instantiated as RNNs [21], [22], convolutional neural networks (CNN) [23] or hyper-networks [24]. The meta-learning loop finds the hyperparameters of the inner loop network yielding good performance.

In non-parametric methods, the inner loop aims to identify feature extractors that enable the matching of validation and training samples to yield an accurate prediction using the matched training label. The meta-loop aims to identify the class of feature extractors that transform the data samples into an appropriate space where matching is viable [25], [26].

2) Implicit Layers: Recent work has proposed a novel viewpoint wherein deep learning can be instantiated using implicit prediction rules rather than as conventional explicit feedforward architectures [8], [27], [28]. In [8] and [28] authors formalize how deep equilibrium models, characterized by nonlinear fixed point equations, represent weight-tied, infinite-depth networks. In this framework, [8] demonstrates how the aforementioned models are able to generalize most of the popular deep learning architectures. In [27], authors propose neural ordinary differential equations (ODE): an alternative instantiation of an implicit layer where the layer output is the solution to an ODE. This is shown to be an expressive model class yielding particularly impressive results when processing sequential data. Implicit layers have also been characterized as differentiable optimization layers. The work [29] introduces differentiable quadratic optimization (QP) layers that can be incorporated within deep learning architectures. In [30] authors develop software to differentiate through defined convex optimization problems. Some notable applications of differentiable optimization layers include parameterizing model predictive control policies [31] and representing a maximum satisfiability (MAXSAT) solver [32].

C. Notations

Throughout this work, we consider the problem of solving a task $\mathcal{T}$ which consists of an optimization problem and an evaluation step. The optimization problems are characterized with a loss function $l(\theta)$ over decision variables θ belonging to some parameter space $\Theta \subseteq \mathbb{R}^p$. For evaluation, we denote a validation criterion l^{val} that assesses the optimizer $\theta^\star$ found in the associated problem. We refer to solving the optimization over θ as the *inner loop* problem. At the meta-level we consider an algorithm $\mathbf{Opt}(\;\cdot\;;\phi) : \Theta \to \Theta$ with meta-parameters ϕ which generates a sequence of parameter updates using first-order information to solve the inner loop problem. We denote the horizon of the parameter update sequence by T. By meta-training, we refer to the optimization over the meta-parameters over a training set of N tasks $\{\mathcal{T}_i\}_{i=1}^N$. For a vector-valued function $f(x) : \mathbb{R}^d \to \mathbb{R}^p$ we let the operator $\nabla_x f(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d\times p}$ denote the gradient. If $f : \mathbb{R}^d \to \mathbb{R}$ is a scalar function, the Hessian of f is denoted by $\nabla_x^2 f(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d\times d}$. We denote that a square symmetric matrix A is positive definite (all eigenvalues strictly positive) by $A \succ 0$. The vectorization of a matrix $A \in \mathbb{R}^{m\times n}$ is denoted by $\text{vec}(A) \in \mathbb{R}^{mn}$ and is constructed by stacking the columns of A. The Kronecker product of two matrices A and B is denoted $A \otimes B$. For a vector $x \in \mathbb{R}^n$ and $p \geq 1$, $\|x\|_p$ denotes the ℓ_p-norm of x. For $m \in \mathbb{N}_+$, we define $[m]$ to be the set $\{a \in \mathbb{N}_+ \mid a \leq m\}$, where $\mathbb{N}_+$ is the set of positive integer numbers. We define the operator $\odot$ as an elementwise multiplication. $\mathcal{U}(a, b)$ denotes the uniform probability distribution with support $[a, b]$ and $\mathcal{N}(\mu, \sigma^2)$ represents a univariate normal distribution centered at μ with standard deviation σ. Finally, we define $E_{\mathcal{D}}[\cdot]$ as the expectation operator over distribution $\mathcal{D}$.

II. Background

This section contextualizes our proposed framework. Section II-A illustrates how conventional first-order update rules can be typically expressed as the solution to a convex optimization problem. Section II-B then elaborates on the differentiable convex optimization methods that can be used to differentiate through the aforementioned inner loop gradient steps to update meta-parameters.

A. First-order methods

We consider a generic unconstrained optimization problem

$$\min_x f(x) \tag{1}$$

with differentiable objective f. First-order methods are a popular means to solve optimization problems of the form (1). The first-order property refers to the underlying methods' use of gradient information to generate a sequence of parameter iterates. Next we briefly survey a subset of important first-order methods that solve optimization problems of the form (1). We highlight how the update rules of these algorithms can be formulated as convex optimization problems themselves. This motivates the formulation of a generic parameterized convex optimizer to yield optimal parameter updates.

1) Gradient descent: Gradient descent (GD) is a standard first-order method used to solve a variety of unconstrained optimization problems. For an unconstrained optimization problem, GD updates aim to reconcile the notion of minimizing a linear approximation of the objective while simultaneously maintaining proximity to the current parameter iterate. This can be cast as the convex optimization

$$x^{(t+1)} = \arg\min_x \{\nabla f(x^{(t)})^\top (x - x^{(t)}) + \frac{\lambda}{2}||x - x^{(t)}||_2^2\} \tag{2}$$

where $\lambda > 0$ is the step size. Solving (2) in closed-form yields the well-known GD update.

2) Gradient descent with Momentum: A popular practical variation of GD is to utilize the history first-order information within the parameter update rule. This is referred to as GD with momentum. The contribution of historic first-order information is captured by the notion of a state. More concretely, we define state update for $t > 1$ as

$$S^{(t+1)} = \beta S^{(t)} + (1-\beta)\nabla f(x^{(t)}), \tag{3}$$

where $\beta \in [0,1]$ is an averaging parameter and we initialize $S^{(1)} := \nabla f(x^{(1)})$. The convex update rule in this method substitutes $\nabla f(x^{(t)})$ with $S^{(t+1)}$:

$$x^{(t+1)} = \arg\min_x \{(S^{(t+1)})^\top (x - x^{(t)}) + \frac{\lambda}{2}||x - x^{(t)}||_2^2\} \tag{4}$$

Other notable first-order methods whose updates are defined via convex optimization problems are the proximal gradient (PG) [33], [34] and mirror descent (MD) [35], [36] methods. The former addresses unconstrained nondifferentiable problems whose objective is a composite function that can be decomposed into the sum of a differentiable and non-differentiable part. The latter targets potentially constrained problems with updates that simultaneously minimize a linear approximation of the objective and a proximity term between parameter updates.

B. Differentiable Optimization Layers

We now present the formulation for a general DCO [30]:

$$\begin{aligned} D(x;\phi) := \arg\min_{y\in\mathbb{R}^n} \quad & f_0(x,y;\phi) \\ \text{s.t.} \quad & f_i(x,y;\phi) \le 0 \quad \text{for } i \in [q], \\ & g_j(x,y;\phi) = 0 \quad \text{for } j \in [r], \end{aligned} \tag{5}$$

where $x \in \mathbb{R}^d$ is the optimization input and $y \in \mathbb{R}^n$ is the solution. Here optimization parameters are defined by a vector ϕ. The functions f_i parameterize inequality constraint functions which are convex in y and g_j parameterize affine equality constraints. As with the constraint functions, the objective f_0 is convex in the optimization variable y.

Note that this formulation defines a general parameterized convex optimization problem in the output y. The solution to the optimization is a function of the input x.

When embedding DCO as a layer within the deep learning context, we require the ability to differentiate through D with respect to ϕ when performing backpropagation. This is achieved via implicit differentiation through the Karush-Kuhn-Tucker (KKT) optimality conditions as proposed in [29], [37]. Particular instantiations of DCO, such as parameterized QPs, can enable more efficient backpropagation of gradients [29].

III. Meta-Optimization Framework

Consider the setting where we have N *training* tasks $\mathcal{T}_i = (l_i, l_i^{\text{val}})$ for $i \in [N]$, where each task consists of a tuple containing a training loss function l_i and an associated performance metric l_i^{val}. Each of these tasks is sampled from an underlying distribution $\mathcal{D}$, i.e $\mathcal{T}_i \sim \mathcal{D}\ \forall i \in [N]$. For task $\mathcal{T}_i$, we consider the optimization

$$\min_{\theta_i \in \Theta} l_i(\theta_i) \tag{6}$$

where we aim to minimize loss l_i over the decision variable θ constrained to the set $\Theta \subset \mathbb{R}^p$. We let ϕ denote the set of meta-parameters that configure the method used to solve optimization (6), e.g. ϕ could include the learning rate in a gradient-based algorithm. The validation loss l_i^{val} is used to evaluate the final $\theta_i^\star$ recovered from solving (6). As motivation for this setup, we consider the general training-validation procedure seen in ML. Here l_i can be seen as the loss on training data with respect to model parameters θ and l_i^{val} denotes the loss on validation data. Note that for problems where the metric of interest is in fact the objective of (6), we can trivially define $l_i^{\text{val}} := l_i$.

In this meta-learning framework, the goal is to perform well on a new task $\mathcal{T}_{\text{target}} = (l_{\text{target}}, l_{\text{target}}^{\text{val}}) \sim \mathcal{D}$ using previous experience from tasks $\{\mathcal{T}_i\}_{i=1}^N$. Since $\mathcal{T}_{\text{target}}$ is sampled from the same distribution $\mathcal{D}$ as the training tasks, it has structural similarities that can be exploited by meta-learning.

A. Inner Optimization Loop

Depending on the structure of (6), several iterative methods exist to solve the considered problem. The chosen algorithm has an update rule that yields a sequence of parameter updates $\{\theta_i^{(t)}\}_{t=1}^T$ where T is defines the total number of updates and i indexes the associated task $\mathcal{T}_i$. Within the class of first-order methods, these update rules require computing or estimating (e.g. within reinforcement learning) the gradient $G_i^{(t)} := \nabla_\theta l_i(\theta_i^{(t)})$ to solve the inner optimization of task $\mathcal{T}_i$. The algorithm **Opt** applies the computed first-order and zeroth-order information at time step t along with an abstraction of past information encapsulated by state $S^{(t)}$ to yield both an updated parameter $\theta_i^{(t+1)}$ and state $S^{(t+1)}$:

$$(\theta_i^{(t+1)}, S_i^{(t+1)}) := \mathbf{Opt}(\theta_i^{(t)}, S_i^{(t)}, G_i^{(t)}; \phi). \tag{7}$$

Here we characterize the optimizer with meta-parameters ϕ. Solving the inner loop problem to completion involves recursively applying (7) T times from an initial condition $\theta_i^{(1)}$ and history state $S_i^{(1)}$, which we denote by $\mathbf{Opt}_T(\theta_i^{(1)}, S_i^{(1)}; \phi)$. Note that moving forward, unless made explicit, we suppress the return argument of the next state, i.e. we utilize the shorthand $\theta_i^{(t+1)} := \mathbf{Opt}(\theta_i^{(t)}, S_i^{(t)}, G_i^{(t)}; \phi)$.

B. Meta-Learning Loop

The meta-learning loop wraps around the inner loop. It aims to find optimal meta-parameters $\phi^\star$ that ensure that for each task $\mathcal{T}_i$ in distribution $\mathcal{D}$, the inner loop optimizer **Opt** produces $\theta_i^\star$ that performs well on metric $l_i^{\text{val}}(\theta_i^\star)$:

$$\min_\phi E_{\mathcal{T}_i\sim\mathcal{D}}[l_i^{\text{val}}(\theta_i^\star(\phi))] \tag{8}$$

where $E_{\mathcal{T}_i \sim \mathcal{D}}$ denotes the expectation over task distribution $\mathcal{D}$. An empirical version of this meta-learning process with training tasks $\{\mathcal{T}_i\}_{i=1}^N$ can be formulated as

$$\begin{aligned}
\phi^\star &= \arg\min_\phi \frac{1}{N}\sum_{i=1}^N l_i^{\text{val}}(\theta_i^\star) \\
&= \arg\min_\phi \frac{1}{N}\sum_{i=1}^N l_i^{\text{val}}(\arg\min_{\theta\in\Theta} l_i(\theta)) \\
&\approx \arg\min_\phi \frac{1}{N}\sum_{i=1}^N l_i^{\text{val}}(\mathbf{Opt}_T(\theta_i^{(1)}, S_i^{(1)};\phi)), \qquad (9) \\
&:= \arg\min_\phi l^{\text{total}}(\phi) \qquad (10)
\end{aligned}$$

where the inner optimization is approximated by running algorithm **Opt** for T time steps. Optimization (9) can be approximated by another iterative gradient-based scheme that estimates $\nabla_\phi l_i^{\text{val}}(\theta_i^\star)$. This requires differentiation through the inner loop update rule **Opt** with respect to meta-parameters ϕ. More specifically, we require differentiation with respect to ϕ through a trajectory of parameter updates with horizon T. The meta-parameters will then be updated using a meta-optimizer of choice that uses first-order information on the meta-parameters:

$$\phi^{(t+1)} := \mathbf{MetaOpt}\left(\phi^{(t)}, \nabla_\phi l^{\text{total}}\left(\phi^{(t)}\right)\right). \qquad (11)$$

Remark 1. Note that an approximated attempt at meta-learning is ubiquitous in practice. More specifically, the notion of hyperparameter selection (e.g. learning rate) for a first-order method is an instance of approximated meta-learning. In this context, we let hyperparameters be viewed as meta-parameters. Given a task, the goal in hyperparameter selection is to identify these such that the algorithm **Opt** generates $\theta_i^\star$ with low $l_i^{\text{val}}(\theta_i^\star)$. In practice, the selection of hyperparameters (i.e. **MetaOpt**) is restricted to crude rules of thumb or grid search guided by previous experience of similar problems. It is clear how such approximations can often fall short especially when considering high-dimensional or even continuous meta-parameter search spaces. Moreover, it does not accommodate parameterizing **Opt** to describe novel update rules. The meta-learning framework in (9) generalizes the hyperparameter selection problem and makes it more rigorous.

C. Meta-Training

The meta-training algorithm for an arbitrary optimizer **Opt** with meta-parameters ϕ is presented in Algorithm 1. For each meta-parameter update, average validation losses across training tasks $\{\mathcal{T}_i\}_{i=1}^N$ are accumulated in l^{total}. For each task $\mathcal{T}_i$, these validation losses are measured after running the inner loop optimization using $\mathbf{Opt}(\cdot;\phi)$ for T iterations. $\mathbf{MetaOpt}(\cdot,\cdot)$ then uses first-order information on l^{total} with respect to ϕ to update the meta-parameters.

Remark 2. For a specific task $\mathcal{T}_i$, the role of the meta-optimizer can be viewed as trying to learn the loss landscape of the inner problem locally around $\theta_i^{(t)}$ for $t \in [T]$ and adapt the optimizer accordingly to encourage efficient descent. Thus, the updates within the inner loop help the meta-optimizer get a better gauge of the loss landscape. In turn, T should be selected based on how complicated or spurious the inner problem's loss landscape is. For more complicated inner problems, more information (i.e. updates) are necessary to gauge the loss landscape. For simpler problems, a smaller horizon should suffice.

Algorithm 1 Meta-Training Framework

```
Input: Training set consisting of N tasks $\{\mathcal{T}_i\}_{i=1}^N$
Design choices: Inner loop horizon $T$, meta-training epochs $M$, optimizer $\mathbf{Opt}(\cdot;\phi)$, meta-optimizer $\mathbf{MetaOpt}(\cdot,\cdot)$
Return: Meta-parameters $\phi$

begin training
1. Initialize meta-parameters $\phi^{(1)}$
2. Initialize inner loop parameters and initial optimizer states $\{\theta_i^{(1)}, S_i^{(1)} : i \in [N]\}$
for $k \in [M]$ do
    3. Initialize $l^{\text{total}} \leftarrow 0$
    for $i \in [N]$ do
        for $t \in [T]$ do
            4. Compute inner loop gradient $G_i^{(t)} \leftarrow \nabla_\theta l_i(\theta_i^{(t)})$
            5. $(\theta_i^{(t+1)}, S_i^{(t+1)}) \leftarrow \mathbf{Opt}(\theta_i^{(t)}, S_i^{(t)}, G_i^{(t)};\phi)$
        end for
        6. $l^{\text{total}} \leftarrow l^{\text{total}} + l_i^{\text{val}}(\theta_i^{(T+1)})/N$
    end for
    7. $\phi^{(k+1)} \leftarrow \mathbf{MetaOpt}(\phi^{(k)}, \nabla_\phi l^{\text{total}})$
end for
end training
```

Remark 3. Algorithm 1 allows for flexibility when choosing **MetaOpt**. Stochastic first-order methods can be employed to solve the meta-training problem. That is, rather than using the entire batch of N tasks $\{\mathcal{T}_i\}_{i=1}^N$, a random minibatch can be selected to perform a meta-parameter update. This strategy becomes particularly useful in settings where N is prohibitively large. Furthermore, adding stochasticity in the **MetaOpt** procedure may reap some known benefits of SGD such as not succumbing to local minima.

IV. Differentiable Convex Optimizers

We now propose various instantiations of the the inner loop optimization step (7) as differentiable convex optimizations. More generally, our proposed DCO meta-learning framework parameterizes optimizer **Opt** as a DCO introduced in (5):

$$\mathbf{Opt}(\cdot;\phi) := D(\cdot;\phi). \qquad (12)$$

As discussed in Section II-A, this formulation contains a range of well-known first-order update rules as special cases.

To demonstrate the representational capacity of general DCOs as formulated in (5) within the meta-learning context, we focus on the subclass of unconstrained differentiable QPs. Note that this is a narrower subclass of DCO as we no longer have an arbitrary convex objective but rather a convex quadratic one. However, as we will demonstrate, this narrower formulation lends itself naturally to generalize the structure of update rules of existing gradient-based methods. While the formulations themselves admit closed-form solutions, we treat these as convex optimizations in our implementations to stay true to the DCO framework.

1) DCO Gradient: We propose DCO Gradient based on the convex optimization (2) that encodes the vanilla GD update rule. The formulation discards the optimizer state $S_i^{(t)}$ and simply encodes the update rule:

$$\theta_i^{(t+1)} := \arg\min_{\theta}\{\big(G_i^{(t)}\big)^\top \theta + \frac{1}{2}||\Lambda \odot (\theta - \theta_i^{(t)})||_2^2\}, \quad (13)$$

where the parameterization is given by $\phi := \Lambda \in \mathbb{R}^p$. In this formulation, learning the parameter Λ can be viewed as optimizing the per-weight learning rate within vanilla GD.

2) DCO General Descent (DCOGD): We introduce a generalization of the previous approach that enables a general linear transformation of the update gradient:

$$\theta_i^{(t+1)} := \arg\min_{\theta}\{\big(BG_i^{(t)}\big)^\top \theta + \frac{1}{2}||\theta - \theta_i^{(t)}||_2^2\}, \quad (14)$$

where $\phi := B \in \mathbb{R}^{p\times p}$.

3) DCO Momentum (DCOM): Finally, we extend formulation (13) to include momentum information:

$$S_i^{(t+1)} := M \odot G_i^{(t)} + (\mathbf{1} - M) \odot S_i^{(t)}, \quad (15)$$

$$\theta_i^{(t+1)} := \arg\min_{\theta}\{\big(S_i^{(t+1)}\big)^\top \theta + \frac{1}{2}||\Lambda \odot (\theta - \theta_i^{(t)})||_2^2\}, \quad (16)$$

where the parameterization is given by $\phi := \{\Lambda, M \in \mathbb{R}^p\}$. Here, the DCO learns both the learning rate and the momentum averaging mechanism on a per-weight basis.

V. Theory

We illustrate the potential of the DCO framework by analyzing the meta-learner process for a class of linear least-squares problems. Specifically, we let the tasks $\mathcal{T}_i$ be the least-squares problems

$$\min_{\theta \in \mathbb{R}^p} \|X\theta - (y + X\Delta_i)\|_2^2, \quad (17)$$

where X is a *fixed* feature matrix with $X^\top X$ invertible and the regression targets vary using task-specific Δ_i. We restrict our task-dependent regression target shifts to lie in the range space of X for theoretical tractability and concreteness: note that each task assumes a shifted version of the same loss landscape, with the optimal weights also shifted by Δ_i.

Namely, let $\theta' = (X^\top X)^{-1}X^\top y$ be the typical least-squares solution to (17) in the case where $\Delta_i = 0$. It is then clear by inspection that $\theta_i^* = \theta' + \Delta_i$, and that the minimizing loss $l_i(\theta_i^*)$ is invariant to i; we thus denote the solution to (17) by l^*. Note that here we consider the case where training and validation data are identical for a particular task; i.e. $l_i = l_i^{\text{val}}$. With some abuse of notation, our kth meta-optimization step target task loss

$$l_{\text{target}}^{(k)} := l_{\text{target}}(\mathbf{Opt}_1(\theta^{(1)}; \phi^{(k)})) \quad (18)$$

consists of an identically constructed task (17) with a distinct Δ and $\theta^{(1)}$. Concretely, we consider the performance on the target loss after one inner loop optimizer step using the meta-parameters $\phi^{(k)}$ obtained from k meta-optimization steps. Naturally, we expect that increasing both the number of meta-optimization steps k and the number of training tasks N should help reduce the target loss, ideally such that $l_{\text{target}}^{(k)}$ approaches the optimum l^*. This is formalized in Theorem 1.

Theorem 1. *Consider executing Algorithm 1 with the DCOGD optimizer* (14) *and* $T = 1$ *on the set of shifted least-squares problems* $\{\mathcal{T}_i\}_{i=1}^N$ *introduced in* (17)*, each with an arbitrary but fixed initial parameter* $\theta_i^{(1)} \in \mathbb{R}^p$*. Instantiate* **MetaOpt** *as standard GD with step size* $\eta > 0$*. Finally, define the set of vectors* $\{Z_i\}_{i=1}^N$ *by*

$$Z_i := \theta_i^{(1)} - \Delta_i - \theta'.$$

Then if $\{Z_i\}_{i=1}^N$ *span* $\mathbb{R}^p$*, there exists a sufficiently small* η *such that the one-step target task loss* (18) *approaches the optimum as the number of meta-steps* $k \to \infty$*; specifically,*

$$l_{\textit{target}}^{(k)} - l^* \le O((1-\epsilon)^k),$$

for some $0 < \epsilon < 1$.

Proof. We can solve the gradient update from (14) in closed form. Doing this yields the following weight vector the ith task after one step on the inner problem:

$$\theta_i^{(2)} := \theta_i^{(1)} - BG_i^{(1)}. \quad (19)$$

The total loss l^{total} with $T = 1$ can therefore be written as:

$$\begin{aligned} l^{\text{total}} &= \frac{1}{N}\sum_{i=1}^N \|X(\theta_i^{(1)} - BG_i^{(1)}) - (y + X\Delta_i)\|_2^2 \\ &= \frac{1}{N}\sum_{i=1}^N \|XBG_i^{(1)} - X\theta_i^{(1)} + y + X\Delta_i\|_2^2. \quad (20) \end{aligned}$$

The meta-learning problem aims to minimize this loss over the meta-parameter $\phi = B$. We will proceed with two steps: (1) show that there exists a meta-parameter B^* for which l^{total} equals the optimal minimizer l^*, and (2) show that B^* is attained by our meta-learning procedure.

The existence of such a minimizing B^* can be shown directly by letting $B^* = (1/2)(X^\top X)^{-1}$. Noting that

$$G_i^{(1)} = 2X^\top X(\theta_i^{(1)} - \Delta_i) - 2X^\top y$$

from differentiation of (17), we can substitute B^* and $G_i^{(1)}$ into (20) to yield:

$$l^{\text{total}} = \frac{1}{N}\sum_{i=1}^N \|(X^\top X)^{-1}X^\top y - y\|_2^2 = l^*. \quad (21)$$

We now show that B^* is attained by the meta-learning procedure. Namely, we consider our total loss for each outer meta-learning step in Algorithm 1 to be a function $l^{\text{total}}(B)$ of our meta-parameter $\phi = B$. We let $l_{\text{target}}(B)$ be defined similarly. It is easy to verify that the gradient $\nabla_B l^{\text{total}}$ is Lipschitz in B; therefore, we aim to show strong convexity of l^{total} in B to complete the proof using standard convex optimization results.

For convenience, define $y_i' := -X\theta_i^{(1)} + y + X\Delta_i$. Substituting into (20), we want to show that the following is strictly convex:

$$l^{\text{total}}(B) = \frac{1}{N}\sum_{i=1}^{N} \|XBG_i^{(1)} + y_i'\|_2^2.$$

Expanding the square, scaling, and dropping terms which are linear in B and thus do not affect convexity, we have that $l^{\text{total}}(B)$ is strictly convex iff $f(B)$ is strictly convex, where

$$f(B) = \sum_{i=1}^{N} (G_i^{(1)})^\top B^\top X^\top XBG_i^{(1)}.$$

With some abuse of notation, we aim to compute the Hessian of $f(B^v)$ with respect to the vectorized $B^v = \text{vec}(B)$. Using standard matrix calculus identities [38] gives

$$\begin{aligned}\frac{\partial f}{\partial B} &= 2\sum_{i=1}^{N} X^\top XBG_i^{(1)}(G_i^{(1)})^\top \\ &= 2X^\top XB\sum_{i=1}^{N} G_i^{(1)}(G_i^{(1)})^\top.\end{aligned}$$

In order to compute the Hessian, we need to express $\text{vec}(\frac{\partial f}{\partial B}) = \frac{\partial f}{\partial B^v}$. Using the standard identity $\text{vec}(ABC) = (C^\top \otimes A)\ \text{vec}(B)$ yields

$$\frac{\partial f}{\partial B^v} = \left(\sum_{i=1}^{N} G_i^{(1)}(G_i^{(1)})^\top \otimes 2X^\top X\right) B^v.$$

Note that the gradient of f with respect to B^v is now linear in B^v. It is therefore immediate that that our desired Hessian is a constant matrix

$$\nabla^2_{B^v} f = \sum_{i=1}^{N} G_i^{(1)}(G_i^{(1)})^\top \otimes 2X^\top X.$$

Note that the Kronecker product of positive definite matrices is positive definite. Since $X^\top X \succ 0$ by assumption, $\nabla^2_{B^v} f \succ 0$ (and therefore $l^{\text{total}}(B)$ is strictly convex) if $\sum_{i=1}^{N} G_i^{(1)}(G_i^{(1)})^\top \succ 0$. This occurs iff the set of vectors $\{G_i^{(1)}\}_{i=1}^N$ spans $\mathbb{R}^p$. Invertibility of $X^\top X$ implies that this is equivalent to the collection of vectors $\{Z_i\}_{i=1}^N$ spanning $\mathbb{R}^p$, where Z_i is as defined in the theorem statement.

Therefore $\nabla^2_{B^v} f \succ 0$, and $l^{\text{total}}(B)$ is strongly convex. Letting $B^{(k)}$ denote the values of the meta-parameters after k iterations of GD, by standard convex optimization results [39, Theorem 3.6] we have that for a sufficiently small step size η,

$$\|B^{(k)} - B^*\|_2^2 \le O\left((1-\epsilon)^k\right),$$

where $0 < \epsilon < 1$. As $l_{\text{target}}(B)$ is Lipschitz on any bounded set around B^*, the linear convergence in parameter space implies linear convergence in value, and we have shown the desired statement. □

Note that the condition that $\{Z_i\}_{i=1}^{\infty}$ span $\mathbb{R}^p$ is satisfied almost surely for typical random initializations of $\theta_i^{(1)}$. Theorem 1 can thus be interpreted as follows: provided at least $N = p$ sensibly initialized meta-training tasks, the meta learner will eventually learn to solve any target task to arbitrary precision with *exactly one* inner-loop gradient descent step. This is an interesting formal guarantee that suggests the expressive power of our DCO meta-learning framework. While this section focuses on a particularly simple and tractable family of shifted least-squares problems as a proof-of-concept, we expect that the DCO meta-learning framework provides a tractable avenue for more sophisticated convergence results.

VI. Experiments

We verify the effectiveness of the proposed DCO meta-learning framework for some illustrative tasks. Specifically, we leverage the DCO optimizer instantiations introduced in Section IV to solve linear least squares, system identification, and smooth function interpolation tasks.

A. Meta-Training Setup

Meta-parameters ϕ were initialized such that the DCO optimizers resemble existing first-order update rules. Λ and M were set to constant vectors in (13) and (15) to mimic the GD update rules introduced in Section II-A. Similarly, we initialized B as the identity matrix in formulation (14).

One potential challenge in training DCO optimizers is in ensuring that the proposed formulations remain well-posed for the entirety of the unrolling of the computational graph represented by Algorithm 1. While the formulations as unconstrained QPs are by themselves well-posed, from a practical viewpoint potentially ill-posed inputs need to be handled. This is especially true for the initial meta-training epochs, where suboptimal meta-parameters may give rise undesirably small or large inner loop gradients. This was overcome by normalizing inner loop gradients before feeding them into the DCO optimizers.

In our experiments we set $T = 1$ with T as defined in Algorithm 1. Restricting T has the effect of explicitly training the DCO optimizer to perform an aggressive inner loop descent step. From a compututational standpoint, this restriction of T allows to reallocate compute resources from solving several DCOs in the inner loop to performing more meta-parameter updates.

Note that Algorithm 1 allows for any first-order meta-optimizer to perform updates on ϕ. For simplicity we restrict ourselves to using RMSProp with default hyperparameter settings as suggested in the `PyTorch` library.

The DCO optimizers were implemented on a 2.2 GHz single-core CPU using the `CVXPYLayers` library [30] and were solved using general-purpose interior-point solvers. While the implementation could be made more efficient, it suffices to outline the potential of the DCO meta-learning framework to outperform existing first-order baselines.

Throughout the experiments a comparison baseline of first-order methods Adam, SGD and RMSProp was considered due

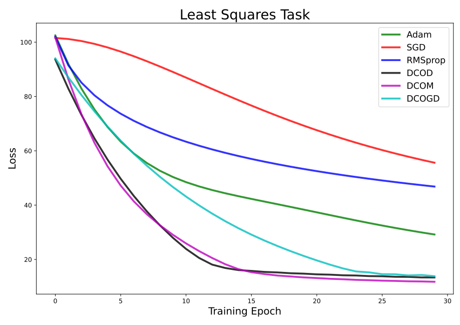

Fig. 1: Optimization performance on 100-dimensional LS tasks. Validation curves are averaged across 100 new tasks.

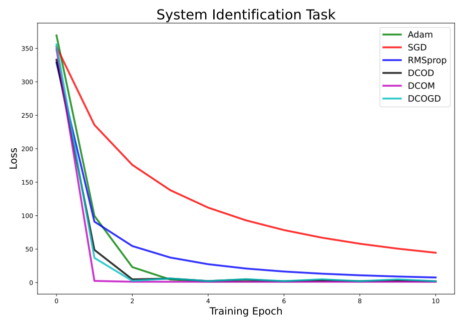

Fig. 2: Optimization performance on approximating the Beverton–Holt dynamics. Validation curves are averaged across 100 new tasks.

to their prevalence in solving unconstrained minimizations. For each baseline optimizer the learning rate was tuned and the best validation performance was reported.

B. *Least Squares Task*

We first focus on solving least-squares (LS) problems

$$\min_{\theta \in \mathbb{R}^{100}} \|X\theta - y\|_2^2, \tag{22}$$

where $X \in \mathbb{R}^{100\times 100}$, $y \in \mathbb{R}^{100}$ with $X_{ij}, y_i \sim \mathcal{N}(0,1)\ \forall i,j \in [100]$. The meta-training set was constructed by sampling 100 tasks according to (22). For each task, a LS objective was sampled which acts as both as l_i and l_i^{val}, i.e. $l_i = l_i^{\text{val}}$ for $i \in [100]$. Meta-training was run for $M = 20$ epochs. Then 100 new tasks were sampled and the evolution of the average loss across tasks over 30 training epochs was compared with existing first-order methods. Figure 1 shows the results. The DCO optimizers exhibit substantially faster convergence compared to classical baselines.

C. *System Identification Task*

Next, we consider the task of identifying the underlying nonlinear discrete-time dynamics for population growth. We approximate the Beverton–Holt model given by

$$n_{t+1} = f(n_t) := \frac{R_0 n_t}{K + n_t}, \tag{23}$$

where n_t represents the population density in generation t, $R_0 > 0$ is the proliferation rate per generation, and $K > 0$ is the carrying capacity of the environment. To introduce stochasticity into the model we include additive disturbance $d \sim \mathcal{N}(0, 0.1)$. In this context, we define a particular task by sampling a system with $R_0, K \sim \mathcal{U}(1,2)$ and then generating training and validation samples $\{n, f(n)\}$ with $n \sim \mathcal{U}(0,10)$. For each task we sample 500 training points and 100 validation points. The goal is to learn the underlying discrete nonlinear dynamics using a feedforward architecture with design (1-5-5-1), i.e. 2 hidden layers with 5 units each. The training of each network is carried out on the training set sampled for each task, and the final performance for that task is measured using the mean-square error (MSE) metric on the associated validation set. Meta-training was run for $M = 20$ epochs. Figure 2 presents the performance comparison between considered methods on 100 newly sampled tasks. The DCO optimizers continue to outperform baselines.

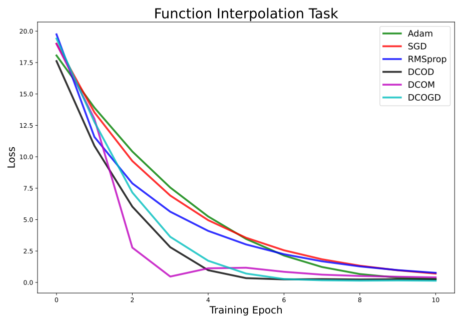

Fig. 3: Optimization performance on smooth interpolation tasks. Validation curves are averaged across 10 new tasks.

D. *Smooth Function Interpolation Task*

We finally consider the task of interpolating a real, nonlinear, smooth, univariate function via regression. As an illustrative example, we consider the smooth function

$$g(x) = a\cos(bx)\exp(-c|x|) \tag{24}$$

where $a, b, c \sim \mathcal{U}(0,1)$. A particular task is constructed by sampling 500 training points and 100 validation points from an instance of $g(x)$. The goal of the task was to learn a feed-forward network (FFN) with architecture (1-10-10-1) consisting of 2 hidden layers with 10 units each that yields low validation loss. As before, meta-training was run for $M = 20$ epochs. The performance comparison with first-order methods on a new set of tasks is shown in Figure 3. The validation loss learning curves are averaged over 10 tasks. Similar to previous settings, we have obtained an improved convergence of DCO optimizers over baselines.

VII. Conclusion

This work introduces a novel DCO-based approach for optimizer design within the context of meta-learning. The DCO meta-learning framework remains loyal to the inherent convex nature of existing first-order update rules. We demonstrate that DCO-based optimizers not only generalize existing first-order methods but also have the potential of representing novel update rules. Theoretically, we show rapid convergence to the optimal update rule when meta-training DCOGD optimizers for a family of linear least-squares tasks. Experimentally, we demonstrate faster convergence of the DCO instantiations as compared to existing first-order methods on a range of illustrative tasks. Exciting future work involves finding a more general instantiation of DCO optimizers and scaling this approach to more complex networks.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks." Red Hook, NY, USA: Curran Associates Inc., 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[5] N. Sünderhauf, O. Brock, W. J. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research*, vol. 37, pp. 405 – 420, 2018.

[6] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400 – 407, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729586

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[8] L. El Ghaoui, F. Gu, B. Travacca, A. Askari, and A. Tsai, "Implicit deep learning," *SIAM Journal on Mathematics of Data Science*, vol. 3, no. 3, pp. 930–958, 2021.

[9] T. Gautam, B. G. Anderson, S. Sojoudi, and L. El Ghaoui, "A sequential greedy approach for training implicit deep models," *Technical report*, 2022. [Online]. Available: https://people.eecs.berkeley.edu/~tgautam23/publications/ImplicitSequential_Preprint.pdf

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[11] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017.

[12] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 44, no. 09, pp. 5149–5169, sep 2022.

[13] E. Grefenstette, B. Amos, D. Yarats, P. M. Htut, A. Molchanov, F. Meier, D. Kiela, K. Cho, and S. Chintala, "Generalized inner loop meta-learning," 2019.

[14] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 1126–1135.

[15] A. Antoniou, H. Edwards, and A. Storkey, "How to train your maml," 2018. [Online]. Available: https://arxiv.org/abs/1810.09502

[16] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few shot learning," *CoRR*, vol. abs/1707.09835, 2017. [Online]. Available: http://arxiv.org/abs/1707.09835

[17] A. Antoniou, H. Edwards, and A. Storkey, "How to train your maml," 2018. [Online]. Available: https://arxiv.org/abs/1810.09502

[18] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *International Conference on Artificial Neural Networks*, 2001.

[19] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 3988–3996.

[20] K. Li and J. Malik, "Learning to optimize," 2016. [Online]. Available: https://arxiv.org/abs/1606.01885

[21] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *International Conference on Learning Representations*, 2016.

[22] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *Artificial Neural Networks — ICANN 2001*, G. Dorffner, H. Bischof, and K. Hornik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 87–94.

[23] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," in *International Conference on Learning Representations*, 2017.

[24] S. Qiao, C. Liu, W. Shen, and A. Yuille, "Few-shot image recognition by predicting parameters from activations," 06 2018, pp. 7229–7238.

[25] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/90e1357833654983612fb05e3ec9148c-Paper.pdf

[26] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 4080–4090.

[27] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[28] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[29] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 136–145.

[30] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter, "Differentiable convex optimization layers," in *Advances in Neural Information Processing Systems*, 2019.

[31] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/ba6d843eb4251a4526ce65d1807a9309-Paper.pdf

[32] P.-W. Wang, P. L. Donti, B. Wilder, and J. Z. Kolter, "Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver," in *International Conference on Machine Learning*, 2019.

[33] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[34] J. Wright and Y. Ma, *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press, 2021.

[35] A. Beck and M. Teboulle, "Mirror descent and nonlinear projected subgradient methods for convex optimization," *Operations Research Letters*, vol. 31, no. 3, pp. 167–175, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167637702002316

[36] C. Blair, "Problem complexity and method efficiency in optimization (a. s. nemirovsky and d. b. yudin)," *SIAM Review*, vol. 27, no. 2, pp. 264–265, 1985. [Online]. Available: https://doi.org/10.1137/1027074

[37] B. Amos, L. Xu, and J. Z. Kolter, "Input convex neural networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 146–155.

[38] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," Oct. 2008, version 20081110. [Online]. Available: http://www2.imm.dtu.dk/pubdb/p.php?3274

[39] G. Garrigos and R. M. Gower, "Handbook of convergence theorems for (stochastic) gradient methods," *arXiv preprint arXiv:2301.11235*, 2023.