IEEE Control Systems Letters paper presented at
2024 IEEE 63rd Conference on Decision and Control (CDC)
December 16-19, 2024. MiCo, Milan, Italy

Learning to optimize with convergence guarantees using nonlinear system theory

Andrea Martin and Luca Furieri

***Abstract*— The increasing reliance on numerical methods for controlling dynamical systems and training machine learning models underscores the need to devise algorithms that dependably and efficiently navigate complex optimization landscapes. Classical gradient descent methods offer strong theoretical guarantees for convex problems; however, they demand meticulous hyperparameter tuning for non-convex ones. The emerging paradigm of learning to optimize (L2O) automates the discovery of algorithms with optimized performance leveraging learning models and data – yet, it lacks a theoretical framework to analyze convergence of the learned algorithms. In this paper, we fill this gap by harnessing nonlinear system theory. Specifically, we propose an unconstrained parametrization of all convergent algorithms for smooth non-convex objective functions. Notably, our framework is directly compatible with automatic differentiation tools, ensuring convergence by design while learning to optimize.**

I. Introduction

Many fundamental tasks in machine learning (ML) and optimal control involve solving optimization problems, that is, computing a solution

$$x^\star = \arg\min_{x\in\mathbb{R}^d} \; f(x)\,, \tag{1}$$

for a given objective function $f(\cdot)$. As the real-world applications of ML and optimal control grow in complexity, from training deep neural networks (NNs) for high-dimensional classification tasks to optimally operating large-scale cyber-physical systems, finding analytical solutions to these optimization problems becomes prohibitive. This has led to the increased use of numerical optimization algorithms, such as gradient descent methods, which iteratively approach the critical points $\hat{x}$ of $f(\cdot)$, i.e., the values $\hat{x}$ such that $\nabla f(\hat{x}) = 0$. As we rely on iterative algorithms to solve complex optimization problems, their ability to quickly and robustly converge to good critical points becomes crucial.

Traditionally, the optimization literature has focused on hand-crafting algorithms tailored to specific instances of (1), such as those showcasing convex objective functions [1]. For instance, widely-used optimization algorithms include vanilla gradient descent, the heavy-ball method, and Nesterov's accelerated method [2]. While these algorithms come with strong theoretical guarantees for convex optimization, their performance on non-convex problems, such as training deep NNs, crucially depends on their hyperparameters [3]. Besides, due to a lack of a general theory, hyperparameter tuning is often performed by domain experts based on best practices and know-how.

In the attempt to provide a unifying take on the analysis and synthesis of optimization algorithms, the control theory and ML communities have increasingly interpreted iterative update rules as evolving discrete-time dynamical systems; we refer to the recent review and road-map paper [4] for a comprehensive list of references. Notably, [5]–[8] have studied robustness and worst-case performance of optimization algorithms, leading to the design of new methods with optimized convergence rates [5], [8] or sublinear regret guarantees in online optimization scenarios [9]. All the results mentioned above are limited to convex objective functions. The paradigm of feedback optimization [10], [11] implements optimization algorithms directly in closed-loop with dynamical systems, endowing them with the ability to self-regulate and converge towards the solution of desired nonlinear optimization programs. Analyzing and shaping the transient performance of the resulting closed-loop behavior remains an open venue for research.

To tackle the non-convex and time-varying optimization landscapes that are ubiquitous in ML and optimal control, a learning to optimize (L2O) shift of paradigm has been emerging: moving from *in silico* algorithms designed by hand based on general problem properties [4], towards embracing ML to discover powerful algorithms from data. In this context, "data" refers to example optimization problems of interest provided during a training phase – which can take place either offline or online. Specifically, the L2O approach parametrizes algorithms in a very general way, and performs meta-training over these parameters; we refer to [12] for a recent overview of L2O and a detailed discussion on its advantages and disadvantages. As observed in [12], when the distribution of sample problems is narrow, learned algorithms can overfit the tasks and discover shortcuts that classic algorithms do not take. When the distribution of sample problems is sufficiently varied, the algorithm's performance transfers well to new tasks [13]–[15]. However, it has been observed that optimizers trained as per [13] may lack convergence guarantees on almost all unseen tasks [12] – even when they are taken from the same task distribution [15]. A mitigation to avoid compounding errors is proposed in [15] based on reinforcement meta-learning. For convex objective functions, provable convergence guarantees of learned optimizers were considered in [16] by exploiting a conservative fall-back mechanism that switches to a fixed convergent algorithm when the learned updates are too aggressive. To the best of the authors' knowledge, despite the outstanding empirical

This research is supported by the Swiss National Science Foundation through the NCCR Automation (grant agreement 51NF40_180545) and the Ambizione grant PZ00P2_208951.

A. Martin and L. Furieri are with the Institute of Mechanical Engineering, EPFL, Switzerland. E-mail addresses: {andrea.martin, luca.furieri}@epfl.ch.

performance, the theoretical underpinnings of L2O such as convergence and robustness guarantees of the learned algorithms stand as uncharted territory.

Contributions: In this paper, we establish methods to learn high-performance optimization algorithms that are inherently convergent for smooth non-convex functions. From control system theory, we inherit the emphasis on convergence guarantees [5], [8], [10], [11], ensuring learned algorithms converge to local solutions in a provable and quantifiable way. From ML, we embrace the ability to tackle user-defined performance metrics through automatic differentiation[1], and the outstanding generalization capabilities to previously unseen optimization problems. Our key contribution is the reformulation of the problem of learning optimal convergent algorithms into an equivalent, unconstrained one that is directly amenable to automatic differentiation tools.

We achieve this by dividing update rules into: 1) a gradient descent step that ensures convergence, and 2) a learnable term that enhances performance without compromising convergence. Notably, our method not only guarantees algorithm convergence, but it also encompasses *all and only* convergent algorithms. As a result, we do not rely on safeguarding and early-stopping mechanisms [12], [13], [16]. Instead, the L2O approach we consider shifts the challenge to selecting a metric for algorithm performance that appropriately reflects the desiderata, for instance, trading off the speed of convergence and the quality of the solution. Furthermore, we achieve convergence even when dealing with incomplete gradient measurements, making our methodology relevant for ML with batch data. We validate the effectiveness and generalizability of our methodology through ML benchmarks.

Notation: The set of all sequences $\mathbf{x} = (x_0, x_1, x_2, \dots)$ where $x_t \in \mathbb{R}^n$ for all $t \in \mathbb{N}$ is denoted as ℓ^n. For $\mathbf{x} \in \ell^n$, we denote by $z\mathbf{x} = (x_1, x_2, \dots)$ the sequence shifted one-time-step forward. Moreover, $\mathbf{x}$ belongs to $\ell_2^n \subset \ell^n$ if $\|\mathbf{x}\|_2 = \sqrt{\sum_{t=0}^{\infty} |x_t|^2} < \infty$, where $|\cdot|$ denotes any vector norm. When clear from the context, we omit the superscript n from ℓ^n and ℓ_2^n. For a function $g : \mathbb{R}^n \to \mathbb{R}^m$, we write $g(\mathbf{x}) = (g(x_0), g(x_1), \dots) \in \ell^m$. A causal operator $\mathbf{A} : \ell^n \to \ell^m$ such that $\mathbf{A}(\mathbf{x}) = (A_0(x_0), A_1(x_{1:0}), \dots, A_t(x_{t:0}), \dots)$ is said to be ℓ_2-stable if $\mathbf{A}(\mathbf{x}) \in \ell_2^m$ for all $\mathbf{x} \in \ell_2^n$. Equivalently, we write $\mathbf{A} \in \mathcal{L}_2$. We denote by $\lfloor x \rfloor$ the greatest integer smaller than $x \in \mathbb{R}$ and use $a \bmod b$ to denote the remainder of $a \in \mathbb{N}$ when divided by $b \in \mathbb{N}$.

II. Problem Formulation

In this paper, we focus on optimization problems in the form (1) where $f(\cdot)$ has β-Lipschitz gradients, that is, $|\nabla f(x) - \nabla f(y)| \le \beta |x - y|$ for all $x, y \in \mathbb{R}^d$. We denote the set of such β-smooth functions by $\mathcal{S}_\beta$. Further, it is assumed that $f(\cdot)$ is bounded from below. We describe an iterative optimization algorithm via the recursion

$$x_{t+1} = x_t + u_t = x_t + \pi_t(f, x_{t:0}), \quad t \in \mathbb{N}, \tag{2}$$

where $x_0 \in \mathbb{R}^d$ is the initial guess, $x_t \in \mathbb{R}^d$ is the candidate solution vector after t iterations, and $u_t = \pi_t(f, x_{t:0}) \in \mathbb{R}^d$ is the algorithm update rule. We can write (2) compactly as

$$z\mathbf{x} = \mathbf{x} + \boldsymbol{\pi}(f, \mathbf{x}) + z\boldsymbol{\delta}^{x_0}, \tag{3}$$

where $\boldsymbol{\pi}(f, \cdot) = (\pi_0(f, x_0), \pi_1(f, x_{1:0}), \dots)$ is a causal operator for any objective function f. The initial state sequence $\boldsymbol{\delta}^{x_0}$ is defined as $\boldsymbol{\delta}^{x_0} = (x_0, 0, \dots) \in \ell_2$. We proceed to define the fundamental notion of convergent algorithms.

Definition 1: Consider the iteration (2). An update rule $\boldsymbol{\pi}(f, \mathbf{x})$ is *convergent* for f if for any $x_0 \in \mathbb{R}^d$

$$\lim_{t\to\infty} \pi_t(f, x_{t:0}) = 0, \quad \lim_{t\to\infty} \nabla f(x_t) = 0. \tag{4}$$

Equivalently, we write $\boldsymbol{\pi} \in \Gamma(f)$. Additionally, if

$$\|\boldsymbol{\pi}(f, \mathbf{x})\|^2 < \infty, \quad \|\nabla f(\mathbf{x})\|^2 < \infty, \tag{5}$$

we say the algorithm is *square-sum convergent* for f. Equivalently, we write $\boldsymbol{\pi} \in \Sigma(f)$.

Note that every update rule in $\Sigma(f)$ also lies in $\Gamma(f)$ for every $f \in \mathcal{S}_\beta$. In particular, although (4) and (5) both guarantee convergence to a critical point of f as $t \to \infty$, (5) only holds for those algorithms in $\Gamma(f)$ that achieve a sufficiently fast asymptotic convergence rate. Further, we observe that classical convergence bounds for smooth convex optimization in the form $|x_t - x^\star| \le K\rho^t |x_0 - x^\star|$, where $K > 0$, see, e.g, [5], [8], readily imply that $\nabla f(\mathbf{x}) \in \ell_2$.

Given a distribution $\mathcal{F}$ over functions in $\mathcal{S}_\beta$ and a distribution $\mathcal{X}_0$ over initial guesses $x_0 \in \mathbb{R}^d$, the problem of designing an optimal convergent algorithm is formulated as

$$\min_{\boldsymbol{\pi}} \quad \mathbb{E}_{f\sim\mathcal{F}, x_0\sim\mathcal{X}_0} \left[\texttt{MetaLoss}(f, \mathbf{x})\right] \tag{6a}$$

$$\text{subject to} \quad x_{t+1} = x_t + \pi_t(f, x_{t:0}), \tag{6b}$$

$$\boldsymbol{\pi}(f, \mathbf{x}) \in \Sigma(f), \ \forall f \in \mathcal{S}_\beta, \tag{6c}$$

where $\Sigma(f)$ can be relaxed to $\Gamma(f)$ depending on the design specifications. As suggested in [13]–[15], a useful choice for $\texttt{MetaLoss}(f, \mathbf{x})$ in (6a) is given by

$$\texttt{MetaLoss}(f, \mathbf{x}) = \sum\nolimits_{t=0}^{T} \alpha_t |\nabla f(x_t)|^2 + \gamma_t f(x_t), \tag{7}$$

where $\alpha_t \ge 0$ and $\gamma_t \ge 0$. Specifically, α_t promotes fast convergence of $\nabla f(x_t)$, while γ_t drives the algorithm closer to the solution of (1). As a result, α_t and γ_t act as hyperparameters that must be tuned to strike a balance between these two competing aspects. At the same time, the constraint (6c) ensures convergence to a critical point $\hat{x}$ satisfying $\nabla f(\hat{x}) = 0$ for any future $f \in \mathcal{S}_\beta$.

Remark 1 (The value of convergence): Excluding update rules that fail to comply with (6c) is crucial, as (6c) ensures algorithm convergence, even if meta-optimization is prematurely stopped. Notably, convergence is necessary for generalizing to unseen problems $f_{\text{new}} \in \mathcal{S}_\beta$ drawn from a different distribution $\mathcal{F}'$ and achieving sublinear meta-regret in online convex optimization [9].

[1]The computational engine to efficiently compute derivatives of functions specified by a computer program using the chain rule repeatedly, e.g., [17].

III. Main Results

This section characterizes update rules that converge according to Definition 1, and describes how to learn over them. First, given full gradient measurements, we establish a complete parametrization of all and only the algorithms that converge in the sum-square sense as per (6c). Second, for the case – common in ML and deep learning applications – where $f(x) = \sum_{i=0}^{M-1} f_i(x)$ and only partial gradients $\nabla f_i(x)$ are available at each step, we parametrize algorithms that converge asymptotically as per (4). In both scenarios, we directly parametrize convergent update rules via a vector $\theta \in \mathbb{R}^D$, thus enabling unconstrained learning of convergent-by-design algorithms via automatic differentiation tools.

A. Learning over all square-sum convergent algorithms

We start by proving that any update rule in the form

$$\boldsymbol{\pi}(f, \mathbf{x}) = -\eta \nabla f(\mathbf{x}) + \mathbf{v}\,, \tag{8}$$

lies in $\Sigma(f)$ for any $\mathbf{v} \in \ell_2$ and any $f \in \mathcal{S}_\beta$, as long as $0 < \eta < \beta^{-1}$. In other words, if we perturb standard gradient descent with an ℓ_2 *"enhancement"* term – designed, e.g., to escape a bad local minimum or a saddle point – we preserve square-sum convergence to a critical point of f.

Lemma 1: Consider the recursion (3). The update rule given by (8) with $0 < \eta < \beta^{-1}$ satisfies (6c) for every choice of $\mathbf{v} \in \ell_2$.

The class of algorithms in the form (8) suggests a useful separation of roles; a gradient descent update can be used to ensure convergence, while an enhancement term $\mathbf{v} \in \ell_2$ can be learned to improve the algorithm performance. Nonetheless, a crucial question regarding the conservatism of searching over $\mathbf{v} \in \ell_2$ in (8) remains.

Can any *convergent algorithm complying with* (6c) *be written as the sum of a gradient-based update and an enhancement signal* $\mathbf{v} \in \ell_2$ *as per* (8)*?*

In what follows, we answer in the affirmative, further revealing that $\mathbf{v} \in \ell_2$ must be parametrized as a function of $\boldsymbol{\delta}^{x_0}$ in order to recover any convergent behavior using (8). Our proof hinges on studying the closed-loop mappings induced by an update rule $\boldsymbol{\pi}(f, \mathbf{x})$.

Definition 2: Consider the recursion (3). For any update rule $\boldsymbol{\pi}(f, \mathbf{x})$, the mapping $(f, \boldsymbol{\delta}^{x_0}) \to (\mathbf{x}, \mathbf{u}, \nabla f(\mathbf{x}))$ is denoted as the *closed-loop mapping* induced by $\mathbf{u} = \boldsymbol{\pi}(f, \mathbf{x})$.

The terminology above is drawn from control system theory. Under a system-theoretic lens, we can view $\boldsymbol{\pi}(f, \mathbf{x})$ as an objective-dependent state feedback control policy, and $(f, \boldsymbol{\delta}^{x_0}) \to (\mathbf{x}, \mathbf{u}, \nabla f(\mathbf{x}))$ as the corresponding closed-loop behavior. The constraint (6c) thus translates to regulating the system output signal $\mathbf{y}$ to 0, further requiring that $\mathbf{y} = \nabla f(\mathbf{x}) \in \ell_2$, robustly for any $x_0 \in \mathbb{R}^d$ and any $f \in \mathcal{S}_\beta$.

Lemma 2: Let $x_0 \in \mathbb{R}^d$ and $f \in \mathcal{S}_\beta$. Define

$$(f, \boldsymbol{\delta}^{x_0}) \to (\mathbf{x}_{\boldsymbol{\pi}}, \mathbf{u}_{\boldsymbol{\pi}}, \nabla f(\mathbf{x}_{\boldsymbol{\pi}}))\,, \tag{9}$$

as the closed-loop mapping induced by a policy $\mathbf{u} = \boldsymbol{\pi}(f, \mathbf{x})$. For any $\boldsymbol{\pi}(f, \mathbf{x})$ complying with (6c), there exists an operator $\mathbf{V} \in \mathcal{L}_2$ such that the closed-loop mapping given by

$$(f, \boldsymbol{\delta}^{x_0}) \to (\mathbf{x}, -\eta \nabla f(\mathbf{x}) + \mathbf{V}(\boldsymbol{\delta}^{x_0}), \nabla f(\mathbf{x}))\,, \tag{10}$$

is equivalent to (9).

The completeness property stated above is key, as it implies that (8) encompasses all sum-square convergent algorithms – including those that *globally* minimize (6a). Together with Lemma 1, Lemma 2 leads to our main result.

Theorem 1: If $0 < \eta < \beta^{-1}$, the meta-optimization problem (6) is equivalent to

$$\min_{\mathbf{V} \in \mathcal{L}_2} \quad \mathbb{E}_{f \sim \mathcal{F}, x_0 \sim \mathcal{X}_0} \left[\texttt{MetaLoss}(f, \mathbf{x}) \right] \tag{11a}$$

$$\text{subject to} \quad z\mathbf{x} = \mathbf{x} - \eta \nabla f(\mathbf{x}) + \mathbf{V}(\boldsymbol{\delta}^{x_0}) + z\boldsymbol{\delta}^{x_0}\,. \tag{11b}$$

A few comments are in order. First, any possibly suboptimal solution $\mathbf{V} \in \mathcal{L}_2$ to (11) yields a converging algorithm complying with (6c). Second, every converging algorithm complying with (6c) is recovered by appropriately choosing $\mathbf{V} \in \mathcal{L}_2$ with no conservatism. Third, as the convergence constraint (6c) simplifies to $\mathbf{V} \in \mathcal{L}_2$, we can use finite-dimensional approximations of operators in $\mathcal{L}_2$, that is

$$\mathbf{V}(\boldsymbol{\delta}^{x_0}, \theta) \in \ell_2, \quad \forall \boldsymbol{\delta}^{x_0} \in \ell_2,\ \forall \theta \in \mathbb{R}^D\,, \tag{12}$$

to translate (11) into the unconstrained[2] optimization problem of learning the best parameter $\theta \in \mathbb{R}^D$. To ensure that (12) holds, one can, for instance, model $\mathbf{V}$ as a stable recurrent NN $v_t = \phi(\theta, v_{t-1}, u_t)$, where $\phi(\cdot)$ is contracting for all $\theta \in \mathbb{R}^D$; several such models have recently been developed in the literature [18], [19] and are readily implementable. Despite approximating the original infinite-dimensional problem (6), these parametrizations have been shown to be highly expressive [19], with formal density and suboptimality bounds for linear operators in $\mathcal{L}_2$ [20].

In practice, it may prove beneficial to introduce explicit dependence of $\mathbf{V}$ in (12) on additional input features besides $\boldsymbol{\delta}^{x_0}$. Indeed, as shown in [15], learning over algorithms that react to $(x_{t:0}, \nabla f(x_{t:0}), f(x_{t:0}))$ can be effective in transferring their meta-performance to ML tasks vastly different from those encountered during training. While Theorem 1 proves that designing an update rule that solely reacts to $x_0 \sim \mathcal{X}_0$ is sufficient for achieving meta-optimal behaviors, additional input features could significantly improve how effectively we navigate the meta-optimization landscape. For instance, by defining $\boldsymbol{\omega} = \boldsymbol{\Omega}(\mathbf{x}, \nabla f(\mathbf{x}), f(\mathbf{x}))$ and $\mathbf{z} = \mathbf{Z}(\boldsymbol{\delta}^{x_0})$, where $\boldsymbol{\Omega} : \ell \to \ell$ and $\mathbf{Z} \in \mathcal{L}_2$ are operators to be freely designed, we can generate $\mathbf{v} \in \ell_2$ as follows

$$v_t = |z_t||\omega_t|^{-1}\omega_t\,. \tag{13}$$

Using (13), sum-square convergence is preserved by design, as we set $|v_t| = |z_t|$ at all times, and $\mathbf{z} \in \ell_2$. Further, completeness as per Lemma 2 is maintained; this is proved by choosing $\mathbf{Z}$ according to (21) in the Appendix and selecting $\boldsymbol{\Omega}(\mathbf{x}, \nabla f(\mathbf{x}), f(\mathbf{x})) = \mathbf{Z}(\boldsymbol{\delta}^{x_0})$. We remark that, even though completeness is guaranteed, discovering alternative parametrizations of converging algorithms beyond (8) with (13) could be beneficial; indeed, despite their theoretical equivalence, different convergence strategies may result in more favorable meta-optimization landscapes.

[2]Note that (11b) defines the signal $\mathbf{x}$ through the recursion (2) and does not pose any constraints on $\mathbf{V} \in \mathcal{L}_2$.

B. The case of gradients with errors

In many ML and deep learning tasks, $f(x)$ is obtained as the empirical average of the cost over a batch of input data, that is, $f(x) = \sum_{i=0}^{M-1} f_i(x)$. In these cases, global gradient information $\nabla f(x)$ may not be available, and the candidate solution x_t is updated based on $\nabla f_i(x_t)$ for some $i \in [0, M-1]$ only. Drawing connections with analysis techniques for stochastic gradient descent (SGD) from [21], we proceed to parametrize a rich class of asymptotically convergent algorithms that rely on partial gradient information.

Theorem 2: Let $f(x) = \sum_{i=0}^{M-1} f_i(x)$ be separable in $M \in \mathbb{N}$ continuously differentiable components $f_i \in \mathcal{S}_\beta$ satisfying, for some non-negative constants A and B,

$$|\nabla f_i(x)| \leq A + B|\nabla f(x)|\,,\ \forall x \in \mathbb{R}^d\,.^3 \tag{14}$$

Choose any stepsize sequence $\boldsymbol{\eta} \in \ell_2$ such that $\sum_{t=0}^{\infty} \eta_t = \infty$ with $\eta_t > 0$ at all times, and any $\mathbf{v}$ such that

$$|v_t| \leq \eta_{\lfloor t/M \rfloor}(C + D|\nabla f(x_t)|)\,, \tag{15}$$

for some non-negative constants C and D. Then, the update

$$\pi_t(f, x_t) = -\eta_{\lfloor t/M \rfloor}(\nabla f_{t \bmod M}(x_t) + v_t)\,,^4 \tag{16}$$

is convergent according to (4), that is, $\boldsymbol{\pi}(f, \mathbf{x}) \in \Gamma(f)$.

The proof of Theorem 2 adapts the analysis of [21, Proposition 2] by accounting for the contribution of v_t as per (16). To ensure (15), while endowing our learned algorithms with the ability to react to input features, we propose using

$$\begin{aligned} &v_t = \eta_{\lfloor t/M \rfloor}|z_t||\omega_t|^{-1}\omega_t\,, \quad \mathbf{z} = \mathbf{Z}(\boldsymbol{\delta}^{x_0})\,, \\ &\omega_t = \Omega_t(x_{t:0}, \nabla f_\tau(x_t), \ldots, \nabla f_0(x_0), f_\tau(x_t), \ldots, f_0(x_0))\,, \end{aligned} \tag{17}$$

where $\tau = t \bmod M$, and $\boldsymbol{\Omega} : \ell \to \ell$ and $\mathbf{Z} \in \mathcal{L}_2$ are operators to be freely designed. In this way, similarly to (13), we have $|v_t| = \eta_{\lfloor t/M \rfloor}|z_t| \leq \eta_{\lfloor t/M \rfloor} \max_{t \in \mathbb{N}}\ z_t$, and we thus satisfy (15) with $C = \max_{t \in \mathbb{N}}\ z_t$ and $D = 0$.

The update rule (16) cycles through the gradients ∇f_i and applies an enhancement signal to be learned. Coherently with standard SGD, Theorem 2 guarantees asymptotic convergence of the learned algorithm – despite the additional presence of $\mathbf{v}$ satisfying (15). Moreover, while (15) may restrict the set of $\boldsymbol{\pi}(f, \mathbf{x}) \in \Gamma(f)$ that our parametrization can achieve, we proceed to illustrate the rich expressivity of learning over enhancement terms modeled as (17).

IV. Experiments

Motivated by [13], we consider the problem of learning to optimize the parameters x_t of a shallow NN for image classification with the MNIST dataset. Further, we investigate how our optimizer generalizes to different network activation functions and different initial parameter distribution $\mathcal{X}_0$.

We model our trainable optimizer as per (16)-(17), using a recurrent equilibrium network[5] [19] with depth of $r = 3$ layers and internal state dimension $n = 3$ as a model for $\mathbf{Z}(\theta) \in \mathcal{L}_2$, and a multilayer perceptron (MLP) with 2 hidden layers as a model for $\boldsymbol{\Omega}(\theta)$. We instead model the shallow NN as a simple perceptron that, given a vectorized image s corresponding to a handwritten digit as input, predicts the label $\hat{l}(s, x_t)$ according to the criterion:

$$\arg\max_i\ o(s, x_t) = \arg\max_i\ [\tanh(sW_t^\top + b_t)]_i\,, \tag{18}$$

where W_t and b_t are the trainable parameters of the classifier, whose scalar entries are collected in x_t, and $[\cdot]_i$ denotes the i-th entry of a vector. Based on (18), we also define the classification loss $g(s, l, x_t)$ on an image s as the cross entropy loss between the softmax transformation of $o(s, x_t)$ and l, the true label of s, encoded as a one-hot vector.

To encourage our learned updates (16) to promote the accuracy of the classifier predictions (18) after $T = 50$ iterations of (2), we consider the meta-loss (7) with $\alpha_t = 0$, $\gamma_t = 0.95^{T-t}$, and let $f(x_t)$ be the cross entropy loss on the training dataset. For fixed parameters θ of $\mathbf{Z}$ and $\boldsymbol{\Omega}$, we approximate (6a) by repeating the training of W_t and b_t in (18) for 10 times, using different initial parameters sampled from a distribution $\mathcal{X}_0$ that is uniform in the interval $[0, 0.01]$. Motivated by Theorem 2, we estimate $f(x_t)$ and $\nabla f(x_t)$ in (11b) using random minibatches of 128 images drawn sequentially. Then, consistently with [13], we perform the minimization of (6a) using Adam with a learning rate of 0.01. We use 80% of the training dataset for optimizing θ.

After 40 training epochs, we freeze the value of θ, and we benchmark the performance of our learned optimizer against standard optimizers including Adam, SGD, Nesterov's accelerated gradient (NAG), and RMSprop. To assess the generalization capabilities of (17), we use the remaining 20% of the training dataset to train a shallow NN that: 1) uses sigmoid or ReLU as activations in (18), and 2) whose initial parameters $[x_0]_i$ are sampled from independent and identically distributed Gaussian distributions $\mathcal{N}(0, 0.1)$.

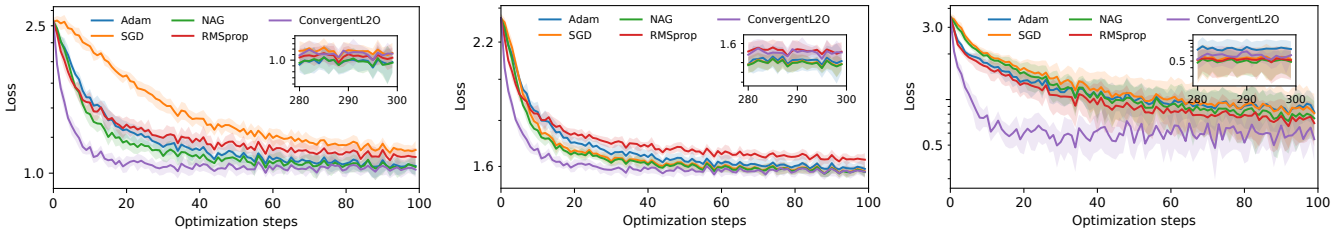
We report training curves for our learned optimizer and for classical hand-crafted algorithms in Figure 1 and the corresponding average test accuracy in the tables below.[6] In all considered scenarios, learned algorithms excel in finding shortcuts, that is, in steering x_0 to a good local minimum within a few iterations; this is reflected in the superior test accuracy achieved after only $t = 20$ optimization steps. As the gradient norm diminishes, our learned algorithm favors simple gradient-based updates to ensure convergence, as predicted by Theorem 2. Despite being trained to optimize for $T = 50$ optimization steps only, the average test accuracy of our method matches that of classical algorithms after 300 iterations of (2) – compare also the zooming at $t = 300$ in

[3]This assumption encompasses the case of Lipschitz continuous function components f_i when $B = 0$, and is therefore common in the analysis of SGD-related methods [21].

[4]Unlike SGD that selects partial gradients ∇f_{k_t} with k_t drawn uniformly at random from $\{0, 1, \ldots, M-1\}$, our analysis considers the case where k_t is deterministically selected in a sequential way, thus resulting in a deterministic convergence result.

[5]This architecture, which subsumes several existing deep NN models, proves convenient as it provides a finite-dimensional approximation of $\mathcal{L}_2$ without imposing any constraint on the parameter vector θ.

[6]Following [13], for each considered scenario, we tune the learning rate of each baseline optimizer to minimize the training loss $f(x_t)$, and we adopt default values for additional hyperparameters.



(a) Activation function: tanh. (b) Activation function: sigmoid. (c) Activation function: ReLU.

Fig. 1. Training curves of learned and hand-crafted optimizers; shaded areas and solid lines denote standard deviations and mean values, respectively.

Step $t = 20$	tanh	sigmoid	ReLU
Adam	$71.7 \pm 5.1\%$	$76.1 \pm 3.1\%$	$52.7 \pm 11.1\%$
SGD	$44.9 \pm 4.2\%$	$79.7 \pm 1.9\%$	$49.8 \pm 9.3\%$
NAG	$79.7 \pm 1.4\%$	$81.1 \pm 1.5\%$	$52.7 \pm 10.2\%$
RMSprop	$69.4 \pm 2.9\%$	$72.8 \pm 2.3\%$	$61.1 \pm 8.9\%$
ConvergentL2O	$\mathbf{87.0 \pm 0.5\%}$	$\mathbf{86.8 \pm 0.6\%}$	$86.3 \pm 0.6\%$
LSTM	$82.2 \pm 0.1\%$	$83.3 \pm 0.1\%$	$\mathbf{88.3 \pm 0.0\%}$

Step $t = 300$	tanh	sigmoid	ReLU
Adam	$\mathbf{89.5 \pm 0.5\%}$	$\mathbf{89.6 \pm 0.3\%}$	$70.3 \pm 12.2\%$
SGD	$87.4 \pm 0.4\%$	$89.3 \pm 0.3\%$	$80.6 \pm 8.1\%$
NAG	$89.4 \pm 0.2\%$	$89.4 \pm 0.2\%$	$82.2 \pm 7.6\%$
RMSprop	$87.6 \pm 2.1\%$	$88.5 \pm 0.4\%$	$81.5 \pm 7.5\%$
ConvergentL2O	$88.5 \pm 0.2\%$	$88.4 \pm 0.3\%$	$87.7 \pm 0.2\%$
LSTM	$81.4 \pm 0.0\%$	$81.4 \pm 0.0\%$	$\mathbf{88.3 \pm 0.0\%}$

Figure 1. Remarkably, our algorithm generalizes well also to the optimization landscape of the ReLU classifier, which may prove particularly challenging, as highlighted in [13], due to its structural difference with respect to tanh in (18). Future work will address the generalization of algorithms trained on the MNIST dataset to different test datasets, e.g., Fashion-MNIST; such generalization was not achieved using the current shallow classifier architecture (18).

To compare with alternative L2O approaches [13], we have trained for 200 epochs a two-layer Long Short-Term Memory (LSTM) optimizer $u_t = \mathrm{LSTM}(x_t, \nabla f(x_t), f(x_t))$. As shown in the table above, the LSTM optimizer achieves similar average test accuracy as our ConvergentL2O algorithm. Nonetheless, the LSTM output u_t does not vanish with time, causing the classifier parameters to diverge[7] – similar phenomena were also observed in [15]. None of our simulations exhibited such divergence as per Theorem 2.

V. Conclusion

In this paper, we have introduced a methodology for learning over all convergent update rules for smooth nonconvex optimization, thus enabling the automated synthesis of more reliable, efficient, and reconfigurable algorithms. By synergizing nonlinear system theory with the emerging L2O paradigm, we aimed to close the gap between offline, theory-based algorithm design and adaptable, example-driven approaches that are the hallmark of ML. Building on the proposed control-theoretic perspective we have embraced, further avenues for future research include certifying stronger convergence guarantees of learned algorithms for convex optimization, analyzing generalization capabilities, extending our framework to online and constrained optimization scenarios, and federated learning.

[7] We refer to https://github.com/andrea-martin/ConvergentL2O for the corresponding plots and the source code reproducing our numerical examples.

References

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

[2] Y. E. Nesterov, "A method of solving a convex programming problem with convergence rate $O\left(\frac{1}{k^2}\right)$," in *Doklady Akademii Nauk*, vol. 269, no. 3. Russian Academy of Sciences, 1983, pp. 543–547.

[3] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 2012, pp. 437–478.

[4] F. Dörfler, Z. He, G. Belgioioso, S. Bolognani, J. Lygeros, and M. Muehlebach, "Towards a systems theory of algorithms," *arXiv preprint arXiv:2401.14029*, 2024.

[5] L. Lessard, B. Recht, and A. Packard, "Analysis and design of optimization algorithms via integral quadratic constraints," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 57–95, 2016.

[6] L. Lessard, "The analysis of optimization algorithms: A dissipativity approach," *IEEE Control Systems Magazine*, vol. 42, no. 3, pp. 58–72, 2022.

[7] B. Goujaud, A. Dieuleveut, and A. Taylor, "On fundamental proof structures in first-order optimization," in *2023 62nd IEEE Conference on Decision and Control (CDC)*. IEEE, 2023, pp. 3023–3030.

[8] C. Scherer and C. Ebenbauer, "Convex synthesis of accelerated gradient algorithms," *SIAM Journal on Control and Optimization*, vol. 59, no. 6, pp. 4615–4645, 2021.

[9] X. Chen and E. Hazan, "Online control for meta-optimization," in *37-th Conference on Neural Information Processing Systems*, 2023.

[10] A. Hauswirth, S. Bolognani, G. Hug, and F. Dörfler, "Optimization algorithms as robust feedback controllers," *arXiv preprint arXiv:2103.11329*, 2021.

[11] G. Belgioioso, D. Liao-McPherson, M. H. de Badyn, S. Bolognani, R. S. Smith, J. Lygeros, and F. Dörfler, "Online feedback equilibrium seeking," *arXiv preprint arXiv:2210.12088*, 2022.

[12] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin, "Learning to optimize: A primer and a benchmark," *Journal of Machine Learning Research*, vol. 23, no. 189, pp. 1–59, 2022.

[13] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," *Advances in neural information processing systems*, vol. 29, 2016.

[14] K. Li and J. Malik, "Learning to optimize," in *International Conference on Learning Representations*, 2017.

[15] ——, "Learning to optimize neural nets," *arXiv preprint arXiv:1703.00441*, 2017.

[16] H. Heaton, X. Chen, Z. Wang, and W. Yin, "Safeguarded learned convex optimization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 6, 2023, pp. 7848–7855.

[17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[18] K.-K. K. Kim, E. R. Patrón, and R. D. Braatz, "Standard representation and unified stability analysis for dynamic artificial neural network models," *Neural Networks*, vol. 98, pp. 251–262, 2018.

[19] M. Revay, R. Wang, and I. R. Manchester, "Recurrent equilibrium networks: Flexible dynamic models with guaranteed stability and robustness," *IEEE Transactions on Automatic Control*, 2023.
[20] M. W. Fisher, G. Hug, and F. Dörfler, "Approximation by simple poles–part i: Density and geometric convergence rate in hardy space," *IEEE Transactions on Automatic Control*, 2023.
[21] D. P. Bertsekas and J. N. Tsitsiklis, "Gradient convergence in gradient methods with errors," *SIAM Journal on Optimization*, vol. 10, no. 3, pp. 627–642, 2000.
[22] L. Furieri, C. L. Galimberti, and G. Ferrari-Trecate, "Neural system level synthesis: Learning over all stabilizing policies for nonlinear systems," in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 2765–2770.

APPENDIX

Proof: [Lemma 1] Let $\nabla f(x_t) = \nabla_t$ for compactness. For any $t \in \mathbb{N}$ and $f \in \mathcal{S}_\beta$, it holds that $f(x_{t+1}) \leq f(x_t) + \nabla_t^\top (x_{t+1} - x_t) + \frac{\beta}{2}|x_{t+1} - x_t|^2$. Substituting $x_{t+1} - x_t = v_t - \eta \nabla_t$ according to (8) yields

$$f(x_{t+1}) \leq f(x_t) - \eta |\nabla_t|^2 + \nabla_t^\top v_t + \frac{\beta}{2}|v_t - \eta \nabla_t|^2 . \quad (19)$$

Observe that for any $a, b \in \mathbb{R}^d$ and any $\epsilon > 0$, we have that $a^\top b \leq |a||b| \leq \frac{|a|^2}{2\epsilon} + \frac{\epsilon |b|}{2}$ and $\frac{1}{2}|a-b|^2 \leq |a|^2 + |b|^2$ thanks to the Cauchy-Schwarz and Young's inequalities. Hence, we can upper-bound the right-hand side of (19) by $f(x_t) - \eta|\nabla_t|^2 + \frac{|\nabla_t|^2}{2\epsilon} + \frac{\epsilon |v_t|^2}{2} + \beta\left(|v_t|^2 + \eta^2 |\nabla_t|^2\right)$. Collecting terms and letting $\rho = 2\eta\epsilon(1-\beta\eta) - 1$, we obtain

$$\frac{\rho}{2\epsilon}|\nabla_t|^2 \leq f(x_t) - f(x_{t+1}) + \left(\frac{\epsilon}{2} + \beta\right)|v_t|^2 . \quad (20)$$

As $1 - \beta\eta > 0$, choosing any $\epsilon > \frac{1}{2\eta(1-\beta\eta)} > 0$ ensures that $\rho > 0$. Then, summing (20) with t ranging from 0 to $T \in \mathbb{N}$ and observing that the term $f(x_t) - f(x_{t+1})$ telescopes and that $f(x_T) \geq \inf_{x \in \mathbb{R}^d} f(x)$, we obtain $\sum_{t=0}^{T} |\nabla_t|^2 \leq \frac{2\epsilon}{\rho}(f(x_0) - \inf_{x\in\mathbb{R}^d} f(x)) + \frac{\epsilon}{\rho}(\epsilon + 2\beta)\sum_{t=0}^{T}|v_t|^2$. As f is bounded from below, $f(x_0) - \inf_{x\in\mathbb{R}^d} f(x)$ is finite and non-negative. Finally, as $\mathbf{v} \in \ell_2$, taking the limit of T to ∞ yields $\|\nabla f(\mathbf{x})\|^2 < \infty$, which concludes the proof. ∎

Proof: [Lemma 2] For any $f \in \mathcal{S}_\beta$ and any $\boldsymbol{\pi}(f, \mathbf{x})$ complying with (6c), select the operator $\mathbf{V}$ as

$$\mathbf{V}(\boldsymbol{\delta}^{x_0}) = \eta \nabla f(\mathbf{x}_{\boldsymbol{\pi}}) + \mathbf{u}_{\boldsymbol{\pi}} . \quad (21)$$

As $\boldsymbol{\pi}$ complies with (6c), we have $\nabla f(\mathbf{x}_{\boldsymbol{\pi}}) \in \ell_2$ and $\mathbf{u}_{\boldsymbol{\pi}} \in \ell_2$ for all $x_0 \in \mathbb{R}^d$. Hence, $\mathbf{V}(\boldsymbol{\delta}^{x_0}) \in \ell_2$ for every $x_0 \in \mathbb{R}^d$ and every $f \in \mathcal{S}_\beta$ as the sum of two signals in ℓ_2 lies in ℓ_2.

It remains to prove that (10) is equivalent to (9) for $\mathbf{V}$ chosen as per (21). We prove this by induction with a similar proof method as [22, Theorem 2]. For the closed-loop mapping (9), we define $\boldsymbol{\Psi} = (\boldsymbol{\Psi}^{\mathbf{x}}, \boldsymbol{\Psi}^{\mathbf{u}}, \boldsymbol{\Psi}^{\boldsymbol{\nabla}})$ such that $\boldsymbol{\Psi}^{\mathbf{x}}(f, \boldsymbol{\delta}^{x_0}) = \mathbf{x}_{\boldsymbol{\pi}}$, $\boldsymbol{\Psi}^{\mathbf{u}}(f, \boldsymbol{\delta}^{x_0}) = \mathbf{u}_{\boldsymbol{\pi}}$ and $\boldsymbol{\Psi}^{\boldsymbol{\nabla}}(f, \boldsymbol{\delta}^{x_0}) = \nabla f(\mathbf{x}_{\boldsymbol{\pi}})$. Similarly, for the closed-loop mapping (10) corresponding to $\mathbf{V}(\boldsymbol{\delta}^{x_0})$ in (21), we define $\boldsymbol{\Phi} = (\boldsymbol{\Phi}^{\mathbf{x}}, \boldsymbol{\Phi}^{\mathbf{u}}, \boldsymbol{\Phi}^{\boldsymbol{\nabla}})$ such that $\boldsymbol{\Phi}^{\mathbf{x}}(f, \boldsymbol{\delta}^{x_0}) = \mathbf{x}$, $\boldsymbol{\Phi}^{\mathbf{u}}(f, \boldsymbol{\delta}^{x_0}) = -\eta \nabla f(\mathbf{x}) + \mathbf{V}(\boldsymbol{\delta}^{x_0})$ and $\boldsymbol{\Phi}^{\boldsymbol{\nabla}}(f, \boldsymbol{\delta}^{x_0}) = \nabla f(\mathbf{x})$. For the inductive step, we assume that, for any $j \in \mathbb{N}$, we have $\Phi_i^u = \Psi_i^u$, $\Phi_i^x = \Psi_i^x$ and $\Phi_i^\nabla = \Psi_i^\nabla$ for all $i \in \mathbb{N}$ with $0 \leq i \leq j$. Note that (2) implies $\Phi_{j+1}^x = \Phi_j^x + \Phi_j^u + I$ and $\Psi_{j+1}^x = \Psi_j^x + \Psi_j^u + I$, which ensure that $\Phi_{j+1}^x = \Psi_{j+1}^x$ by inductive assumption. Hence, it follows that $\Phi_{j+1}^\nabla = \nabla f(\Phi_{j+1}^x) = \nabla f(\Psi_{j+1}^x) = \Psi_{j+1}^\nabla$. For $\mathbf{V}$ chosen as per (21), we also have $\Phi_{j+1}^u = -\eta \nabla f(\Phi_{j+1}^x) + \eta \nabla f(\Psi_{j+1}^x) + \Psi_{j+1}^u$, which simplifies to $\Phi_{j+1}^u = \Psi_{j+1}^u$. For the base case $j = 0$, we have $\Phi_0^x = \Psi_0^x = I$ by inspection of (3), and thus $\Phi_0^\nabla = \nabla f(\Phi_0^x) = \nabla f(\Psi_0^x) = \Psi_0^\nabla$. Last, we also have $\Phi_0^u = -\eta \nabla f(\Phi_0^x) + \eta \nabla f(\Psi_0^x) + \Psi_0^u = \Psi_0^u$. ∎

Proof: [Theorem 1] If $0 < \eta < \beta^{-1}$, any algorithm in the form $\boldsymbol{\pi}(f, \mathbf{x}) = -\eta \nabla f(\mathbf{x}) + \mathbf{v}$, with $\mathbf{v} \in \ell_2$, belongs to $\Sigma(f)$ for any $f \in \mathcal{S}_\beta$ by Lemma 1. Since $\mathbf{V} \in \mathcal{L}_2$ and $\boldsymbol{\delta}^{x_0} \in \ell_2$, we have that $\mathbf{v} = \mathbf{V}(\boldsymbol{\delta}^{x_0}) \in \ell_2$, that is, any $\mathbf{V} \in \mathcal{L}_2$ leads to a feasible solution of (6). Hence, (6c) becomes redundant, and can thus be removed from (11), after plugging the update rule (10) in (6b). Last, Lemma 2 ensures that for any $x_0 \in \mathbb{R}^d$, $f \in \mathcal{S}_\beta$, and $\boldsymbol{\pi}(f, \mathbf{x}) \in \Sigma(f)$, there exists $\mathbf{V} \in \mathcal{L}_2$ such that the update rule in (10) reproduces the trajectories of $\boldsymbol{\pi}(f, \mathbf{x})$. In turn, any feasible solution of (6) is considered in (11), implying equivalence. ∎

Proof: [Theorem 2] Rolling out (2) for M steps, starting from any t such that $t \bmod M = 0$, and using the update rule (16) with $\tau = \lfloor t/M \rfloor$, we obtain the iteration:

$$x_{t+M} = x_t - \eta_\tau \nabla f(x_t) + \eta_\tau w_t - \eta_\tau \sum\nolimits_{i=0}^{M-1} v_{t+i} , \quad (22)$$

where the term w_t, which represents an error in the gradient direction relative to the gradient iteration (8), is given by $w_t = \sum_{i=1}^{M-1} \nabla f_i(x_t) - \nabla f_i(x_{t+i})$. Under the assumptions of Lemma 2, we have that

$$\left|\sum\nolimits_{i=0}^{M-1} v_{t+i}\right| \leq \sum\nolimits_{i=0}^{M-1} |v_{t+i}| \leq \eta_\tau M (C + sD|\nabla f(x_t)|); \quad (23)$$

we now proceed to show that a similar bound also holds for $|w_t|$. By the triangle inequality and observing that $f_i \in \mathcal{S}_\beta$ ensures that ∇f_i is β-Lipschitz continuous, we have that $|w_t| \leq \sum_{i=1}^{M-1} |\nabla f_i(x_t) - \nabla f_i(x_{t+i})| \leq \beta \sum_{i=1}^{M-1} |x_t - x_{t+i}|$, which further implies:

$$|w_t| \leq \eta_\tau \beta |\nabla f_0(x_t) + v_t| + \beta \sum\nolimits_{i=2}^{M-1} |x_t - x_{t+i}| , \quad (24)$$

where the term $\sum_{i=2}^{M-1} |x_t - x_{t+i}|$ is given by:

$$\eta_\tau \sum\nolimits_{i=2}^{M-1} \left| \nabla f_0(x_t) + v_t s + \sum\nolimits_{j=1}^{i-1} \nabla f_j(x_{t+j}) + v_{t+j} \right| . \quad (25)$$

Then, we observe that (25) can be upper-bounded by $\eta_\tau (M-2)|\nabla f_0(x_t) + v_t| + \eta_\tau \sum_{i=2}^{M-1} \sum_{j=1}^{i-1} |\nabla f_j(x_{t+j}) + v_{t+j}| \leq \eta_\tau (M-2)\Big(|\nabla f_0(x_t) + v_t| + \sum_{i=1}^{M-2} |\nabla f_i(x_{t+i}) + v_{t+i}|\Big)$. By combining these inequality with (24), we deduce that $\eta_\tau \beta (M-1)\Big(|\nabla f_0(x_t)| + \sum_{i=0}^{M-2} |v_{t+i}| + \sum_{i=1}^{M-2} |\nabla f_i(x_{t+i})|\Big)$ upper-bounds $|w_t|$. Moreover, for any $i = 1, \ldots, M-2$, it holds that $|\nabla f_i(x_{t+i})| = |\nabla f_i(x_{t+i-1} - \eta_\tau (\nabla f_{i-1}(x_{t+i-1}) + v_{t+i-1}))| \leq |\nabla f_i(x_{t+i-1})| + \eta_\tau \beta |\nabla f_{i-1}(x_{t+i-1})| + \eta_\tau \beta |v_{t+i-1}|$. By iterating the reasoning above, we have that $|\nabla f_i(x_{t+i})|$ is upper-bounded by $\sum_{j=0}^{i} E|\nabla f_j(x_t)| + F|v_{t+j}|$ for appropriately defined positive constant E and F. Finally, leveraging (14) and (23), we conclude that there exists positive constant P and Q such that $\left|w_t - \sum_{i=0}^{M-1} v_{t+i}\right| \leq \eta_\tau (P + Q|\nabla f(x_t)|)$. Having established this upper bound, the results of Lemma 2 follow from applying [21, Proposition 1] to the recursion (22), since f is bounded from below. ∎