MARTHE: Scheduling the Learning Rate Via Online Hypergradients

Michele Donini[1*] , **Luca Franceschi**[2,3†] , **Orchid Majumder**[1] ,
Massimiliano Pontil[2,3] and **Paolo Frasconi**[4]

[1]Amazon
[2]University College London, London, UK
[3]Istituto Italiano di Tecnologia, Genova, Italy
[4]Università di Firenze, Firenze, Italy
donini@amazon.com; luca.franceschi@iit.it

Abstract

We study the problem of fitting task-specific learning rate schedules from the perspective of hyperparameter optimization, aiming at good generalization. We describe the structure of the gradient of a validation error w.r.t. the learning rate schedule – the hypergradient. Based on this, we introduce MARTHE, a novel online algorithm guided by cheap approximations of the hypergradient that uses past information from the optimization trajectory to simulate future behaviour. It interpolates between two recent techniques, RTHO [Franceschi *et al.*, 2017] and HD [Baydin *et al.*, 2018], and is able to produce learning rate schedules that are more stable leading to models that generalize better.

1 Introduction

Learning rate (LR) adaptation for first-order optimization methods is one of the most widely studied aspects in optimization for learning methods, in particular neural networks. Recent research in this area has focused on developing complex schedules that depend on a small number of hyperparameters [Loshchilov and Hutter, 2017; Orabona and Pál, 2016] or proposed methods to optimize LR schedules w.r.t. the training objective [Schaul *et al.*, 2013; Baydin *et al.*, 2018; Wu *et al.*, 2018]. While quick optimization is desirable, the true goal of supervised learning is to minimize the generalization error, which is commonly estimated by holding out part of the available data for validation. Hyperparameter optimization (HPO), a related but distinct branch of the literature, specifically focuses on this aspect, with less emphasis on the goal of rapid convergence on a single task. Research in this direction is vast and includes model-based, model-free, and gradient-based approaches (see [Hutter *et al.*, 2019] for an overview). Additionally, works in the area of learning to optimize [Andrychowicz *et al.*, 2016; Wichrowska *et al.*, 2017] have focused on the problem of tuning parameterized optimizers on whole classes of learning problems but require prior expensive optimization and are not designed to speed up training on a single task.

The goal of this paper is to automatically compute *in an online fashion* a learning rate schedule for stochastic optimization methods (such as SGD) only on the basis of the given learning task, aiming at producing models with associated small validation error. We study the problem of finding a LR schedule under the framework of gradient-based hyperparameter optimization [Franceschi *et al.*, 2017]: we consider an optimal schedule $\eta^* = (\eta_0^*, \ldots, \eta_{T-1}^*) \in \mathbb{R}_+^T$ as a solution to the following constrained optimization problem

$$\min\{f_T(\eta) = E(w_T(\eta)) : \eta \in \mathbb{R}_+^T\} \qquad (1)$$

$$\text{s.t.} \quad w_0 = \bar{w}, \quad w_{t+1}(\eta) = \Phi_t(w_t(\eta), \eta_t)$$

for $t = \{0, \ldots, T-1\} = [T]$, where $E : \mathbb{R}^d \to \mathbb{R}_+$ is an objective function, $\Phi_t : \mathbb{R}^d \times \mathbb{R}_+ \to \mathbb{R}^d$ is a (possibly stochastic) weight update dynamics, $\bar{w} \in \mathbb{R}^d$ represents the initial model weights (parameters) and finally w_t are the weights after t iterations. We can think of E as either the training or the validation loss of the model, while the dynamics Φ describe the update rule (such as SGD, SGD-Momentum, Adam etc.). For example in the case of SGD, $\Phi_t(w_t, \eta_t) = w_t - \eta_t \nabla L_t(w_t)$, with $L_t(w_t)$ the training loss on the t-th minibatch. The *horizon* T should be large enough so that the training error can be effectively minimized to avoid underfitting. A too large value of T does not necessarily harm since $\eta_k = 0$ for $k > \bar{T}$ is still a feasible solution, implementing early stopping in this setting. There is vast empirical evidence that the LR is among the most critical hyperparameters affecting the performances of learnt statistical models. Beside convergence arguments from the stochastic optimization literature, for all but the simplest problems, non-constant schedules yield generally better results [Bengio, 2012].

Problem (1) can be in principle solved by any HPO technique. However, most HPO techniques, including those based on hypergradients [Maclaurin *et al.*, 2015] or on a bilevel programming formulation [Franceschi *et al.*, 2018; MacKay *et al.*, 2019] would not be suitable for the present purpose since they require multiple evaluations of f (which, in turn, require executions of the weight optimization routine). This clearly defeats one of the main goals of determining LR schedules, i.e. speed. In fact, several other researchers [Almeida *et al.*, 1999; Schraudolph, 1999; Schaul *et al.*, 2013; Franceschi *et al.*, 2017; Baydin *et al.*, 2018; Wu *et al.*, 2018] have investigated related solutions for deriving greedy update rules for the learning rate. A common

---

*Contact Author, Equal Contribution.
†Contact Author, Equal Contribution.

characteristic of methods in this family is that the LR update rule does not take into account information from the future. At a high level, we argue that any method should attempt to produce updates that approximate the true and computationally unaffordable hypergradient of the *final* objective with respect to the current learning rate. In relation to this, [Wu *et al.*, 2018] discuss the bias deriving from greedy or short-horizon optimization. In practice, different methods resort to different approximations or explicitly consider greedily minimizing the performance after a single parameter update [Almeida *et al.*, 1999; Schaul *et al.*, 2013; Baydin *et al.*, 2018]. The type of approximation and the type of objective (i.e. the training or the validation loss) are in principle separate issues although comparative experiments with both objectives and the same approximation are never reported in the literature. The validation loss is only used in experiments reported by Franceschi *et al.* [2017].

In this work, we make a step forward in understanding the behavior of online gradient-based hyperparameter optimization techniques by (i) analyzing in Section 2 the structure of the true hypergradient that could be used to solve Problem (1) if wall-clock time was not a concern, and (ii) by studying in Section 3 some failure modes of previously proposed methods along with a detailed discussion of the type of approximations that these methods exploit. In Section 4, based on these considerations, we develop a new hypergradient-based algorithm which reliably produces competitive learning rate schedules aimed at lowering the final validation error. The algorithm, which we call MARTHE (Moving Average Real-Time Hyperparameter Estimation), has a moderate computational cost and can be interpreted as a generalization of the algorithms described by Baydin *et al.* [2018] and Franceschi *et al.* [2017]. In Section 5, we empirically compare the quality of different hypergradient approximations in a small scale task where true hypergradient can be exactly computed. In Section 6, we present a set of real world experiments showing the validity of our approach. We finally discuss potential future applications and research directions in Section 7.

2 Structure of the Hypergradient

We study the optimization problem (1) under the perspective of gradient-based hyperparameter optimization, where the learning rate schedule $\eta = (\eta_0, \ldots, \eta_{T-1})$ is treated as a vector of hyperparameters and T is a fixed horizon. Since the learning rates are positive real-valued variables, assuming both E and Φ are smooth functions, we can compute the gradient of $f \in \mathbb{R}^T$, which is given by

$$\nabla f_T(\eta) = \dot{w}_T^{\mathsf{T}} \nabla E(w_T), \quad \dot{w}_T = \frac{\mathrm{d} w_T}{\mathrm{d} \eta} \in \mathbb{R}^{d \times T}, \quad (2)$$

where "T" means transpose. The total derivative $\dot{w}_T$ can be computed iteratively with forward-mode algorithmic differentiation [Griewank and Walther, 2008; Franceschi *et al.*, 2017] as

$$\dot{w}_0 = 0, \quad \dot{w}_{t+1} = A_t \dot{w}_t + B_t, \quad (3)$$

$$\text{with} \quad A_t = \frac{\partial \Phi_t(w_t, \eta_t)}{\partial w_t}, \quad B_t = \frac{\partial \Phi_t(w_t, \eta_t)}{\partial \eta}. \quad (4)$$

The Jacobian matrices A_t and B_t depend on w_t and η_t, but we will leave these dependencies implicit to ease our notation. In the case of SGD[1], $A_t = I - \eta_t H_t(w_t)$ and $[B_t]_j = -\delta_{tj} \nabla L_t(w_t)^{\mathsf{T}}$, where subscripts denote columns (starting from 0), $\delta_{tj} = 1$ if $t = j$ and 0 otherwise and H_t is the Hessian of the training error on the t−th mini-batch.

Given the high dimensionality of η, reverse-mode differentiation would result in a more efficient (running-time) implementation. We use here forward-mode both because it is easier to interpret and because it is closely related to the computational scheme behind MARTHE, as we will show in Section 4. We note that stochastic approximations of Eq. (2) may be obtained with randomized telescoping sums [Beatson and Adams, 2019] or hyper-networks based stochastic approximations [MacKay *et al.*, 2019].

Eq. (3) describes the so-called tangent system [Griewank and Walther, 2008] which is a discrete affine time-variant dynamical system that measures how the parameters of the model would change for infinitesimal variations of the learning rate schedule, after t iterations of the optimization dynamics. Notice that the "translation matrices" B_t are very sparse, having, at any iteration, only one non-zero column. This means that $[\dot{w}_t]_j$ remains 0 for all $j \geq t$: η_t affects only the future parameters trajectory. Finally, for a single learning rate η_t, the derivative (a scalar) is

$$\frac{\partial f_T(\eta)}{\partial \eta_t} = [\nabla f_T(\eta)]_t = \left[\left(\prod_{s=t+1}^{T-1} A_s \right) B_t \right]_t^{\mathsf{T}} \nabla E(w_T) \quad (5)$$

$$= -\nabla L_t(w_t)^{\mathsf{T}} P_{t+1}^{T-1} \nabla E(w_T), \quad (6)$$

where the last equality holds true for SGD. Eq. (5) can be read as the scalar product between the gradients of the training error at the t-th step and the objective E at the final iterate, *transformed by* the accumulated (transposed) Jacobians of the optimization dynamics, shorthanded by P_{t+1}^{T-1} in (6). As it is apparent from Eq. (5), given w_t, the hypergradient of η_t is affected only by the future trajectory and does not depend explicitly on η_t.

In its original form, where each learning rate is left free to take any permitted value, Problem (1) presents a highly nonlinear setup. Although in principle it could be tackled by a projected gradient descent method, in practice this is not feasible even for relatively small problems: evaluating the gradient with forward-mode is inefficient in time since it requires maintaining a large matrix tangent system. Evaluating it with reverse-mode is inefficient in memory since the entire weight trajectory $(w_i)_{i=0}^T$ should be stored[2]. Furthermore, it can be expected that several updates of η are necessary to reach convergence – each update requiring the computation of f_T and the entire parameter trajectory in the weight space. Since this

---

[1]Throughout we use SGD to simplify the discussion, however, similar arguments hold for any smooth optimization dynamics such as those including momentum terms.

[2]Techniques based on implicit differentiation [Pedregosa, 2016; Agarwal *et al.*, 2017] or fixed-point equations [Griewank and Faure, 2002] (also known as recurrent backpropagation [Pineda, 1988]) cannot be readily applied to compute ∇f_T since the training loss L does not depend explicitly on η.

approach is computationally very expensive, we turn out attention to online updates where η_t is required to be updated online based only on trajectory information up to time t.

3 Online Gradient-Based Adaptive Schedules

Before developing and motivating our proposed technique, we discuss two previous methods to compute the learning rate schedule online. The real-time hyperparameter optimization (RTHO) algorithm suggested in [Franceschi *et al.*, 2017], reminiscent of stochastic meta-descent [Schraudolph, 1999], is based on forward-mode differentiation and uses information from the entire weight trajectory by accumulating partial hypergradients. Hypergradient descent (HD), proposed in [Baydin *et al.*, 2018] and closely related to the earlier work by Almeida *et al.* [1999], aims at minimizing the loss with respect to the learning rate after one step of optimization. It uses information only from the past and current iterate.

Both methods implement an update rules of the type

$$\eta_t = \max\left[\eta_{t-1} - \beta \Delta\eta_t, 0\right], \tag{7}$$

where $\Delta\eta_t$ is an online estimate of the hypergradient, $\beta > 0$ is a step-size or *hyper-learning rate* and the $\max$ ensures positivity[3]. To ease the discussion, we omit the stochastic (mini-batch) evaluation of the training error L and possibly of the objective E.

The update rules[4] are given by

$$\Delta^{\mathrm{RTHO}}\eta_t = \left[\sum_{i=0}^{t-1} P_{i+1}^{t-1} B_i\right]^\intercal \nabla E(w_t); \tag{8}$$

$$\Delta^{\mathrm{HD}}\eta_t = B_{t-1}^\intercal \nabla E(w_t) \tag{9}$$

for RTHO and HD respectively, where $P_t^{t-1} := I$. Thus $\Delta^{\mathrm{RTHO}} = \Delta^{\mathrm{HD}} + r((w_i, \eta_i)_{i=0}^{t-2})$: the correction term r can be interpreted as an "on-trajectory approximations" of longer horizon objectives as we will discuss in Section 4.

Although successful in some learning scenarios, we argue that both these update rules suffer from (different) pathological behaviors, as HD may be "shortsighted", being prone to underestimate the learning rate (as noted by Wu *et al.* [2018]), while RTHO may be too slow to adapt to sudden changes of the loss surface or, worse, it may be unstable, with updates growing uncontrollably in magnitude. We exemplify these behaviors in Figure 1, using two bidimensional test functions[5] from the optimization literature, where we set $E = L$ and we perform 500 steps of gradient descent. The Beale function,

on the left, presents sharp peaks and large plateaus. RTHO consistently outperforms HD for all probed values of β that do not lead to divergence (Figure 1 upper center). This can be easily explained by the fact that in flat regions gradients are small in magnitude, leading to $\Delta^{\mathrm{HD}}\eta_t$ to be small as well. RTHO, on the other hand, by accumulating all available partial hypergradients and exploiting second order information, is capable of making faster progress. We use a simplified and smoothed version of the Bukin function N.6 to show the opposite scenario (Figure 1 lower center and right). Once the optimization trajectory closes the valley of minimizers $y = 0.01x$, RTHO fails to discount outdated information, bringing the learning rate first to grow exponentially, and then to suddenly vanish to 0, as the gradient changes direction. HD, on the other hand, correctly damps η and is able to maintain the trajectory close to the valley.

These considerations suggest that neither Δ^{RTHO} nor Δ^{HD} provide *globally useful* update directions, as large plateaus and sudden changes on the loss surface are common features of the optimization landscape of neural networks [Bengio *et al.*, 1994; Glorot and Bengio, 2010]. Our proposed algorithm smoothly interpolates between these two methods, as we will present next.

4 MARTHE

In this section, we develop and motivate MARTHE, an algorithm for computing LR schedules online during a single training run. This method maintains a moving-average over approximations of (5) of increasingly longer horizon, using the past trajectory and gradients to retain a low computational overhead. Further, we show that RTHO [Franceschi *et al.*, 2017] and HD [Baydin *et al.*, 2018] outlined above, can be interpreted as special cases of MARTHE, shedding further light on their behaviour and shortcomings.

Shorter horizon auxiliary objectives. For $K > 0$, define $g_K(u, \xi)$, with $\xi \in \mathbb{R}_+^K$ as

$$g_K(u, \xi) = E(u_K(\xi)) \quad \text{s.t.} \quad u_0 = u, \tag{10}$$

$$u_{i+1} = \Phi(u_i, \xi_i) \quad \text{for} \quad i = [K]. \tag{11}$$

The g_Ks define a class of shorter horizon objective functions, indexed by K, which correspond to the evaluation of E after K steps of optimization, starting from $u \in \mathbb{R}^d$ and using ξ as the LR schedule[6]. Now, the derivative of g_K with respect to ξ_0, denoted g_K', is given by

$$g_K'(u, \xi) = \frac{\partial g_K(u, \xi)}{\partial \xi_0} = [B_0]_0^\intercal P_1^{K-1} \nabla E(u_K) \tag{12}$$

$$= -\nabla L(u)^\intercal P_1^{K-1} \nabla E(u_K), \tag{13}$$

where the last equality holds for SGD dynamics. Once computed on subsets of the original optimization dynamics $(w_i)_{i=0}^T$, the derivative reduces for $K = 1$ to $g_1'(w_t, \eta_t) = -\nabla E(w_{t+1}) \nabla L(w_t)^\intercal$ (for SGD dynamics), and for $K = T - t$ to $g_{T-t}'(w_t, (\eta_i)_{i=t}^{T-1}) = [\nabla f(\eta)]_t$. Intermediate values of K yield cheaper, shorter horizon approximations of (5).

---

[3]Updates could be also considered in the logarithmic space, e.g. by Schraudolph [1999]; we find it useful, to let η reach 0 whenever needed, offering a natural way to implement early stopping.

[4]In [Franceschi *et al.*, 2017], the hyperparameter is updated every K iterations. Here we focus on the case $K = 1$ which better allows for a unifying treatment. HD is developed using as objective the training loss L rather than the validation loss E. We consider here without loss of generality the case of optimizing E.

[5]We use the Beale function defined as $L(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$ and a simplified smoothed version of Buking N.6: $L(x, y) = \sqrt{((y - 0.01x)^2 + \varepsilon)^{1/2} + \varepsilon}$, with $\varepsilon > 0$.

[6]Formally, ξ and u are different from η and w from the previous sections; later, however, we will evaluate the g_K's on subsequences of the optimization trajectory.
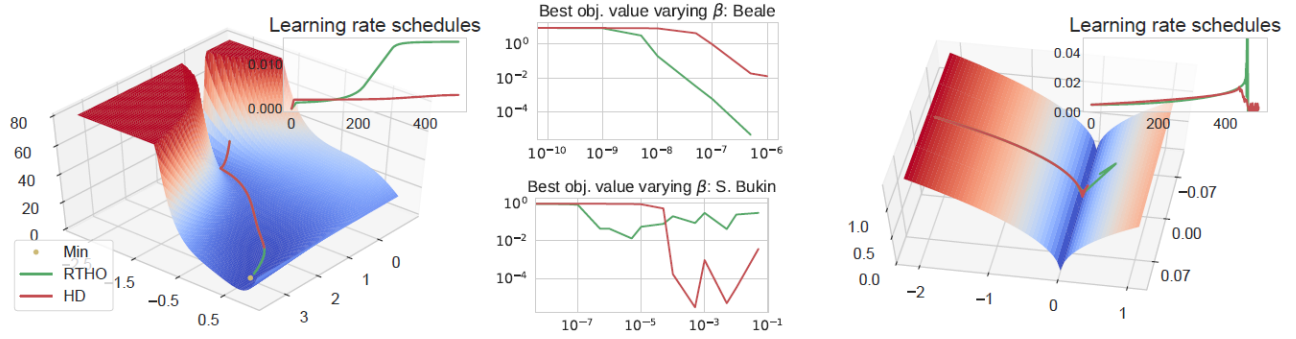
Figure 1: Loss surface and trajectories for 500 steps of gradient descent with HD and RTHO for Beale function (left) and (smoothed and simplified) Bukin N.6 (right). Center: best objective value reached within 500 iterations for various values of β that do not lead to divergence.

Approximating the future trajectory with the past. Explicitly using any of the approximations given by $g'_K(w_t, \eta)$ as $\Delta \eta_t$ is, however, still largely impractical, especially for $K \gg 1$. Indeed, it would be necessary to iterate the map Φ for K steps in the future, with the resulting $(w_{t+i})_{i=1}^K$ iterations discarded after a single update of the learning rate. For $K \in [t]$, we may then consider evaluating g'_K *exactly K steps in the past*, that is evaluating $g'_K(w_{t-K}, (\eta_i)_{i=t-K}^{t-1})$. Selecting $K = 1$ is indeed equivalent to Δ^{HD}, which is computationally inexpensive. However, when past iterates are close to future ones (such as in the case of large plateaus), using larger K's would allow in principle to capture longer horizon dependencies present in the hypergradient structure of Eq. (5). Unfortunately the computational efficiency of $K = 1$ does not generalize to $K > 1$, since setting $\Delta \eta_t = g'_K$ would require maintaining K different tangent systems.

Discounted accumulation of g'_ks. The definition of the g_Ks, however, allows one to highlight the recursive nature of the *accumulation* of g'_K. Indeed, by maintaining the vector tangent system,

$$Z_0 = [B_0(u_0, \xi_0)]_0 \tag{14}$$
$$Z_{i+1} = \mu A_i(u_i, \xi_i) Z_i + [B_i(u_i, \xi_i)]_i \quad \text{for } i \geq 0, \tag{15}$$

with $Z_i \in \mathbb{R}^d$, computing the moving average

$$S_{K,\mu}(u, \xi) = \sum_{i=0}^{K-1} \mu^{K-1-i} g'_{K-i}(u_i, (\xi_j)_{j=i}^{K-1}) = Z_K^{\mathsf{T}} \nabla E(u_K)$$

from S_{K-1} requires only updating (14) and recomputing the gradient of E. The total cost of this operation is $O(c(\Phi))$ per step both in time and memory using fast Jacobians vector products [Pearlmutter, 1994] where $c(\Phi)$ is the cost of computing the optimization dynamics (typically $c(\Phi) = O(d)$). The parameter $\mu \in [0, 1]$ allows to control how quickly past history is forgotten. One can notice that $\Delta^{\mathrm{RTHO}} \eta_t = S_{t,1}(w_0, (\eta_j)_{i=0}^{t-1})$, while $\mu = 0$ recovers $\Delta^{\mathrm{HD}} \eta_t$. Values of $\mu < 1$ help discounting outdated information, while as μ increases so does the horizon of the hypergradient approximations. The computational scheme of Eq. (14) is quite similar to that of forward-mode algorithmic differentiation for computing $\dot{w}$ (see Section 2 and Eq. (3)); however, the "tangent system" in Eq. (14), exploiting the sparsity of the matrices B_t, only keeps track of the variations with respect to the first component ξ_0, drastically reducing the running time.

Algorithm 1 MARTHE; requires $\beta, \mu, \eta_0[= 0]$

Initialization of w and $Z_0 \leftarrow 0$
for $t = 0$ **to** T **do**
  $\eta_t \leftarrow \max[\eta_{t-1} - \beta \Delta \eta_t, 0]$ {Update LR if $t > 0$}
  $Z_{t+1} \leftarrow \mu A_t(w_t, \eta_t) Z_t + [B_t(w_t, \eta_t)]_t$ {Eq. (15)}
  $w_{t+1} \leftarrow \Phi_t(w_t, \eta_t)$ {Parameter update}
end for

Algorithm 1 presents the pseudocode of MARTHE. The runtime and memory requirements of the algorithm are dominated by the computation of the variables Z. Being these structurally identical to the tangent propagation of forward mode algorithmic differentiation, we conclude that the runtime complexity is only a multiplicative factor higher than that of the underlying optimization dynamics Φ and requires two times the memory (see Griewank and Walther [2008], Sec. 4).

5 Optimized Schedules and Quality of MARTHE Approximations

In this section, we empirically compare the optimized LR schedules found by approximately solving Problem 1 by gradient descent (denoted LRS-OPT), where the hypergradient is given by Eq. (5), against those generated by MARTHE, for a wide range of hyper-momentum factors μ (including HD and RTHO) and hyper-learning rates β. We are interested in understanding and visualizing the qualitative similarities among the schedules, as well as the effect of μ and β on the final performance measure. To this end, we trained three-layers feed forward neural networks with 500 hidden units per layer on a subset of 7000 MNIST [LeCun *et al.*, 1998] images. We used a cross-entropy loss and SGD as optimization dynamics Φ, with a mini-batch size of 100. We further sampled 700 images to form the validation set and defined E to be the validation loss after $T = 512$ optimization steps (about 7 epochs). For LRS-OPT, we randomly generated different mini-batches at each iteration, to prevent the schedule from unnaturally adapting to a specific progression of mini-batches[7]. We initialized $\eta = 0.01 \cdot \mathbf{1}_{512}$ for LRS-OPT and set

---

[7]We retained, however, the random initialization of the network weights, to account for the impact that this may have on the initial part of the trajectory. This offers a fairer comparison between LRS-OPT and online methods, which compute the trajectory only once.
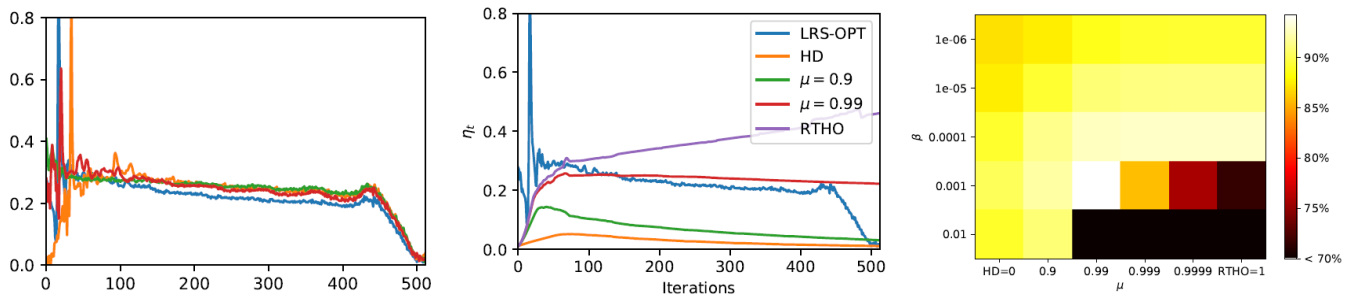
Figure 2: Left: schedules found by LRS-OPT (after 500 iterations of SGD) on 4 different random seeds. Center: comparison between optimized and MARTHE schedules for one seed, for indicative values of μ. We report the schedule generated with the hyper-learning rate β that achieves the best final validation accuracy. Right: Average validation accuracy of MARTHE over 20 random seeds, for various values of β and μ. The best performance of 94.2% is obtained with $\mu = 0.99$. For reference, the average validation accuracy of the network trained with $\eta = 0.01 \cdot \mathbf{1}_{512}$ is 87.5%, while LRS-OPT obtains an average accuracy of 96.1%. For $\mu \in [0.9, 1)$, when MARTHE converges it consistently outperforms HD and it performs at least as well as RTHO, but converges for a wider range of β.

$\eta_0 = 0.01$ for MARTHE, and repeated the experiments for 4 random seeds.

Figure 2 (left) shows the LRS-OPT schedules found after 5000 iterations of gradient descent: the plot reveals a strong initialization (random seed) specific behavior of η^* for approximately the first 100 steps. The LR schedule then stabilizes or slowly decreases up until around 50 iterations before the final time, at which point it quickly decreases (recall that, in this setting, all η_i, including η_0, are optimized "independently" and may take any permitted value). Figure 2 (center) present a qualitative comparison between the offline LRS-OPT schedule and the online ones, for indicative values of μ. Too small values of μ result in an underestimation of the learning rates, with generated schedules that quickly decay to very small values – this is in line with what observed in [Wu *et al.*, 2018]. For too high values of μ ($\mu = 1$ i.e. RTHO [Franceschi *et al.*, 2017] in the figure) the schedules linger or fail to decrease, possibly causing instability and divergence. For certain values of μ, the schedules computed by MARTHE seems to capture the general behaviour of the optimized ones. Finally, Figure 2 (right) shows the average validation accuracy (rather than loss, for easier interpretation) of MARTHE methods varying β and μ. Higher values of μ translate to higher final performances – with a clear jump occurring between $\mu = 0.9$ and $\mu = 0.99$ – but may require a smaller hyper learning rate to prevent divergence.

6 Experiments

We performed an extensive set of experiments in order to compare MARTHE, RTHO, and HD. We also considered a classic LR scheduling baseline in the form of exponential decay (Exponential) where the LR schedule is defined by $\eta_t = \eta_0 \gamma^t$. The purpose of these experiments is to perform a thorough comparison of various learning-rate scheduling methods, with a focus on those that are (hyper-)gradient based, in the fairest possible manner: indeed, these methods have very different running-time per iteration – HD and Exponential being much faster than MARTHE and RTHO – as well as different configuration spaces. It would be unfair to compare them using the number of iterations as computational budget. We therefore designed an experimental setup that al-

lowed us to account for it: we implemented a random search strategy over the respective algorithms' configuration spaces and early-stopped each run with a 10-epochs patience window. We repeatedly drew configurations parameters (hyper-hyperparameters) and run respective experiments until a fixed time budget of 36 hours was reached. The proposed experimental setting tries to mimic how machine learning practitioners may approach the parameter-tuning problem.

We used two alternative optimization dynamics: SGDM with the momentum hyperparameter fixed to 0.9 and Adam with the commonly suggested default values $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We fixed the batch size to 128, the initial learning rate $\eta_0 = 0.1$ for SGDM and 0.003 for Adam, and the weight decay (i.e. 2-norm) to $5 \cdot 10^{-4}$. For the adaptive methods, we sampled β in $[10^{-3}, 10^{-6}]$ log-uniformly, and for our method, we sampled μ between 0.9 and 0.999. Finally, we picked the decay factor γ for Exponential log-uniformly in $[0.9, 1.0]$.

In our first set of experiments, we used VGG-11 [Simonyan and Zisserman, 2014] with batch normalization [Ioffe and Szegedy, 2015] after the convolutional layers, on CIFAR10 [Krizhevsky and Hinton, 2009], and SGDM as the inner optimizer. In the second set of experiments, we used ResNet-18 [He *et al.*, 2016] on CIFAR100 [Krizhevsky and Hinton, 2009], in this case with Adam. For both CIFAR10 and CIFAR100, we used 45000 images as training images and 5000 images as the validation dataset. An additional set of 10000 test images was finally used to estimate generalization accuracy. We used standard data augmentation, including mean-std normalization, random crops and horizontal flips. Gradients were clipped to an absolute value of 100.0.

We kept track of the model with the best validation accuracy found so far, reporting in Figure 3 (left and center) the relative mean test accuracy (solid line) and minimum and maximum (limits of the shaded regions) across 5 repetitions. Inspecting the figure, it is possible to identify which method is the best performing one for any give time budget, both in average and in the worst/best case scenario. Figures 3 (right) shows examples of the LR schedules obtained by using the different methods.

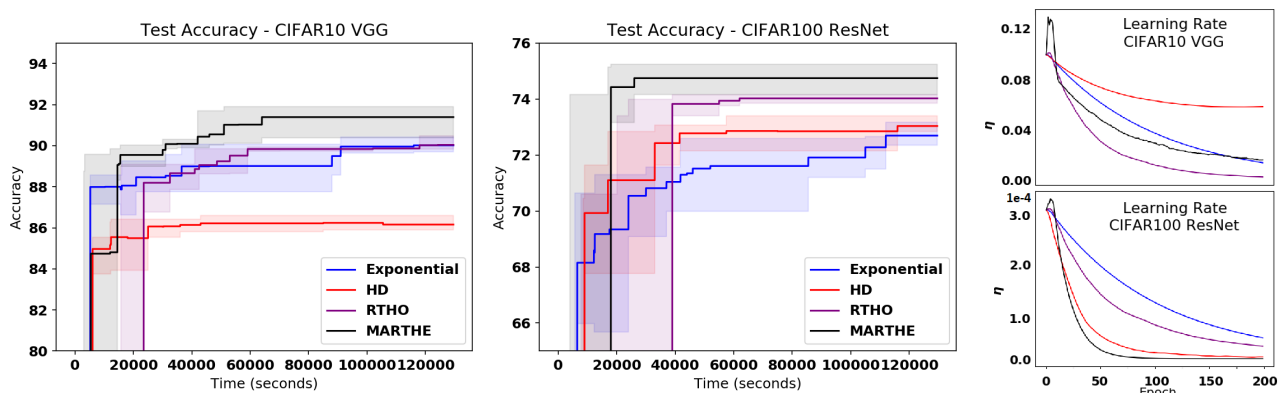We performed all experiments using AWS P3.2XL in-

Figure 3: Left and center: we randomly draw parameters from each algorithm's configuration space (hyper-hyperparameters) and run the resulting experiments using early stopping with a patience window of 10 epochs. We keep track of the best model (i.e. the model with the highest validation accuracy) found so far and we report the relative test accuracy as a function of time. The solid line represents average accuracy, while shaded regions depict minimum and maximum accuracy across different seeds. On the left we show results for the experiments with VGG networks trained on CIFAR10 with SGDM as inner optimization method. The center plot reports experiments with ResNet models trained on CIFAR100 with Adam as optimization dynamics. Right: samples of learning rate schedules that lead to the best found model for each scheduling method and for the relative seed. Experiments on CIFAR10 (top) and CIFAR100 (bottom).

stances, each providing one NVIDIA Tesla V100 GPU. Finally, our PyTorch implementation of the methods and the experimental framework to reproduce the results is available at https://github.com/awslabs/adatune.

6.1 Discussion

In the analysis of our results, we will mainly focus on the accuracy on the test dataset achieved by different methods within a fixed time budget. For all the experiments, results summarized in Figure 3 show that both Exponential and HD were able to obtain a reasonably good accuracy within the first 4 hours, while RTHO and MARTHE required 6 hours at least to reach the same level of accuracy. This is due to the fact that the wall-clock time required to process a single minibatch is different: MARTHE takes approximately 4 times the wall-clock time of HD; there is negligible wall-clock time difference between MARTHE and RTHO or between HD and Exponential. MARTHE was able to surpass all the other methods consistently after 10 hours.

Our experimental protocol resulted in HD and Exponential getting more trials compared to RTHO and MARTHE (in average around 24 trials for the first two compared to 8 of RTHO and MARTHE). Despite the fact that MARTHE could only afford fewer trials, it could still achieve better performance, suggesting that it is able to produce better learning rate schedules more reliably. Moreover, MARTHE maintains a better peak accuracy compared to RTHO showing the effectiveness of down-weighting outdated information.

Our experimental setup helped us investigate the robustness of the methods with respect to the choice of the hyper-hyperparameters[8]. To that end, we can see from Figure 3 (left) that the average and worst case test set accuracies (measured across multiple seeds) of MARTHE are better in comparison to the other methods. This is a strong indication that

MARTHE demonstrated superior adaptability with respect to different hyper-hyperparameters and seeds compared to other methods. This is also reflected by the result that MARTHE outperforms other strategies on a consistent basis if given a sufficient time budget (4-6 hours in our experiments): the higher computational cost of MARTHE is outbalanced by the fact that it needs fewer trials to reach the same performance level of faster methods like Exponential or HD.

Overall, our experiments reveal that RTHO and MARTHE provide better performance, giving a clear indication of the importance of the past information. Due to its lower computational overhead, Exponential should be still preferred under tight budget constraints, while MARTHE with $\mu \in [0.9, 0.999]$ should be preferred if enough time is available.

7 Conclusion

Finding a good learning rate schedule is an old but crucially important issue in machine learning. This paper makes a step forward, analyzing previously proposed online gradient-based methods and introducing a more general technique to obtain performing LR schedules based on an increasingly long moving average over hypergradient approximations. MARTHE interpolates between HD and RTHO, and its implementation is fairly simple within modern automatic differentiation, adding only a moderate computational overhead over the underlying optimizer complexity.

In this work, we studied the case of optimizing the learning rate schedules for image classification tasks; we note, however, that MARTHE is a general technique for finding online hyperparameter schedules (albeit it scales linearly with the number of hyperparameters), possibly implementing a competitive alternative in other application scenarios, such as tuning regularization parameters [Luketina *et al.*, 2016]. We plan to further validate the method both in other learning domains for adapting the LR and also to automatically tune other crucial hyperparameters. We believe that another interesting future research direction could be to derive or learn adaptive rules for μ and β.

---

[8]We note that it is not common in the existing literature to mention the necessity of tuning hyper-hyperparameters for adaptive learning-rate methods, although different datasets and/or networks may require some tuning that may strongly affect the results.

References

[Agarwal *et al.*, 2017] N. Agarwal, B. Bullins, and E. Hazan. Second-order stochastic optimization for machine learning in linear time. *Journal of Machine Learning Research*, 18(1):4148–4187, 2017.

[Almeida *et al.*, 1999] L.B. Almeida, T. Langlois, J.D. Amaral, and A. Plakhov. Parameter adaptation in stochastic optimization. In *On-line learning in neural networks*, pages 111–134. Cambridge University Press, 1999.

[Andrychowicz *et al.*, 2016] M. Andrychowicz, M. Denil, S. Gomez, M.W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, pages 3981–3989, 2016.

[Baydin *et al.*, 2018] A.G. Baydin, R. Cornish, D. Martínez-Rubio, M. Schmidt, and F.D. Wood. Online learning rate adaptation with hypergradient descent. In *ICLR*, 2018.

[Beatson and Adams, 2019] A. Beatson and R.P. Adams. Efficient optimization of loops and limits with randomized telescoping sums. *ICML*, 2019.

[Bengio *et al.*, 1994] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. on Neural Networks*, 5(2):157–166, 1994.

[Bengio, 2012] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[Franceschi *et al.*, 2017] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil. Forward and reverse gradient-based hyperparameter optimization. In *ICML*, pages 1165–1173, 2017.

[Franceschi *et al.*, 2018] L. Franceschi, P. Frasconi, S. Salzo, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*, pages 1568–1577, 2018.

[Glorot and Bengio, 2010] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.

[Griewank and Faure, 2002] A. Griewank and C. Faure. Reduced functions, gradients and hessians from fixed-point iterations for state equations. *Numerical Algorithms*, 30(2):113–139, 2002.

[Griewank and Walther, 2008] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam, 2008.

[He *et al.*, 2016] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, June 2016.

[Hutter *et al.*, 2019] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

[Ioffe and Szegedy, 2015] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[Krizhevsky and Hinton, 2009] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Univ. Toronto, 2009.

[LeCun *et al.*, 1998] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Loshchilov and Hutter, 2017] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. *ICLR*, 2017.

[Luketina *et al.*, 2016] J. Luketina, M. Berglund, K. Greff, and T. Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *ICML*, pages 2952–2960, 2016.

[MacKay *et al.*, 2019] M. MacKay, P. Vicol, J. Lorraine, D. Duvenaud, and R. Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *AISTATS*, 2019.

[Maclaurin *et al.*, 2015] D. Maclaurin, D. Duvenaud, and R.P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *ICML*, 2015.

[Orabona and Pál, 2016] F. Orabona and D. Pál. Coin betting and parameter-free online learning. In *NeurIPS*, pages 577–585, 2016.

[Pearlmutter, 1994] B.A. Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

[Pedregosa, 2016] F. Pedregosa. Hyperparameter optimization with approximate gradient. In *ICML*, 2016.

[Pineda, 1988] F.J. Pineda. Generalization of back propagation to recurrent and higher order neural networks. In *Neural information processing systems*, pages 602–611, 1988.

[Schaul *et al.*, 2013] T. Schaul, S. Zhang, and Y. LeCun. No more pesky learning rates. In *ICML*, pages 343–351, 2013.

[Schraudolph, 1999] N.N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *ICANN*, volume 2, pages 569–574. IET, 1999.

[Simonyan and Zisserman, 2014] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2014.

[Wichrowska *et al.*, 2017] O. Wichrowska, N. Maheswaranathan, M.W. Hoffman, S.G. Colmenarejo, M. Denil, N. Freitas, and J. Sohl-Dickstein. Learned optimizers that scale and generalize. In *ICML*, pages 3751–3760, 2017.

[Wu *et al.*, 2018] Y. Wu, M. Ren, R. Liao, and R. Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *ICLR*, 2018.