NEWTON-BASED TRAINABLE LEARNING RATE

George Retsinas[1] *Giorgos Sfikas*[2] *Panagiotis Paraskevas Filntisis*[1] *Petros Maragos*[1]

[1]School of E.C.E., National Technical University of Athens, 15773, Athens, Greece
[2]Dept. of Surveying & Geoinformatics Engineering, University of West Attica, 12243, Athens, Greece
Email: gretsinas@central.ntua.gr, gsfikas@uniwa.gr, filby@central.ntua.gr, maragos@cs.ntua.gr

ABSTRACT

Selecting an appropriate learning rate for efficiently training deep neural networks is a difficult process that can be affected by numerous parameters, such as the dataset, the model architecture or even the batch size. In this work, we propose an algorithm for automatically adjusting the learning rate during the training process, assuming a gradient descent formulation. The rationale behind our approach is to train the learning rate along with the model weights. Specifically, we formulate first and second-order gradients w.r.t. the learning rate as functions of consecutive weight gradients, leading to a cost-effective implementation. Our extensive experimental evaluation validates the effectiveness of the proposed method for a plethora of different settings. The proposed method has proven to be robust to both the initial learning rate and the batch size, making it ideal for an off-the-shelf optimizing scheme.

Index Terms— gradient descent, adaptive learning rate

1. INTRODUCTION

Deep learning ushered in an era where AI applications are abundant around us, and yet, despite its great success, an important practical shortcoming is that at its core lies a very difficult computational optimization problem. Training requires optimizing a non-convex loss over multiple parameters, finding a global optimum over which is known to be NP-hard [1]. Due to their complex loss structure, existing algorithms aim to discover well-performing local minima using a gradient-based optimization scheme, the most common of which is the Stochastic Gradient Descent (SGD) algorithm.

Arguably, SGD is still widely used for training deep neural networks, decades after its inception. Its efficiency is further supported by recent theoretical developments concerning its stability, as well as its convergence speed in the context of deep learning [2, 3]. Its simple rationale is to choose an improvement over current parameters along the path defined by the loss (sub-)gradient. Other gradient-based methods such as Adam [4] or momentum-based methods [5, 6, 7, 8, 9] share the logic of first choosing and then moving along a "good" search direction that is to be understood as an improvement over the direction defined by the gradient. Adam, in particular, is a recent widely-used algorithm in the genre of adaptive gradient methods, closely related to other popular algorithms such as RMSProp and Adagrad [4, 10, 11], where the search direction is adapted according to local geometry estimates that are computed as moving averages. For all these gradient-based algorithms, the choice of the step size ("learning rate") is arguably perhaps the weakest point of these methods, and their performance is known to rely heavily on its choice. An inappropriate choice of learning rate value can easily lead to a suboptimal local minimum, leading in practice to inferior network efficiency.

To address the choice of learning rate, several works have explored whether a schedule that leads to theoretical guarantees can be obtained; we know that a constant step size leads to convergence to a neighbourhood of the solution, and using a decreasing step size can guarantee convergence to an exact optimum [12]. These ideas led to a variety of effective scheduling algorithms, from multi-step/exponential decay to more complex ones [13, 14, 15], which typically require their behavior to be set by the user via a set of hyper-parameters. Alternate ways of dealing with choosing learning rate include using backtracking line-search methods [16, 17, 18], typically relying on the Armijo-Goldstein conditions at the cost of significantly increasing computational load (requiring multiple function and gradient computations per search direction iterate). Another interesting direction towards step size adaptation, is based on the so-called Polyak's step size, originally proposed for deterministic projected subgradient descent [19], where the step size is proportional to the current loss difference to the global minimum [20, 12, 21].

In this work, we propose a method that casts the learning rate itself as a trainable parameter, motivated by line-search and the closely related hypergradient concept [22, 23]. Thus, the proposed update employs an explicit derivation of gradients with respect to the learning rate. Instead of performing several steps within the same search direction using a line-search approach, we perform only a single "correction" step towards a better learning rate in a gradient descent scheme. While related interpretations and formulations concerning meta-optimization and first-order gradient properties have appeared in earlier [22, 24] and more recent works [23, 25, 26], we introduce a novel second-order gradient derivation and analysis (following a "Newton-Raphson" rationale). The proposed second-order derivation leads to a simple formula of consecutive weight gradients, implemented as a constant-time operation on the backward pass, avoiding the computationally intensive and impractical Hessian derivation/approximation (w.r.t. weights), typically used in such settings.

2. TRAINABLE LEARNING RATE

Problem Statement: We develop our algorithm within the framework of gradient descent, where we want to minimize a loss function $\mathcal{L}$ with respect to model parameters $\mathbf{w}$. The generic formulation of an update rule for a gradient descent algorithm is: $\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha\,\mathbf{u}_t$, where w_t are the model parameters at iteration t, $\mathbf{u}$ is the update direction and α is the learning rate hyperparameter. Since this is a gradient descent paradigm, update direction $\mathbf{u}$ is a function of the gradient: $\mathbf{u}_t = f(\mathbf{g}_t)$, where $\mathbf{g}_t = \nabla\mathcal{L}(\mathbf{w}_{t-1})$.

Most gradient descent variants can be written in the aforementioned formulation. For example, GD in its vanilla form assumes

This research work was supported by the Hellenic Foundation for Research & Innovation (H.F.R.I.) under the "2nd Call for H.F.R.I. Research Projects to support Faculty Members & Researchers" (Project Number:2656, Acronym: TROGEMAL).

$\mathbf{u}_t = \nabla\mathcal{L}(\mathbf{w}_{t-1})$. Respectively, using the sub-gradients $\mathbf{g}_t = \nabla\mathcal{L}_t(\mathbf{w}_{t-1})$ at each step, we can describe a variety of widely-used stochastic versions of GD, such as SGD, momentum-based SGD (e.g. Nesterov [8]) or even Adam [4]. While typically, α is treated as a (user-defined) hyper-parameter and does not contribute to the loss, here, we consider it as a learnable variable through the introduction of an auxiliary function $\mathcal{L}_\alpha(\mathbf{w}_{t-1}) = \mathcal{L}(\mathbf{w}_{t-1} - \alpha\,\mathbf{u}_t)$. Following this formulation, a straightforward meta-algorithm from learning rate adaptation can be developed: One can apply gradient descent on $\mathcal{L}_\alpha$ with respect to α as the the update rule of Eq. 1 suggests, where a meta-learning rate hyper-parameter η is introduced. This way, we can further optimize the process with finer control over the scale (α) of the updates $\mathbf{u}_t$. In fact, Eq. 1 simply describes the proposed meta-algorithm as the iteration between the presented equations of updating the rate and then the model weights with the updated rate.

$$\alpha_t = \alpha_{t-1} - \eta\frac{\partial\mathcal{L}_\alpha(\mathbf{w}_{t-1})}{\partial\alpha}\Big|_{\alpha=\alpha_{t-1}}, \quad \mathbf{w}_t = \mathbf{w}_{t-1} - \alpha_t\mathbf{u}_t \quad (1)$$

The term $\partial\mathcal{L}_\alpha/\partial\alpha$ of Eq. 1, required for learning rate adaptation, can be computed in a cost-effective manner as we will see in the following subsection, and has been already explored in the literature [22, 23]. Nonetheless, such an approach relies on the selection of a new hyper-parameter η, albeit less sensitive compared to manually selecting the learning rate α [23]. Contrary to existing approaches based on this meta-GD paradigm, often dubbed as hypergradient-based [23], we aim to provide a cost-effective computation for the second-order derivative $\partial^2\mathcal{L}_\alpha/\partial\alpha^2$ in order to design a Newton-based algorithm for updating α_t.

First-order derivative: This step has been thoroughly explored in previous works [22, 23]. For the sake of completeness, we include a brief analysis of the computing the first-order derivative via the chain-rule:

$$\begin{aligned}\frac{\partial\mathcal{L}_\alpha(\mathbf{w}_{t-1})}{\partial\alpha} &= \frac{\partial\mathcal{L}(\mathbf{w}_{t-1} - \alpha\,\mathbf{u}_t)}{\partial\alpha}\\ &= \big\langle\nabla\mathcal{L}(\mathbf{w}_{t-1} - \alpha\,\mathbf{u}_t)\,,\,\frac{\partial(\mathbf{w}_{t-1} - \alpha\,\mathbf{u}_t)}{\partial\alpha}\big\rangle\\ &= -\,\langle\nabla\mathcal{L}_\alpha(\mathbf{w}_{t-1})\,,\,\mathbf{u}_t\rangle \end{aligned} \quad (2)$$

Thus the update term of Eq. 1, can be then written as follows:

$$\frac{\partial\mathcal{L}_\alpha(\mathbf{w}_{t-1})}{\partial\alpha}\Big|_{\alpha=\alpha_{t-1}} = -\langle\nabla\mathcal{L}(\hat{\mathbf{w}}_t)\,,\,\mathbf{u}_t\rangle, \quad (3)$$

where $\hat{\mathbf{w}}_t$ is a "look-ahead" term: $\hat{\mathbf{w}}_t = \mathbf{w}_{t-1} - \alpha_{t-1}\,\mathbf{u}_t$. Note that this term differs from the updated parameter $\mathbf{w}_t$ of Eq. 1, which uses α_t instead of α_{t-1}.

For the case of the vanilla GD algorithm, this expression equals to the inner product of the two successive gradients, if one assumed that α is not updated. The derived gradient has an intuitive interpretation:

- $\partial\mathcal{L}_\alpha/\partial\alpha > 0$: If two consecutive weight updates move towards the same direction, the learning rate should be increased.
- $\partial\mathcal{L}_\alpha/\partial\alpha < 0$: Conversely, when we have opposite directions (local optimum nearby), the learning rate should be decreased.
- $\partial\mathcal{L}_\alpha/\partial\alpha = 0$: When the inner product is zero, either we reached a converged state ($||\mathbf{g}_t|| = 0$) or gradient directions are perpendicular. The latter case corresponds to an optimal learning rate according to the line-search formulation and thus no change on the learning rate should be made.

Approximation of the Second-order Derivative: A popular direction towards an adaptive and fast converging learning rate is adopting Newton-based methods, which include the derivation of second-order gradients. Despite the fact that second-order derivatives typically include computationally demanding Hessian computations/approximations (multivariate case), the problem at hand has an intuitive analytical form if we use a Taylor approximation step. We first write the second-order derivative with respect to α:

$$\begin{aligned}&\frac{\partial^2\mathcal{L}_\alpha(\mathbf{w}_{t-1})}{\partial\alpha^2} \overset{(2)}{=} \frac{\partial\big(-\langle\nabla\mathcal{L}_\alpha(\mathbf{w}_{t-1})\,,\,\mathbf{u}_t\rangle\big)}{\partial\alpha} \overset{(2)}{=}\\ &\big\langle\nabla\big(\langle\nabla\mathcal{L}_\alpha(\mathbf{w}_{t-1})\,,\,\mathbf{u}_t\rangle\big)\,,\,\mathbf{u}_t\big\rangle = \mathbf{u}_t^T\,\mathbf{H}(\mathcal{L}_\alpha(\mathbf{w}_{t-1}))\,\mathbf{u}_t \end{aligned} \quad (4)$$

Equation 4 corresponds to a quadratic form, where the Hessian matrix $\mathbf{H}$ is required. As we have already mentioned, such a computation is impractical for modern deep learning models that may consist of millions of parameters. To circumvent this problem, we use a first-order (linear) Taylor approximation on the multivariate function $f(\mathbf{x}) = \nabla\mathcal{L}(\mathbf{x})$. We consider that one step of the descent algorithm generates neighboring solution points and thus we apply the approximation around these successive points, $\mathbf{w}_{t-1}$ and $\hat{\mathbf{w}}_t = \mathbf{w}_{t-1} - \alpha_{t-1}\mathbf{u}_t$, as Eq. 5 suggests.

$$\nabla\mathcal{L}(\mathbf{w}_{t-1}) \approx \nabla\mathcal{L}(\hat{\mathbf{w}}_t) + \mathbf{H}(\mathcal{L}(\hat{\mathbf{w}}_t))\big(\mathbf{w}_{t-1} - \hat{\mathbf{w}}_t\big), \quad (5)$$

where $\mathbf{H}(\mathcal{L}(\mathbf{w}_t))$ is the Hessian matrix of the real-valued loss function $\mathcal{L}(\mathbf{x})$. Using the definition of the look-ahead term $\hat{\mathbf{w}}_t$, the approximation is then written as:

$$\nabla\mathcal{L}(\mathbf{w}_{t-1}) \approx \nabla\mathcal{L}(\hat{\mathbf{w}}_t) + \alpha_{t-1}\mathbf{H}(\mathcal{L}(\hat{\mathbf{w}}_t))\,\mathbf{u}_t \quad (6)$$

The derived approximation relies on the matrix-vector product $\mathbf{H}_t\,\mathbf{u}_t$ that also exists in the second-order gradient of Eq. 4. Therefore, we can substitute this term, which contains the "unwelcome" Hessian matrix, as follows:

$$\frac{\partial^2\mathcal{L}_\alpha(\mathbf{w}_{t-1})}{\partial\alpha^2}\Big|_{\alpha=\alpha_{t-1}} \overset{(4,6)}{=} \frac{\langle\mathbf{u}_t\,,\,\nabla\mathcal{L}(\mathbf{w}_{t-1}) - \nabla\mathcal{L}(\hat{\mathbf{w}}_t)\rangle}{\alpha_{t-1}} \quad (7)$$

Note that the final form of Eq. 7 does not contain the Hessian matrix and can be computed with only the dot products of first-order gradients and their by-products (e.g. $\mathbf{u}_t$).

Newton-based Trainable Learning Rate: Having derived analytical equations for both first- and second-order derivatives, we substitute η in Eq. 1 with $\big[\partial^2\mathcal{L}_\alpha/\partial\alpha^2\big]^{-1}$, as follows:

$$\begin{aligned}\alpha_t &= \alpha_{t-1} - \gamma\Big(\Big[\frac{\partial^2\mathcal{L}_\alpha}{\partial\alpha^2}\Big]^{-1}\frac{\partial\mathcal{L}_\alpha}{\partial\alpha}\Big)\Big|_{\alpha=\alpha_{t-1}}\\ &= \alpha_{t-1}\left(1 + \gamma\frac{\langle\mathbf{u}_t\,,\,\nabla\mathcal{L}(\hat{\mathbf{w}}_t)\rangle}{\langle\mathbf{u}_t\,,\,\nabla\mathcal{L}(\mathbf{w}_{t-1}) - \nabla\mathcal{L}(\hat{\mathbf{w}}_t)\rangle}\right). \end{aligned} \quad (8)$$

Note that Eq. 8 describes a multiplicative rule. In practice, we used the damped Newton's method, with a hyper-parameter $\gamma \in (0, 1]$ to restrict aggressive adaptation. Following the rationale of the Newton-Raphson method, in order to attain a minimum, the second derivative should be positive. Nonetheless, Eq. 7 can take negative values. Moreover, as this derivative is the denominator to the Newton-Raphson formulation, an impractical update direction which tends to infinity would arise when it takes values close to zero.

Given the quadratic form of Eq. 4 and the symmetry of the Hessian (a typical term appearing in Taylor expansion of second order), we can deduce when our approach "miss-behaves": • $\partial^2\mathcal{L}_\alpha/\partial\alpha^2$

can be zero if $\mathbf{u}_t$ belongs to the nullspace of $\mathbf{H}$. If $\mathbf{H}$ is full-rank, $\mathbf{u}_t$ must be zero which implies convergence. • assuming $\mathcal{L}$ to be locally convex around the point of interest $\hat{\mathbf{w}}$ leads to a (semi-)positive definite Hessian matrix and thus a well-behaving positive denominator. Negatives values appear when the underlying function is concave around $\hat{\mathbf{w}}$, typically during the first steps of the descent process.

In practice, such cases are very rare and we overcome this problem by keeping the learning rate unchanged when the denominator is negative, while constraining the multiplicative factor -to avoid exploding gradients- to a maximum value of 100, i.e. large enough to quickly adapt to very different settings.

Technical Considerations: The derived update rule of Eq. 8 contains the look-ahead term $\nabla\mathcal{L}(\hat{\mathbf{w}}_t)$ which introduces a computational overhead. Instead of computing this term, which approximates the gradient at the next step, we consider the gradients of the two previous consecutive steps, as done in [23]. Moreover, concerning the mini-batch versions of the gradient descent, consecutive sub-gradients $\nabla\mathcal{L}_i$ would correspond to different batches of data contrary to the formulation. The inner product of such consecutive gradients could be negative for the majority of batch pairs (different optimization directions for different data), leading to a rapidly diminishing learning rate. To address this problem, we rely on the notion of expecting well-performing gradients, in average, after several iterations. To this end, we accumulate the calculated gradients across several consecutive iterations. For this work, we further simplify our method by assuming that despite the underlying optimizer, we assume that a high-level gradient descent algorithm is approximated every K steps, i.e. $\mathbf{w}_k = \mathbf{w}_{k-1} - \alpha_{k-1}\mathbf{g}_{k-1}$, and thus $\mathbf{g}_{k-1}$ can be easily computed as the difference of the model's weights between K optimization steps divided by α_{k-1}. In other words, we update the learning rate every K iterations by averaging the weights' gradients/update directions. In order to provide finer control over the whole process, the value of K corresponds to a specific percentage p of the entire set.

The main advantage of the proposed method is its trivial computational overhead; only pre-calculated values attained by GD are used. Implementation-wise, we should additionally store the weights of a previous update step $k-1$ and the previous approximated gradient $\mathbf{g}_{k-2}$ as temporary variables.

3. EXPERIMENTAL EVALUATION

Experimental Setup: The proposed methodology is evaluated over a set of different tasks, datasets and models, focusing on deep learning and its applications. We select the following four datasets for the image classification task: MNIST, CIFAR10/100 [27] and ImageNet [28]. For MNIST, we use a multi-layer perceptron (MLP) with a single hidden layer of width 1000, as in [18] & [12]. For CIFAR10 and CIFAR100, we selected the Wide-ResNet (WRNET) architectures [29] which can be modified easily, allowing us to study the proposed algorithm across various network width and depth settings. Finally, the ResNet50 architecture [30] is selected for the ImageNet setting. We also evaluated our method on a Seq2Seq *Transformer* model with 8 heads and 3 encoder and decoder layers used for machine translation from German to English (Multi30k dataset) [1]. All tasks are trained using a standard cross-entropy loss. The proposed method is dubbed *TLR* -Trainable Learning Rate- in the upcoming comparisons. Code is publicly available at `https://github.com/georgeretsi/NTLR`.

[1] We used the implementation in `https://github.com/pytorch/tutorials`.

Hyper-parameters Exploration: We start by exploring the impact of the two user-defined hyperparameters: γ of Eq. 8 and the update percentage p. We performed a grid-search over possible p and γ by training the MNIST+MLP setting for 20 epochs over a set of diverse initial learning rates: $lr = 10^{-i}$, $i = 1, \ldots, 5$ and batch sizes $bs \in \{64, 128, 256, 512\}$. According to this grid search, we selected default hyper-parameters (used for the rest of the paper) as $p = 0.20$ and $\gamma = 0.33$.

Concerning p, frequent updates would lead to a fast diminishing rate that may result to a suboptimal solution; in contrast, updating at the end of each epoch would slow down the whole procedure. On the other hand, small γ values lead to slow adaptation, while large ones correspond to an "aggresive" descent towards the gradient direction, a behavior related to short-horizon bias [31]. However, for an intuitive range of the hyper-parameters values (i.e 0.2 - 0.5) the proposed method indicates robustness across both initial learning rate α_0 and batch size. For reference, concerning the case of batch size 128, typical SGD is very sensitive with 0.661 ± 0.774 loss across the different learning rates, compared to 0.095 ± 0.054 of our approach. Note that the proposed hyper-parameters are intuitive and have a clear interpretation and a constrained range of values, compared to the typical step size formulation. The effect of these hyper-parameters on the learning rate adaptation can be seen in Figure 1. Interestingly enough, the reported behavior resembles an "auto-tuned" scheduler comprised of two phases: warmup and exponential decay.

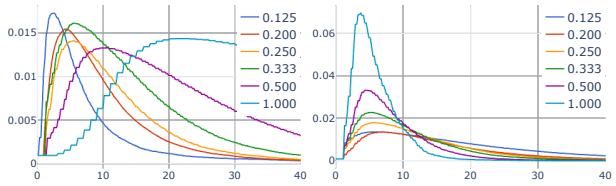

Fig. 1: MNIST+MLP: Learning rate curves for different p values when $\gamma = 0.2$ (left) and different γ values when $p = 0.33$ (right)

Next, we report in detail the behavior of the proposed algorithm under a wide range of initial learning rates ($\alpha_0 = 10^{-i}$, $i = 1, \ldots, 9$), showcasing the adaptation ability of the proposed method. The resulting training curves are depicted in Figure 2 for the MNIST+MLP setting. It is evident that the proposed method quickly adapts to this vast variety of initial learning rates. The case of α_0 corresponds to an under-performing solution when using SGD (loss: 0.326, acc.: 92.3%) due to the large step size; TLR manages to improve this performance (loss: 0.095, acc.: 96.0%), but not to the extent of other initializations since it has already done several "wrong" steps with a large learning rate.

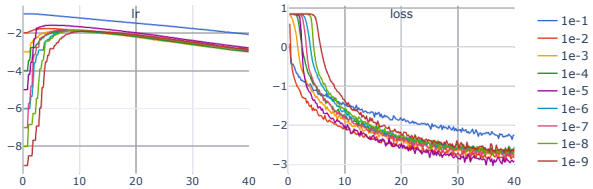

Fig. 2: Loss (left) and learning rate (curve) logarithmic curves of different initial learning rates α_0 for the setting MNIST + MLP.

State-of-the-art Comparisons: Having established the robustness of the proposed method, we continue by comparing our method to several existing adaptive methods. First, we compare our approach to the closely related approach of hypergradients [23]. To promote

a fair comparison, we adopt the first-order method of hypergradients to our framework of updating the learning rate every K iterations[2]. In this experiment, a meta-learning rate η should be set in an additive formulation, i.e. $\alpha_k = \alpha_{k-1} + \eta\, \mathbf{g}_{k-1} \cdot \mathbf{g}_{k-2}$, or in a multiplicative formulation, i.e. $\alpha_k = \alpha_{k-1}[1 + \eta\, cos(\mathbf{g}_{k-1}, \mathbf{g}_{k-2})]$. For a set of different a_0 (10^{-i}, $i = 1, 2, 3$) and batch-sizes (64, 128, 256, 512), we explore the behavior of this approach for different η values, as shown in Table 1 in the form of mean/std value. The main drawback of the first-order hypergradient approach is that different values of η are required for different settings, even if a relative robustness to α_0 is achieved when η is properly selected [23]. This is also evident in the results of Table 1, where we present the two best-performing η out of the set 10^{-i}, $i = 1, \ldots, 5$.

	additive		multiplicative		
α_0	$\eta = 10^{-1}$	$\eta = 10^{-2}$	$\eta = 10^{-4}$	$\eta = 10^{-5}$	TLR
10^{-1}	0.48 ± 0.07	0.38 ± 0.26	0.57 ± 0.08	0.10 ± 0.03	$\mathbf{0.09 \pm 0.01}$
10^{-2}	0.47 ± 0.06	0.59 ± 0.03	1.06 ± 0.14	0.62 ± 0.24	$\mathbf{0.16 \pm 0.02}$
10^{-3}	0.36 ± 0.07	0.49 ± 0.22	1.61 ± 0.14	2.02 ± 0.71	$\mathbf{0.18 \pm 0.03}$

Table 1: Comparison between first-order [23] and second-order (ours-TLR) α updates. Mean & std values of loss are reported for a range of different α_0 and batch-sizes, evaluated over the CIFAR100+WRNET_16_4 setting.

Next, we consider a variety of state-of-the-art adaptive optimizers: Adam [4], SLS (based on the line-search formulation) [18] and SPS (based on the Polyak step size) [12]. We also evalute the popular multistep scheduler with vanilla SGD, denoted as *sgd-mstep*, where rates are decayed by 0.1 at 50% and 75% of the total number of epochs (with $\alpha_0 = 0.1$, a typical well-performing initialization for CIFAR). A comparison of the loss/accuracy curves for the case of batch-size 128 is depicted in Figure 3.

The following observations can be made: 1) The proposed approach achieves fast convergence to well-performing minimum on par with the fine-tuned *sgd-mstep*. 2) *sgd-mstep* provides the best-performing model in this setting at the cost of manually setting when the rate should be decreased (and how much) - note that for different α_0, *sgd-mstep* under-performs since it cannot adapt to such change. 3) SLS produces "trembling" curves for the examined setting, indicating sensitivity. 4) SPS and Adam provide sub-par performance for this vision task, indicating possible generalization problems [32].

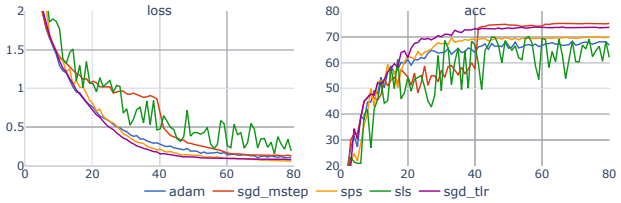

Fig. 3: Loss (left) and accuracy (right) curves of different optimizers/schedulers for the setting CIFAR100 + WRNET_16_4 (128 batch size).

The effectiveness of the proposed method is further validated on the large-scale ImageNet setting, as shown in Figure 4. The overall number of epochs were 80 and the batch size was set to 96 for all experiments due to GPU memory constraints. SLS approach diverges in this setting and thus is not included. Once again, SPS has identical behavior to Adam. Even though these two approaches exhibit convergence to slightly better overall loss, they have sub-optimal performance on the test set, similar to the CIFAR100 case. In contrast, our approach efficiently converges (we achieve 90% of the maximum accuracy at around 30 epochs, faster than any other method) into a well-performing optimum, providing the best accuracy results.

[2]The implementation of [23] under-performs compared to the considered gradient-averaging alternative in our experiments and thus it is omitted.

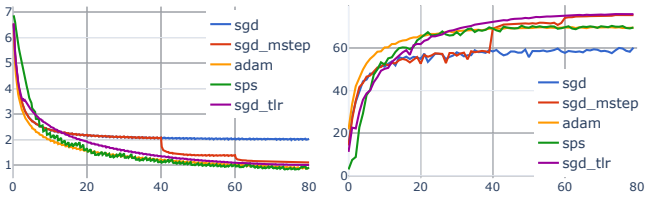

Fig. 4: Loss (left)/accuracy (right) curves for ImageNet+ResNet50 setting.

To further test our algorithm, we also considered tasks where Adam optimizer is usually used effectively. Specifically, we evaluated our method on a Seq2Seq *Transformer* for machine translation from German to English and the training/validation loss curves can be found in Figure 5. For both SGD and Adam, the best-performing learning rate was selected according to a grid-search. Then, we applied the proposed approach over both SGD and Adam to highlight its adaptability. It is evident that the proposed method converges to a very low validation minimum very fast for both cases - typically the final model is the model with the lowest validation loss/error.

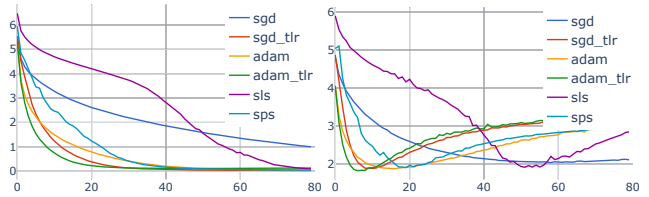

Fig. 5: Train (left)/validation (right) loss curves for the translation task.

Final Remark: The proposed algorithm cannot provide a convergence guarantee for non-convex loss functions, where the increase of learning rate may lead to exploding gradients. Moreover, the algorithm could be trapped at local minima if the learning rate decreases fast enough (a common problem to such adaptive optimizers). Nonetheless, the proposed method can achieve accelerated convergence to the proximity of a well-performing optimum, practically regardless of the setting or the hyperparameters, reporting only minor sensitivity to such selections. Even though it cannot replace modern fine-tuned schedulers [15], it can be an ideal component for more complex heuristic schedulers, such as restarting schemes.

4. CONCLUSIONS

Our contribution can be summarized as follows: this work aims to provide a novel, cost-efficient and easy-to-use optimization meta-algorithm with adaptive learning rate, orthogonal to scheduling ideas such as restarts [14], able to provide well-performing solutions without the need to carefully tune any optimization hyper-parameters. Experimental evaluation over a plethora of settings support the effectiveness of the proposed algorithm, which rapidly converges to well-performing local minima without any setting-specific tuning required. This work paves the way towards the following research steps: 1) assume a finer control over the adaptation by introducing a different learning rate for each layer - this is straightforward under our setting, 2) examine how the proposed approach can be compared/combined with scheduling tactics [14] and 3) analyze the conditions where such an algorithm can guarantee convergence (consider using Wolfe/Armijo-Goldstein conditions).

5. REFERENCES

[1] Ju Sun, *When are nonconvex optimization problems not scary?*, Ph.D. thesis, Columbia University, 2016.

[2] Moritz Hardt, Ben Recht, and Yoram Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *International Conference on Machine Learning*. PMLR, 2016, pp. 1225–1234.

[3] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song, "A convergence theory for deep learning via over-parameterization," in *International Conference on Machine Learning*. PMLR, 2019, pp. 242–252.

[4] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations*, 2015.

[5] Elijah Polak and Gerard Ribière, "Note sur la convergence de méthodes de directions conjuguées," *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique*, vol. 3, no. R1, pp. 35–43, 1969.

[6] Boris T. Polyak, "The conjugate gradient method in extremal problems," *USSR Computational Mathematics and Mathematical Physics*, vol. 9, no. 4, pp. 94–112, 1969.

[7] Jonathan Barzilai and Jonathan M. Borwein, "Two-point step size gradient methods," *IMA journal of numerical analysis*, vol. 8, no. 1, pp. 141–148, 1988.

[8] Yuri Nesterov, "A method of solving a convex programming problem with convergence rate O($1/k^2$)," in *Sov. Math. Dokl.*, 1983, vol. 27.

[9] Jian Zhang and Ioannis Mitliagkas, "Yellowfin and the art of momentum tuning," in *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., 2019, vol. 1, pp. 289–308.

[10] John Duchi, Elad Hazan, and Yoram Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[11] Tijmen Tieleman and Geoffrey Hinton, "Lecture 6.5-RMSProp, Coursera: Neural Networks for machine learning," *University of Toronto, Technical Report*, 2012.

[12] Nicolas Loizou, Sharan Vaswani, Issam Hadj Laradji, and Simon Lacoste-Julien, "Stochastic Polyak step-size for SGD: An adaptive learning rate for fast convergence," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 1306–1314.

[13] Leslie N Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2017, pp. 464–472.

[14] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proceedings of the International Conference on Learning Representations*, 2017.

[15] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He, "Accurate, large minibatch SGD: Training Imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

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

[17] Clément W. Royer and Stephen J. Wright, "Complexity analysis of second-order line-search algorithms for smooth nonconvex optimization," *SIAM Journal on Optimization*, vol. 28, no. 2, pp. 1448–1477, 2018.

[18] Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien, "Painless stochastic gradient: Interpolation, line-search, and convergence rates," *arXiv preprint arXiv:1905.09997*, 2019.

[19] Boris T. Polyak, "Introduction to optimization," in *Optimization Software, Publications Division*. Citeseer, 1987.

[20] Mathieu Barré, Adrien Taylor, and Alexandre d'Aspremont, "Complexity guarantees for Polyak steps with momentum," in *Conference On Learning Theory*. PMLR, 2020, pp. 452–478.

[21] Ryan D'Orazio, Nicolas Loizou, Issam Laradji, and Ioannis Mitliagkas, "Stochastic mirror descent: Convergence analysis and adaptive variants via the mirror stochastic polyak stepsize," *arXiv preprint arXiv:2110.15412*, 2021.

[22] Luís Almeida, Thibault Langlois, D. Amaral José, and Alexander Plakhov, "Parameter adaptation in stochastic optimization," in *Publications of the Newton Institute*, pp. 111–134. 1999.

[23] Atılım Günes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood, "Online learning rate adaptation with hypergradient descent," in *Proceedings of the International Conference on Learning Representations*, 2018.

[24] Nicol N. Schraudolph, "Local gain adaptation in stochastic gradient descent," Tech. Rep. IDSIA-09-99, IDSIA, 1999.

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

[26] Ehsan Amid, Rohan Anil, Christopher Fifty, and Manfred K. Warmuth, "Step-size adaptation using exponentiated gradient updates," in *In ICML Workshop on Beyond First-Order Methods in ML Systems*, 2020.

[27] Alex Krizhevsky and Geoffrey Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., Citeseer, 2009.

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[29] Sergey Zagoruyko and Nikos Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference (BMVC)*, Edwin R. Hancock Richard C. Wilson and William A. P. Smith, Eds., 2016.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[31] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse, "Understanding short-horizon bias in stochastic meta-optimization," in *Proceedings of the International Conference on Learning Representations*, 2018.

[32] Nitish Shirish Keskar and Richard Socher, "Improving generalization performance by switching from Adam to SGD," *arXiv preprint arXiv:1712.07628*, 2017.