Assessment of a Two-step Integration Method as an Optimizer for Deep Learning

Paul Rodriguez
Department of Electrical Engineering
Pontifical Catholic University of Peru
Lima, Peru
prodrig@pucp.edu.pe / ORCID 0000-0002-8501-0907

***Abstract*—It is a known fact that accelerated (non-stochastic) optimization methods can be understood as multi-step integration ones: e.g. the heavy ball's Polyak and Nesterov accelerations can be derived as particular instances of a two-step integration method applied to the gradient flow. However, in the stochastic context, to the best of our knowledge, multi-step integration methods have not been exploited as such, only as some particular instances, i.e. SGD (stochastic gradient descent) with momentum or with the Nesterov acceleration.**

In this paper we propose to directly use a two-step (TS) integration method in the stochastic context. Furthermore, we assess the computational effectiveness of selecting the TS method's weights after considering its lattice representation. Our experiments includes several well-known multiclass classification architectures (AlexNet, VGG16 and EfficientNetV2) as well as several established stochastic optimizer e.g. SGD along with momentum/Nesterov acceleration and ADAM. The TS based method attains a better test accuracy than the first two, whereas it is competitive with to a well-tuned (ϵ / learning rate) ADAM.

***Index Terms*—stochastic gradient descent, gradient flow.**

I. Introduction

Let $F(\mathbf{u})$ represent a loss function, then a linear two-step (TS) optimization method [1, Ch. 4] can be summarized by

$$\mathbf{u}_{k+1} \;=\; \rho_0 \mathbf{u}_k + \rho_1 \mathbf{u}_{k-1} - \alpha \cdot (\sigma_0 \nabla F_k + \sigma_1 \nabla F_{k-1}), \quad (1)$$

where ∇F_k represents $\nabla F(\mathbf{u}_k)$, and $\boldsymbol{\rho} = \{\rho_0, \rho_1\}$ and $\boldsymbol{\sigma} = \{\sigma_0, \sigma_1\}$ with $\rho_j, \sigma_j \in \mathbb{R}$, are the TS' weights.

While a brief theory background on multi-step integration methods [1, Ch. 4] and the gradient flow [2] is given in Section II-B along with the conditions that $\boldsymbol{\rho}$ and $\boldsymbol{\sigma}$ should satisfy in order to obtain a *proper* method (i.e. (1) converges to the minimizer of $F(\mathbf{u})$), here we highlight that the well-known (non-stochastic) Polyak's [3] heavy ball and Nesterov [4] accelerations, can be written (see Section II-B as well as [5]) as a particular instances of (1); e.g., by taking $\boldsymbol{\rho} = \{1+\gamma, -\gamma\}$ and $\boldsymbol{\sigma} = \{1, 0\}$, Fig. 1 summarizes the former case.

Linear two-step (TS) method	Polyak's heavy ball
$\mathbf{u}_{k+1} = (1+\gamma)\cdot\mathbf{u}_k - \gamma\cdot\mathbf{u}_{k-1} - \alpha\nabla F_k$	$\mathbf{z}_{k+1} = \gamma\cdot\mathbf{z}_k - \alpha\nabla F_k$
$= \mathbf{u}_k + \gamma\cdot\underbrace{(\mathbf{u}_k - \mathbf{u}_{k-1})}_{\mathbf{z}_k} - \alpha\nabla F_k$	$\mathbf{u}_{k+1} = \mathbf{u}_k + \mathbf{z}_{k+1}$

Fig. 1: Summary of the relationship between the generic TS method (1) when $\boldsymbol{\rho} = \{1+\gamma, -\gamma\}$ and $\boldsymbol{\sigma} = \{1, 0\}$ and the Polyak's [3] heavy ball optimization method.

This research was supported by the Army Research Office (ARO) under Grant W911NF-22-1-0296.

If we let $F(\mathbf{u})$ to be defined as in (2a) and consider that $N \to \infty$, such in the case of applications associated with deep learning (DL), then the well-known first order optimization methods or any of its accelerated versions, e.g. gradient descent (GD), Polyak's [3] heavy ball, Nesterov's [4] or Anderson's [6] acceleration, etc., become impractical.

$$F(\mathbf{u}) = \frac{1}{N}\sum_{n=1}^{N} f_n(\mathbf{u}), \quad \mathbf{g}_k = \frac{1}{\#\mathcal{I}_k}\sum_{n\in\mathcal{I}_k} \nabla f_n(\mathbf{u}) \quad (2a,2b)$$

In this scenario, stochastic GD (SGD), which uses a noisy gradient approximation (computed over set $\mathcal{I}_k$, a random fraction of the $\{1, 2, \ldots, N\}$ set; see (2b)), has became crucial. In its most basic form, the main inner loop of the SGD algorithm can be written as

$$\mathbf{z}_{k+1} = \gamma\cdot\mathbf{z}_k - \alpha\cdot\mathbf{g}_k, \qquad \mathbf{u}_{k+1} = \mathbf{u}_k + \mathbf{z}_{k+1}, \qquad (3a,3b)$$

where $\mathbf{g}_k$ is defined as in (2b); if $\gamma = 0$ then (3a)-(3b) sumarize the SGD algorithm, whereas, if $\gamma > 0$, they summarized the well-known SGD+momentum (SGD+M) variant.

While there exists several more elaborated SGD variants (see Section II-C along with Algorithm 1 and Fig. 3) such RMSprop [7], AdaDelta [8], ADAM (and its variant AdaMax) [9], AMSGrad [10], AdaBound / AMSbound [11], Nadam [12], EAdam [13], AdaBelief [14], adaptive SGD+M [15], etc. none of them share the structure of a "direct" linear two-step optimization adapted to the stochastic scenario, i.e.

$$\mathbf{u}_{k+1} \;=\; \rho_0\cdot\mathbf{u}_k + \rho_1\cdot\mathbf{u}_{k-1} - \alpha\cdot(\sigma_0\cdot\mathbf{g}_k + \sigma_1\cdot\mathbf{g}_{k-1}), \quad (4)$$

nor, to the best of our knowledge [1], the stochastic algorithm summarized by (4), where $\mathbf{g}_k$ is defined as in (2b), has been assessed for the DL scenario. Given the practical performance [17] of SGD+M, this observation is worth pursuing.

In particular, in this paper, we focus on assessing the performance of (4) when training several well-known classification architectures (e.g. AlexNet [18], VGG16 [19] and EfficientNetV2 [20]); furthermore, in our reproducible (TensorFlow based) experiments we include the performance of three well-established optimizers as baseline. Moreover, we also propose a simple procedure to select the $\boldsymbol{\rho}$ and $\boldsymbol{\sigma}$ weights in (4) base

[1]Here we highlight that the unpublished work [16] proposed to use the "effective gradient flow" instead of using the gradient norm of a network as a proxy to study its optimization dynamics. However (4) is not even indirectly mentioned in such work.

on the lattice representation associated to the interpretation of (4) as a FIR filter (see Section III). In summary, in our experiments (multiclass classification problem) the "stochastic TS" or S-TS (i.e. iteration (4)) along with the proposed weight selection/adaptation attains a better test accuracy value than SGD+M and SGD-Nesterov. However a well-tuned ADAM optimizer is slightly superior to the S-TS method.

II. Related work

A. *Non-stochastic accelerated methods*

In this Section we succinctly review the Polyak's [3] heavy ball and Nesterov [4] accelerations; the rationality is to then highlight, in Section II-B, that both methods are particular instances of the two-step linear optimization method. While other accelerated methods (i.e. Anderson [6]) may be cast as linear multistep methods, they are not considered here.

1) Polyak's [3] heavy ball (PHB) : extends the standard GD algorithm by incorporation a short-term memory (for the gradient): while the classical PHB method is summarized in Fig. 1, its current iterate can be interpreted as the result of a standard GD step plus a momentum, which depends on past gradient values and constant γ (easily observed if $\mathbf{z}_{k+1}$ is replaced into $\mathbf{u}_{k+1}$ recursively in Fig. 1). [15] is a recently proposed adaptive variant.

2) Nesterov's [4] acceleration (NTRV) : also uses a short term memory: it first generates an extrapolation ($\mathbf{y}_{k+1}$ iterate in Fig. 2) between the current and past iterates, which is controlled by the inertial sequence γ_k, to then proceed with a standard GD step ($\mathbf{u}_{k+1}$ iterate in Fig. 2). The NTRV method is proven to be optimal on the class of strongly convex functions.

B. *Linear two-step (TS) method (see (1))*

For a detailed review of the general linear multi-step integration method we recommend [1, Ch. 4]. On what follows we will focus on the main results associated with the TS method, which are mainly summarized from [5].

For smooth convex and continuous functions, the gradient flow [2] is a smooth curve defined by (5). While its numerical

$$\dot{\mathbf{x}}(t) = \nabla F(\mathbf{x}(t)) \qquad (5)$$

interpretation leads to integration methods (e.g. Euler, Runge-Kutta and, in general, multi-step integration methods [1]), it is also related to optimization methods. For instance, if we consider the Taylor expansion of $\mathbf{x}(t+h)$ along with $t = k\cdot h$ (i.e. the Euler method to integrate (5)) then we obtain GD:

$$\mathbf{x}(t+h) \approx \mathbf{x}(t) + h\cdot\dot{\mathbf{x}}(t) \longrightarrow \mathbf{x}_{k+1} = \mathbf{x}_k + h\cdot\nabla F(\mathbf{x}_k). \quad (6)$$

Multi-step integration methods [1, Ch. 4] improve the rate of convergence of the classical Euler method by using more previous information (past gradients and iterates). As we mentioned in the Introduction, given the success of the SGD+M [17], which is PHB in the stochastic context (see also Fig. 1), it seems natural to speculate (stochastic context) about the performance of other TS's direct instances when compared to established SGD variants.

Here we acknowledge that, contrary to the PHB algorithm, the NTRV one (which is also related to the TS method, see Fig. 2) is not a popular choice in the stochastic context, although it may attain comparative or even better performance than SGD+M for some classification problems [21].

Nesterov	TS method
$\mathbf{u}_{k+1} = \boxed{\mathbf{y}_k - \alpha\nabla F(\mathbf{y}_k)}$ $\mathbf{y}_{k+1} = \mathbf{u}_{k+1} + \gamma_k\cdot(\mathbf{u}_{k+1} - \mathbf{u}_k)$	$\mathbf{y}_{k+1} = (1+\gamma_k)\cdot\mathbf{y}_k - \gamma_k\cdot\mathbf{y}_{k-1}$ $-\alpha\cdot(1-\gamma_k)\cdot\nabla F(\mathbf{y}_k)$ $+\alpha\cdot\gamma_k\cdot\nabla F(\mathbf{y}_{k-1})$

Fig. 2: Summary of the relationship between the generic two-step integration method (1) when $\boldsymbol{\rho} = \{1+\gamma_k, -\gamma_k\}$ and $\boldsymbol{\sigma} = \{1+\gamma_k, -\gamma_k\}$ and the Nesterov's [4] accelerated method.

The convergence of a multistep linear method is determined [5, Section 3.1] by the selection of its weights $\boldsymbol{\rho}$ and $\boldsymbol{\sigma}$. In particular, the TS (see (1)) results in a convergent method if

$$\rho 0 + \rho_1 = 1, \quad \sigma_0 + \sigma_1 = -\rho_0 + 2, \quad |\textsc{Roots}(\boldsymbol{\rho})| \leq 1, \text{(7a,7b,7c)}$$

where $\textsc{Roots}(\boldsymbol{\rho})$ are the roots of the polynomial $\boldsymbol{\rho}(z)$, i.e. $-\rho_1 - \rho_0\cdot z + z^2$. Furthermore, the region of absolute stability (see [5, definition 2.6]) of a TS method is the set of values $\alpha\cdot\lambda$ for which all the roots of the polynomial $\boldsymbol{\rho}(z) + \alpha\cdot\lambda\cdot\boldsymbol{\sigma}(z)$, where $\boldsymbol{\sigma}(z) = \sigma_1 + \sigma_0\cdot z$, are inside the unit circle.

C. *SGD's variants*

There exist several alternative approaches to summarized the SGD algorithm as well as its variants. For that objective, in particular, here we use the structure of algorithm 1 (adapted from [22]; for alternative interpretations, among others, see [23], [24]). Furthermore, Fig. 3 may be derived from it, and naturally describes SGD variants as an "add on" set of features that may be applied over the "vanilla" SGD.

```
Algorithm 1: Main inner loop of a generalized SGD
Inputs: $\alpha$: Step-size. $\beta, \gamma_1, \gamma_2$: Decay rates. $\mathbf{g}_k$: (2b);
for $k \geq 0$ do
    $\alpha_k = \phi(\alpha, k)$    (e.g. $\alpha/\sqrt{k}$. Details in [25], [26])
    $\mathbf{v}_k = Q(\mathbf{g}_k, \gamma_2)$    (usually $\gamma_2\cdot\mathbf{v}_{k-1} + (1-\gamma_2)\cdot\mathbf{g}_k\odot\mathbf{g}_k$)
    $\mathbf{w}_k = \frac{c_k}{\sqrt{\mathbf{v}_k+\epsilon}}$    (or $\frac{c_k}{\sqrt{\mathbf{v}_k}+\epsilon}$; $c_k$ : "bias/unit correction")
    $\mathbf{u}_{k+1} = \mathbf{u}_k - \alpha_k\cdot\mathbf{w}_k\odot\mathcal{A}(\mathbf{g}_k, \gamma_1) + \beta\cdot(\mathbf{u}_k - \mathbf{u}_{k-1})$
end
```

For instance, the standard SGD along with its momentum and Nesterov variants can be obtained by setting $\alpha_k = \alpha$, $\mathbf{v}_k = 1$ and $\mathbf{w}_k = 1$ along with the proper choice for $\beta > 0$ and $\mathcal{A}(\cdot)$, i.e. $\mathbf{g}_k$ (former) or $\mathbf{g}_k + \beta\cdot(\mathbf{g}_k - \mathbf{g}_{k-1})$ (latter).

The well-known AdaGrad [27] and RMSprop [7] variants took advantage of the problem's local geometry by setting $\mathbf{v}_k = \gamma_2\cdot\mathbf{v}_{k-1} + (1-\gamma_2)\cdot\mathbf{g}_k\cdot\mathbf{g}_k$ with $\gamma_2 = 0$ (former) or $0 < \gamma_2 < 1$ (latter), i.e. exponential moving average (EMA). Both cases also set $\mathbf{w}_k = \frac{1}{\sqrt{\mathbf{v}_k+\epsilon}}$, $\mathcal{A}(\cdot) = \mathbf{g}_k$ and $\beta = 0$.

Taking an alternative direction Adadelta [8] proposed to used a customization of the Newton method in the SGD context (which can also be summarized by Algorithm 1).

Taking AdaGrad and RMSprop as starting point, ADAM [9] proposed to also use an EMA for the gradient, i.e. $\mathcal{A}(\cdot) =$

$\gamma_1 \cdot \mathcal{A}_{k-1} + (1-\gamma_1) \cdot \mathbf{g}_k$ along with a bias correction which was encoded into c_k (see line 4 in Algorithm 1).

There are several ADAM variants: AdaMax (also detailed in [9]) used $\mathbf{v}_k = \max\{\gamma_2 \cdot \mathbf{v}_{k-1}, |\mathbf{g}_k|\}$ instead of the standard choice (also used in AdaGrad and RMSprop); Nadam [12] proposed to used the the Nesterov's acceleration [4] on the gradient's EMA (not on the solution itself, as was originally intended); AdaBelief [14] also used ADAM's structure along with $\mathbf{v}_k = \gamma_2 \cdot \mathbf{v}_{k-1} + (1-\gamma_2) \cdot (\mathbf{g}_k - \mathcal{A}_k)^2 + \epsilon$, while AMSGrad [10] considers $\hat{\mathbf{v}}_k = \max\{\hat{\mathbf{v}}_{k-1}, \mathbf{v}_k\}$ as means to improve ADAM's convergence; On the other hand, AdaBound [11] may be described as a "ADAM more robust to large learning rate values" for which $\phi(\cdot)$ uses dynamic bounds; EAdam [13] studied the impact of the ϵ safeguard (see line 4 in Algorithm 1) and proposed to change its location (see [13, Algorithm 2]).

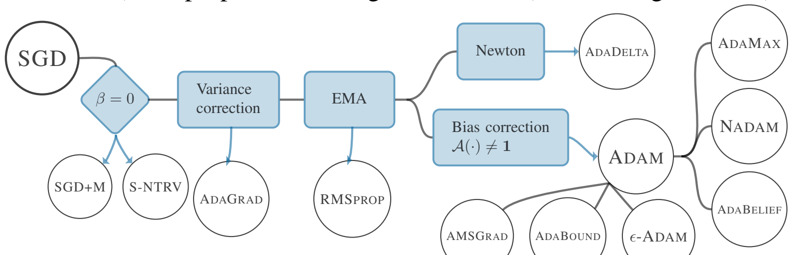

Fig. 3: The relationship between SGD variants may be understood as applying a set of "add on" features (highlighted in blue) over the "vanilla" SGD. See also Section II-C.

III. Stochastic TS and weight selection

The rules to be enforced in order to obtain a *proper* TS method (see (7) as well as [5, Section 3.1]) can be used to design a family of (non-stochastic) TS for quadratic functions; such approach leads to the well-known PHB and NTRV optimizers and was carried on [5, Section 3.2] by directly analyzing the roots of the polynomial $\mathbf{\Pi}_{\alpha,\lambda}(z) = \boldsymbol{\rho}(z) + \alpha \cdot \lambda \cdot \boldsymbol{\sigma}(z)$.

Clearly, in the case of functions that are not strongly convex, such the loss associated with a multi-class classification problem, such straightforward approach can not be applied. However, here we propose to take an alternative route: Instead of directly using the $\mathbf{\Pi}_{\alpha,\lambda}(z)$'s roots in order to enforce the condition of them being inside the unit circle, we propose to interpret $\mathbf{\Pi}_{\alpha,\lambda}(z)$ as a FIR filter and use its lattice [28, Chapter 6] coefficients to meet the unit circle condition.

Given a monic polynomial which represents the $\mathcal{Z}$ transform of a FIR filter, e.g. $H(z) = z^2 + m_0 \cdot z + m_1$, then its lattice representation may be computed by means of the Step-down recursion (associated with the Levinson-Durbin [28, Chapter 5]). The original filter is minimum phase (i.e. all its roots are inside the unit circle) if and only if all its lattice coefficients are bounded by 1. The relationship between $H(z)$ and its lattice representation (Γ-coefficients) is given by (8), which, if applied to the case of $\boldsymbol{\rho}(z)$ (see 7c)), we obtain (9).

$$\Gamma_2 = m_1, \quad \Gamma_1 + \Gamma_1 \cdot \Gamma_2 = m_0 \longrightarrow \Gamma_1 = \frac{m_0}{1+m_1} \tag{8}$$

$$|-\rho_1| \leq 1, \qquad \left| \frac{-\rho_0}{1-\rho_1} \right| \leq 1. \tag{9a,9b}$$

For either case, SGD+M or NTRV, the weights related to the past iterates (see (3a-3b)) are given by $\rho_0 = 1 + \gamma_k$ and $\rho_1 = -\gamma_k$, which clearly comply with (9); moreover, while γ_k may be a fixed value (SGD+M), it may also vary (e.g. as for NTRV), as long as $0 < \gamma_k < 1$.

If we use (8) for the case of $\mathbf{\Pi}_\alpha(z)$, we get

$$|-\rho_1 + \alpha\sigma_1| \leq 1, \quad \left| \frac{-\rho_0 + \alpha \cdot \sigma_0}{1 - \rho_1 + \alpha \cdot \sigma_1} \right| \leq 1; \tag{10a,10b}$$

taking $\rho_1 = -\gamma_k$, and considering[2] $\alpha\sigma_1 > 0$ then (10a) implies that as $\gamma_k \to 1$ then $\alpha \cdot \sigma_1 \to 0$; in a practical scenario, if we fix σ_1, then any rule that increases γ_k can be linked to a decreasing learning rate schedule (while motivated by different observations, this is a well-known fact, with works that span several decades, e.g. [29] to [30]). Furthermore, if we consider[3] that $\alpha > 0$, $\sigma_0 > 0$ and $\sigma_1 > 0$, and re-write (10b) as $\left| \frac{\rho_0 - \alpha \cdot \sigma_0}{\rho_0 + \alpha \cdot \sigma_1} \right| \leq 1$, then, within reasonable limits, we can consider (11), where k represents the iteration index.

$$\sigma_0^{(k)} > \sigma_1, \quad \sigma_0^{(k)} \geq \sigma_0^{(k-1)}. \tag{11}$$

Based on the above analysis, then the stochastic two-step (TS) is summarized in Algorithm 2.

Algorithm 2: Main inner loop of the stochastic-TS.

```
Inputs: $\alpha$: Step-size. $\rho$, $\sigma$: TS weights. $\mathbf{g}_k$: (2b);
1 for $k \geq 0$ do
2     $\alpha_k = \phi(\alpha, k)$        (e.g. $\alpha_k = c^{\lfloor \frac{\text{Epoch}}{L} \rfloor} \cdot \alpha$.)
3     $\rho_0, \rho_1, \sigma_0$ : update s.t. (9), (11) are observed.
4     $\mathbf{u}_{k+1} = \rho_0 \cdot \mathbf{u}_k + \rho_1 \cdot \mathbf{u}_{k-1} - \alpha_k \cdot (\sigma_0 \cdot \mathbf{g}_k + \sigma_1 \cdot \mathbf{g}_{k-1})$
5 end
```

IV. Computational results

The simulations presented below were carried out on an Intel i9-12900H (2.90 GHz, 24MB Cache, 64GB RAM) based laptop equipped with a GeForce RTX 3080 graphic card. All SGD variants implementations used in our simulations correspond to those found in the TensorFlow 2.11 library [31]; furthermore, such methods will be labeled as "TF *method*", where *method* corresponds to the particular SGD variant (SGD+M, NTRV or ADAM). Our publicly available code [32], which includes the TensorFlow-based implementation of our proposed algorithm, labeled "TS", can be used to reproduce our experimental results.

A. Hyperparameter selection

As it has been noted in [33] (and elsewhere) ADAM's default $\epsilon = 10^{-8}$ value (see Line 4 in Algorithm 1) does not yield the best results. For our experiments we used $\epsilon = 10^{-3}$ along with $\alpha \approx 0.001$, since such combination gave, statistically, as good results as other ϵ/α combinations suggested by [33, Fig. 1]. ADAM's other parameters (EMA related) were fixed to $\gamma_1 = 0.9$ and $\gamma_2 = 0.999$.

[2]Clearly $\alpha\sigma_1 < 0$ is possible; moreover, this is the case for NTRV: see [5, Section 4.2]) as well as Fig. 2.

[3]We do not consider the case where $\sigma_0 > 0$ and $\sigma_1 < 0$ since such choice would lead us to an algorithm similar to NTRV.

For the SGD+M and NTRV we used a simple grid search, which looked for the best learning rate α. For both cases we fixed the momentum parameter to 0.9.

The stochastic-TS's weight were selected heuristically, as a result of a *guided* search based on the observations related to (9) and (11). As a result we use:

- $\rho_0^{(k)} = \min(\rho_0 + (2.0 - \rho) \cdot (k/L_\rho),\, 2.0)$, $\rho_1^{(k)} = 1 - \rho_0^{(k)}$,
- $\alpha_k = c^{\lfloor \frac{\text{Epoch}}{L_\alpha} \rfloor} \cdot \alpha$,
- $\sigma_0^{(k)} = \min(\sigma_0 + (1.5 - \sigma) \cdot (k/L_\sigma),\, 1.5)$,

along with $L_\rho = 10^3$, $L_\alpha = 20, L_\sigma = 10^4$ for all the multi-class architectures described next (Section IV-B).

B. Experiments on multiclass classification

To experimentally assess Algorithm 2, we focus on the multi-class classification problem along with the CIFAR-10 dataset, which consists on 50K 32×32 color images for training, divided into 10 classes, and 10K images for testing. No data augmentation is considered, only the standard pre-processing[4]. We consider 150 (AlexNet, VGG16) and 50 (EfficientNetV2) epochs and the selected parameters are associated with the best performance (test accuracy point of view).

1) AlexNet [18] : In Fig. 4 we compare the performance of the TS (Algorithm 2) optimizer along with that of SGD+M, SGD-NTRV and Adam. For completeness' sake, for ADAM we also report the $\epsilon = 10^{-8}$ and consider two learning rate schedules (LR-sch): the initial LR is (i) multiplied by 0.1 after 100 epochs, or (ii) multiplied by 0.5 every 20 epochs; the latter is the preferred LR-sch (see (10) and related comments) for TS (Algorithm 2); for the same reason, SGD+M and SGD-NTRV also take advantage of the same LR-sch.

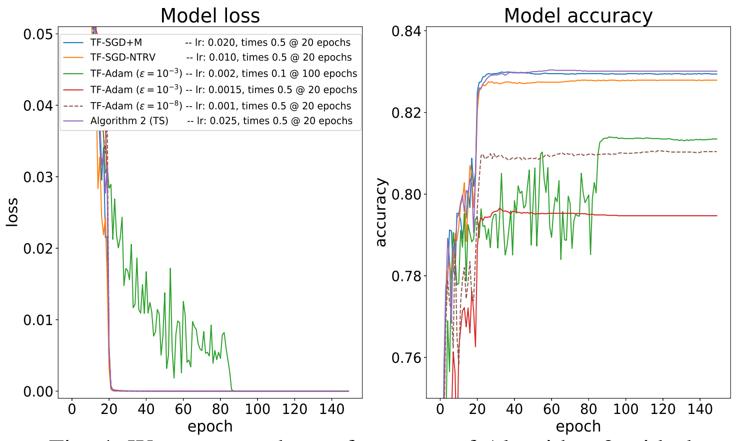

Fig. 4: We compare the performance of Algorithm 2 with that of SGD+M, SGD-NTRV and ADAM on AlexNet.

2) VGG16 [19]: Contrary to what was observed in Fig. 4, in Fig. 5, (i) ADAM along with the preferred LR-sch ($0.5^{\lfloor \frac{\text{Epoch}}{20} \rfloor} \cdot \alpha$) gives better performance than Adam along with the other LR-sch (not shown), and SGD-NTRV attains better results than SGD+M (although not as good as ADAM). Overall TS (Algorithm 2) is competitive with Adam, however its loss evolution is far smoother than the former.

[4]As it is customary, the input images' pixel are normalized between [0, 1] and then force to be zero mean.

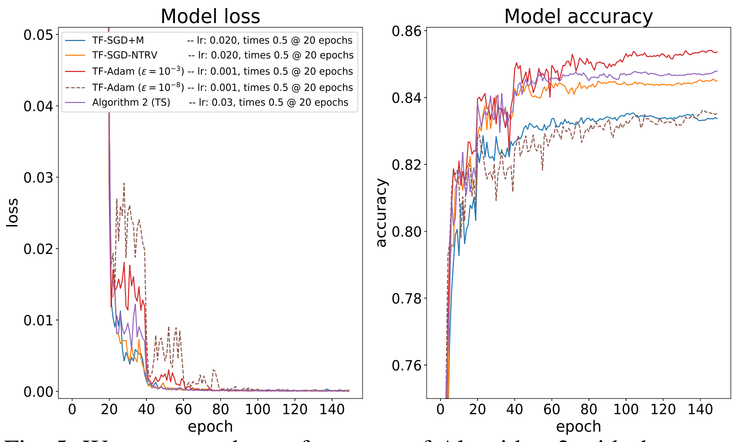

Fig. 5: We compare the performance of Algorithm 2 with that of SGD+M, SGD-NTRV and ADAM on VGG16.

3) EfficientNetV2 [20]: For this case we used a partially trained EfficientNetV2 as a staring point in order to be able to attain competitive results when using CIFAR's original image size. It can be argued that the well-tuned ADAM optimizer attains the best results, while TS and NTRV had a similar performance but superior to SGD+M. The loss evolution (all cases) is not as smooth as in the previous examples.

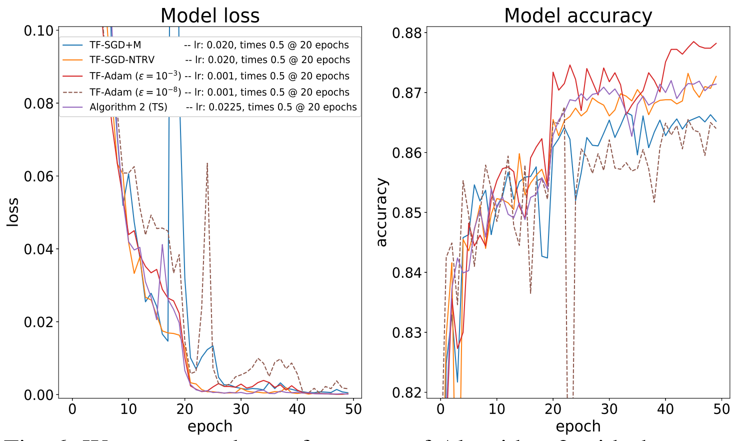

Fig. 6: We compare the performance of Algorithm 2 with that of SGD+M, SGD-NTRV and ADAM on EfficientNetV2.

V. Conclusions

Given that two SGD (stochastic gradient descent) variants, namely SGD+momentum and SGD-Nesterov, can be interpreted as a particular instances of the stochastic counterpart of a linear two-step (TS) integration method, in this paper we have assessed the performance of such generic method.

Based on the lattice representation associated with the interpretation of the stochastic TS (S-TS) method as a FIR filter, we derive simple rules for the S-TS method's parameters such they differ from its well-established SGD variants.

Our reproducible experiments lead us to the conclusion that the S-TS method has superior performance (multiclass classification problem's test accuracy) than its well-established SGD variants. While a well-tuned ADAM optimizer attains a slightly superior performance, the loss function's evolution of the S-TS method is, in general, smooth while ADAM's tends to exhibit an oscillatory behavior for all the considered cases.

References

[1] J. C. Butcher, *Numerical Methods for Ordinary Differential Equations*, John Wiley & Sons, Ltd, 2016.

[2] L. Ambrosio, N. Gigli, and G. Savare, *Gradient Flows in Metric Spaces and in the Space of Probability Measures*, Lectures in Mathematics ETH Zürich. Birkhäuser, 2 edition.

[3] B. Polyak, "Some methods of speeding up the convergence of iteration methods," *Ussr Computational Mathematics and Mathematical Physics*, vol. 4, pp. 1–17, 12 1964.

[4] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o(1/k)^2," *Soviet Mathematics Doklady*, vol. 27, no. 2, pp. 372–376, 1983.

[5] D. Scieur, V. Roulet, F. Bach, and A. d'Aspremont, "Integration methods and optimization algorithms," in *Advances in Neural Information Processing Systems*, 2017, vol. 30.

[6] D. Anderson, "Iterative procedures for nonlinear integral equations," *J. ACM*, vol. 12, no. 4, pp. 547–560, Oct. 1965.

[7] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," 2012.

[8] Matthew Zeiler, "Adadelta: An adaptive learning rate method," vol. https://arxiv.org/abs/1212.5701, 12 2012.

[9] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Int'l Conf. on Learning Representations (ICLR)*, 2015.

[10] S. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *International Conference on Learning Representations*, 2018.

[11] L. Luo, Y. Xiong, and Y. Liu, "Adaptive gradient methods with dynamic bound of learning rate," in *International Conference on Learning Representations*, 2019.

[12] J. Ma and D. Yarats, "Quasi-hyperbolic momentum and adam for deep learning," in *Int'l Conf. on Learning Representations (ICLR)*, 2019.

[13] W. Yuan and K. Gao, "Eadam optimizer: How ϵ impact adam," 2020.

[14] J. Zhuang, T. Tang, Y. Ding, S. C Tatikonda, N. Dvornek, X. Papademetris, and J. Duncan, "Adabelief optimizer: Adapting stepsizes by the belief in observed gradients," *Adv. in neural information proc. syst.*, vol. 33, pp. 18795–18806, 2020.

[15] S. Saab, S. Phoha, M. Zhu, and A. Ray, "An adaptive polyak heavy-ball method," *Machine Learning*, vol. 111, 07 2022.

[16] K. Tessera, S. Hooker, and B. Rosman, "Keep the gradients flowing: Using gradient flow to study sparse network optimization," 2021.

[17] Y. Liu, Y. Gao, and W. Yin, "An improved analysis of stochastic gradient descent with momentum," in *Int'l Conf. on Neural Information Processing Systems*, Red Hook, NY, USA, 2020, Curran Associates Inc.

[18] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

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

[20] M. Tan and Q. Le, "Efficientnetv2: Smaller models and faster training," in *ICML*, 2021, vol. 139, pp. 10096–10106.

[21] D. Choi, C. Shallue, Z. Nado, J. Lee, C. Maddison, and G. Dahl, "On empirical comparisons of optimizers for deep learning," 2019.

[22] P. Rodriguez, "Improving the stochastic gradient descent's test accuracy by manipulating the ℓ_∞ norm of its gradient approximation," in *IEEE ICASSP*, 2023, pp. 1–5.

[23] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[24] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," in *Int'l Conf. on Learning Representations*, 2018.

[25] L. Bottou, "Online algorithms and stochastic approximations," in *Online Learning and Neural Networks*, David Saad, Ed. Cambridge University Press, Cambridge, UK, 1998.

[26] L. Bottou, *Stochastic Gradient Descent Tricks*, pp. 421–436, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[27] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[28] M.H. Hayes, *Statistical Digital Signal Processing and Modeling*, Wiley, 1996.

[29] C. Darken and J. Moody, "Note on learning rate schedules for stochastic optimization," in *Advances in Neural Information Processing Systems*, 1990, vol. 3.

[30] F. Schneider, L. Balles, and P. Hennig, "Deepobs: A deep learning optimizer benchmark suite," in *Int'l Conference on Learning Representations*, 2019.

[31] M. Abadi, A. Agarwal, and et.al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, https://www.tensorflow.org/.

[32] P. Rodriguez, "Simulations for SGD variants," https://hands-on-sgd.readthedocs.io/en/latest/sims.

[33] P. Savarese, D. McAllester, S. Babu, and M. Maire, "Domain-independent dominance of adaptive methods," in *IEEE CVPR*, June 2021, pp. 16286–16295.