Stochastic Runge-Kutta methods and adaptive SGD-G2 stochastic gradient descent

Imen Ayadi, Gabriel Turinici

Stochastic Runge-Kutta methods and adaptive SGD-G2 stochastic gradient descent

Imen AYADI and Gabriel TURINICI
CEREMADE, Universit Paris Dauphine - PSL Research University, Paris, France

correspondence to Gabriel.Turinici@dauphine.fr

February 9, 2020

Abstract

The minimization of the loss function is of paramount importance in deep neural networks. On the other hand, many popular optimization algorithms have been shown to correspond to some evolution equation of gradient flow type. Inspired by the numerical schemes used for general evolution equations we introduce a second order stochastic Runge Kutta method and show that it yields a consistent procedure for the minimization of the loss function. In addition it can be coupled, in an adaptive framework, with a Stochastic Gradient Descent (SGD) to adjust automatically the learning rate of the SGD, without the need of any additional information on the Hessian of the loss functional. The adaptive SGD, called SGD-G2, is successfully tested on standard datasets.

Keywords: Machine Learning, ICML, SGD, stochastic gradient descent, adaptive stochastic gradient, deep learning optimization, neural networks optimization

1 Introduction and related literature

Optimization algorithms are at the heart of neural network design in deep learning. One of the most studied procedures is the fixed step Stochastic Gradient Descent (SGD) [1]; although very robust, SGD may converge too slow for small learning rates or become unstable if the learning rate is too large. Each problem having its own optimal learning rate, there is no general recipe to adapt it automatically; to address this issue, several approaches have been put forward among which the use of momentum [14], Adam [4], RMSprop [16] and so on. On the other hand, recent research efforts have been directed towards finding, heuristically or theoretically, the best learning rate [11, 13, 12] with [17], which uses an estimate of the Lipschitz constant, being a very recent example.

Another interpretation of the minimization procedure it to see it as a time evolution (a flow) in the space $\mathbb{R}^d$ of neural network parameters. Denote such

a parameter by X and let $X \in \mathbb{R}^d \mapsto f(X) \in \mathbb{R}$ be the the loss functional; the flow interpretations recognizes that the minimization of $f(X)$ is related to the solution of the following evolution equation

$$X'(t) = \nabla f(X(t)). \tag{1}$$

In this work we consider the flow (1) and apply two numerical schemes to evolve it in time: the Explicit Euler scheme (which will correspond to the SGD algorithm) and a numerical scheme of second order in time, labeled "SH" (like in "stochastic Heun") belonging to the class of stochastic Runge-Kutta methods; this second scheme allows to have a more precise estimation of the flow and in turn provides essential information to adapt the learning rate of the SGD.

We prove theoretically in section 3 that the SH scheme is indeed of second order; then we explain how it allows to choose the optimal learning rate for the SGD and build the SGD-G2 algorithm. Numerical results on standard datasets (MNIST, F-MNIST, CIFAR10) are presented in section 4 followed by a discussion and concluding remarks.

2 Notations

The fit of the neural networks is formalized through the introduction of a loss functional f depending on the network parameters X and the input data $\omega \in \Omega$ presented to it. In full generality the input data belongs to some probability space $(\Omega, \mathbb{P})$ and the optimization aims to find the value X^{opt} minimizing the mapping $X \mapsto \mathbb{E}_\omega f(\omega, X)$. However, for instance for classification purposes, not all ω have a label attached to it, so in practice only a limited amount of values $\omega_1, ..., \omega_N \in \Omega$ can be used. So the loss functional becomes:

$$f(X) := \frac{1}{N} \sum_{i=1}^{N} f(\omega_i, X). \tag{2}$$

For $i \leq N$ we introduce the functions $f_i = f(\omega_i, \cdot) : \mathbb{R}^d \to \mathbb{R}$ to represent the loss due to the i^{th} training sample ω_i.

To minimize the loss functional one can think of an deterministic procedure (of gradient descent type) which can be written as:

$$X_{n+1} = X_n - h\nabla f(X_n), \tag{3}$$

where $h > 0$ denotes the learning rate (also called "step size"). Note that this update rule requires N gradient evaluations per step which is prohibitively large in applications in deep learning that involve networks with many parameters (tenths of thousands up to billions). To this end, the deterministic procedure is replaced by its stochastic counterpart, the Stochastic Gradient Descent (SGD). Let $(\gamma_n)_{n\geq 1}$ be i.i.d uniform variables taking values in $\{1, 2, .., N\}$. Then, the SGD is defined as:

$$X_{n+1} = X_n - h\nabla f_{\gamma_n}(X_n), \;\; X_0 = X(0). \tag{4}$$

The advantage of the stochastic algorithm is that the gradient is evaluated once per iteration which makes its computational complexity independent of N. This explains why this method is preferred for large data sets.[1]

2.1 The construction of the SGD-G2 algorithm: the principle

The state X_{n+1} in (4) can be also seen as an approximation of the solution $X(t)$ of the flow in (1) at the "time" $t_{n+1} = (n+1)h$: $X_{n+1} \simeq X(t_{n+1})$. But there are many ways to obtain approximations of $X(t_{n+1})$, for instance one can use a second order in h scheme (such as the so-called Runge-Kutta schemes to name but a few [9]) and construct another approximation Y_{n+1} at the price of computing another gradient. If Y_{n+1} is a better approximation then it closer to $X(t_{n+1})$ and thus at the leading order $Y_{n+1} - X_{n+1}$ is an estimation of the error $X_{n+1} - X(t_{n+1})$. With such an approximation one can extrapolate the behavior of f near X_n and compute for what values of the learning rate h we still have stability (the precise computations are detailed in the next section). We adapt then the learning rate to go towards the optimal value, that is, large enough to advance fast but still stable. This will be encoded in the SGD-G2 algorithm we propose. Two questions arise:

- how to design a high order scheme consistent with the equation (1): this is the object of section 3;

- is the numerical procedure performing well in practice : this is the object of section 4.

3 Theoretical results

3.1 Choice of the stochastic Runge-Kutta scheme

First we need to choose a numerical scheme that solves the equation (1) by using only partial information on the samples, i.e., we have to use some stochastic numerical scheme. Among the possible variants we choose the stochastic-Heun method described below (see also [10, 18] for related works, although not with the same goal); while the SGD updates the state by relation (4) the stochastic Heun scheme (named "SH" from now on) reads:

$$\begin{aligned} Y_0 &= X(0) \\ \tilde{Y}_{n+1} &= Y_n - h\nabla f_{\gamma_n}(Y_n) \\ Y_{n+1} &= Y_n - \frac{h}{2}\left[\nabla f_{\gamma_n}(Y_n) + \nabla f_{\gamma_n}(\tilde{Y}_{n+1})\right] \end{aligned} \tag{5}$$

[1] In practice γ_n are drawn without replacement from $\{1, ..., N\}$ until all values are seen. This is called an epoch. Then all values re-enter the choice set and a new epoch begins. We will not take discuss this refinement in our procedures which is independent of the segmentation in epochs or not. Same for mini-batch processing which consists in drawing several γ_k at once; the method proposed in the sequel adapts out-of-the-box to such a situation too.

Note that a step requires two evaluations of the gradient, but this is the price to pay for higher precision. Note also that the same random sample γ_n is used for both gradient computations. In fact SH this can be also seen as a SGD that uses a sample **twice** to advance and then do a linear combination of the gradients thus obtained.

In order to prove relevant properties of SH scheme, we need to make clear some details concerning the evolution equation (1). In fact, since only one sample is used at the time, the evolution X_n will depend on the order in which samples ω_{γ_n} are chosen. This means that in fact there is some randomness involved and we cannot hope to approach exactly the solution $X(t)$ of (1). In fact, see [5, 6], it is known that the output of, let's say, the SGD algorithm is close (in mathematical terms "weakly converging") when $h \to 0$ to the solution of the following stochastic differential equation:

$$dZ_t = b(Z_t)dt + \sigma(Z_t)dW_t,\ Z(0) = X(0), \tag{6}$$

with $b(z) = -\nabla f(z)$ and $\sigma(z) = (h\mathcal{V}(z))^{1/2}$, where

$$\mathcal{V}(z) = \frac{\sum_{k=1}^{N}(\nabla f_{\gamma_k}(z) - \nabla f(z))(\nabla f_{\gamma_k}(z) - \nabla f(z))^T}{N}, \tag{7}$$

is the covariance matrix of $\nabla_X f(\omega, z)$ taken as a random variable of the samples ω (see also [5] equation (4)). Here W_t is a standard Brownian motion. With these provisions we can formally state the following result:

Theorem 1 (Convergence of SGD and SH schemes)**.** *Suppose f, f_k are Lipschitz functions having at most linear increase for $|X| \to \infty$[2]. The SGD scheme converges at (weak) order* 1 *(in h) to the solution Z_t of* (6) *while the SH scheme* (5) *at (weak) order* 2*.*

Proof: Te recall what weak convergence means we need to introduce some notations: we designate by $\mathcal{G}$ the set of function having at most polynomial growth at infinity and $W^{1,\infty}$ the set of Lipschitz functions. Given a numerical scheme U_n of step h (SGD, SH, etc.) that approximates the solution Z_t of the SDE (6) with $U_n \simeq Z_{nh}$, weak convergence at order p means that, for any t (kept fixed) and any function $G \in \mathcal{G}$ we have[3] :

$$|\mathbb{E}G(U_{\lfloor t/n \rfloor}) - \mathbb{E}G(Z_t)| = O(h^p). \tag{8}$$

It is known from [5] (Theorem 1 point "i") that SGD is of weak order 1. It remains to prove that SH is of weak order 2; the proof uses Theorem 2 from the same reference (see also [7]) that is recalled below:

Theorem 2 (Milstein,1986)**.** *Suppose $\forall i \geq 1 : \nabla f, \nabla f_i \in \mathcal{G} \cap W^{1,\infty}$, and have at most linear growth at infinity. Suppose that $Z_0 = U_0 = z \in \mathbb{R}^d$. Let*

[2]The linear growth at infinity is a technical hypothesis. It is for instance true when the parameter domain is closed and bounded and the function continuous.

[3]We used the notation $\lfloor x \rfloor$ to designate the integer part of a real number x, that is the largest integer smaller than x.

$\Delta = (\Delta_1, ..., \Delta_d) = Z_h - z$ *and* $\overline{\Delta} = (\overline{\Delta}_1, ..., \overline{\Delta}_d) = U_1 - z$. *If in addition, there exist* $K_1, K_2 \in \mathcal{G}$ *such that for any* $s \in \{1, 2, ..., 2p+1\}$ *and any* z*:*

$$\left| \mathbb{E}(\prod_{j=1}^{s} \Delta_{i_j}) - \mathbb{E}(\prod_{j=1}^{s} \overline{\Delta}_{i_j}) \right| \leq K_1(x)\ h^{p+1}, \tag{9}$$

and

$$\mathbb{E}(\prod_{j=1}^{2p+1} |\overline{\Delta}_{i_j}|) \leq K_2(x)\ h^{p+1}, \tag{10}$$

the numerical scheme U_n *converges at weak order* p.

We return to the proof of theorem 1. Let L be an operator acting over sufficiently smooth functions $\zeta : \mathbb{R}^d \to \mathbb{R}$ by:

$$L\zeta = - < \nabla f, \nabla \zeta > + \frac{h}{2} \sum_{i,j=1}^{d} \mathcal{V}_{ij} \partial^2_{ij} \zeta. \tag{11}$$

Let $\Phi \in \mathcal{G}$ and suppose it is 6 times differentiable; using a classical result of semi-groups expansions (see [8]), we have:

$$\mathbb{E}(\Phi(Z_h)) = \Phi(z) + hL\Phi(z) + \frac{h^2}{2} L^2\Phi(z) + O(h^3). \tag{12}$$

For $\xi \in \mathbb{R}^d$, we define $\Phi_\xi : y \mapsto e^{i\langle \xi, y-z \rangle}$. Let $\mathcal{M} : \mathbb{R}^d \to \mathbb{R}$ defined by $\mathcal{M}(\xi) = e^{i<\xi, \Delta>}$. Note that the function $\mathcal{M}$ belongs to the class $\mathcal{C}^\infty(\mathbb{R}^d)$ and for $s = 1, .., d$,

$$\left. \frac{\partial^s \mathcal{M}}{\prod_{j=1}^{s} \partial \xi_{i_j}} \right|_{\xi=0} = i^s \mathbb{E}(\prod_{j=1}^{s} \Delta_{i_j}). \tag{13}$$

To determine the partial derivatives of $\mathcal{M}$ in 0 , we use (12) to get:

$$\mathcal{M}(\xi) = 1 + hL\Phi_t(z) + \frac{h^2}{2} L^2\Phi_\xi(z) + O(h^3). \tag{14}$$

After calculating the explicit expressions of $L\Phi_\xi(z)$ and $L^2\Phi_\xi(z)$, we obtain:

$$\begin{aligned} \mathcal{M}(\xi) &= 1 - ih\langle \nabla f(z), \xi \rangle \\ &+ h^2 \left(\frac{i}{2} \sum_{k=1}^{d} \partial_k f(z) \langle \partial_k \nabla f(z), \xi \rangle + \langle \nabla f(z), \xi \rangle^2 - \frac{1}{2} \xi^T \mathcal{V} \xi \right) \\ &+ O(h^3). \end{aligned} \tag{15}$$

Therefore,

$$\mathbb{E}(\Delta_j) = -h\partial_j f(z) + \frac{h^2}{2} \sum_{k=1}^{d} \partial_k f(z) \partial^2_{kj} f(z) + O(h^3) \tag{16}$$

$$\mathbb{E}(\Delta_j \Delta_l) = h^2[\partial_j f(z)\partial_l f(z) + \mathcal{V}_{jl}] + O(h^3) \tag{17}$$

$$\mathbb{E}(\prod_{j=1}^{s} \Delta_{i_j}) = O(h^3), \ for \ s \geq 3. \tag{18}$$

For the SH scheme:

$$E(\overline{\Delta}_k) = -h\partial_k f(z) - \frac{h^2}{2} < \partial_i \nabla f(z), \nabla f(z) > + O(h^3), \tag{19}$$

$$\mathbb{E}(\overline{\Delta}_k \overline{\Delta}_l) = \frac{1}{N} h^2 \sum_{\ell=1}^{N} \partial_k f_\ell(z) \partial_l f_\ell(z) + O(h^3), \tag{20}$$

$$\mathbb{E}(\prod_{j=1}^{s} \overline{\Delta}_{i_j}) = O(h^3), \ \text{for } s \geq 3. \tag{21}$$

Therefore, all hypotheses of the theorem 2 are satisfied for $p = 2$ which gives the conclusion.

Given theorem 1 we can trust SH to produce high order approximations of the solution which in turn will help obtain information on the Hessian and thus calibrate automatically the learning rate.

3.2 Rationale for the adaptive step proposal

In what follows, $\langle X, Y \rangle$ denotes the usual scalar product in $\mathbb{R}^d$ and $\|X\|$ the associated euclidean norm. For a matrix $A \in \mathbb{R}^{d \times d}$, $\|A\|$ denotes the matrix norm defined as: $\|A\| = \sup_{X \in \mathbb{R}^d, X \neq 0} \frac{\|AX\|}{\|X\|}$.

If the loss function f is smooth enough, a Taylor expansion around a current point Y allows to write:

$$\begin{aligned} f(X) &= f(Y) + \langle \nabla f(Y), X - Y \rangle \\ &+ \frac{1}{2} \langle \nabla^2 f(Y)(X - Y), X - Y \rangle + O(\|X - Y\|^3), \end{aligned} \tag{22}$$

where ∇f is the gradient of f computed at Y and $\nabla^2 f(Y)$ is the Hessian matrix of f at the same point.

Note that the Hessian provides detailed information over the behavior of f around the current point Y but in practice $\nabla^2 f(Y)$ is a high dimensional object and it is impossible to deal with it directly. We will not try to compute it but will exploit its structure.

Neglecting higher order terms, the loss functional can be written as:

$$f(X) \simeq \frac{1}{2} \langle AX, X \rangle - \langle b, X \rangle, \tag{23}$$

for some matrix A in $\mathbb{R}^{d\times d}$ and vector b in $\mathbb{R}^d$. To explain our method we will consider that in equation (23) we have equality. Then, $\nabla f(X) = AX - b$. If $X^{opt} \in \mathbb{R}^d$ is the minimum of f, then $AX^{opt} = b$ and thus

$$\nabla f(X) = A(X - X^{opt}). \tag{24}$$

We will forget for a moment that the gradient of f is not computed exactly (only an unbiased estimator being available in practice under the form of ∇f_γ). To minimize the function f, the gradient descent (also called Explicit Euler) scheme with learning rate $h_n > 0$ reads:

$$X_{n+1} = X_n - h_n \nabla f(X_n). \tag{25}$$

Then, denoting I_d the identity matrix of d dimensions, we obtain:

$$X_{n+1} - X^{opt} = (I_d - h_n A)(X_n - X^{opt}). \tag{26}$$

On the other hand the Heun scheme (which is a Runge Kutta method of the second order) reads:

$$Y_{n+1} = Y_n - \frac{h_n}{2}(\nabla f(Y_n) + \nabla f(Y_n - h_n \nabla f(Y_n))). \tag{27}$$

Then,

$$Y_{n+1} - X^{opt} = \left[I_d - hA + \frac{h_n^2 A^2}{2}\right](Y_n - X^{opt}). \tag{28}$$

At the step n, suppose that the two schemes start from the same point i.e., $X_n = Y_n$. Therefore, using (26) and (28):

$$Y_{n+1} - X_{n+1} = \frac{h_n^2 A^2}{2}(X_n - X^{opt}) = \frac{h_n^2 A}{2}\nabla f(X_n) \tag{29}$$

On the other hand from (25) and (27) we can also write:

$$Y_{n+1} - X_{n+1} = \frac{h_n}{2}(\nabla f(X_n) - \nabla f(X_{n+1})) \tag{30}$$

Combining (29) and (30) we get:

$$(I_d - h_n A)\nabla f(X_n) = \nabla f(X_{n+1}). \tag{31}$$

From (24) and (26) the stability criterion for the gradient descent (Explicit Euler) scheme is that $\|\nabla f(X_n)\|$ needs to be bounded, which is verified in particular when $\|I_d - h_n A\| < 1$. If this condition is true then for each step n:

$$\frac{\|(I_d - h_n A)\nabla f(X_n)\|}{\|\nabla f(X_n)\|} < 1. \tag{32}$$

Although the reciprocal is false, it is not far from true because, if $\frac{\|(I_d - h_n A)\nabla f(X_n)\|}{\|\nabla f(X_n)\|} > 1$ for some n, then this means in particular $\|(I_d - h_n A)\| > 1$ (because the matricial norm is a supremum) and then, except degenerate initial conditions X_0,

we obtain that X_n will diverge (unless h_n is adapted to ensure stability). So to enforce stability we have to request:

$$\|\nabla f(X_{n+1})\| \leq \|\nabla f(X_n)\|. \tag{33}$$

On the other hand, at every iteration n, we attempt to choose the biggest possible learning rate that guarantees (33), that is we accelerate the rate of convergence without breaking the stability criterion.

A natural question arises : what is the maximum value of h so that (33) still holds ?

Let us denote $\xi_n(h) = ||(I_d - hA)\nabla f(X_n)||^2$. With X_n being given, this is a second order polynomial in h; let h_n^{opt} be the maximum value of the learning rate h such (33) still holds. In other words, $\xi_n(h_n^{opt}) = \|\nabla f(X_n)\|^2$.

Note that:

$$\begin{aligned}\xi_n(h) &= \|A\nabla f(X_n)\|^2 h^2 \\ &-2\langle A\nabla f(X_n), \nabla f(X_n)\rangle h + \|\nabla f(X_n)\|^2.\end{aligned} \tag{34}$$

Then, $\xi_n(h_n^{opt}) = ||\nabla f(X_n)||^2$ implies that $h_n^{opt} = 0$ or $h_n^{opt} = \frac{2\langle A\nabla f(X_n), \nabla f(X_n)\rangle}{\|A\nabla f(X_n)\|^2}$. Since $(I_d - h_n A)\nabla f(X_n) = \nabla f(X_{n+1})$, we have that $A\nabla f(X_n) = \frac{\nabla f(X_n) - \nabla f(X_{n+1})}{h_n}$. Then, unless $\nabla f(X_n) = \nabla f(X_{n+1})$, which would imply that a critical point has already been reached:

$$h_n^{opt} = \max\left(0, 2h_n \frac{\langle \nabla f(X_n) - \nabla f(X_{n+1}), \nabla f(X_n)\rangle}{||\nabla f(X_n) - \nabla f(X_{n+1})||^2}\right). \tag{35}$$

Note that it is important that in (35) the matrix A, which is impossible to compute, does not appear; only appear $\nabla f(X_n)$ and $\nabla f(X_{n+1})$ that are known.[4].

To conclude: if the current learning rate is h_n then it should be put to h_n^{opt} (given in equation (35)) to have the best convergence and stability properties.

Note that when A is definite positive, the scalar product $p_n := \langle A\nabla f(X_n), \nabla f(X_n)\rangle$ must be positive. Therefore, if in a given iteration n, $\langle \nabla f(X_n) - \nabla f(X_{n+1}), \nabla f(X_n)\rangle$ (which equals $h_n p_n$) happens to be negative this means that the second order assumption (23) made on f breaks down around the current point X_n; we cannot trust any computation made above and thus when $p_n < 0$ we can set for instance $h_n^{opt} = h_n$. Therefore, denoting now h_n the learning rate at step n we will define:

$$h_n^{opt} = \begin{cases} 2h_n \frac{p_n}{||\nabla f(X_n) - \nabla f(X_{n+1})||^2} & \text{if } p_n > 0 \\ h_n & \text{otherwise.} \end{cases} \tag{36}$$

3.3 Update policy

At a given iteration n, if $h_n \ll h_n^{opt}$, this means that the gradient descent is progressing too slow with the current learning rate and thus we need to accelerate

[4]Recall that do not discuss here the stochastic part; in practice an unbiased estimator of the gradient $\nabla f(X)$ is available.

it by increasing the learning rate. In practice, to not break the stability condition, the new learning rate must not be very close to h_n^{opt}. To avoid this risk, we choose a gradual update with an hyper-parameter β close to 1:

$$h_{n+1} = \beta h_n + (1-\beta) h_n^{opt}. \tag{37}$$

Suppose now that $h_n \gg h_n^{opt}$. This means that the current learning rate breaks the convergence criteria. Then, we have to decrease it; contrary to previous policy, here we do not want a slow update because instability is already set in. A drastic measure is required, otherwise the whole optimization may become useless. We propose the following update rule:

$$h_{n+1} = (1-\beta) h_n^{opt}. \tag{38}$$

This update is not necessarily close to h_n but is a conservative choice to enter again the stability region, which, in practice gives good results.

3.4 The SGD-G2 algorithm

We present in this section the algorithm resulting from the above considerations; due to the stochastic nature of the gradient that is to be computed, for each iteration n in all the formulas in the former section the full gradient ∇f must be replaced by ∇f_{γ_n} . Then, our suggested adaptive SGD called "SGD-G2" is described by the Algorithm 1 which includes the provision for mini-batch processing.

Algorithm 1 SGD-G2

Set hyper-parameter: β, mini-batch size M, choose stopping criterion
Input: initial learning rate h_0, initial guess X_0
Initialize iteration counter: $n = 0$
while stopping criterion not met **do**
  select next mini-batch γ_n^m, $m = 1, ..., M$
  Compute $g_n = \frac{1}{M} \sum_{m=1}^{M} \nabla f_{\gamma_n^m}(X_n)$
  Compute $\tilde{g}_n = \frac{1}{M} \sum_{m=1}^{M} \nabla f_{\gamma_n^m}(X_n - h_n g_n)$
  Compute

$$h_n^{opt} = \begin{cases} \frac{2h_n \langle g_n - \tilde{g}_n, g_n \rangle}{\|g_n - \tilde{g}_n\|^2} & \text{if } \langle g_n - \tilde{g}_n, g_n \rangle > 0 \\ h_n & \text{otherwise.} \end{cases}$$

  if $h_n^{opt} \geq h_n$ **then**
    $h_{n+1} = \beta h_n + (1-\beta) h_n^{opt}$
  else
    $h_{n+1} = (1-\beta) h_n^{opt}$
  end if
  Update $X_{n+1} = X_n - h_{n+1} g_n$
  Update $n \to n+1$
end while

Remark 1. *Several remarks are in order:*

1. *The computation of both g_n and $\tilde{g}_n$ allows in principle to construct a more precise, second order in the learning rate, estimate of the next step X_{n+1} of the form $X_n - \frac{h_n}{2}(g_n + \tilde{g}_n)$; this is not what we want here, the precise estimate is only used to calibrate the learning rate, in the end the SGD update formula is invoked to advance the network parameters X_n to X_{n+1}.*

2. *It is crucial to have the same randomness in the computation of $\tilde{g}_n$ as the one present in the computation of g_n. This ensures that a consistent approximation is obtained as detailed in Theorem 1.*

3. *We recommend to take the initial learning rate h_0 very small, in order to be sure to start in the stability region around X_0, for instance $h_0 = 10^{-6}$; however numerical experiments seem to be largely insensitive to this value as detailed in section 4, figure 2.*

4 Numerical experiments

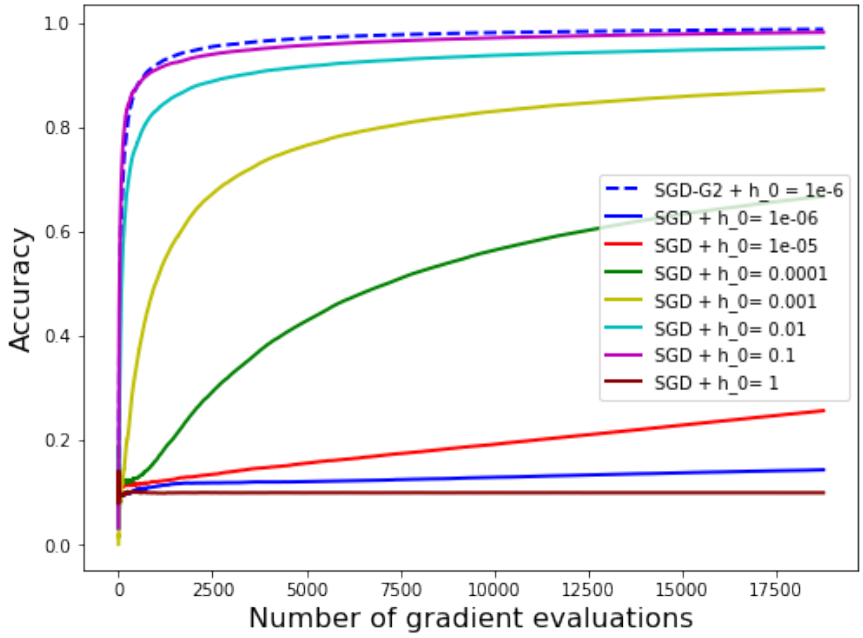

Figure 1: Numerical results for the SGD and SGD-G2 algorithms on the MNIST database. Here $\beta = 0.9$.

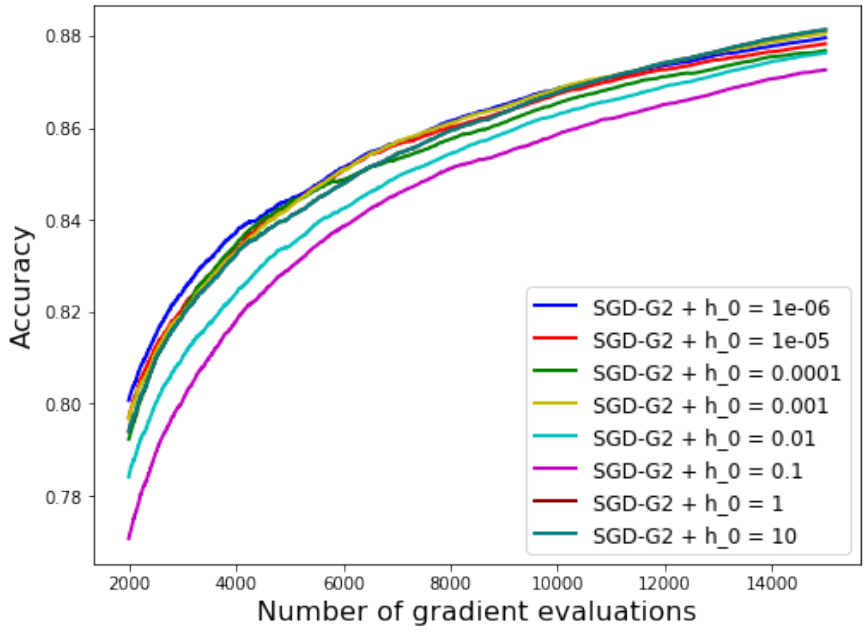

Figure 2: Numerical results for the SGD-G2 algorithm on the FMNIST database with several choices of the initial learning rate h_0. Here $\beta = 0.9$. Similar results are obtained for the MNIST and CIFAR10 databases.

4.1 Network architecture

We conducted experiments on three different data sets (MNIST, Fashion MNIST and CIFAR-10) using neural networks (CNNs) developed for image classification. We follow in this section the specifications in [3, 15] and in [2] and reproduce below the corresponding architectures as given in the reference:
MNIST/Fashion-MNIST (28×28 sized images): three dense 256 neurons feed-forward ReLU layers followed by a final dense 10 neurons feed-forward layer.
CIFAR-10 (32×32 images with 3 color layers): a convolution layer with 3×3 filters, a 2×2 max pooling layer, a convolution layer with 3×3 filters, a 2×2 max pooling layer, a convolution layer with 3×3 filters followed by a flatten layer, a dense layer with a 64 neurons and a final dense layer with 10 neurons. All activations are ReLU except last one which is a softmax.

The last layer returns the classification result.

All hyper-parameters are chosen as in the references: the loss was minimized with the SGD / SGD-G2 algorithms and hyper-parameters $\beta = 0.9$ or as indicated in the figures; we used 10 epochs. The batch size is 32.

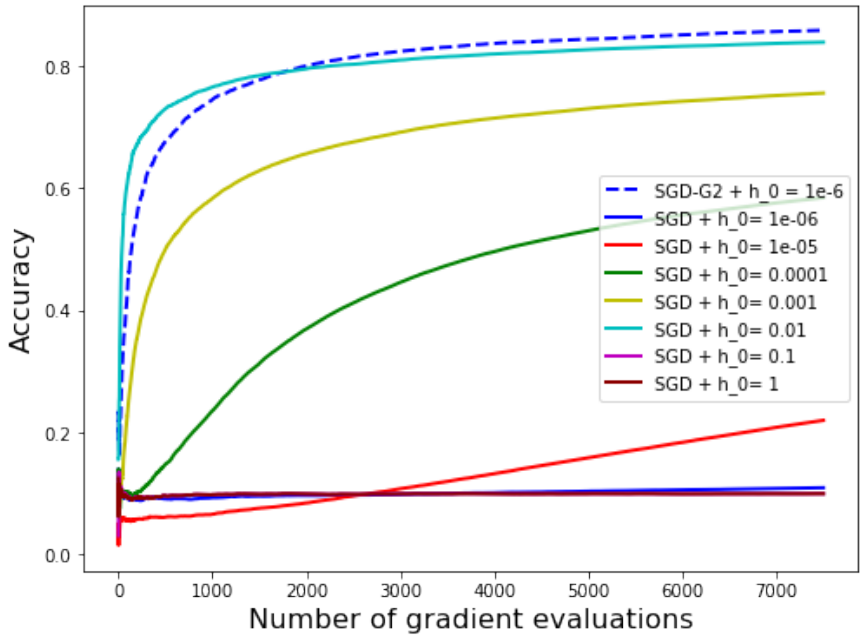

Figure 3: Numerical results for the SGD and SGD-G2 algorithms on the FMNIST database. Here $\beta = 0.9$.

4.2 Discussion

First, we compare the performance of the adaptive SGD algorithm for different choices of the h_0 parameter. The results do not vary much among databases, we plot in figure 2 the ones for FMNIST. The variability is similar for MNIST but slightly larger for CIFAR10.

We come next to the heart of the procedure and we compare the performance of the adaptive SGD algorithm with the standard SGD algorithm for different initial learning rates. Recall that the goal of the adaptive procedure is not to beat the best possible SGD convergence but to identify fast enough the optimal learning rate. We recommend thus to start from a very small value of h_0; the algorithm will increase it up to the stability threshold. This is indeed what happens, see figures 1, 3 and 4. As the adaptive algorithm uses two (mini-batch) gradient evaluations per iteration, we take as x-axis in the plots the number of gradient evaluations and not the iteration counter, (in order not to favor the adaptive procedure which is more costly per iteration). We see that in all situations, starting from a tiny value of h_0, the SGD-G2 algorithm quickly reaches the stability region and converge accordingly.

For MNIST and FMNIST the SGD-G2 cannot be surpassed by SGD, even

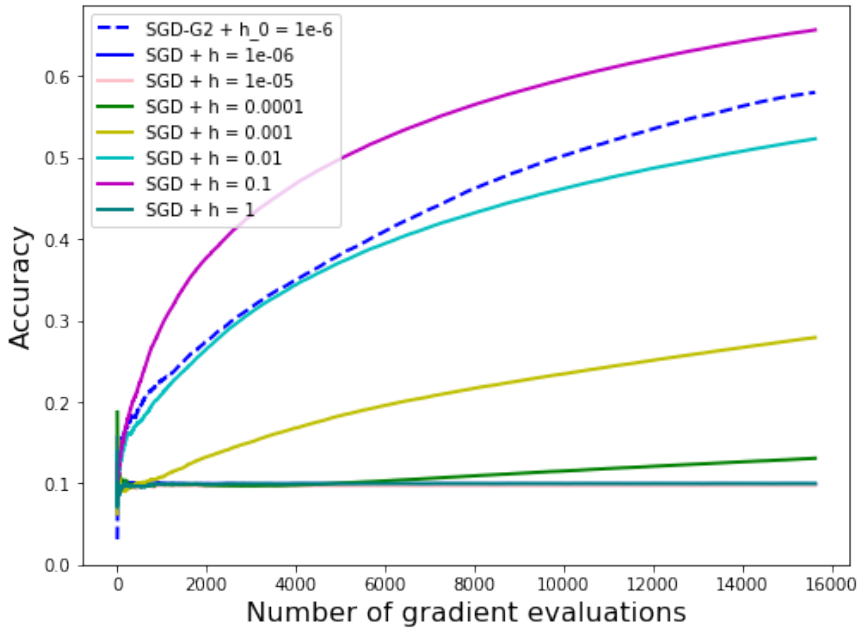

Figure 4: Numerical results for the SGD and SGD-G2 algorithms on the CIFAR10 database. Here $\beta = 0.9$.

for the optimal SGD learning rate; on the contrary for CIFAR10 the SGD with the optimal learning rate (which has to be searched through repeated runs) does converge better than the SGD-G2, but this is due to the counting procedure: if instead of number of gradient evaluations we count the iterations, the two are comparable as shown in figure 5 (same considerations apply for the Adam algorithm as illustrated in figure 6); so one can imagine that the adaptive part is only switched on from time to time (for instance once every 10 iterations), which will make its overhead negligible and still reach the optimal learning rate regime. Such fine tuning remains for future work.

In conclusion, an adaptive SGD algorithm (called SGD-G2) is proposed which is both computationally convenient and provides a stable, optimal learning rate value. The procedure is tested successfully on three image databases (MNIST, FMNIST, CIFAR10).

References

[1] Lon Bottou. Stochastic Gradient Descent Tricks. In Grgoire Montavon, Genevive B. Orr, and Klaus-Robert Mller, editors, *Neural Networks: Tricks*

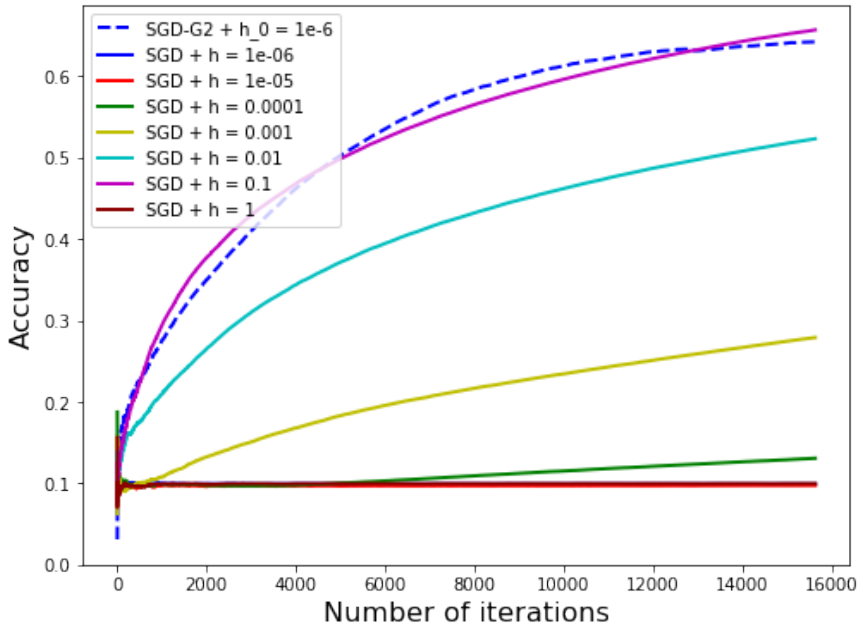

Figure 5: Numerical results for the SGD and SGD-G2 algorithms on the CIFAR10 database. Here $\beta = 0.9$, and we compare in number of iterations instead of gradient evaluations.

of the Trade: Second Edition, Lecture Notes in Computer Science, pages 421–436. Springer, Berlin, Heidelberg, 2012.

[2] Feiyang Chen, Nan Chen, Hanyang Mao, and Hanlin Hu. Assessing four neural networks on handwritten digit recognition dataset (mnist), 11 2018. arXiv:1811.08278.

[3] Akinori Hidaka and Takio Kurita. Consecutive dimensionality reduction by canonical correlation analysis for visualization of convolutional neural networks. *Proceedings of the ISCIE International Symposium on Stochastic Systems Theory and its Applications*, 2017:160–167, 12 2017.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. arXiv:1412.6980.

[5] Qianxiao Li, Cheng Tai, and Weinan E. Stochastic modified equations and adaptive stochastic gradient algorithms. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages

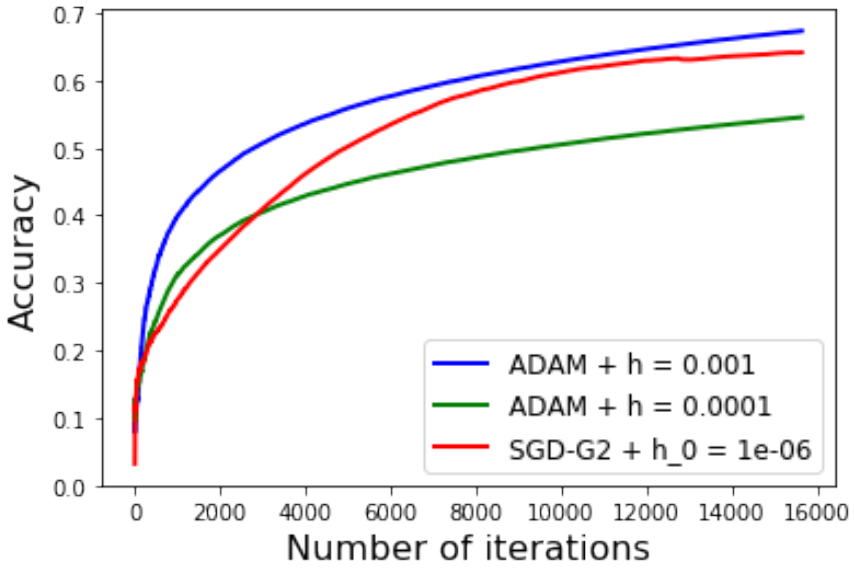

Figure 6: Numerical results for the SGD-G2 algorithm and the ADAM algorithm on the CIFAR10 database. Here, $\beta = 0.9$.

2101–2110, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[6] Qianxiao Li, Cheng Tai, and Weinan E. Stochastic Modified Equations and Dynamics of Stochastic Gradient Algorithms I: Mathematical Foundations. *Journal of Machine Learning Research*, 20(40):1–47, 2019.

[7] G. N. Milshtein. Weak Approximation of Solutions of Systems of Stochastic Differential Equations. *Theory of Probability & Its Applications*, 30(4):750–766, December 1986.

[8] E. Hille and R. S. Phillips. *Functional Analysis and Semi-groups*. American Mathematical Society, Providence, revised edition edition, February 1996.

[9] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *"Numerical Recipes 3rd Edition: The Art of Scientific Computing"*. Cambridge University Press, 2007.

[10] Lars Ruthotto and Eldad Haber. Deep Neural Networks Motivated by Partial Differential Equations. *arXiv:1804.04272 [cs, math, stat]*, December 2018. arXiv: 1804.04272.

[11] Sihyeon Seong, Yekang Lee, Youngwook Kee, Dongyoon Han, and Junmo Kim. Towards Flatter Loss Surface via Nonmonotonic Learning Rate

Scheduling. In *UAI2018 Conference on Uncertainty in Artificial Intelligence*. Association for Uncertainty in Artificial Intelligence (AUAI), 2018.

[12] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

[13] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. *arXiv preprint arXiv:1708.07120*, 2017.

[14] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[15] TensorFlow. Convolutional Neural Network (CNN) TensorFlow Core, retrieved 2020-02-06. https://www.tensorflow.org/tutorials/images/cnn.

[16] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[17] Rahul Yedida and Snehanshu Saha. Lipschitzlr: Using theoretically computed adaptive learning rates for fast convergence, 2019.

[18] Mai Zhu, Bo Chang, and Chong Fu. Convolutional Neural Networks combined with Runge-Kutta Methods. *arXiv:1802.08831 [cs]*, January 2019. arXiv: 1802.08831.