Automatic hyperparameter tuning in on-line learning: Classic Momentum and ADAM

Paweł Wawrzyński, Paweł Zawistowski, Łukasz Lepak
Institute of Computer Science
Warsaw University of Technology
Nowowiejska 15/19, 00-665 Warsaw, Poland
pawel.wawrzynski@pw.edu.pl, pawel.zawistowski@pw.edu.pl, lukasz.lepak.stud@pw.edu.pl

***Abstract*—We propose a method that adapts hyperparameters, namely step-sizes and momentum decay factors, in on-line learning with classic momentum and ADAM. The approach is based on the estimation of the short- and long-term influence of these hyperparameters on the loss value. In the experimental study, our approach is applied to on-line learning in small neural networks and deep autoencoders. Automatically tuned coefficients surpass or roughly match the best ones selected manually in terms of learning speed. As a result, on-line learning can be a fully automatic process, producing results from the first run, without preliminary experiments aimed at manual hyperparameter tuning.**

***Index Terms*—online optimization, Stochastic Gradient Descent, hyperparameter tuning, neural networks, Classic Momentum, ADAM**

I. Introduction

In this paper, we consider the typical setting for on-line learning: we wish to optimize a parameter, $\theta \in \mathbb{R}^d$, of a learning system. For each time step, there exists a known (momentary) loss function $J(\theta, \xi)$, where ξ denotes a randomly generated data sample. The goal of learning is to find the point $\theta^* \in \mathbb{R}^d$ for which the global loss function

$$\bar{J}(\theta) = EJ(\theta, \xi) \tag{1}$$

attains its minimum. We assume that only the gradient of the momentary loss function, $\nabla_\theta J(\theta, \xi)$, is available, which is an unbiased estimate of the (unavailable) global loss gradient $\nabla \bar{J}(\theta)$.

Most fundamental methods of on-line learning, like stochastic gradient descent [1] (SGD) and classic momentum [2] (CM) require hyperparameters called step-sizes and momentum decay factors that generally depend on the problem and process stage.

In practice, on-line learning is usually conducted by selecting the aforementioned parameters using trial-and-error, which is time consuming and not satisfying. There have been attempts, like AdaGrad [3] or ADAM [4], to design an algorithm that does not depend on any manually tuned hyperparameters, or at least works well with the defaults. However, these algorithms also require step-sizes and momentum decay factors, and their default values do not guarantee good performance for all learning problems.

In this paper, we design an algorithm that optimizes the step-size and the momentum decay factor in CM and ADAM while these methods are running. The algorithm presented here extends and refines the ASDM method presented by Wawrzynski [5]. The focus is put on a novel method of analyzing the long-term influence of the parameters on the momentary loss value. This method overcomes important ASDM deficiencies.

In addition, the analysis of the short-term influence of the step-size and the momentum decay factor on a momentary value of θ introduced in the previous paper for CM is here extended to ADAM. The resulting learning algorithm has been analyzed experimentally and compared in simulations with CM, accelerated gradient, ADAM, AdaGrad, and AdaDelta. All algorithms are applied to 14 different learning problems and two parameter settings.

The main contributions of this paper are thus as follows:

- we propose to estimate the initial step-size in ASDM using the estimate of the largest eigenvalue of the Hessian,
- we propose to define a quality measure for the algorithm's hyperparameters' adjustments based on their influence on the long-term performance of the learning process,
- we introduce indicators of optimization instability and propose heuristics to ensure that the process remains stable,
- we conduct extensive experiments to test the approach against current state-of-the art methods.

The paper is organized as follows. The next section presents related work. In Sec. III, the formal definition of the considered problem is given. In Sec. IV, the influence of hyperparameters on the learning process is derived, while Sec. V presents the resulting algorithm. Sec. VI reports the experimental study with on-line learning in feedforward neural networks. The last section concludes the paper.

II. Related work

The earliest methods of on-line learning include SGD, which has its roots in the aforementioned works of Robbins and Monro [1] and Kiefer et al. [6]. Two other classic approaches, which build upon the concepts of SGD and are directly related to this paper, are classic momentum [2, 7] (CM) and accelerated gradient [8] (AG). Both CM and AG are based on the concept of extracting a velocity component

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

from the iterative update procedure utilized by SGD; they only differ in the way this component gets updated.

The CM approach, which is formally defined in Sec. III of this paper, has been thoroughly analyzed by Qian [9], where the convergence bounds for the learning rate and momentum parameters have been analyzed. Bhaya and Kaszkurewicz [10] present a view on CM from the control perspective on the conjugate gradient algorithm [11] and perform an analysis in terms of Lyapunov stability to choose the learning rate and momentum parameters.

There have been multiple attempts to create algorithms which adapt the momentum term to improve CM performance. These include using one dimensional minimisation to estimate this term [12].

Swanston et al. [13] propose to use the angle between the previous update steps and the current one to determine the magnitude of the momentum term.

Graepel and Schraudolph [14] utilize a curvature matrix connected with a forgetting factor to formulate the Stable Adaptive Momentum approach.

The back-propagation Gradient Descent Adaptive Momentum Algorithm [15] (GDAM) adapts the momentum while maintaining a fixed learning rate.

Hameed et al. [16] utilize eigenvalues of the autocorrelation matrix of the optimized model's inputs to adapt momentum in a back-propagation algorithm.

YellowFin [17] is an approach that adapts the learning rate and momentum term by approximating the curvature using single-dimensional noisy quadratic models.

In order to get rid of free parameters, the classic methods have also been combined with gradient normalization in multiple approaches. AdaGrad [3] scales the gradient descent step at time t with an inversed diagonal matrix constructed using a cumulative sum of gradient products from times $1, \ldots, t$. However, it requires setting a global learning rate manually and results in a decay of learning rates as the optimization progresses. The AdaDelta method [18] addresses these issues by restricting the cumulative sum to a time window of fixed size and utilizing a diagonal Hessian approximation to obtain correct units for the parameter update vectors. A similar remedy to the decaying learning rates problem present in AdaGrad was suggested for RMSProp [19], which also suggests applying a running average of the gradient magnitude to scale the global learning rate.

ADAM and ADAMAX [4] are currently among the most widely used optimization algorithms that utilize first and second-order moment corrections in their parameter update rules. A version incorporating Nesterov momentum into ADAM is introduced by Dozat [20] and called NADAM.

Although ADAM is a robust and widely used approach in the field of deep learning models, there exist cases in which it is not able to converge [21]. This has been tackled by the AMSGRAD [21] approach. Furthermore, AdaBound and AmsBound methods [22] try to address the problem (signalled by Wilson et al. [23]) of extreme learning rates which occurs in, respectively, ADAM and AMSGRAD and slows down the optimization process in the long run.

To the best of our knowledge, the only method that adjusts the momentum decay factor on-line in CM was introduced by Schraudolph and Graepel [24], where a stochastic version of the conjugate gradient algorithm is presented.

In recent years, tremendous progress has been made in the field of online convex optimization, which considers a simplified version of the problem analyzed here in which $\bar{J}$ is convex. One interesting branch of relevant research is connected with meta-descent methods like the stochastic meta-descent algorithm [25] (SMD) which adapts the learning rates in successive iterations using an exponentiated gradient descent controlled by a global learning rate.

The hypergradient descent method [26] (HD) applies gradient descent directly on the learning rate parameters in an online fashion. The L^4 adaptation scheme [27] is a meta-algorithm which calculates step size at every iteration using a lineariztion of the loss function (inspired by root-finding Newton's method).

The limitations of current meta-optimization approaches are explored by Wu et al. [28]. The authors argue that short time horizons typically used in meta-optimization settings lead to using too small learning rates — this poses a significant challenge in that area of research.

III. Notation and problem formulation

In the below equations, t denotes discrete time, $\theta_t \in \mathbb{R}^d$ is the optimized vector, $m_t \in \mathbb{R}^d$ is an auxiliary vector called momentum, $\beta_t > 0$ is the step-size, $\lambda_t \in (0, 1)$ denotes the momentum decay factor, the symbol $\circ$ denotes the Hadamard (elementwise) product, and $s_t \in \mathbb{R}^d$ is a vector that normalizes different coordinates of θ.

Additionally, we denote $g(\theta, \xi) = \nabla_\theta J(\theta, \xi)$. In the context of feedforward neural network training, θ is a vector of neural weights, ξ_t is a data sample (an input–output pair or a minibatch of these), and $g(\theta, \xi)$ is computed by means of gradient backpropagation.

We wish to find the minimum of $\bar{J}$ (1) using the following procedure:

$$\begin{aligned} m_t &= \lambda_t m_{t-1} - \beta_t g(\theta_t, \xi_t) \circ s_t \\ \theta_{t+1} &= \theta_t + m_t, \; t = 1, 2, \ldots \end{aligned} \tag{2}$$

In original CM, the elements of s_t are uniformly equal to 1. The ADAM algorithm implements (2) with s_t containing the inverses of square roots of average squares of elements of $g(\theta_{t-i}, \xi_{t-i}), i \geq 0$.

The problem considered here is how to tune β_t and λ_t on the run of procedure (2) to make it most efficient.

IV. Quality index for optimization of β and λ

The approach to β and λ optimization in the course of learning can be summarized as follows:

- the influence of previous values of β and λ on the current θ_t and the exponentially smoothed θ_t, denoted

by $\bar{\theta}_t$, is analyzed; the parameters of smoothing also undergo constant optimization,
- the values of β and λ are being incrementally adjusted in the direction, that if these parameters had been pushed before, then current positions of θ_t and $\bar{\theta}_t$ would be better.

The idea of optimization of the current loss, $J(\theta_t, \xi_t)$, by manipulating previous values of β and λ is not new. However, when applied directly, it yields small β and λ, since their small values are the best way to suppress random fluctuations of θ_t thereby minimizing the current loss. But small β and λ lead to slow learning in the long term. Therefore, we analyze how β and λ influence $\bar{\theta}_t$, whose improvement rate roughly reflects the long-term speed of learning.

Technically the aggregation of the impact of β_i and λ_i on future values of θ_t is done by means of an operator, $\mathcal{S}_\gamma$, which is defined for $\gamma \in (0, 1]$ as follows

$$\mathcal{S}_\gamma \frac{\mathrm{d}v}{\mathrm{d}\alpha_k} = \sum_{i \le k} \gamma^{k-i} \frac{\mathrm{d}v}{\mathrm{d}\alpha_i}. \tag{3}$$

It is a discounted sum of derivatives of the same value v with respect to parameters α_i, in which the weight of a specific derivative decreases with growing $k - i$. Short-term influence of β_k, λ_k on $m_t, \theta_{t+1}, t \ge k$ can be expressed in the following recursive equations:

$$\mathcal{S}_\gamma \frac{\mathrm{d}m_t}{\mathrm{d}\lambda_t} = m_{t-1} + \lambda_t \gamma \mathcal{S}_\gamma \frac{\mathrm{d}m_{t-1}}{\mathrm{d}\lambda_{t-1}} - \beta_t \gamma \frac{\partial g_t}{\partial \theta_t} \mathcal{S}_\gamma \frac{\mathrm{d}\theta_t}{\mathrm{d}\lambda_{t-1}} \circ s_t \tag{4}$$

$$\mathcal{S}_\gamma \frac{\mathrm{d}\theta_{t+1}}{\mathrm{d}\lambda_t} = \gamma \mathcal{S}_\gamma \frac{\mathrm{d}\theta_t}{\mathrm{d}\lambda_{t-1}} + \mathcal{S}_\gamma \frac{\mathrm{d}m_t}{\mathrm{d}\lambda_t}, \tag{5}$$

$$\mathcal{S}_\gamma \frac{\mathrm{d}m_t}{\mathrm{d}\beta_t} = \lambda_t \gamma \mathcal{S}_\gamma \frac{\mathrm{d}m_{t-1}}{\mathrm{d}\beta_{t-1}} - g_t \circ s_t - \beta_t \gamma \frac{\partial g_t}{\partial \theta_t} \mathcal{S}_\gamma \frac{\mathrm{d}\theta_t}{\mathrm{d}\beta_{t-1}} \circ s_t \tag{6}$$

$$\mathcal{S}_\gamma \frac{\mathrm{d}\theta_{t+1}}{\mathrm{d}\beta_t} = \gamma \mathcal{S}_\gamma \frac{\mathrm{d}\theta_t}{\mathrm{d}\beta_{t-1}} + \mathcal{S}_\gamma \frac{\mathrm{d}m_t}{\mathrm{d}\beta_t}. \tag{7}$$

For brevity, g_t is written in here in place of $g(\theta_t, \xi_t)$. The above equations can be derived following the original work of Wawrzynski [5].

A. Influence of β and λ on the trend in θ_t

Here we analyze the influence of β and λ on the long-term movement of θ_t. In this order, we define $\bar{\theta}_t$ as a smoothed θ_t, namely

$$\bar{\theta}_1 = \theta_1, \quad \bar{\theta}_{t+1} = \mu \bar{\theta}_t + (1 - \mu)\theta_t, \tag{8}$$

where $\mu \in [0, 1)$ is optimized to minimize $J(\bar{\theta}_t, \xi_t)$.

Let us visualize the optimization process as a descent down a multidimensional valley between steep slopes. The goal is to descend in the direction of the bottom of this valley and $\bar{\theta}_t$ plays the role of the projection of θ_t onto its bottom. We will optimize β_t and λ_t to approximately minimize a linear combination of $J(\theta_t, \xi_t)$ and $J(\bar{\theta}_t, \xi_t)$. For this purpose we analyze how fast $\bar{\theta}_t$ is moving depending on β and λ.

Using the $\mathcal{S}_\gamma$ operator, it is easy to quantify the influence of β and λ on $\bar{\theta}_t$ (8), namely

$$\mathcal{S}_\gamma \frac{\mathrm{d}\bar{\theta}_{t+1}}{\mathrm{d}\beta_t} = \mu\gamma \mathcal{S}_\gamma \frac{\mathrm{d}\bar{\theta}_t}{\mathrm{d}\beta_{t-1}} + (1 - \mu)\gamma \mathcal{S}_\gamma \frac{\mathrm{d}\theta_t}{\mathrm{d}\beta_{t-1}}, \tag{9}$$

$$\mathcal{S}_\gamma \frac{\mathrm{d}\bar{\theta}_{t+1}}{\mathrm{d}\lambda_t} = \mu\gamma \mathcal{S}_\gamma \frac{\mathrm{d}\bar{\theta}_t}{\mathrm{d}\lambda_{t-1}} + (1 - \mu)\gamma \mathcal{S}_\gamma \frac{\mathrm{d}\theta_t}{\mathrm{d}\lambda_{t-1}}. \tag{10}$$

For $\bar{\theta}_t$ to play its role in the projection of θ_t on the valley's bottom, $\bar{J}(\bar{\theta}_t)$ needs to be minimized with respect to μ. In order to adjust μ in the course of learning with CM, we notice that from (8) we have

$$\frac{\mathrm{d}\bar{\theta}_{t+1}}{\mathrm{d}\mu} = \bar{\theta}_t - \theta_t + \mu \frac{\mathrm{d}\bar{\theta}_t}{\mathrm{d}\mu}, \tag{11}$$

and then

$$\frac{\mathrm{d}J(\bar{\theta}_t, \xi_t)}{\mathrm{d}\mu} = \frac{\mathrm{d}J(\bar{\theta}_t, \xi_t)}{\mathrm{d}\bar{\theta}_t} \frac{\mathrm{d}\bar{\theta}_t}{\mathrm{d}\mu}. \tag{12}$$

The above derivative will indicate the direction of incremental adjustments of μ. In order to compute $\bar{\theta}_t - \theta_t$ in (11) conveniently we define

$$\phi_1 = 0, \quad \phi_{t+1} = m_t + \mu\phi_t. \tag{13}$$

A simple induction combining (8) and (13) reveals that $\bar{\theta}_t - \theta_t = -\phi_t$.

B. Optimization of β and λ

In order to maintain the desired trend in θ_t both in the short and long-term we propose here to minimize (with respect to β and λ) the following quality index

$$Q_t = J(\bar{\theta}_t, \xi_t) + r_t^T(\theta_t - \theta_{t-1}). \tag{14}$$

where $r_t = g(\bar{\theta}_t, \xi_t)$ is treated as a constant.

The first term of the sum in (14) is crucial as it directly leads to decreasing the loss value observed for $\bar{\theta}_t$. However, a second term needs to be included in order to penalize the fluctuations of consecutive θ_t values. This penalty is for the increment θ_t along the gradient of the loss function, thereby increasing the loss.

Then, we propose recognizing how infinitesimal changes of β_k, λ_k for $k < t$ would influence (14), which can be quantified as, respectively,

$$\mathcal{S}_\gamma \frac{\mathrm{d}Q_t}{\mathrm{d}\beta_{t-1}} = g(\bar{\theta}_t, \xi_t)^T \left(\mathcal{S}_\gamma \frac{\mathrm{d}\bar{\theta}_t}{\mathrm{d}\beta_{t-1}} + \mathcal{S}_\gamma \frac{\mathrm{d}m_{t-1}}{\mathrm{d}\beta_{t-1}} \right), \tag{15}$$

$$\mathcal{S}_\gamma \frac{\mathrm{d}Q_t}{\mathrm{d}\lambda_{t-1}} = g(\bar{\theta}_t, \xi_t)^T \left(\mathcal{S}_\gamma \frac{\mathrm{d}\bar{\theta}_t}{\mathrm{d}\lambda_{t-1}} + \mathcal{S}_\gamma \frac{\mathrm{d}m_{t-1}}{\mathrm{d}\lambda_{t-1}} \right). \tag{16}$$

Next, β and λ are adjusted in the directions opposite to the above derivatives.

V. Algorithm ASDM2

Algorithm 1 is based on pillars presented in the previous sections. When the symbol "$\leftarrow$" is used, the full assignment is given on the right side of the symbol. The notation introduced above is generally preserved in the algorithm but simplified e.g., the subscript t is omitted. The algorithm has two versions:

1 Assign 0 to all estimators
2 Initialize θ
3 Assign $t \leftarrow 1$, set any $\beta \neq 0$
4 (If needed) Update s
5 $\frac{\mathrm{d}Q}{\mathrm{d}\beta} \leftarrow g(\bar{\theta}, \xi_t)^T \left(\mathcal{S}_\gamma \frac{\mathrm{d}\bar{\theta}}{\mathrm{d}\beta} + \mathcal{S}_\gamma \frac{\mathrm{d}m}{\mathrm{d}\beta} \right)$
6 $\frac{\mathrm{d}Q}{\mathrm{d}\lambda} \leftarrow g(\bar{\theta}, \xi_t)^T \left(\mathcal{S}_\gamma \frac{\mathrm{d}\bar{\theta}}{\mathrm{d}\lambda} + \mathcal{S}_\gamma \frac{\mathrm{d}m}{\mathrm{d}\lambda} \right)$
7 $\frac{\mathrm{d}\bar{J}}{\mathrm{d}\mu} \leftarrow g(\bar{\theta}, \xi_t)^T \frac{\mathrm{d}\bar{\theta}}{\mathrm{d}\mu}$
8 **if** $t \leq t_0$ **then**
9 $\quad \beta \leftarrow \frac{1}{2} \left(\frac{t-1}{t} \frac{1}{2} \beta^{-1} + \frac{1}{t} \left\| \frac{\partial g(\theta, \xi_t)}{\partial \theta} g_t \right\| / \|g_t\| \right)^{-1}$
10 $\quad \alpha \leftarrow \ln \beta$
11 $\quad \lambda \leftarrow 0.5$
12 $\quad \eta \leftarrow -\ln(1-\lambda)$
13 **else**
14 $\quad \alpha \leftarrow \alpha - \delta \frac{\mathrm{d}Q}{\mathrm{d}\beta} / \sqrt{\mathcal{A}(\frac{\mathrm{d}Q}{\mathrm{d}\beta})^2}$
15 $\quad \beta \leftarrow \exp(\alpha)$
16 $\quad \eta \leftarrow \eta - \delta \frac{\mathrm{d}Q}{\mathrm{d}\lambda} / \sqrt{\mathcal{A}(\frac{\mathrm{d}Q}{\mathrm{d}\lambda})^2}$
17 $\quad \eta \leftarrow \max\{-\ln(1-\lambda_{\min}), \eta\}$
18 $\quad \lambda \leftarrow 1 - \exp(-\eta)$
19 $\quad \nu \leftarrow \nu - \delta \frac{\mathrm{d}\bar{J}}{\mathrm{d}\mu} / \sqrt{\mathcal{A}(\frac{\mathrm{d}\bar{J}}{\mathrm{d}\mu})^2}$
20 $\quad \mu \leftarrow 1 - \exp(-\nu)$
21 $\mathcal{S}_\gamma \frac{\mathrm{d}g}{\mathrm{d}\beta} \leftarrow \frac{\mathrm{d}g(\theta, \xi_t)}{\mathrm{d}\theta} \mathcal{S}_\gamma \frac{\mathrm{d}\theta}{\mathrm{d}\beta}$
22 $\mathcal{S}_\gamma \frac{\mathrm{d}g}{\mathrm{d}\lambda} \leftarrow \frac{\mathrm{d}g(\theta, \xi_t)}{\mathrm{d}\theta} \mathcal{S}_\gamma \frac{\mathrm{d}\theta}{\mathrm{d}\lambda}$
23 **if** $t > 1$ **then**
24 $\quad \alpha_0 \leftarrow \ln \left(\frac{1}{2} \min \left\{ \frac{\|\mathcal{S}_\gamma \mathrm{d}\theta/\mathrm{d}\beta\|}{\|\mathcal{S}_\gamma \mathrm{d}g/\mathrm{d}\beta\|}, \frac{\|\mathcal{S}_\gamma \mathrm{d}\theta/\mathrm{d}\lambda\|}{\|\mathcal{S}_\gamma \mathrm{d}g/\mathrm{d}\lambda\|} \right\} \right)$
25 $\quad$ **if** $\alpha > \alpha_0$ **then**
26 $\quad\quad \alpha \leftarrow \alpha - 2\delta$
27 $\quad\quad \beta \leftarrow \exp(\alpha_0)$
28 $\quad \gamma \leftarrow \min \left\{ 1, C \frac{\mathcal{A}\|g\|^2}{\|\mathcal{S}_\gamma \frac{\mathrm{d}\theta}{\mathrm{d}\beta}\|^2}, C \frac{\mathcal{A}\|m\|^2}{\|\mathcal{S}_\gamma \frac{\mathrm{d}\theta}{\mathrm{d}\lambda}\|^2} \right\}$
29 $\quad$ **if** $\lambda > \gamma$ **then**
30 $\quad\quad \eta \leftarrow \eta - 2\delta$
31 $\quad\quad \lambda \leftarrow \gamma$
32 Update $\bar{\theta}$ with (8)
33 Update $\mathcal{S}_\gamma \frac{\mathrm{d}\bar{\theta}}{\mathrm{d}\beta}$ with (9)
34 Update $\mathcal{S}_\gamma \frac{\mathrm{d}\bar{\theta}}{\mathrm{d}\lambda}$ with (10)
35 Update $\frac{\mathrm{d}\bar{\theta}}{\mathrm{d}\mu}$ with (11)
36 Update $\mathcal{S}_\gamma \frac{\mathrm{d}m}{\mathrm{d}\beta}$ with (6)
37 Update $\mathcal{S}_\gamma \frac{\mathrm{d}m}{\mathrm{d}\lambda}$ with (4)
38 Update m and θ with (2)
39 Update ϕ with (13)
40 Update $\mathcal{S}_\gamma \frac{\mathrm{d}\theta}{\mathrm{d}\beta}$ with (7)
41 Update $\mathcal{S}_\gamma \frac{\mathrm{d}\theta}{\mathrm{d}\lambda}$ with (5)
42 $\mathcal{A}(\frac{\mathrm{d}Q}{\mathrm{d}\beta})^2 \leftarrow w_t^\rho \mathcal{A}(\frac{\mathrm{d}Q}{\mathrm{d}\beta})^2 + (1-w_t^\rho)(\frac{\mathrm{d}Q}{\mathrm{d}\beta})^2$
43 $\mathcal{A}(\frac{\mathrm{d}Q}{\mathrm{d}\lambda})^2 \leftarrow w_t^\rho \mathcal{A}(\frac{\mathrm{d}Q}{\mathrm{d}\lambda})^2 + (1-w_t^\rho)(\frac{\mathrm{d}Q}{\mathrm{d}\lambda})^2$
44 $\mathcal{A}(\frac{\mathrm{d}J}{\mathrm{d}\mu})^2 \leftarrow w_t^\rho \mathcal{A}(\frac{\mathrm{d}J}{\mathrm{d}\mu})^2 + (1-w_t^\rho)(\frac{\mathrm{d}J}{\mathrm{d}\mu})^2$
45 $\mathcal{A}\|g\|^2 \leftarrow w_t^\rho \mathcal{A}\|g\|^2 + (1-w_t^\rho)\|g_t\|^2$
46 $\mathcal{A}\|m\|^2 \leftarrow w_t^\rho \mathcal{A}\|m\|^2 + (1-w_t^\rho)\|m\|^2$
47 Increment t and go to Line 4

Algorithm 1: Autonomous stochastic descent with momentum version 2: ASDM2.

- ASDM2/b is based on CM and applies no gradient normalization: s_t is the vector of 1s ("/b" stands for "bare"),
- ASDM2/n is based on ADAM i.e., CM with gradient normalization (hence "/n").

The core of the algorithm is Line 38, where the basic adjustment of m and θ is done. The algorithm utilizes several technicalities such as (i) initialization of β based on a rough estimate of the inverse of the largest eigenvalue of the Hessian $\nabla^2 \bar{J}$ (Lines 3 and 9), (ii) re-parameterization of β, λ, and μ (Lines 10, 12, 15, 18, 20, 27) (iii) keeping β, λ, and γ small enough to prevent abnormal operation of the algorithm (Lines 25–28), and (iv) exponential smoothing of some estimates (Lines 42–46). All the aforementioned technicalities are discussed below.

A. Initialization of β and λ

According to [5], a good initial β would be the inverse of the largest eigenvalue of the global loss function Hessian i.e.,

$$\beta \approx \left(\max_{v \in \mathbb{R}^d} \|\nabla^2 \bar{J}(\theta) v\| / \|v\| \right)^{-1}. \tag{17}$$

As it is not possible to determine that value, we notice that

$$\nabla^2 \bar{J}(\theta) = E \nabla^2 J(\theta, \xi) = E \frac{\partial g(\theta, \xi)}{\partial \theta} \tag{18}$$

and set β_t equal to half the inverse of the average of values maximized in (17) with $v = g_i$, namely

$$\beta_t = \frac{1}{2} \left(\frac{1}{t} \sum_{i=1}^{t} \left\| \frac{\partial g(\theta_i, \xi_i)}{\partial \theta_i} g_i \right\| / \|g_i\| \right)^{-1} \tag{19}$$

for $t \leq t_0$, where $t_0 > 0$ determines how long the initial stage of learning lasts. For $t \leq t_0$, we set $\lambda_t \equiv 0.5$. See Lines 8–12, where β_t is calculated recursively on the basis of β_{t-1} and (19).

B. Re-parameterization and adjusting β, λ, and μ

We want to adjust β, λ, and μ with increments of controlled magnitude, regardless of the actual scale of β, $1-\lambda$, and $1-\mu$. In this order, the parameters α, η, and ν are introduced and the following equivalence is established

$$\beta = \exp(\alpha), \quad \alpha = \ln(\beta), \tag{20}$$

$$\lambda = 1 - \exp(-\eta), \quad \eta = -\ln(1-\lambda), \tag{21}$$

$$\mu = 1 - \exp(-\nu), \quad \nu = -\ln(1-\mu). \tag{22}$$

Technically, these are α, η, and ν that are incrementally adjusted. Since the updates are normalized, α, η, and ν are updated on average $\pm\delta$, thus β, $1-\lambda$, and $1-\mu$ are updated by factor $(1 \pm \delta)$, where δ is a meta-stepsize. See Lines 10, 12, 15, 18, 20, and 27.

Effectively, the step-size β and the momentum decay factor λ are adjusted according to the discussion in Sec. IV-B. The main steps of these operations are in Lines 5, 6, 14, and 16. The supplementary steps are in Lines 32–34, 35–37, 40–41.

Additionally, in order to locate $\bar{\theta}_t$ properly, the μ parameter needs to be adjusted. That happens in Lines 7, 19, and 35.

C. Keeping β, λ, and γ small enough

As discussed above, the learning becomes unstable when the step-size β becomes significantly larger than

$$\|v\|/\|\nabla^2 \bar{J}(\theta_t)v\| \tag{23}$$

for a certain $v \in \mathbb{R}^d$. We utilize the fact that $\nabla^2 J(\theta_t, \xi_t)$ is multiplied by a vector in (6) and (4). Namely, early signs of instability are detected when β is larger than

$$\beta_0 = \frac{1}{2}\min\left\{ \frac{\left\|\mathcal{S}_\gamma \frac{\mathrm{d}\theta_t}{\mathrm{d}\beta_{t-1}}\right\|}{\left\|\frac{\partial g_t}{\partial \theta_t}\mathcal{S}_\gamma \frac{\mathrm{d}\theta_t}{\mathrm{d}\beta_{t-1}}\right\|}, \frac{\left\|\mathcal{S}_\gamma \frac{\mathrm{d}\theta_t}{\mathrm{d}\lambda_{t-1}}\right\|}{\left\|\frac{\partial g_t}{\partial \theta_t}\mathcal{S}_\gamma \frac{\mathrm{d}\theta_t}{\mathrm{d}\lambda_{t-1}}\right\|} \right\}. \tag{24}$$

When that happens (Line 25), β_0 is used instead of β (Line 27), and β is decreased (Line 26).

Both γ and λ are exponential decay factors that determine the memory lengths of certain estimators. The term $\mathcal{S}_\gamma(\mathrm{d}\theta_{t+1}/\mathrm{d}\lambda_t)$ captures the influence of the previous values of λ_i on θ_{t+1}. The memory length defined by γ should be at least as large as the memory length defined by λ. Therefore, whenever $\gamma < \lambda$, λ is being decreased (Lines 29, 30). Also, λ is kept equal to at least $\lambda_{\min}$, as very small values of this parameter hardly ever work well in practice.

It can be seen from (7) and (5) that $\mathcal{S}_\gamma(\mathrm{d}\theta_{t+1}/\mathrm{d}\lambda_t)$ and $\mathcal{S}_\gamma(\mathrm{d}\theta_{t+1}/\mathrm{d}\beta_t)$ are adjusted additively by g_t and m_t, respectively. Considering the limited precision of floating point numbers, $\|\mathcal{S}_\gamma(\mathrm{d}\theta_{t+1}/\mathrm{d}\lambda_t)\|$ and $\|\mathcal{S}_\gamma(\mathrm{d}\theta_{t+1}/\mathrm{d}\beta_t)\|$ can not be too many orders of magnitude larger than average $\|g_t\|$ and $\|m_t\|$, respectively.

We apply γ small enough to assure

$$\|\mathcal{S}_\gamma(\mathrm{d}\theta_{t+1}/\mathrm{d}\lambda_t)\|^2 \le C \cdot \text{average}\|g_t\|^2$$
$$\|\mathcal{S}_\gamma(\mathrm{d}\theta_{t+1}/\mathrm{d}\beta_t)\|^2 \le C \cdot \text{average}\|m_t\|^2,$$

where C is a large constant depending on the representation of numbers. It expresses how many times one number can be larger than another with their addition still being effective. In our experiments, we set $C = 10^8$.

D. Exponential smoothing

Let $x_t, t = 1, 2, 3, ...$ be a sequence. Let $\mathcal{A}x_t$ be the exponentially moving average of (x_t), namely

$$\mathcal{A}x_t = \frac{\sum_{i=0}^{t-1} \rho^i x_{t-i}}{\sum_{i=0}^{t-1} \rho^i} \tag{25}$$

for $\rho \in (0, 1]$. Elementary transformations reveal that

$$\mathcal{A}x_t = w_t^\rho \mathcal{A}x_{t-1} + (1 - w_t^\rho)x_t \tag{26}$$

for $w_t^\rho = \rho(1 - \rho^{t-1})/(1 - \rho^t)$. Exponential smoothing is applied in Lines 42–46 of ASDM2. The terms $\mathcal{A}(\frac{\mathrm{d}Q}{\mathrm{d}\beta})^2$, $\mathcal{A}(\frac{\mathrm{d}Q}{\mathrm{d}\lambda})^2$, and $\mathcal{A}(\frac{\mathrm{d}\bar{J}}{\mathrm{d}\mu})^2$ play the role of scalar variables in the algorithm.

E. Gradient normalization

If the algorithm applies gradient normalization (Line 4), the normalization takes the following form

$$\mathcal{A}(g \circ g)_i \leftarrow w_t^\rho \mathcal{A}(g \circ g)_i + (1 - w_t^\rho) g_i(\theta_t, \xi_t)^2, \tag{27}$$
$$s_i \leftarrow 1/\sqrt{\mathcal{A}(g \circ g)_i + \epsilon}, , \tag{28}$$

where $\epsilon > 0$ is a coefficient that prevents division by zero, ρ is a decay factor (like $\rho = 0.999$), and subscript i denotes the i-th coordinate of the vector.

F. Coefficients

The algorithm requires several coefficients to be provided. Their values applied in the experiments discussed below are based mainly on common-sense, and are as follows: $\epsilon = 10^{-8}$, $\delta = 0.0005$, $\lambda_{\min} = 0.5$, $\rho = 0.999$, $t_0 = 10$, and $C = 10^8$. On several occasions, a certain value is set below a certain threshold (Line 24) or above a certain value (Line 26, 30). In all such cases, "below" is twice smaller, and "above" is twice larger.

The initial μ is set equal to 0.99.

While all these coefficients may undergo some optimization, their default values give good performance over diverse testbed learning tasks.

VI. Experimental study

This section reports experiments with the algorithms presented in the previous section. The algorithms are tested in three settings: firstly, training shallow neural classifiers for 10 arbitrary classification problems from the UCI Machine Learning Repository [29], secondly, creating three classic deep dense autoencoders [30], and thirdly a convolutional autoencoder for CIFAR-10 dataset, taken from [31].

The two new algorithms, namely ASDM2/b and ASDM2/n, are compared with the following known ones: CM, AG, ADAM, AdaGrad, and AdaDelta. The known algorithms are considered in two settings: with optimized parameters e.g., CM/o, and with default parameters e.g., ADAM/d. The optimized parameters are the momentum decay factor and the step-size selected from the Cartesian product $\{0.9, 0.99\} \times \{\dots, 0.1, 0.05, 0.02, 0.01, \dots\}$ such that they give the smallest ultimate loss. The default parameters are ones applied by Tensorflow when their values are not provided when calling the algorithm.

A. Optimization tasks' details

In case of the shallow neural classifiers, we take 10 arbitrary classification problems from the UCI Machine Learning Repository [29]. They are listed in the first part of Tab. I. For each, we build a neural classifier with a single hidden, logistic sigmoidal layer, and a linear output layer. The number of neurons in the output layer is equal to the number of classes, and the number of hidden neurons is roughly optimized in preliminary experiments aiming to minimize the test error. Initial weights of the hidden neurons are drawn from normal distribution $N(0, \sigma^2)$, where $\sigma = 1/\sqrt{\dim(\text{input})}$. Networks inputs are scaled with their means and standard deviations.

TABLE I

BASIC PARAMETERS OF ANALYZED LEARNING PROBLEMS. ABR — PROBLEM NAME ABBREVIATION, SIZE — NUMBER OF SAMPLES, IDIM — INPUT DIMENSION, ODIM — OUTPUT DIMENSION, NCNT — HIDDEN LAYER SIZE, MBS — MINI-BATCH SIZE, LEN — NUMBER OF SAMPLES PROCESSED BEFORE TERMINATION (I.E., A RUN TAKES LEN/MBS STEPS).

Problem	Abr	Size	Idim	Odim	Ncnt	Mbs	Len
Credit card defaults	CCard	30000	32	2	21	200	10^7
Dota2 games results	Dota2	102944	116	2	55	200	10^7
HTRU2	Htru2	17898	8	2	34	200	10^7
Sensorless drive	Motor	58509	48	11	89	200	10^7
Poker hand	Poker	1025010	10	10	89	200	10^7
Robot navigation	Robot	5456	24	4	34	200	10^7
Statlog shuttle	Shuttle	58000	9	7	34	200	10^7
Skin segmentation	Skin	245057	3	2	21	200	10^7
Spambase	Spam	4601	57	2	34	200	10^7
Theorem proving	Theo	6118	51	6	89	200	10^7
Curves	Curves	20000	784	784	—	200	10^8
Handwritten Digits	MNIST	60000	784	784	—	200	10^8
Olivetti Faces	Faces	165600	625	625	—	200	10^8
CIFAR10 autoencoder	CF10ae	50000	3072	3072	—	200	$2{\cdot}10^7$

Required outputs are one-hot vectors. The loss reported is mean-square error.

Experiments connected with deep autoencoders are based on three datasets containing grayscale images. They are fed to a dense autoencoder neural network. The task for the network is to produce output equal to input. Hidden layers of the networks have the following sizes:

- Curves: 400, 200, 100, 50, 25, 6, 25, 50, 100, 200, 400,
- MNIST: 1000, 500, 250, 30, 250, 500, 1000,
- Faces: 2000, 1000, 500, 30, 500, 1000, 2000.

All layers in the networks are logistic sigmoidal, with the exception of bottleneck layers and the output layer in Faces, which are linear. Sparse weights initialization is applied. Details of the experimental setting are adopted from [32] and [33]. The loss reported is mean-square error.

Images from CIFAR10 dataset are fed to a convolutional autoencoder. All details of the network and training procedure are taken from [31].

Table II shows how optimal step-sizes for CM, AG, ADAM, AdaGrad, and AdaDelta may differ for various problems. The differences reach four orders of magnitude.

B. Software and computational complexity

Our experimental software has been written in two versions: in C++ with CUDA, and in Python with Tensorflow. The Python version is freely available on https://github.com/Bestest96/ASDM2-TF. The most computationally expensive operations in Algorithm 1 are gradient back-propagation performed twice in each loop step and Hessian—vector multiplication, also performed twice. The latter operation is slightly more expensive than the first one. Consequently, ASDM2 needs 3-3.5 times more real time per loop step than CM applied to the same problem.

TABLE II

STEP-SIZES AND MOMENTUM DECAY FACTORS OPTIMAL FOR EACH PROBLEM. SS — STEP-SIZE, MDF — MOMENTUM DECAY FACTOR.

Alg.	CM		AG		ADAM		AGrad	ADelta	
Problem	ss	mdf	ss	mdf	ss	mdf	ss	ss	mdf
CCard	0.5	0.9	0.02	0.99	0.05	0.9	0.2	1	0.99
Dota2	0.05	0.99	0.05	0.99	0.002	0.99	0.2	1	0.99
Htru2	0.02	0.99	0.05	0.99	0.05	0.9	0.2	1	0.99
Motor	0.02	0.99	0.05	0.99	0.002	0.99	0.05	1	0.9
Poker	0.01	0.99	0.05	0.99	0.005	0.9	0.2	1	0.99
Robot	0.2	0.99	0.1	0.99	0.05	0.9	0.2	1	0.99
Shuttle	0.2	0.99	0.1	0.99	0.02	0.9	0.2	1	0.99
Skin	0.05	0.99	0.1	0.99	0.05	0.9	0.5	1	0.99
Spam	0.2	0.99	0.1	0.99	0.02	0.9	0.2	1	0.99
Theo	0.01	0.99	0.02	0.99	0.01	0.9	0.2	1	0.99
Curves	0.01	0.9	0.002	0.9	0.002	0.9	0.02	0.5	0.99
MNIST	0.02	0.9	0.02	0.9	0.002	0.9	0.02	1	0.99
Faces	0.001	0.99	0.001	0.99	0.0002	0.99	0.01	0.5	0.99
CF10ae	10^{-5}	0.99	10^{-5}	0.99	0.001	0.9	0.005	0.5	0.99

C. Results

The results are depicted in Tab. III in the form of average losses attained at the end of training. The table also presents the accuracy of errors in the form of standard deviations. Only training losses are reported, as here we focus on optimization rather than the quality of models that could be demonstrated by test losses. The smallest losses for each problem are indicated by bold face font. The observations made are summarized below.

1) In most cases (12 cases out of 14) the winner is either form of ASDM2.
2) In most cases (12/14), the manual optimization of the parameters of ADAM yield significant improvement of its behavior. However, its default parameters make that algorithm perform well in comparison to others, except all variants of ASDM2.
3) ASDM2/n may be understood as ADAM with β and λ tuned on-the-fly by the method introduced here. In (13/14) cases, ASDM2/n outperformed ADAM/d (with default parameters). In (12/14) cases, ASDM2/n also outperformed ADAM/o (with optimized parameters).
4) The performance of AdaGrad and AdaDelta is especially disappointing. Those algorithms were presented as a way to optimize the step-size on-the-fly in SGD and CM, respectively. That does not check out in our experiments.

Table IV presents parameters β, λ, γ, and μ that ASDM2 was using in the middle of training. Note that these parameters were being constantly adjusted. The β and λ parameters determined by the algorithm may be compared to those optimized manually, which are depicted in Tab. II. Analogies are rather vague, but do exist. ASDM2 adjusts the parameters to the current stage of the learning process, and what is seen in Tabs. IV are points in certain trajectories. The γ and μ parameters are close to 1, as was expected.

In general, the ASDM2 algorithm in its various forms yields encouraging results. Usually it reaches its goal, which is

TABLE III

FINAL LOSS ESTIMATES OBTAINED BY AVERAGING 10 INDEPENDENT RUNS AND REPORTING THE MEAN LOSS VALUE. THE ACCURACY IS DEFINED AS THE STANDARD DEVIATION OF THE SAMPLE MEAN. /D — DEFAULT STEP-SIZE AND MOMENTUM DECAY FACTOR, /O — OPTIMIZED ONES. THE ACCURACY IS PRESENTED WITH THE SAME NUMBER OF DIGITS AFTER THE DECIMAL POINT BUT WITHOUT LEADING ZEROS E.G., 15.35 ±31 DENOTES 15.35 ±0.31, AND 0.0055 ±4 DENOTES 0.0055±0.0004.

Alg.	CM	AG	ADAM		AdaGrad	AdaDelta		ASDM2	
Problem			/d	/o		/d	/o	/b	/n
CCard	0.262 ±1	0.264 ±1	0.262 ±1	0.260 ±1	0.266 ±1	0.342 ±2	0.265 ±1	0.257 ±1	**0.256 ±1**
Dota2	0.422 ±1	0.418 ±1	0.412 ±1	0.394 ±1	0.405 ±1	0.500 ±1	0.423 ±1	0.414 ±1	**0.379 ±1**
Htru2	0.0326±7	0.0300±3	0.0313±5	0.0293±5	0.0294±4	0.1033±16	0.0321±5	0.0307±2	**0.0291±4**
Motor	0.0642±10	0.0480±8	0.0668±90	0.0415±5	0.0829±9	0.802 ±16	0.0956±13	0.0614±24	**0.0413±7**
Poker	0.451 ±10	0.315 ±13	0.448 ±15	0.327 ±3	0.477 ±7	0.569 ±1	0.524 ±3	0.499 ±6	**0.302 ±3**
Robot	**0.0642±21**	0.0908±23	0.1198±14	0.1064±12	0.1270±13	0.6178±19	0.1251±17	0.0843±32	0.0925±44
Shuttle	0.0064±7	0.0045±3	0.0084±3	**0.0038±2**	0.0093±3	0.2768±41	0.0135±4	0.0100±3	0.0059±2
Skin	0.0062±2	0.0055±1	0.0074±1	0.0043±2	0.0074±2	0.2573±75	0.0089±2	0.0053±1	**0.0037±2**
Spam	0.0245±10	0.0230±3	0.0279±3	0.0172±4	0.0257±3	0.4270±56	0.0390±5	0.0243±3	**0.0171±3**
Theo	0.442 ±2	0.409 ±1	0.430 ±2	0.393 ±1	0.440 ±1	0.720 ±1	0.454 ±1	0.408 ±1	**0.381 ±2**
Curves	0.178 ±2	0.312 ±16	0.158 ±4	0.157 ±3	0.444 ±8	15.930±1	0.303 ±5	0.129 ±1	**0.109 ±1**
MNIST	1.03 ±1	1.03 ±1	1.27 ±6	1.12 ±1	2.11 ±1	15.35 ±31	1.31 ±1	**0.90 ±1**	0.92 ±1
Faces	15.99 ±3	16.81 ±38	15.32 ±25	14.61 ±2	39.63 ±32	103.26±9	19.73 ±3	**14.45 ±5**	15.76 ±8
CF10ae	5.11 ±12	3.75 ±8	2.29 ±2	2.29 ±2	4.98 ±9	23.75 ±7	2.69 ±4	3.73 ±10	**2.22** ±2

TABLE IV

AVERAGE (OVER TRAINING RUNS) VALUES OF β, λ, γ, AND μ IN THE MIDDLE OF TRAINING.

Alg.	ASDM2/b				ASDM2/n			
Problem	β	λ	γ	μ	β	λ	γ	μ
CCard	0.33	0.92	1	0.997	0.0024	0.81	1	0.997
Dota2	0.24	0.93	0.9999	0.998	0.0024	0.74	0.9994	0.9994
Htru2	0.19	0.96	0.9998	0.995	0.0025	0.89	0.9999	0.995
Motor	0.11	0.93	0.998	0.995	0.0011	0.76	0.995	0.996
Poker	0.076	0.93	1	0.997	0.00022	0.97	0.9997	0.996
Robot	0.26	0.97	0.9998	0.997	0.0033	0.84	1	0.996
Shuttle	0.20	0.86	0.9999	0.990	0.00048	0.89	0.998	0.985
Skin	0.31	0.98	0.9992	0.993	0.0028	0.96	0.9994	0.994
Spam	0.22	0.97	0.9996	0.997	0.0019	0.78	0.9999	0.996
Theo	0.11	0.94	1	0.997	0.0013	0.71	0.9992	0.996
Curves	0.001	0.996	0.9995	0.998	3.2e-5	0.97	0.9998	0.9991
MNIST	0.016	0.91	0.9999	0.9995	7.5e-5	0.87	1	0.9994
Faces	0.004	0.94	1	0.9996	3.4e-5	0.84	0.9998	0.9996
CF10ae	4.2e-5	0.94	0.997	0.97	2.5e-5	0.98	0.994	0.998

approximate optimization on-the-fly of the β and λ parameters for CM, AG, and ADAM. Consequently, ASDM2 yields good speed of learning from the first run. However, in some cases, the algorithm does not provide optimal β or λ, and those cases are especially interesting. Such cases are Robot and Shuttle.

What is conspicuous about those problematic cases in Tab. IV are the relatively small values of γ. While for the sake of numeric accuracy (see Sec. V-C), it does not make sense to set γ equal to 1, there is a systematic reason why the larger that parameter is, the better. Rough interpretation of the terms $\mathcal{S}_\gamma \mathrm{d}J(\bar{\theta}_t, \xi_t)/\mathrm{d}\beta_{t-1}$ and $\mathcal{S}_\gamma \mathrm{d}J(\bar{\theta}_t, \xi_t)/\mathrm{d}\lambda_{t-1}$ is as follows: ASDM2 at each t looks at what would happen with $J(\bar{\theta}_t, \xi_t)$ if β_i and λ_i had been larger in preceding $(1-\gamma)^{-1}$ steps. Therefore, if the algorithm is forced to set small γ, it has only a myopic view on the influence of β and λ on the learning process. That, in turn, leads to suboptimal values of β and λ. How to enable large γ for all problems and learning stages is a curious research topic.

VII. CONCLUSIONS

The manual tuning of hyperparameters in on-line learning slows research down since it requires a whole process to be repeated many times. It also makes many potential applications of machine learning unavailable, since in most cases one can not tell the user to go pick the right hyperparameters by trial and error.

In this paper, a step has been made to get rid of "pesky" hyperparameters such as step-size and momentum decay factor, and to be able to get results of on-line learning after a single run. A method was introduced that adjusts those hyperparameters in CM and ADAM. The method is based on the recognition of the short- and long-term influence of the hyperparaemters on the learning process. The hyperparameters are tuned to make the process fast yet stable. The method does not depend on any preliminary knowledge of the learning problem, thereby making on-line learning a process that needs to be run only once. In the experimental study, the method was applied to shallow neural networks, as well as deep auto-encoders, and in most cases it performed better than any manually selected hyperparameters.

ACKNOWLEDGMENT

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

REFERENCES

[1] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Matchematical Statistics*, vol. 22, pp. 400–407, 1951.

[2] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational*

Mathematics and Mathematical Physics, vol. 4, pp. 1–17, 1964.

[3] 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.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *CoRR*, vol. abs/1412.6980, 2014.

[5] P. Wawrzynski, "ASD+M: Automatic parameter tuning in stochastic optimization and on-line learning," *Neural Networks*, vol. 96, pp. 1–10, 2017.

[6] J. Kiefer, J. Wolfowitz *et al.*, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, no. 323, pp. 533–536, 1986.

[8] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o(1/sqr(k))," *Soviet Mathematics Doklady*, vol. 27, pp. 372–376, 1983.

[9] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, 1999.

[10] A. Bhaya and E. Kaszkurewicz, "Steepest descent with momentum for quadratic functions is a version of the conjugate gradient method," *Neural Networks*, vol. 17, pp. 65–71, 2004.

[11] T. A. Straeter, "On the extension of the davidon-broyden class of rank one, quasi-newton minimization methods to an infinite dimensional hilbert space with applications to optimal control problems," 1971.

[12] G. Qiu, M. Varley, and T. Terrell, "Accelerated training of backpropagation networks by using adaptive momentum step," *Electronics letters*, vol. 28, no. 4, pp. 377–379, 1992.

[13] D. Swanston, J. Bishop, and R. J. Mitchell, "Simple adaptive momentum: new algorithm for training multilayer perceptrons," *Electronics Letters*, vol. 30, no. 18, pp. 1498–1500, 1994.

[14] T. Graepel and N. N. Schraudolph, "Stable adaptive momentum for rapid online learning in nonlinear systems," in *International Conference on Artificial Neural Networks*. Springer, 2002, pp. 450–455.

[15] M. Z. Rehman and N. M. Nawi, "The effect of adaptive momentum in improving the accuracy of gradient descent back propagation algorithm on classification problems," in *International Conference on Software Engineering and Computer Systems*. Springer, 2011, pp. 380–390.

[16] A. A. Hameed, B. Karlik, and M. S. Salman, "Back-propagation algorithm with variable adaptive momentum," *Knowledge-Based Systems*, vol. 114, pp. 79–87, 2016.

[17] J. Zhang and I. Mitliagkas, "Yellowfin and the art of momentum tuning," *arXiv preprint arXiv:1706.03471*, 2017.

[18] M. D. Zeiler, "Adadelta: An adaptive learning rate method," in *arXiv:1212.5701*, 2012.

[19] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," 2012.

[20] T. Dozat, "Incorporating Nesterov momentum into Adam," in *ICLR*, 2016.

[21] L. Luo, Y. Xiong, and Y. Liu, "Adaptive gradient methods with dynamic bound of learning rate," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg3g2R9FX

[22] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," *arXiv preprint arXiv:1902.09843*, 2019.

[23] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4148–4158.

[24] N. Schraudolph and T. Graepel, "Towards stochastic conjugate gradient methods," in *Proceedings of the 9th International Conference on Neural Information Processing*, 2002, pp. 853–856.

[25] N. N. Schraudolph, "Local gain adaptation in stochastic gradient descent," 1999.

[26] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood, "Online learning rate adaptation with hypergradient descent," *arXiv preprint arXiv:1703.04782*, 2017.

[27] M. Rolinek and G. Martius, "L4: Practical loss-based stepsize adaptation for deep learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 6433–6443.

[28] Y. Wu, M. Ren, R. Liao, and R. Grosse., "Understanding short-horizon bias in stochastic meta-optimization," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=H1MczcgR-

[29] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[30] A. P. George and W. B. Powell, "Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming," *Machine Learning*, vol. 65, pp. 167–198, 2006.

[31] P. Wawrzynski, "Efcient on-line learning with diagonal approximation of loss function hessian," in *IJCNN*, 2019.

[32] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, pp. 504–507, 2006.

[33] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *ICML*, vol. 28, no. 3. JMLR Workshop and Conference Proceedings, 2013, pp. 1139–1147.