Efficient on-line learning with diagonal approximation of loss function Hessian

Paweł Wawrzyński
Institute of Computer Science
Warsaw University of Technology
Warsaw, Poland
pawel.wawrzynski@pw.edu.pl

***Abstract*—The subject of this paper is stochastic optimization as a tool for on-line learning. New ingredients are introduced to Nesterov's Accelerated Gradient that increase efficiency of this algorithm and determine its parameters that are otherwise tuned manually: step-size and momentum decay factor. In this order a diagonal approximation of the Hessian of the loss function is estimated. In the experimental study the approach is applied to various types of neural networks, deep ones among others.**

***Index Terms*—on-line learning, accelerated gradient, parameter autotuning, deep learning.**

I. Introduction

The general problem of stochastic optimization is considered here [1]: There exists a loss function, $J : \Re^n \mapsto \Re$, whose value and gradient are not given, and only estimates of its gradient are available. Namely, a generator produces data samples on the basis of which vectors are computed, whose expected values are equal to the gradient of the loss function. This problem is commonly encountered in machine learning. A system e.g., a neural network, is to behave efficiently on average, but a single instant of its operation only provides a clue how the system could behave better in this instant, which is an estimate of the average behavior improvement.

One of algorithms applicable in on-line learning is Accelerated Gradient (AG) [2]. As most such algorithms, its application to a given problem requires setting coefficients, namely the step-size and the momentum decay factor, that are problem dependent and fundamentally impact overall training efficiency. Generally the speed of the learning process is increasing with those parameters, hence they should be as large as possible. However, when they are too large they make the process unstable.

In this paper a bound for step-sizes in AG is derived above which the learning process becomes unstable. This bound is related to the largest eigenvalue of the loss function's Hessian. A method of estimation of a diagonal approximation of the Hessian is introduced here. Those results lead to a family of on-line learning algorithms. The first of them extends the Accelerated Gradient such that it applies the approximation of the Hessian to determine the largest step-size and the momentum decay factor that are just below the stability bounds. Another algorithm within this family introduces scaling of gradients.

The reminder of the paper is organized as follows. The next section presents related work. In Sec. III formal definition of the problem considered here is given. In Sec. IV a simplified stochastic optimization problem is analyzed. Sec. VI discusses the design of a stochastic optimization algorithm inspired by Newton's method. Sec. VII presents the family of algorithms. The following section reports the experimental study with multiple learning problems. The last section concludes the paper.

II. Related work

Most popular methods of stochastic optimization and on-line learning are stochastic gradient descent (SGD) [3], classic momentum (CM) [4]–[7], and accelerated gradient (AG) [2]. All these methods require a coefficient called step-size. Also, CM and AG require another coefficient, called momentum decay factor. Those coefficients fundamentally impact the optimization process, and their right values are generally different for different problems. Providing those coefficients is easier when the gradients are scaled. Scaling of gradients for SGD is applied in AdaGrad [8] and AdaDelta [9], and scaling them for CM is applied in ADAM [10].

Another approach to on-line learning is based on the Extended Kalman Filter (EKF) [11]. In principle, it is parameterless and efficient, but its computational complexity is overproportional in number of optimized parameters. Therefore, EKF is in fact not applicable in deep learning, and thus not popular nowadays.

In recent years, tremendous progress has been made in the field of online convex optimization [12] which considers a simplified version of the problem analyzed here in which J is convex. [13] introduced MetaGrad that runs several instances of SGD with different step-sizes and aggregates their results. That method achieves logarithmic regret bound on the unregularized hinge loss. It is generally parameter-free but requires a maximum Lipschitz constant of the losses. The paper [14] introduced a method that does not need any initial knowledge of the function being optimized, neither it requires any parameters, but that function needs to be α-acutely convex. In every step the method minimizes a certain approximation

This work was supported in part by the FlexNet project: "Flexible IoT Networks for Value Creators" and it was co-funded in part by the National Centre for Research and Development in Poland under the EUREKA 2018 Programme. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

978-1-7281-2009-6/$31.00 ©2019 IEEE

of the J function based on all gradient estimates registered so far.

The Hessian of the loss function has been considered in [15] and [16] where the loss meant the error of neural function approximation. The diagonal of the loss function's Hessian was derived there from the structure of the neural approximator.

There have been several attempts to design a method of computing step-sizes during the run of SGD. These methods include *delta-delta* [17] and *delta-bar-delta* [18]. A method of [19] computes step-sizes for SGD with variance reduction [20]. In some methods step-sizes are defined separately for each training sample to assure Lyapunov stability of the learning process [21], [22]. Methods of [23] and [24] vary step-sizes on the basis of the concept of terminal attractors.

Many techniques of adaptive step-size estimation were designed for specific applications. Those include approximate dynamic programming [25], recursive least-squares approximation [26], stochastic variational inference [27], and online linear optimization [28].

To the authors' best knowledge the only methods that adjust on-line the momentum decay factor in the classic momentum have been introduced in [29] and [30].

III. Problem formulation

The general issue of stochastic optimization is considered to be defined as follows. There is a loss function,

$$J : \Re^n \mapsto \Re. \tag{1}$$

Its gradient, ∇J, is not available but a generator is available that produces independent, identically distributed random samples, ξ, that are applied to compute unbiased estimators of ∇J. Namely, a function g is given such that

$$E[g(\theta, \xi)] = \nabla J(\theta). \tag{2}$$

The random element in the above expectation is ξ. In the context of neural networks θ denotes neural weights, ξ denotes data samples, and $g(\theta, \xi)$ is computed by means of gradient backpropagation.

In the below equations t denotes discrete time, θ_t denotes points in the domain, m_t is an auxiliary vector called momentum, $\beta_t > 0$ is the step-size, $\mu_t \in (0, 1)$ denotes the momentum decay factor.

The currently best known procedure to find minimum of J is the classic momentum (CM) [4], [5], namely

$$\begin{aligned} m_t &= \mu_t m_{t-1} - \beta_t g(\theta_t, \xi_t) \\ \theta_{t+1} &= \theta_t + m_t,\ t = 0, 1, 2, \dots \end{aligned} \tag{3}$$

A little less known procedure is the accelerated gradient (AG) [2], namely

$$\begin{aligned} m_t &= \mu_t m_{t-1} - \beta_t g(\theta_t + \mu_t m_{t-1}, \xi_t) \\ \theta_{t+1} &= \theta_t + m_t,\ t = 0, 1, 2, \dots \end{aligned} \tag{4}$$

CM and AG are similar. CM first computes the gradient, and pushes θ_t along the momentum, and the gradient. AG uses the gradient computed at the point to which θ_t is already pushed by the momentum. In words, AG uses more up-to-date gradient, therefore it is usually a little faster than CM.

In this paper we consider a procedure that is a generalization of (4), namely

$$\begin{aligned} m_t &= \mu_t m_{t-1} - b_t \circ g(\theta_t + \mu_t m_{t-1}, \xi_t) \\ \theta_{t+1} &= \theta_t + m_t,\ t = 0, 1, 2, \dots \end{aligned} \tag{5}$$

where b_t is a vector and "$\circ$" denotes Haddamard (elementwise) product. The problem considered here is to tune b_t and μ_t on the run of the procedure (66) to make it most efficient.

IV. Quadratic case

Let us consider application of AG (4) to a quadratic case with H being a non-negative matrix and ξ being a random vector, namely

$$J(\theta) = (1/2)\theta^T H \theta \tag{6}$$

$$g(\theta, \xi) = H\theta + \xi \tag{7}$$

$$E\xi = 0,\ E\xi\xi^T = \sigma^2 I. \tag{8}$$

Let W be a quadratic matrix whose columns are eigenvectors of H, orthogonal to one another. Then

$$W^T W = I,\ H = W \Lambda W^T, \tag{9}$$

where Λ is a diagonal matrix that contains egienvalues of H. Let us express the J function (6), and all the entities necessary to find its minimum with AG (4) in the coordinate system defined by W. The entities in the new coordinate system will be denoted with prims. Namely, we have

$$\theta = W\theta',\ \theta' = W^T\theta \tag{10}$$

$$\xi = W\xi',\ \xi' = W^T\xi \tag{11}$$

$$J'(\theta') = (1/2)\theta'^T \Lambda \theta' = (1/2)\sum_i \lambda_i \theta_i'^2 \tag{12}$$

$$g'(\theta', \xi') = W^T g(\theta, \xi) = \Lambda\theta' + \xi' \tag{13}$$

$$E\xi' = 0,\ E\xi'\xi'^T = W^T\sigma^2 I W = \sigma^2 I. \tag{14}$$

We can see that minimization of J is equivalent to n independent scalar minimization problems along eigenvectors of H. Let us take a closer look at the scalar problem.

Scalar model

Let us consider application of AG (4) where $\theta_t, m_t \in \Re$ to a scalar quadratic function

$$J(\theta_t) = (1/2)\lambda\theta_t^2 \tag{15}$$

$$g_t = \lambda(\theta_t + \mu m_{t-1}) + \xi_t \tag{16}$$

$$m_t = \mu m_{t-1} - \beta g_t \tag{17}$$

$$\theta_{t+1} = \theta_t + m_t \tag{18}$$

for $\lambda > 0$.

Corollary:
On the above assumptions, and for

$$\beta\lambda < \frac{4\mu^2 + \mu - 1 + \sqrt{9\mu^2 + 6\mu + 1}}{2\mu(2\mu + 1)}, \tag{19}$$

it is true that

$$\lim_{t\to\infty} E\theta_t^2 = \frac{\sigma^2(\beta/\lambda)(1+\mu-\mu\beta\lambda)}{2-2\mu^2+\beta\lambda(4\mu^2+\mu-1)-\beta^2\lambda^2\mu(2\mu+1)}. \tag{20}$$

Proof:
In order to derive (20) one analyzes Em_t^2 from (17), $E\theta_{t+1}^2$ from (18), and $Em_t\theta_t$ from both these equations.

From (18) we have

$$E\theta_{t+1}^2 = E\theta_t^2 + 2E\theta_t m_t + Em_t^2. \tag{21}$$

One considers $t\to\infty$ and limit distributions for θ_t, m_t, and $\theta_t m_t$. All the expected values till the end of this derivation should be understood as if they had the prefix $\lim_{t\to\infty}$. Therefore $E\theta_t^2 = E\theta_{t-1}^2$, and from (21) one has

$$E\theta_t m_t = -(1/2)Em_t^2. \tag{22}$$

From (17) and (18) one obtains

$$\begin{aligned} m_{t+1} &= \mu m_t - \beta\lambda(\theta_{t+1}+\mu m_t) - \beta\xi_{t+1} && (23)\\ &= \mu m_t - \beta\lambda(\theta_t + m_t + \mu m_t) - \beta\xi_{t+1} && (24)\\ &= (\mu-\beta\lambda-\mu\beta\lambda)m_t - \beta\lambda\theta_t - \beta\xi_{t+1}. && (25)\end{aligned}$$

Since ξ_{t+1} is zero-mean noise, the expected squares of the sides of the above equation are equal to

$$\begin{aligned} Em_{t+1}^2 =& (\mu-\beta\lambda-\mu\beta\lambda)^2 Em_t^2 + \beta^2\lambda^2 E\theta_t^2 && (26)\\ &- 2\beta\lambda(\mu-\beta\lambda-\mu\beta\lambda)Em_t\theta_t + \beta^2\sigma^2. && (27)\end{aligned}$$

Next, from the above and (22) one obtains

$$\begin{aligned} & Em_t^2\Big(1-(\mu-\beta\lambda-\mu\beta\lambda)^2 - \beta\lambda(\mu-\beta\lambda-\mu\beta\lambda)\Big) && (28)\\ &= \beta^2\lambda^2 E\theta_t^2 + \beta^2\sigma^2 && (29)\\ & Em_t^2\Big(1-(\mu-\beta\lambda-\mu\beta\lambda)\mu(1-\beta\lambda)\Big) && (30)\\ &= \beta^2\lambda^2 E\theta_t^2 + \beta^2\sigma^2 && (31)\end{aligned}$$

By multiplying (18) and (25) one obtains

$$\begin{aligned} & \theta_{t+1}m_{t+1} && (32)\\ &= (\theta_t+m_t)\big((\mu-\beta\lambda-\mu\beta\lambda)m_t - \beta\lambda\theta_t - \beta\xi_{t+1}\big) && (33)\\ &= -\beta\lambda\theta_t^2 + (\mu-\beta\lambda-\mu\beta\lambda)m_t^2 && (34)\\ &\quad + (\mu-2\beta\lambda-\mu\beta\lambda)\theta_t m_t + \xi_{t+1}const_t && (35)\end{aligned}$$

Simple transformations of the above equations and introducing (22) lead to

$$\begin{aligned} & [1-\mu+2\beta\lambda+\mu\beta\lambda]E\theta_t m_t = && (36)\\ &\quad = -\beta\lambda E\theta_t^2 + (\mu-\beta\lambda-\mu\beta\lambda)Em_t^2 && (37)\\ & [1-\mu+2\beta\lambda+\mu\beta\lambda]Em_t^2 = && (38)\\ &\quad = 2\beta\lambda E\theta_t^2 - 2(\mu-\beta\lambda-\mu\beta\lambda)Em_t^2 && (39)\\ & [1+\mu-\mu\beta\lambda]Em_t^2 = 2\beta\lambda E\theta_t^2 && (40)\end{aligned}$$

Combination of (31) and (40) leads to

$$\frac{2\beta\lambda[1-(\mu-\beta\lambda-\mu\beta\lambda)\mu(1-\beta\lambda)]}{1+\mu-\mu\beta\lambda}E\theta_t^2 = \beta^2\lambda^2 E\theta_t^2 + \beta^2\sigma^2 \tag{41}$$

$$\frac{\beta\lambda[2-2\mu^2+\beta\lambda(4\mu^2+\mu-1)-\beta^2\lambda^2\mu(2\mu+1)]}{1+\mu-\mu\beta\lambda}E\theta_t^2 = \beta^2\sigma^2 \tag{42}$$

which, in turn, leads to (20). ■

Although the term (20) is hardly self-explanatory, it leads to useful conclusions:

1) The steady-state loss i.e.,
$$\lim_{t\to\infty} EJ(\theta_t) = (1/2)\lambda \lim_{t\to\infty} E\theta_t^2 \tag{43}$$
is growing with β.
2) For β above a certain threshold related to λ the process (18) becomes unstable, and then $E\theta_t^2$ grows to infinity.
3) For $\mu = 0$ the aforementioned threshold is equal to $2\lambda^{-1}$; it decreases for larger μ, and for μ approaching 1 it is equal to $(4/3)\lambda^{-1}$.
4) For $\beta = \lambda^{-1}$ we have
$$\lim_{t\to\infty} EJ(\theta_t) = (1/2)\beta\sigma^2, \tag{44}$$
regardless of μ.
5) For $\mu \ll 1-\beta/\lambda$ we have
$$\lim_{t\to\infty} EJ(\theta_t) \cong \frac{\beta\sigma^2}{4(1-\mu)}. \tag{45}$$
6) For $\mu = 1$ we have
$$\lim_{t\to\infty} EJ(\theta_t) = \frac{\sigma^2(2-\beta\lambda)}{2\lambda(4-3\beta\lambda)}. \tag{46}$$
Hence, the steady-state loss grows with μ, but only to the above threshold.

The above analysis leads to the following conclusion about the multidimensional case (6): AG is stable as long as the steps-size, β, is smaller than $(4/3)\lambda^{-1}$ where λ is the largest eigenvalue of the Hessian of the loss function, J. Therefore, we set the step-size equal to the inverse of the largest eigenvalue of the Hessian approximation, and devote another section to estimation of this approximation.

V. Hessian

A. Hessian approximation

Let $D: \Re^n \mapsto \Re^{n\times n}$ be a function that transforms a vector to a diagonal matrix such that

$$D_{i,i}(\eta) = \exp(\eta_i). \tag{47}$$

An approximation of a Hessian analyzed in this paper takes the form

$$\widehat{H} = D(\eta) \tag{48}$$

where $\eta \in \Re^n$ is a parameter. It is obvious that the above matrix is symmetrical and positively definite (also easily invertible). Most importantly, it is easy to see that multiplication

of $\widehat{H}$ by any vector requires $O(n)$ operations even though $\widehat{H}$ has n^2 elements.

Because $\widehat{H}$ is diagonal and positively definite, it is easy to indicate its largest eigenvalue. It is equal to

$$\lambda^* = \exp(\max_i \eta_i). \tag{49}$$

The Hessian, $\nabla^2 J$, has the following property for $r \in \Re^n$

$$\nabla J(\theta + r) = \nabla J(\theta) + \nabla^2 J(\theta) r + o(r). \tag{50}$$

Basing on this property we require that the approximation of the Hessian minimizes the following quality index

$$Q(\eta) = \text{Average}\, Q_t(\eta) \tag{51}$$

$$Q_t(\eta) = \|g(\theta + r_t, \xi_t) - \widehat{H} r_t - g(\theta, \xi_t)\|^2, \tag{52}$$

where $(r_t, \xi_t), t = 1, 2, \ldots$ is a certain sequence. For a given $b = g(\theta + r_t, \xi_t) - g(\theta, \xi_t)$ and $\widehat{H} = D(\eta)$, $Q_t(\eta)$ has the following derivative with respect to η:

$$\frac{\partial Q_t}{\partial \eta^T} = \widehat{H} r_t \circ (\widehat{H} r_t - b) + r_t \circ \widehat{H}(\widehat{H} r_t - b) \tag{53}$$

In order to approximate the Hessian, η is adjusted conversely to the above derivative. Section V-B is devoted to derivation of eq. (53).

We will apply the type of adjustment inspired by ADAM [10]. Namely, average square derivatives (53) are registered in a vector, ψ_η, and then the adjustments are scaled with inverse root squares of the elements of ψ_η.

B. Derivation of $\partial Q_t / \partial \eta^T$

In order to derive (53) some elementary properties need to be recalled.

Let $q : \Re^n \mapsto \Re$ be a function that transforms a vector to a scalar. By

$$\frac{\partial q(v)}{\partial v^T} \tag{54}$$

we understand a vector with the shape of v that contains partial derivatives, namely

$$\left[\frac{\partial q(v)}{\partial v^T}\right]_i = \frac{\partial q(v)}{\partial v_i}. \tag{55}$$

Let us consider D (47), vectors η, b, and c of the same shape. We have

$$\frac{\partial b^T D(\eta) c}{\partial \eta_i} = \frac{\partial \sum_i b_i \exp(\eta_i) c_i}{\partial \eta_i} = b_i \exp(\eta_i) c_i. \tag{56}$$

Therefore,

$$\frac{\partial b^T D(\eta) c}{\partial \eta^T} = b \circ D(\eta) c = D(\eta) b \circ c. \tag{57}$$

Now we are closer to derive (53). Namely,

$$Q_t(\eta) = \|D(\eta) r_t - b\|^2 \tag{58}$$

$$= r_t^T D(\eta) D(\eta) r_t - 2 b^T D(\eta) r_t + b^T b. \tag{59}$$

Derivation of the derivative of the above function with respect to η requires several applications of (57).

VI. Momentum, stability, and normalization

A. Momentum decay factor

In order to control the momentum decay factor, i.e., the term μ_t in (3) and (4), we notice that this parameter does not influence stability of AG, which is entirely determined by the step-size. We introduce the following principle for setting that parameter. Eq. (20) clearly indicates that θ wanders longer along those eigenvectors of the Hessian with small eigenvalues. We want to delimit such wandering, and set the following principle: Each element of θ, denoted by θ_i, should traverse at most δ_i when it is pushed only by its inertia, where δ_i is the radius of the interval in which the optimal value of θ_i is expected to be. If i-th element of m is denoted by m_i, then under such circumstances θ_i will traverse in the future the distance

$$m_i + \mu m_i + \mu^2 m_i + \cdots = (1 - \mu)^{-1} m_i. \tag{60}$$

This distance is to be at most as long as δ_i. Therefore

$$\delta_i \geq (1 - \mu)^{-1} \max_i |m_i| \tag{61}$$

and

$$\mu = 1 - \max_i |m_i / \delta_i|. \tag{62}$$

The term δ_i defined above results from the same reasoning as the radius of the interval from which the initial value of θ_i is sampled. Here we focus on neural network training, and therefore we set

$$\delta_i = 1/2 \tag{63}$$

when θ_i being a bias in a neuron, and

$$\delta_i = 1/\sqrt{d_i} \tag{64}$$

for d_i being the dimension of the input of the neuron to which θ_i belongs.

B. Instability detection

In the previous sections the principle was established to set the step-size equal to the inverse of the largest eigenvalue of the Hessian (approximation). However, these eigenvalues are estimated during the learning process. Before they are estimated with sufficient accuracy, the step-size may be large enough to cause instability of the process.

Obvious symptoms of instability are excessive values of m_i. Therefore, we propose to detect instability by checking whether m_i is large in comparison to δ_i. Namely, instability is detected when

$$\max_i |m_i / \delta_i| > c \tag{65}$$

where $c \in (0, 1)$ is a certain threshold parameter. In the experimental study we apply $c = 1/2$. When instability is detected, the estimates of the eigenvalues are increased, and consequently the step-size is decreased.

C. Normalization

As long as AG (4) is convergent, a generalized procedure,

$$\begin{aligned} m_t &= \mu_t m_{t-1} - \beta_t M g(\theta_t + \mu_t m_{t-1}, \xi_t) \\ \theta_{t+1} &= \theta_t + m_t,\ t = 0, 1, 2, \ldots \end{aligned} \tag{66}$$

is also convergent, where M is a positively definite matrix. Procedure (66) may by understood as (4) in a linearly transformed domain. In ADAM [10] elements of g are scaled by root squares of their average squares. A corresponding version of AG could have the following form

$$\begin{aligned} \psi_\theta &= \text{Average}_{i<t}(g_i \circ g_i) \\ m_t &= \mu_t m_{t-1} - \beta_t g(\theta_t + \mu_t m_{t-1}, \xi_t) \oslash \sqrt{\psi_\theta + \epsilon} \\ \theta_{t+1} &= \theta_t + m_t,\ t = 0, 1, 2, \ldots \end{aligned} \tag{67}$$

where "$\circ$" denotes elementwise product and "$\oslash$" denotes elementwise division, ϵ is small positive constant that prevents division by zero, and $\sqrt{v + \epsilon}$ denotes a vector of root squares of the elements of v increased by ϵ.

Technically, the way to calculate average squares on-line follows. Let $x_t, t = 1, 2, 3, ...$ be a sequence. Let $\bar{x}_t$ be the exponentially moving average of (x_t), namely

$$\bar{x}_t = \frac{\sum_{i=0}^{t-1} \gamma^i x_{t-i}}{\sum_{i=0}^{t-1} \gamma^i}. \tag{68}$$

Elementary transformation reveal that

$$\bar{x}_t = \gamma \frac{1 - \gamma^{t-1}}{1 - \gamma^t} s_{t-1} + \frac{1 - \gamma}{1 - \gamma^t} x_t. \tag{69}$$

Hence, the way to keep ψ_θ in (67) updated is as follows

$$\psi_\theta \leftarrow \gamma \frac{1 - \gamma^{t-1}}{1 - \gamma^t} \psi_\theta + \frac{1 - \gamma}{1 - \gamma^t} g_t \circ g_t. \tag{70}$$

VII. Algorithm

Algorithms 1 (S2AG) and 2 (S2AGS) result from the discussion in previous sections. Alg. 1 applies the Hessian approximation (48) to determine the steps-size and the concept presented in Sec. VI-A to compute the momentum decay factor. Alg. 2 also encompasses gradient scaling similar to that applied by ADAM [10].

Initially, both algorithms operate exactly like Nesterov's Accelerated Gradient (AG) with step-size β_0 and momentum decay factor μ_0, where β_0 and μ_0 are given as coefficients. S2AG eventually operates as AG with step-size equal to β_1/λ and momentum decay factor equal to μ. The term λ is an estimate of the largest eigenvalue of the Hessian of the cost function. The term $\beta_1 \in (0, 1]$ is a coefficient. The term μ is computed within the algorithm according to the discussion in Sec. VI-A. In most cases when the algorithm uses a certain estimate e.g. λ, it is a result exponential smoothing, with the decay factor γ, of the the raw estimate.

The algorithm computes gradient estimates at two points, namely θ_t and $\theta_t + 2\mu_t m$. The difference between these estimates is applied to estimate the Hessian. Their average estimates the gradient at the point $\theta_t + \mu_t m$ as required by the AG algorithm.

Algorithm 1 S2AG: Self-Stabilizing Accelerated Gradient

1: Initialize θ
2: $\lambda \leftarrow \beta_1/\beta_0$, $\mu \leftarrow \mu_0$, $\gamma \leftarrow \gamma_0$, $t \leftarrow 0$
3: Fill η with $\ln(\beta_1/\beta_0)$
4: Fill m with 0-s and ψ_η with 1-s
5: $t \leftarrow t + 1$
6: Get ξ_t
7: $g' \leftarrow g(\theta, \xi_t)$
8: $g'' \leftarrow g(\theta + 2\mu m, \xi_t)$
9: $r \leftarrow 2\mu m$
10: $g \leftarrow 0.5(g' + g'')$
11: $m \leftarrow \mu m - (\beta_1/\lambda) g$
12: If $\max_i |m_i/\delta_i| > 1/2$:
13: $\quad \eta \leftarrow \eta + \ln 2$
14: $\quad \lambda \leftarrow 2\lambda$
15: $\quad$ Fill m with 0-s
16: $\quad$ Go to Line 5
17: $\theta \leftarrow \theta + m$
18: $\lambda \leftarrow \exp(\max_i \eta_i)$
19: $\mu \leftarrow \gamma\mu + (1 - \gamma)(1 - \max_i |m_i/\delta_i|)$
20: $\gamma \leftarrow \max\{\mu, \gamma_0\}$
21: $b \leftarrow g'' - g'$
22: $\frac{\partial Q_t}{\partial \eta^T} \leftarrow \widehat{H} r \circ (\widehat{H} r - b) + r \circ \widehat{H}(\widehat{H} r - b)$
23: $\psi_\eta \leftarrow \gamma \frac{1-\gamma^{t-1}}{1-\gamma^t} \psi_\eta + \frac{1-\gamma}{1-\gamma^t} \frac{\partial Q_t}{\partial \eta^T} \circ \frac{\partial Q_t}{\partial \eta^T}$
24: $\eta \leftarrow \eta - \beta_2 \frac{\partial Q_t}{\partial \eta^T} \oslash \sqrt{\psi_\eta + \epsilon_0}$
25: If the stop criterion is not met, go to Line 5

Algorithm 2 S2AGS: Self-Stabilizing Accelerated Gradient with gradient Scaling.

$\ldots$ Like in Alg. 1
4: Fill m with 0-s and ψ_η, ψ_θ with 1-s
$\ldots$ Like in Alg. 1
11a: $\psi_\theta \leftarrow \gamma \frac{1-\gamma^{t-1}}{1-\gamma^t} \psi_\theta + \frac{1-\gamma}{1-\gamma^t} g_t \circ g_t.$
11b: $m \leftarrow \mu m - (\beta_1/\lambda) g \oslash \sqrt{\psi_\theta + \epsilon_1}$
$\ldots$ Like in Alg. 1
22: $b \leftarrow (g'' - g') \oslash \sqrt{\psi_\theta + \epsilon_0}$
$\ldots$ Like in Alg. 1

In Lines 1–4 the algorithm initializes its data structures to operate at first as discussed above. In Lines 6–8 it retrieves data and computes gradient estimates at two points. In Lines 11 and 17 the algorithm performs basic operations of AG. In Lines 12–16 it verifies stability and react accordingly in case of instability as discussed in Sec. VI-B. In Line 18 it updates the maximum eigenvalue of the Hessian as discussed in Sec. V-A. In Line 19 it updates the momentum decay factor according to Sec. VI-A. In the remaining part the algorithm updates the Hessian approximation according to Sec. V-A. In Lines 22 and 24 the term "$\circ$" denotes Haddamard (elementwise) product, and "$\oslash$" denotes elementwise division. Also, the square root that appears in Lines 11b and 24 is elementwise.

The algorithms require the following coefficients:

β_0 — the initial learning rate; default $\beta_0 = 1$,

μ_0 — the initial momentum decay factor; default $\mu_0 = 1/2$,
β_1 — eventually gradients are applied multiplied by β_1/λ; default $\beta_1 = 1$ but it may be smaller if for some reasons the learning should be more averaging e.g., in its final stage,
γ_0 — minimal decay factor for exponential smoothing; in our experiments $\gamma_0 = 0.99$,
β_2 — meta-step-size for adjustments of η; default $\beta_2 = 1 - \gamma_0 = 10^{-2}$,
ϵ_0 — a small constant that prevents division by zero in scaling increments of η; default $\epsilon_0 = 10^{-32}$,
ϵ_1 — a small constant that prevents division by zero in scaling derivatives; default $\epsilon_1 = 10^{-8}$ is taken from Tensorflow implementation of ADAM.

In all experiments reported in the following section default values of the coefficients are applied.

VIII. Experimental study

This section reports experiments with the algorithms presented in the previous section. The algorithms are tested on training shallow neural classifiers, deep dense autoencoders, and deep convolutional autoencoders.

Four new algorithms are examined:

- Self-Stabilizing Accelerated Gradient (S2AG) — Algorithm 1
- Self-Stabilizing Accelerated Gradient with gradient Scaling (S2AGS) — Algorithm 2.

The new algorithms are compared to the following known ones: the classic momentum (CM), the accelerated gradient (AG), ADAM, AdaGrad, AdaDelta. These 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 gave the best ultimate error. The default parameters are ones applied by Tensorflow when their values are not provided.

Table I presents the analyzed learning problems along with their basic parameters, Table II demonstrates the parameters that resulted from the aforementioned optimization, and Table III presents ultimate errors obtained by all algorithms for all learning problems.

A. Shallow neural classifiers

We take random 10 classification problems from UCI Machine Learning Repository [31]. 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 at minimization of the test error. (Below we only report the training error, as the subject of the study is optimization rather than generalization.) Initial weights of the hidden neurons are drawn from the normal distribution $N(0, \sigma^2)$ where $\sigma = 1/\sqrt{\dim(\text{input})}$. Networks inputs are scaled with their means and standard deviations. Required outputs are one-hot vectors. The loss reported is mean-square error.

B. Dense autoencoders

There are three tasks, each based on a database that contains 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].

C. Convolutional autoencoders

A convolutional autoencoder neural network is fed with color images from CIFAR10 dataset. The required output is equal to the input. The layers in the network are as follows:

1) Convolution with 32 filters 3×3, ReLU activation,
2) Max pooling 2×2, step 2,
3) Convolution with 64 filters 3×3, ReLU activation,
4) Dense layer with 512 units, linear activation,
5) Dense layer with 16·16·3 units, logistic sigmoid activation,
6) Transposed convolution with 32 filters 3×3, ReLU activation,
7) Resizing ×2,
8) Transposed convolution output with 3 filters 3×3, logistic sigmoid activation.

Initial weights of the hidden neurons are drawn with the xavier method. The loss reported is mean-square error.

D. Results

The results are depicted in Tab. III. The smallest error for each problem is indicated by bold face font. The following observations can be made:

- In most cases (11 cases out of 14) the winner is either S2AGS or S2AG.
- Usually (in 9/14 cases) ADAM with optimized parameters (ADAM/o) gives slightly worse results than S2AGS. Rarely (in 3/14 cases) ADAM/o outperforms S2AGS.
- In most cases (13/14) manual optimization of parameters of ADAM yield significant improvement of its behavior. However, its default parameters make that algorithm perform well in comparison others, except S2AG and S2AGS.
- CM and AG also have their moments of glory. Each of them yield the smallest error twice. The case of CM and Robot is especially interesting: The algorithm outperformed all the rest so much to oblige us to double-check the setting and repeat all the experiments with Robot. However, the results were the same.

TABLE I
BASIC PARAMETERS OF ANALYZED LEARNING PROBLEMS.

Problem	Abr*	Size	Idim	Odim	Ncnt	Mbs	Len
Default of credit card clients	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 diagnosis	Motor	58509	48	11	89	200	10^7
Poker hand	Poker	1025010	10	10	89	200	10^7
Wall-following 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
First order theorem prooving	Theo	6118	51	6	89	200	10^7
Curves	Curves	20000	784	784	—	200	10^8
Handwritten Digits MNIST	MNIST	60000	784	784	—	200	10^8
Olivetti Faces	Faces	165600	625	625	—	200	10^8
CIFAR10	CIFAR10	50000	3072	3072	—	200	$2{\cdot}10^7$

* Abr — problem name abbreviation, Size — number of samples in the dataset, Idim — input dimension, Odim — output dimension, Ncnt — number of neurons in hidden layer, Mbs — mini-batch size, Len — number of samples processed before termination (i.e., a run takes Len/Mbs steps).

TABLE II
STEP-SIZES AND MOMENTUM DECAY FACTORS OPTIMAL FOR EACH PROBLEM.

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	0.05	0.99
Dota2	0.05	0.99	0.05	0.99	0.002	0.99	0.2	0.2	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	0.1	0.99
Robot	0.2	0.99	0.1	0.99	0.05	0.9	0.2	0.2	0.99
Shuttle	0.2	0.99	0.1	0.99	0.02	0.9	0.2	0.05	0.99
Skin	0.1	0.9	0.1	0.99	0.05	0.9	0.5	1	0.9
Spam	0.2	0.99	0.1	0.99	0.02	0.9	0.2	0.2	0.99
Theo	0.01	0.99	0.02	0.99	0.01	0.9	0.2	0.2	0.99
Curves	0.01	0.9	0.002	0.9	0.002	0.9	0.02	1	0.9
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
CIFAR10	10^{-5}	0.99	10^{-5}	0.99	0.001	0.9	0.005	0.5	0.99

* ss — step-size, mdf — momentum decay factor.

TABLE III
RESULTING AVERAGE ERRORS. EACH NUMBER AVERAGES 10 RUNS.

Alg.	CM	AG	ADAM		AdaGrad	AdaDelta		S2AG	S2AGS
Problem			/d*	/o*		/d	/o		
CCard	0.262	0.264	0.262	0.260	0.266	0.342	0.274	0.265	**0.258**
Dota2	0.422	0.418	0.412	**0.394**	0.405	0.500	0.417	0.426	**0.394**
Htru2	0.0326	0.0300	0.0313	0.0293	0.0294	0.1033	0.0295	**0.0290**	0.0299
Motor	0.0642	0.0480	0.0668	0.0415	0.0829	0.802	0.0872	0.0526	**0.0386**
Poker	0.451	**0.315**	0.448	0.327	0.477	0.569	0.532	0.358	0.519
Robot	**0.0642**	0.0908	0.1198	0.1064	0.1270	0.6178	0.1311	0.0975	0.0941
Shuttle	0.0064	0.0045	0.0084	0.0038	0.0093	0.2768	0.0218	0.0053	**0.0033**
Skin	0.0070	0.0055	0.0074	**0.0043**	0.0074	0.2573	0.0102	0.0054	0.0048
Spam	0.0245	0.0230	0.0279	0.0172	0.0257	0.4270	0.0449	0.0240	**0.0151**
Theo	0.442	0.409	0.430	0.393	0.440	0.720	0.457	0.419	**0.380**
Curves	0.178	0.312	0.158	0.157	0.444	15.930	0.348	0.122	**0.104**
MNIST	**1.03**	**1.03**	1.27	1.12	2.11	15.351	1.30	**1.03**	1.07
Faces	15.99	16.81	15.32	14.61	39.63	103.26	19.88	**13.1**	14.8
CIFAR10	5.11	3.75	2.29	2.29	4.98	23.75	3.29	2.95	**2.12**

* /d — default step-size and momentum decay factor, /o — optimized ones.

- 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.
- Performance of AdaGrad and AdaDelta is especially disappointing. Those algorithm 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.

IX. Conclusions and future work

In this paper a method has been introduced to approximate the Hessian of the loss function within the process of on-line learning. That enable estimation of the approximately optimal step-size of the method of accelerated gradient.

The experimental study with on-line learning in several types of neural networks confirms that the presented learning algorithms are efficient. Moreover, they are practically parameter-free, as default values of the required parameters are as good for all problems.

The presented material is a proof-of-concept that justifies the following claims:

- The step-size in SGD-like algorithms may be estimated with good results with the use of a certain approximation of the Hessian of the loss function.
- The momentum decay factor may be estimated just on the basis of large components of the momentum vector.

All the above observations require further studies to produce even better performing learning algorithms with strict convergence guarantees.

Acknowledgments

We gratefully acknowledge the support of the following students in this research: Karol Chęciński, Łukasz Lepak, Daniel Klepacki, and Jakub Łyskawa.

References

[1] H. J. Kushner and G. Yin, *Stochastic Approximation Algorithms and Applications*. Springer-Verlag, 1997.

[2] 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.

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

[4] 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.

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

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

[7] 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.

[8] 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.

[9] M. D. Zeiler, "Adadelta: An adaptive learning rate method," *http://arxiv.org/abs/1212.5701*, 2012.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[11] Y. Iiguni, H. Sakai, and H. Tokumaru, "A real time learning lagorithm for a multilayered neural network based on extended kalman filter," *IEEE Trans. on Signal Processing*, vol. 45, no. 6, pp. 959–966, 1992.

[12] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Machine Learning*, vol. 69, no. 2–3, pp. 169–192, 2007.

[13] T. van Erven and W. M. Koolen, "Metagrad: Multiple learning rates in online learning," in *Advances in NIPS*, 2016, pp. 3666–3674.

[14] A. Cutkosky and K. A. Boahen, "Stochastic and adversarial online learning without hyperparameters," in *Advances in NIPS*, 2017, pp. 5059–5067.

[15] T. Schaul, S. Zhang, and Y. LeCun, "No More Pesky Learning Rates," in *ICML*, 2013, pp. 343–351.

[16] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *ICML*, 2015, pp. 573–582.

[17] F. M. Silva and L. B. Almeida, "Acceleration techniques for the backpropagation algorithm," in *Neural Networks EURASIP Workshop*, Sesim, 1990.

[18] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, no. 4, pp. 295–308, 1988.

[19] C. Tan, S. Ma, Y.-H. Dai, and Y. Qian, "Barzilai-borwein step size for stochastic gradient descent," in *ICML*, 2016, pp. 685–693.

[20] L. Xiao and T. Zhang, "A proximal stochastic gradient method with progressive variance reduction," *SIAM Journal on Optimization*, vol. 24, no. 4, pp. 2057–2075, 2014.

[21] L. Behera, S. Kumar, and A. Patnaik, "On adaptive learning rate that guarantees convergence in feedforward networks," *IEEE Trans. on Neural Networks*, vol. 17, no. 5, pp. 1116–1125, 2006.

[22] T. Kathirvalavakumar and S. J. Subavathi, "Neighborhood based modified backpropagation algorithm using adaptive learning parameters for training feedforward neural networks," *Neurocomputing*, vol. 72, pp. 3915–3921, 2009.

[23] M. Zak, "Terminal attractors in neural networks," *Neural Networks*, vol. 2, pp. 259–274, June 1989.

[24] X. Yu, B. Wang, B. Batbayar, L. Wang, and Z. Man, "An improved training algorithm for feedforward neural network learning based on terminal attractors," *Journal of Global Optimization*, pp. 1–14, September 2010.

[25] 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.

[26] Y. Zhang, F. Cao, and Z. Xu, "Estimation of learning rate of least square algorithm via jackson operator," *Neurocomputing*, vol. 74, pp. 516–521, 2011.

[27] R. Ranganath, C. Wang, D. M. Blei, and E. P. Xing, "An adaptive learning rate for stochastic variational inference," in *ICML*, 2013, pp. 298–306.

[28] F. Orabona and D. Pl, "Coin betting and parameter-free online learning," in *Advances in NIPS*, 2016, pp. 577–585.

[29] 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.

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

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

[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.