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, \( J : \mathbb{R}^n \rightarrow \mathbb{R} \), 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 \( \alpha \)-acutely convex. In every step the method minimizes a certain approximation of the loss function.
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 [28] 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,

$$J : \mathcal{R}^n \rightarrow \mathcal{R}.$$  

(1)

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

$$E[g(\theta, \xi)] = \nabla J(\theta).$$  

(2)

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

In the below equations $t$ denotes discrete time, $\theta_t$ 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

$$m_t = \mu_t m_{t-1} - \beta_t g(\theta_t, \xi_t)$$  

$$\theta_{t+1} = \theta_t + m_t, \; t = 0, 1, 2, \ldots$$  

(3)

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

$$m_t = \mu_t m_{t-1} - \beta_t g(\theta_t + \mu_t m_{t-1}, \xi_t)$$  

$$\theta_{t+1} = \theta_t + m_t, \; t = 0, 1, 2, \ldots$$  

(4)

CM and AG are similar. CM first computes the gradient, and pushes $\theta_t$ along the momentum, and the gradient. AG uses the gradient computed at the point to which $\theta_t$ 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

$$m_t = \mu_t m_{t-1} - \beta_t g(\theta_t + \mu_t m_{t-1}, \xi_t)$$  

$$\theta_{t+1} = \theta_t + m_t, \; t = 0, 1, 2, \ldots$$  

(5)

where $b_t$ is a vector and “$\circ$” denotes Haddamard (elementwise) product. The problem considered here is to tune $b_t$ and $\mu_t$ 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 $\xi$ being a random vector, namely

$$J(\theta) = (1/2)\theta^T H \theta$$  

(6)

$$g(\theta, \xi) = H \theta + \xi$$  

(7)

$$E\xi = 0, \; E\xi\xi^T = \sigma^2 I.$$  

(8)

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

$$W^T W = I, \; H = W \Lambda W^T,$$  

(9)

where $\Lambda$ is a diagonal matrix that contains eigenvalues 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

$$\theta = W \theta', \; \theta' = W^T \theta$$  

(10)

$$\xi = W \xi', \; \xi' = W^T \xi$$  

(11)

$$J'(\theta') = (1/2)\theta'^T \Lambda \theta' = (1/2) \sum_i \lambda_i \theta'^2$$  

(12)

$$g'(\theta', \xi') = W^T g(\theta, \xi) = \Lambda \theta' + \xi'$$  

(13)

$$E\xi' = 0, \; E\xi'\xi'^T = W^T \sigma^2 I W = \sigma^2 I.$$  

(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 $\theta_t, m_t \in \mathcal{R}$ to a scalar quadratic function

$$J(\theta_t) = (1/2)\lambda \theta_t^2$$  

(15)

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

(16)

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

(17)

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

(18)

for $\lambda > 0$.

**Corollary:**

On the above assumptions, and for

$$\beta \lambda < 4 \mu^2 + \mu - 1 + \sqrt{9 \mu^2 + 6 \mu + 1}$$

$$2 \mu (2 \mu + 1),$$  

(19)}
it is true that

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

**Proof:**

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

From (18) we have

\[
E \theta_{t+1}^2 = E \theta_t^2 + 2 E \theta_t m_t + E m_t^2.
\] (21)

One considers \( t \to \infty \) and limit distributions for \( \theta_t, m_t, \) 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

\[
E \theta_t m_t = -(1/2) E m_t^2.
\] (22)

From (17) and (18) one obtains

\[
m_{t+1} = \mu m_t - \beta \lambda (\theta_{t+1} + m_t) - \beta \xi_{t+1} = \mu m_t - \beta \lambda (\theta_t + m_t) - \beta \xi_{t+1} = (\mu - \beta \lambda - \mu \beta \lambda) m_t - \beta \theta_t - \beta \xi_{t+1}.
\] (23)

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

\[
E m_{t+1}^2 = (\mu - \beta \lambda - \mu \beta \lambda)^2 E m_t^2 + \beta^2 \lambda^2 E \theta_t^2 - 2 \beta \lambda (\mu - \beta \lambda - \mu \beta \lambda) E m_t \theta_t + \beta^2 \sigma^2.
\] (24)

Next, from the above and (22) one obtains

\[
E m_t^2 \left(1 - (\mu - \beta \lambda - \mu \beta \lambda)^2 - \beta \lambda (\mu - \beta \lambda - \mu \beta \lambda)\right) = \beta^2 \lambda^2 E \theta_t^2 + \beta^2 \sigma^2.
\] (25)

By multiplying (18) and (25) one obtains

\[
\theta_{t+1} = \theta_t + m_t \left((\mu - \beta \lambda - \mu \beta \lambda) m_t - \beta \theta_t - \beta \xi_{t+1}\right) = -\beta \lambda \theta_t^2 + (\mu - \beta \lambda - \mu \beta \lambda) m_t^2 + (\mu - 2 \beta \lambda - \mu \beta \lambda) \theta_t m_t + \xi_{t+1} \text{const}.
\] (26)

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

\[
[1 - \mu + 2 \beta \lambda + \mu \beta \lambda] E \theta_t m_t = \beta \lambda \theta_t^2 + (\mu - \beta \lambda - \mu \beta \lambda) E m_t^2.
\] (27)

\[
[1 - \mu + 2 \beta \lambda + \mu \beta \lambda] E \theta_t^2 = 2 \beta \lambda \theta_t^2 - 2(\mu - \beta \lambda - \mu \beta \lambda) E m_t^2.
\] (28)

\[
[1 + \mu - \mu \beta \lambda] E m_t^2 = 2 \beta \lambda \theta_t^2
\] (29)

Combination of (31) and (40) leads to

\[
\begin{align*}
\frac{2 \beta \lambda[1 - (\mu - \beta \lambda - \mu \beta \lambda)\mu(1 - \beta \lambda)]}{1 + \mu - \mu \beta \lambda} E \theta_t^2 &= \beta^2 \lambda^2 E \theta_t^2 + \beta^2 \sigma^2, \\
\frac{\beta \lambda[2 - 2 \mu^2 + \beta \lambda(4 \mu^2 + \mu - 1) - \beta^2 \lambda^2 \mu(2 \mu + 1)]}{1 + \mu - \mu \beta \lambda} E \theta_t^2 &= \beta^2 \sigma^2.
\end{align*}
\] (30)

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.,

\[
\lim_{t \to \infty} E J(\theta_t) = (1/2) \lambda \lim_{t \to \infty} E \theta_t^2
\] (31)

is growing with \( \beta \).

2) For \( \beta \) above a certain threshold related to \( \lambda \) 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 \( \mu \), and for \( \mu \) approaching \( 1 \) it is equal to \( (4/3) \lambda^{-1} \).

4) For \( \beta = \lambda^{-1} \) we have

\[
\lim_{t \to \infty} E J(\theta_t) = (1/2) \beta \sigma^2,
\] (32)

regardless of \( \mu \).

5) For \( \mu \ll 1 - \beta/\lambda \) we have

\[
\lim_{t \to \infty} E J(\theta_t) \equiv \frac{\beta \sigma^2}{4(1 - \mu)}.
\] (33)

6) For \( \mu = 1 \) we have

\[
\lim_{t \to \infty} E J(\theta_t) = \frac{\sigma^2(2 - \beta \lambda)}{2 \lambda(4 - 3 \beta \lambda)}.
\] (34)

Hence, the steady-state loss grows with \( \mu \), 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 step-size, \( \beta \), is smaller than \( (4/3) \lambda^{-1} \) where \( \lambda \) 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 \( D : \mathbb{R}^n \to \mathbb{R}^{n \times n} \) be a function that transforms a vector to a diagonal matrix such that

\[
D_{i,i}(\eta) = \exp(\eta_i).
\] (35)

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

\[
\tilde{H} = D(\eta)
\] (36)

where \( \eta \in \mathbb{R}^n \) 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 \(\tilde{H}\) by any vector requires \(O(n)\) operations even though \(\tilde{H}\) has \(n^2\) elements.

Because \(\tilde{H}\) is diagonal and positive definite, it is easy to indicate its largest eigenvalue. It is equal to

\[
\lambda^* = \exp(\max_i \eta_i).
\]

(49)

The Hessian, \(\nabla^2 J\), has the following property for \(r \in \mathbb{R}^n\)
\[
\nabla J(\theta + r) = \nabla J(\theta) + \nabla^2 J(\theta)r + o(r).
\]

(50)

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

\[
Q(\eta) = \text{Average} Q_t(\eta)
\]

(51)

\[
Q_t(\eta) = \|g(\theta + r_t, \xi_t) - \tilde{H} r_t - g(\theta, \xi_t)\|^2,
\]

(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 \(\tilde{H} = D(\eta)\), \(Q_t(\eta)\) has the following derivative with respect to \(\eta\):

\[
\frac{\partial Q_t}{\partial \eta^T} = \tilde{H} r_t \circ (\tilde{H} r_t - b) + r_t \circ \tilde{H} (\tilde{H} r_t - b)
\]

(53)

In order to approximate the Hessian, \(\eta\) 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, \(\psi_\eta\), and then the adjustments are scaled with inverse root squares of the elements of \(\psi_\eta\).

B. Derivation of \(\partial Q_t / \partial \eta^T\)

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

Let \(q : \mathbb{R}^n \rightarrow \mathbb{R}\) be a function that transforms a vector to a scalar. By

\[
\frac{\partial q(v)}{\partial v^T}
\]

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

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

(55)

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

\[
\frac{\partial b^\sigma D(\eta)c}{\partial \eta_i} = \sum_i b_i \frac{\exp(\eta_i) c_i}{\partial \eta_i} = b_i \exp(\eta_i) c_i.
\]

(56)

Therefore,

\[
\frac{\partial b^\sigma D(\eta)c}{\partial \eta^T} = b \circ D(\eta)c = D(\eta)b \circ c.
\]

(57)

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

\[
Q_t(\eta) = \|D(\eta) r_t - b\|^2 = r_t^T D(\eta) r_t - 2 b^T D(\eta) r_t + b^T b.
\]

(58)

(59)

Derivation of the derivative of the above function with respect to \(\eta\) 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 \(\mu_i\) 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 \(\theta\) 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 \(\theta_i\), denoted by \(\theta_i\), should traverse at most \(\delta_i\) when it is pushed only by its inertia, where \(\delta_i\) is the radius of the interval in which the optimal value of \(\theta_i\) is expected to be. If \(i\)-th element of \(m\) is denoted by \(m_i\), then under such circumstances \(\theta_i\) will traverse in the future the distance

\[
m_i + \mu m_i + \mu^2 m_i + \cdots = (1 - \mu)^{-1} m_i.
\]

(60)

This distance is to be at most as long as \(\delta_i\). Therefore

\[
\delta_i \geq (1 - \mu)^{-1} \max_i |m_i|
\]

(61)

and

\[
\mu = 1 - \max_i |m_i/\delta_i|.
\]

(62)

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

\[
\delta_i = 1/2
\]

(63)

when \(\theta_i\) being a bias in a neuron, and

\[
\delta_i = 1/\sqrt{d_i}
\]

(64)

for \(d_i\) being the dimension of the input of the neuron to which \(\theta_i\) 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 \(\delta_i\). Namely, instability is detected when

\[
\max_i |m_i/\delta_i| > c
\]

(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, $m_0 = \mu + \beta \Delta g (\theta + \mu_1 m_{t-1}, \xi_t)$

$$\theta_{t+1} = \theta_t + m_t, \quad t = 0, 1, 2, \ldots$$ (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

$$\psi_t = \text{Average}_i (g_{i} \circ g_{i})$$

$$m_t = \psi_t + \epsilon \theta_{t} + \mu_1 m_{t-1}, \xi_t) \otimes \sqrt{\psi_0 + \epsilon}$$ (67)

$\theta_{t+1} = \theta_t + m_t, \quad t = 0, 1, 2, \ldots$

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

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

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

Elementary transformation reveal that

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

Hence, the way to keep $\psi_0$ in (67) updated is as follows

$$\psi_t \leftarrow \gamma \frac{1 - \gamma^{t-1}}{1 - \gamma} \psi_0 + \frac{1 - \gamma}{1 - \gamma^T} g_t \circ g_t$$. (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 $\beta_0$ and momentum decay factor $\mu_0$, where $\beta_0$ and $\mu_0$ are given as coefficients. S2AG eventually operates as AG with step-size equal to $\beta_1/\lambda$ and momentum decay factor equal to $\mu$. The term $\lambda$ is an estimate of the largest eigenvalue of the Hessian of the cost function. The term $\beta$ 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. $\lambda$, it is a result exponential smoothing, with the decay factor $\gamma$, of the the raw estimate.

The algorithm computes gradient estimates at two points, namely $\theta_t$ and $\hat{\theta}_t + 2\mu_1 m$. The difference between these estimates is applied to estimate the Hessian. Their average estimates the gradient at the point $\theta_t + \mu_1 m$ as required by the AG algorithm.

Algorithm 1 S2AG: Self-Stabilizing Accelerated Gradient

1: Initialize $\theta$
2: $\lambda \leftarrow \beta_1/\beta_0, \mu \leftarrow \mu_0, \gamma \leftarrow \gamma_0, t \leftarrow 0$
3: Fill $\eta$ with $\ln(\beta_1/\beta_0)$
4: Fill $m$ with 0-s and $\psi_0$ with 1-s
5: $t \leftarrow t + 1$
6: Get $\xi_t$
7: $g' \leftarrow g(\theta, \xi_t)$
8: $g'' \leftarrow g(\theta + 2\mu_1, \xi_t)$
9: $r \leftarrow 2\mu_1$
10: $g \leftarrow 0.5(g' + g'')$
11: $m \leftarrow \mu_1 m - (\beta_1/\lambda) g$
12: If $\max_i |m_i/\delta_i| > 1/2$
13: $\eta \leftarrow \eta + \ln 2$
14: $\lambda \leftarrow 2\lambda$
15: Fill $m$ with 0-s
16: 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: $\frac{\partial Q_t}{\partial \eta_t} \leftarrow H_r \circ (H_r - b) + r \circ H(H_r - b)$
23: $\psi_0 \leftarrow \gamma \frac{1 - \gamma^{t-1}}{1 - \gamma} \psi_0 + \frac{1 - \gamma}{1 - \gamma^T} g_t \circ g_t$
24: $\eta \leftarrow \eta - \beta_2 \frac{\partial Q_t}{\partial \eta_t} \circ \sqrt{\psi_0 + \epsilon}$
25: If the stop criterion is not met, go to Line 5

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

... Like in Alg. 1
4: Fill $m$ with 0-s and $\psi_0, \psi_0$ with 1-s
... Like in Alg. 1
11a: $\psi_0 \leftarrow \gamma \frac{1 - \gamma^{t-1}}{1 - \gamma} \psi_0 + \frac{1 - \gamma}{1 - \gamma^T} g_t \circ g_t$
11b: $m \leftarrow \mu_1 m - (\beta_1/\lambda) g \circ \sqrt{\psi_0 + \epsilon}$
... Like in Alg. 1
22: $b \leftarrow (g'' - g') \circ \sqrt{\psi_0 + \epsilon}$
... 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 “$\otimes$” denotes elementwise division. Also, the square root that appears in Lines 11b and 24 is elementwise.

The algorithms require the following coefficients:

- $\beta_0$ — the initial learning rate; default $\beta_0 = 1,$
\(\mu_0\) — the initial momentum decay factor; default \(\mu_0 = 1/2\),
\(\beta_1\) — eventually gradients are applied multiplied by \(\beta_1/\lambda\);
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,
\(\gamma_0\) — minimal decay factor for exponential smoothing; in
our experiments \(\gamma_0 = 0.99\),
\(\beta_2\) — meta-step-size for adjustments of \(\eta\); default \(\beta_2 = 1 -
\gamma_0 = 10^{-2}\),
\(\epsilon_0\) — a small constant that prevents division by zero in
scaling increments of \(\eta\); default \(\epsilon_0 = 10^{-32}\),
\(\epsilon_1\) — 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 Scal-
ing (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 opti-
mized parameters are the momentum decay factor and the
step-size selected from the Cartesian product \{0.9, 0.99\} ×
\{0.1, 0.05, 0.02, 0.01, \ldots\} 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 Ta-
ble III presents ultimate errors obtained by all algorithms for
all learning problems.

A. Shallow neural classifiers

We take random 10 classification problems from UCI Ma-
chine 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
gray scale 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.

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 activa-
tion,
6) Transposed convolution with 32 filters 3 × 3, ReLU ac-
tivation,
7) Resizing × 2,
8) Transposed convolution output with 3 filters 3 × 3, logis-
tic 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 param-
ters (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.**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Abr*</th>
<th>Size</th>
<th>Idim</th>
<th>Odim</th>
<th>Ncnt</th>
<th>Mbs</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default of credit card clients</td>
<td>CCard</td>
<td>30000</td>
<td>32</td>
<td>2</td>
<td>21</td>
<td>200</td>
<td>10^6</td>
</tr>
<tr>
<td>Dota2 games results</td>
<td>Dota2</td>
<td>102944</td>
<td>116</td>
<td>2</td>
<td>55</td>
<td>200</td>
<td>10^7</td>
</tr>
<tr>
<td>HORU2</td>
<td>Hru2</td>
<td>17089</td>
<td>8</td>
<td>2</td>
<td>34</td>
<td>200</td>
<td>10^7</td>
</tr>
<tr>
<td>Sensorless drive diagnosis</td>
<td>Motor</td>
<td>58509</td>
<td>48</td>
<td>11</td>
<td>89</td>
<td>200</td>
<td>10^7</td>
</tr>
<tr>
<td>Poker hand</td>
<td>Poker</td>
<td>1025010</td>
<td>10</td>
<td>10</td>
<td>89</td>
<td>200</td>
<td>10^7</td>
</tr>
<tr>
<td>Wall-following Robot navigation</td>
<td>Robot</td>
<td>5456</td>
<td>24</td>
<td>4</td>
<td>34</td>
<td>200</td>
<td>10^7</td>
</tr>
<tr>
<td>Statlog shuttle</td>
<td>Shuttle</td>
<td>58000</td>
<td>9</td>
<td>7</td>
<td>34</td>
<td>200</td>
<td>10^7</td>
</tr>
<tr>
<td>Skin segmentation</td>
<td>Skin</td>
<td>245057</td>
<td>3</td>
<td>2</td>
<td>34</td>
<td>200</td>
<td>10^7</td>
</tr>
<tr>
<td>Spambase</td>
<td>Spam</td>
<td>4601</td>
<td>57</td>
<td>2</td>
<td>34</td>
<td>200</td>
<td>10^7</td>
</tr>
<tr>
<td>First order theorem proving</td>
<td>Theo</td>
<td>6118</td>
<td>51</td>
<td>6</td>
<td>89</td>
<td>200</td>
<td>10^7</td>
</tr>
<tr>
<td>Curves</td>
<td>Curves</td>
<td>20000</td>
<td>784</td>
<td>—</td>
<td>200</td>
<td>10^5</td>
<td></td>
</tr>
<tr>
<td>Handwritten Digits MNIST</td>
<td>MNIST</td>
<td>60000</td>
<td>784</td>
<td>—</td>
<td>200</td>
<td>10^5</td>
<td></td>
</tr>
<tr>
<td>Olivetti Faces</td>
<td>Faces</td>
<td>165600</td>
<td>625</td>
<td>—</td>
<td>200</td>
<td>10^5</td>
<td></td>
</tr>
<tr>
<td>CIFAR10</td>
<td>CIFAR10</td>
<td>50000</td>
<td>3072</td>
<td>—</td>
<td>200</td>
<td>10^5</td>
<td></td>
</tr>
</tbody>
</table>

* 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.**

<table>
<thead>
<tr>
<th>Alg. Problem</th>
<th>CM</th>
<th>ss*</th>
<th>mdf*</th>
<th>AG</th>
<th>ss</th>
<th>mdf</th>
<th>ADAM</th>
<th>ss</th>
<th>mdf</th>
<th>AGrad</th>
<th>ss</th>
<th>mdf</th>
<th>ADelta</th>
<th>ss</th>
<th>mdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCard</td>
<td></td>
<td>0.5</td>
<td>0.9</td>
<td></td>
<td>0.02</td>
<td>0.99</td>
<td></td>
<td>0.05</td>
<td>0.9</td>
<td>0.2</td>
<td>0.05</td>
<td>0.99</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dota2</td>
<td></td>
<td>0.05</td>
<td>0.99</td>
<td></td>
<td>0.05</td>
<td>0.99</td>
<td></td>
<td>0.002</td>
<td>0.99</td>
<td>0.2</td>
<td>0.2</td>
<td>0.09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hru2</td>
<td></td>
<td>0.02</td>
<td>0.99</td>
<td></td>
<td>0.05</td>
<td>0.99</td>
<td></td>
<td>0.002</td>
<td>0.99</td>
<td>0.5</td>
<td>1</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motor</td>
<td></td>
<td>0.02</td>
<td>0.99</td>
<td></td>
<td>0.05</td>
<td>0.99</td>
<td></td>
<td>0.005</td>
<td>0.99</td>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Poker</td>
<td></td>
<td>0.2</td>
<td>0.99</td>
<td></td>
<td>0.1</td>
<td>0.99</td>
<td></td>
<td>0.05</td>
<td>0.9</td>
<td>0.2</td>
<td>0.2</td>
<td>0.09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robot</td>
<td></td>
<td>0.2</td>
<td>0.99</td>
<td></td>
<td>0.1</td>
<td>0.99</td>
<td></td>
<td>0.02</td>
<td>0.9</td>
<td>0.2</td>
<td>0.5</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shuttle</td>
<td></td>
<td>0.1</td>
<td>0.99</td>
<td></td>
<td>0.1</td>
<td>0.99</td>
<td></td>
<td>0.05</td>
<td>0.9</td>
<td>0.5</td>
<td>1</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skin</td>
<td></td>
<td>0.2</td>
<td>0.99</td>
<td></td>
<td>0.1</td>
<td>0.99</td>
<td></td>
<td>0.02</td>
<td>0.9</td>
<td>0.2</td>
<td>0.2</td>
<td>0.09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Theo</td>
<td></td>
<td>0.01</td>
<td>0.99</td>
<td></td>
<td>0.02</td>
<td>0.99</td>
<td></td>
<td>0.01</td>
<td>0.9</td>
<td>0.2</td>
<td>0.2</td>
<td>0.09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Curves</td>
<td></td>
<td>0.01</td>
<td>0.99</td>
<td></td>
<td>0.002</td>
<td>0.9</td>
<td></td>
<td>0.002</td>
<td>0.9</td>
<td>0.2</td>
<td>1</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MNIST</td>
<td></td>
<td>0.02</td>
<td>0.99</td>
<td></td>
<td>0.02</td>
<td>0.99</td>
<td></td>
<td>0.002</td>
<td>0.9</td>
<td>0.2</td>
<td>1</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Faces</td>
<td></td>
<td>0.001</td>
<td>0.99</td>
<td></td>
<td>0.001</td>
<td>0.99</td>
<td></td>
<td>0.0002</td>
<td>0.99</td>
<td>0.01</td>
<td>0.5</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIFAR10</td>
<td></td>
<td>10^-5</td>
<td>0.99</td>
<td></td>
<td>10^-5</td>
<td>0.99</td>
<td></td>
<td>0.001</td>
<td>0.9</td>
<td>0.005</td>
<td>0.5</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

**TABLE III**

**RESULTING AVERAGE ERRORS. EACH NUMBER AVERAGES 10 RUNS.**

<table>
<thead>
<tr>
<th>Alg. Problem</th>
<th>CM</th>
<th>AG</th>
<th>ADAM</th>
<th>AdaGrad</th>
<th>AdaDelta</th>
<th>S2AG</th>
<th>S2AGS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCard</td>
<td>0.262</td>
<td>0.264</td>
<td>0.262</td>
<td>0.260</td>
<td>0.266</td>
<td>0.342</td>
<td>0.274</td>
</tr>
<tr>
<td>Dota2</td>
<td>0.422</td>
<td>0.418</td>
<td>0.412</td>
<td>0.394</td>
<td>0.405</td>
<td>0.500</td>
<td>0.417</td>
</tr>
<tr>
<td>Hru2</td>
<td>0.0326</td>
<td>0.0300</td>
<td>0.0313</td>
<td>0.0293</td>
<td>0.0294</td>
<td>0.1033</td>
<td>0.0295</td>
</tr>
<tr>
<td>Motor</td>
<td>0.0642</td>
<td>0.0480</td>
<td>0.0668</td>
<td>0.0415</td>
<td>0.0829</td>
<td>0.802</td>
<td>0.872</td>
</tr>
<tr>
<td>Poker</td>
<td>0.451</td>
<td>0.315</td>
<td>0.448</td>
<td>0.327</td>
<td>0.477</td>
<td>0.569</td>
<td>0.532</td>
</tr>
<tr>
<td>Robot</td>
<td>0.0642</td>
<td>0.0908</td>
<td>0.1198</td>
<td>0.1064</td>
<td>0.1270</td>
<td>0.6178</td>
<td>0.1311</td>
</tr>
<tr>
<td>Shuttle</td>
<td>0.0064</td>
<td>0.0045</td>
<td>0.0084</td>
<td>0.0038</td>
<td>0.0093</td>
<td>0.2768</td>
<td>0.0218</td>
</tr>
<tr>
<td>Skin</td>
<td>0.0070</td>
<td>0.0055</td>
<td>0.0074</td>
<td>0.0043</td>
<td>0.0074</td>
<td>0.2573</td>
<td>0.0102</td>
</tr>
<tr>
<td>Spam</td>
<td>0.0245</td>
<td>0.0230</td>
<td>0.0279</td>
<td>0.0172</td>
<td>0.0257</td>
<td>0.4270</td>
<td>0.0449</td>
</tr>
<tr>
<td>Theo</td>
<td>0.442</td>
<td>0.409</td>
<td>0.430</td>
<td>0.393</td>
<td>0.440</td>
<td>0.720</td>
<td>0.457</td>
</tr>
<tr>
<td>Curves</td>
<td>0.178</td>
<td>0.312</td>
<td>0.158</td>
<td>0.157</td>
<td>0.444</td>
<td>15.930</td>
<td>0.348</td>
</tr>
<tr>
<td>MNIST</td>
<td>1.03</td>
<td>1.03</td>
<td>1.27</td>
<td>1.12</td>
<td>2.11</td>
<td>15.351</td>
<td>1.30</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>5.11</td>
<td>3.75</td>
<td>2.29</td>
<td>2.29</td>
<td>4.98</td>
<td>23.75</td>
<td>3.29</td>
</tr>
</tbody>
</table>

* /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 Chęciniński, Łukasz Lepak, Daniel Klepcki, and Jakub Łyskawa.

REFERENCES


