On Exploiting Layerwise Gradient Statistics for Effective Training of Deep Neural Networks

Guoqiang Zhang, Kenta Niwa and W. Bastiaan Kleijn

Abstract—Adam and AdaBelief compute and make use of elementwise adaptive stepsizes in training deep neural networks (DNNs) via tracking the exponential moving average (EMA) of the squared-gradient $g_t^2$ and the squared prediction error $(m_t - g_t)^2$, respectively, where $m_t$ is the first momentum at iteration $t$ and can be viewed as a prediction of $g_t$. In this work, we investigate if layerwise gradient statistics can be exploited in Adam and AdaBelief to allow for more effective training of DNNs. We address the above research question in two steps. Firstly, we slightly modify Adam and AdaBelief by introducing layerwise adaptive stepsizes in their update procedures via either pre- or post-processing. Our empirical results indicate that the slight modification produces comparable performance for training VGG and ResNet models over CIFAR10 and CIFAR100, suggesting that layer-wise gradient statistics play an important role towards the success of Adam and AdaBelief for at least certain DNN tasks. In the second step, we propose Aida, a new optimisation method, with the objective that the elementwise stepsizes within each layer have significantly smaller statistical variances, and the layerwise average stepsizes are much more compact across all the layers. Motivated by the fact that $(m_t - g_t)^2$ in AdaBelief is conservative in comparison to $g_t$ in Adam in terms of layerwise statistical averages and variances, Aida is designed by tracking a more conservative function of $m_t$ and $g_t$ than $(m_t - g_t)^2$ via layerwise vector projections. Experimental results show that Aida produces either competitive or better performance with respect to a number of existing methods including Adam and AdaBelief for a set of challenging DNN tasks.

Index Terms—Adaptive gradient descent, Adam, AdaBelief, Aida

I. INTRODUCTION

Iterative optimisation is one key step in machine learning (ML) that enables an ML model to gradually capture useful information or patterns embedded in a dataset via either supervised or unsupervised learning. Generally speaking, different ML models require the design of specific optimisation methods for effective learning. For instance, compressive sensing (CS) can be solved efficiently by forward-backward splitting and its variants [1]. This paper focuses on effective training of deep neural networks (DNNs) via adaptive gradient descent.

Stochastic gradient descent (SGD) and its variants have been widely applied in deep learning due to their simplicity and effectiveness [2]. In the literature, SGD with momentum, also known as heavy ball (HB) method [3], [4], has been widely used for image classification tasks [5], [6]. Suppose the objective function $f(\theta) : \mathbb{R}^d \rightarrow \mathbb{R}$ of a DNN model is differentiable, where $\theta \in \mathbb{R}^d$ denotes the model parameters. The update expression of HB for minimising $f(\theta)$ can be represented as

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

$$\theta_t = \theta_{t-1} - \gamma_t m_t,$$

where $g_t = \nabla f(\theta_{t-1})$ is the gradient at $\theta_{t-1}$, $\beta_t$ is the momentum coefficient, and $\alpha_t$ with fixed $\gamma_t = 1$ or $\gamma_t$ with fixed $\alpha_t = 1$ is the common step-size for all the coordinates of $\theta$. In practice, the HB method is often combined with a certain step-size scheduling method when training DNNs.

To bring flexibility to the HB method, one active research trend in last decade has been to introduce additional adaptive step-sizes for all the components of $m_t$ in (2), referred to as adaptive optimisation. Duchi et al [7], were the first to track the moment of the squared-gradient in computation of the adaptive step-sizes. The basic principle is to identify and multiply those coordinates of $m_t$ that have historically small
squared-gradients on average with relatively large learning rates to enhance their contributions to the updates of $\theta_t$, and vice versa. The proposed method, Adagrad, is found to converge considerably fast when the gradients are sparse. Following the work of [7], various adaptive optimisation methods have been proposed for computing more effective adaptive step-sizes (e.g., [8], [9]).

In the literature, Adam [9] is probably most popular among the existing adaptive optimisation methods (e.g., [10], [11], [12], [13]). The update expression of Adam at iteration $t$ can be written as:

$$
m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\theta_t = \theta_{t-1} - \eta t \frac{m_t}{\sqrt{v_t} \delta (1 - \beta_2)} + \epsilon$$

where $0 < \beta_1, \beta_2 < 1$, and $\epsilon > 0$. The two vector operations ($\cdot)^2$ and $\cdot \delta$ are conducted in an elementwise manner. The two quantities $1 - \beta_1^2$ and $1 - \beta_2^2$ in (5) are introduced to compensate for the estimation bias in the two exponential moving averages (EMAs) $m_t$ and $v_t$, respectively. The parameter $\eta_t$ is the common stepsize while $1/((\sqrt{v_t} \delta (1 - \beta_2^2) + \epsilon) \in R^d$ represents the elementwise adaptive stepsizes.

Insightful theoretical attempts have also been made to study the convergence behaviours of Adam. The incorrectness of the convergence proof of Adam in the original paper [9] was spotted [14], [15]. It was found in [16] that Adam does not converge for a class of stochastic convex optimisation problems. [17] studied the convergence of Adam and its variant for non-convex optimisation.

In recent years, due to the great success of Adam in training DNNs, various extensions of Adam have been proposed. For instance, AdamW [18] proposes to decouple the gradient descent and the weight decay operation. NAdam [19] combines the update expressions of Adam with the techniques developed by Nesterov [20] to accelerate convergence speed. Yogi [21] modifies the computation of $v_t$ in Adam for the purpose of increasing controllability of the elementwise stepsizes. The method AdaBelief [22] proposes to track the EMA of $(m_t - g_t)^2$ and then updates the DNN model $\theta$ accordingly, which can be represented as

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2)(m_t - g_t)^2 + \epsilon$$

$$\theta_t = \theta_{t-1} - \eta_t \frac{1}{1 - \beta_1^2} \sqrt{s_t} \delta (1 - \beta_2^2) + \epsilon$$

where the vector $1/((\sqrt{v_t} \delta (1 - \beta_2^2) + \epsilon)$ denotes the elementwise adaptive stepsizes. It is noted that in principle, the layerwise averages and variances of the subvectors in $(m_t - g_t)^2$ should be smaller than those in $g_t^2$ since the first momentum $m_t$ provides a reliable prediction of $g_t$ in general. We will use the above insight in designing our new optimisation method later on.

In this paper, we determine to find out if layerwise gradient statistics can be used for designing more effective optimisation methods. Our motivation stems from the fact that both Adam and AdaBelief produce promising performance as reported in the literature while the EMA $v_t$ in Adam and $s_t$ in AdaBelief track different dynamics of gradients over iterations (see also [21] for the details of Yogi). This suggests that the layerwise gradient statistics might play an important role in Adam and AdaBelief, which remain to be exploited explicitly to improve the performance.

We consider exploiting layerwise gradient statistics in two steps. Firstly, we make a slight modification to Adam and AdaBelief by introducing and using layerwise adaptive stepsizes instead of elementwise ones. Two approaches are proposed to serve the above purpose. The first approach takes the mean value per layer of the obtained individual stepsizes in Adam and AdaBelief as the layerwise stepsizes. The resulting methods are referred to as AdamL and AdaBeliefL (see Table I). Denote $g_l$ and $m_l$ as the subvectors of $g$ and $m_t$ for the $l$th layer of a DNN model. In the second approach, we propose to track the EMA of the layerwise mean value of $g_{l,t}$ or $(m_{l,t} - g_{l,t})^2$ and then use them to compute the layerwise adaptive stepsizes accordingly, which are referred to as LAdam and LAdaBelief (see Table I). We point out that LAdam
and LAdaBelief are considerably more memory-efficient than Adam and AdaBelief. Empirical study indicates that the four new methods produce comparable validation performance as Adam and AdaBelief for training VGG and ResNet models over CIFAR10 and CIFAR100. This suggests that layerwise gradient statistics are indeed the key factors contributing to the success of Adam and AdaBelief for at least certain DNN tasks. However, it is noted that the four proposed methods are not able to push the layerwise stepsizes get close to each other as the pre and post processing only perform certain layerwise averaging operations.

Secondly, we propose a new method, referred to as Aida, with the objective that the elementwise stepsizes within each neural layer have significantly small statistic variances, and the layerwise average stepsizes are compact accross all the layers. Inspired by the extension from Adam to AdaBelief, Aida tracks the EMA of a more conservative function of $m_{l,t}$ and $g_{l,t}$ for the $l$th layer. In particular, at iteration $t$, a sequence of mutual vector projections starting from the pair $(m_{l,t}, g_{l,t})$ is performed to obtain a set of projected vectors $\{ (m_{l,t}^{(k)}, g_{l,t}^{(k)}) | k = 1, \ldots, K \}$. The EMA of $(m_{l,t}^{(K)} - g_{l,t}^{(K)})^2$ is then tracked and used for computing the elementwise stepsizes for the $l$th layer in Aida.

The results in Fig. 1 and 2 indicate that the layerwise standard deviations of the elementwise stepsizes in Aida are significantly smaller than those of AdaBelief and Adam+. Furthermore, the layerwise average stepsizes of Aida tend to be much more desirably compact than those of the other two methods. This indicates that the behavior of Aida is closer to that of SGD with momentum than the other two methods in that the elementwise stepsizes of Aida are compact both layerwisely and globally. Experimental results in Section III show that Aida yields either competitive or better performance in comparison to a number of existing methods including AdaBelief and Adam for a set of challenging DNN tasks.

**Notations:** We use bold small letters to denote vectors. The $l_2$ norm of a vector $\mathbf{y}$ is denoted as $\| \mathbf{y} \|$. Given an $L$-layer DNN model $\theta$ of dimension $d$, we use $\theta_i$ of dimension $d_i$ (i.e., $\theta_i \in \mathbb{R}^{d_i}$) to denote the subvector of $\theta$ for the $i$th layer. Thus, there is $\sum_{i=1}^{L} d_i = d$. The $i$th element of $\theta_i$ is represented by $\theta_i[i]$. The notation $[L]$ stands for the set $[L] = \{1, \ldots, L\}$.

II. ALGORITHMIC DESIGN BY EXPLOITING LAYERWISE GRADIENT STATISTICS

In this section, we first consider introducing layerwise stepsizes to Adam and AdaBelief by pre- and post-processing. Our main objective is to demonstrate that it is the layerwise gradient statistics that contribute to the success of Adam and AdaBelief for typical DNN tasks. After that, we present Aida, a new optimisation method as an extension of AdaBelief.

A. Introduction of Layerwise Stepsizes in Adam and AdaBelief

We provide two approaches to introduce layer-wise stepsizes into two representative Adam-type methods: Adam and AdaBelief. The first approach uses layerwise post-processing which leads to AdamL and AdaBeliefL while the second one uses layerwise pre-processing that leads to LAdam and LAdaBelief. It is noted that the pre-processing makes LAdam and LAdaBelief considerably more memory-efficient than Adam and AdaBelief. We would like to iteratively optimise an $L$-layer DNN model $\theta$.

**On computing layerwise stepsizes by post-processing:** We first modify Adam to obtain AdamL. Suppose at iteration $t$, the EMA $\nu_t$ is computed as

$$\nu_t = \beta_2 \nu_{t-1} + (1 - \beta_2) g_t^2 + \epsilon,$$

where $1 > \beta_2 > 0$, $\epsilon > 0$. The parameter $\epsilon$ is placed in the update expression of $\nu_t$ to be consistent with the computation of $s_t$ in AdaBelief. Since (8) already guarantees the positiveness of $\nu_t$, there is no need to introduce $\epsilon$ again when computing the adaptive stepsizes later on. We calculate the mean value $\gamma_{l,t}$ of the individual stepsizes of the $l$th layer as

$$\gamma_{l,t} = \frac{1}{d_l} \sum_{i=1}^{d_l} \frac{1}{\sqrt{\nu_{l,t}[i] / (1 - \beta_2)}}, \quad l \in [L],$$

where $\nu_{l,t}[i]$ is a subvector of $\nu_t$ corresponding to the $i$th layer.

Intuitively speaking, the parameters within the same neural layer in a DNN model are functionally homogeneous in that they follow the same rule in processing data from the layer below and producing output to the layer above. As a result, their gradient statistics should be similar. Since the parameter
\( \gamma_{l,t} \) represents the statistic average of the individual stepsizes of the \( l \)th layer, which, in principle, can be used to replace the corresponding individual stepsizes. Thus, the subvector \( \theta_{l,t} \) for the \( l \)th layer can be updated with \( \gamma_{l,t} \) as

\[
\theta_{l,t} = \theta_{l,t-1} - \eta_t \left( 1 - \beta_2^t \right) \gamma_{l,t} m_{l,t} \quad l \in [L],
\]

(10)

See Table I for a summary of AdamL. Similarly, AdaBeliefL can be obtained by replacing \( g_{l,t}^2 \) with \((m_{l,t} - g_{l,t})^2\) in AdamL.

**On computing layerwise stepsizes by pre-processing:** We now introduce layerwise stepsizes in Adam and AdaBelief via pre-processing to reduce their memory consumption. Even though the memory space of modern GPUs keeps increasing over time, the sizes of advanced DNN models also tend to increase to deal with complicated learning tasks. Furthermore, for certain applications, only small memory space might be available for training DNN models (e.g., learning over phones).

As an example, we consider modifying Adam to obtain LAdam. To do so, a scalar parameter \( q_t \in \mathbb{R} \) is introduced for the \( l \)th layer. At iteration \( t \), the EMA \( q_{l,t} \) is computed as

\[
g_{l,t} = \beta_2 g_{l,t-1} + (1 - \beta_2^t) \frac{1}{d_l} \sum_{i=1}^{d_l} g_{l,t}[i]^2 + \epsilon \quad l \in [L],
\]

(11)

where again \( 1 > \beta_2 > 0 \) and \( \epsilon > 0 \). In the above expression, the statistic average of \( g_{l,t}^2 \) is computed and used because of the functional homogeneity of the parameters within the same layer as explained earlier. With the scalar \( q_{l,t} \), \( \theta_{l,t} \) of the \( l \)th layer can be updated accordingly (see Table I). LAdaBelief can be obtained similarly as AdaBeliefL.

In practice, the number of neural layers including bias vectors and scalar vectors of, for example, batch normalisation, is significantly smaller than the total number of parameters in a typical DNN model. As a result, the number of parameters \( \{q_t\}_{t=1}^L \) in LAdam is much smaller than the dimensional \( d \) of \( \theta_t \in \mathbb{R}^d \) in Adam. To summarise, both LAdam and LAdaBelief save almost half of memory needed in Adam or AdaBelief which are required to store both \( m_t \) and \( v_t \) (or \( s_t \)).

Fig. 3 displays the layerwise (average) stepsizes of five methods for training VGG11. The layerwise average stepsizes of AdaBelief at iteration \( t \) are computed as \( \left\{ \frac{1}{d_l} \sum_{i=1}^{d_l} [1/\sqrt{s_{l,i}[i]/(1 - \beta_2^t) + \epsilon}] \right\}_{l=1}^L \), where \( s_l \) is from (6) and \( s_{l,t} \) is a subvector of \( s_l \) for the \( l \)th layer. On the other hand, the layerwise stepsizes for AdamL at iteration \( t \) are \( \left\{ q_{l,t} \right\}_{l=1}^L \) by following (6). And those for LAdam are \( \left\{ 1/\sqrt{q_{l,t}[(1 - \beta_2^t)]} \right\}_{l=1}^L \), where \( q_{l,t} \) is computed by (11). It is seen from the figure that for each layer index of \( \{2, 6, 11\} \), the layerwise (average) stepsizes of all the five methods have similar trajectories over epochs. It is expected that their validation performance should also be similar. As will be demonstrated in Section III the five methods in Fig. 3 indeed produce comparable performance for training VGG and ResNet models over CIFAR10 and CIFAR100. This indicates that layerwise gradient statistics truly play an important role for at least certain DNN tasks.

**B. Aida: tracking more conservative gradient aspects than AdaBelief**

In this subsection, we first explain in detail why our new method Aida is designed to track a more conservative function of \( m_{l,t} \) and \( g_{l,t} \) for the \( l \)th layer of a DNN model than \((m_{l,t} - g_{l,t})^2\) being employed in AdaBelief. After that, we present the update expressions of Aida.

**Motivation:** We would like to design a new method, of which the individual stepsizes within each neural layer have significantly smaller variances than those of Adam and AdaBelief while at the same time, the layerwise average stepsizes of the new method are much more compact than those of Adam and AdaBelief. This is motivated partially by the fact that the parameters within the same neural layer are functionally homogeneous. Therefore, their individual adaptive stepsizes should be relatively close to each other. Furthermore, it is also desirable that all the layerwise average stepsizes are relatively close to each other so that the impact of the gradient descent operation is not significantly different across different neural layers. SGD with momentum is the extreme case which does not take any layerwise or elementwise variation of gradient into account, which may not always be an optimal method for a broad class of DNN tasks. The four proposed methods in Subsection II-A follow two approaches to enforce each neural layer in a DNN model to use the same adaptive stepsize. As illustrated in Fig. 3 the four methods are not able to push the layerwise average stepsizes to get close to each other.

Now let us revisit AdaBelief. It proposes to track the EMA of \((m_t - g_t)^2\) in comparison to \(g_t^2\) in Adam. By assuming the mini-batches of training samples are identically and independently distributed, it is reasonable to take the first momentum \( m_t \) as a reliable prediction of the current gradient \( g_t \). Considering the \( l \)th layer, the prediction error \((m_{l,t} - g_{l,t})^2\) should be relatively small compared to the gradient \( g_{l,t}^2 \) in terms of statistic average and variance across all its coordinates. That is, the dynamics of \((m_{l,t} - g_{l,t})^2\) is more conservative than that of \(g_{l,t}^2\). As a result, the layerwise standard deviations of the individual stepsizes of AdaBelief should be smaller than those of Adam while the layerwise average stepsizes should be larger than those of Adam (see Fig. 1 for empirical verification).

By following the above reasoning of extending Adam to AdaBelief, we design the new method, named Aida, as an extension of AdaBelief by tracking the EMA of a more conservative function of \( m_{l,t} \) and \( g_{l,t} \) for the \( l \)th layer. This would likely lead to the desired property that the new method pushes the layerwise standard deviations of the individual stepsizes to be significantly smaller than those of AdaBelief. We will explain later that it is doable to design a layerwise conservative function of \( m_{l,t} \) and \( g_{l,t} \) so that the resulting layerwise average stepsizes of Aida are more compact than those of AdaBelief.

**Update expressions of Aida:** Consider the \( l \)th layer of a DNN model at iteration \( t \). We would like to track the EMA of a more conservative function of \( m_{l,t} \) and \( g_{l,t} \) than the squared prediction error \((m_{l,t} - g_{l,t})^2\) as employed in AdaBelief. To do so, we propose to first perform a sequence
Adam: \( \{ g^0_{l,t} \mid t \geq 0 \} \)

AdaBelief: \( \{ (m^1_{l,t} - g^1_{l,t})^2 \mid t \geq 0 \} \)

Aida: \( \{ (m^{(K)}_{l,t} - g^{(K)}_{l,t})^2 \mid t \geq 0 \} \)

of mutual vector projections to obtain a set of projected vectors \( \{ (m^{(k)}_{l,t}, g^{(k)}_{l,t}) \mid k = [K] \} \) starting from the initial pair \( (m^{(0)}_{l,t}, g^{(0)}_{l,t}) = (m_{l,t}, g_{l,t}) \). By using algebra, the two vectors at iteration \( 2k + 1 \) can be represented as

\[
\begin{align*}
    m^{(2k+1)}_{l,t} &= \frac{(g^{(2k)}_{l,t}, m^{(2k)}_{l,t})}{\|g^{(2k)}_{l,t}\|^2 + \xi} \cdot m^{(2k)}_{l,t} \quad \text{(12)} \\
    g^{(2k+1)}_{l,t} &= \frac{(g^{(2k)}_{l,t}, m^{(2k)}_{l,t})}{\|m^{(2k)}_{l,t}\|^2 + \xi} \cdot g^{(2k)}_{l,t}, \quad \text{(13)}
\end{align*}
\]

where \( \langle \cdot, \cdot \rangle \) denotes inner product, and \( \xi > 0 \) is a scalar parameter to ensure the division operations are valid. It is not difficult from Fig. 4 and (12)-(13) to obtain the update expressions for \( (m^{(2k)}_{l,t}, m^{(2k)}_{l,t}) \).

Once \( (m^{(K)}_{l,t}, g^{(K)}_{l,t}) \) are obtained for the \( l \)th layer, Aida tracks the EMA of the squared difference \( (m^{(K)}_{l,t} - g^{(K)}_{l,t})^2 \), given by

\[
v_{l,t} = \beta_2 v_{l,t-1} + (1 - \beta_2) (m^{(K)}_{l,t} - g^{(K)}_{l,t})^2 + \epsilon, \quad \text{(14)}
\]

where \( 1 > \beta_2 > 0 \) and \( \epsilon > 0 \). With \( v_{l,t} \), the model parameters \( \theta_{l,t} \) of the 1th layer can be updated accordingly. See Table III for the key update steps of Aida. We point out that since the vector projections are layerwise oriented, Aida implicitly makes use of layerwise gradient statistics in its update expressions while Adam and AdaBelief only exploit the elementwise gradient statistics.

Next we consider the geometrical properties of the set of projected vectors. It is not difficult to show that after projection, the obtained vectors have either smaller or equal lengths in comparison to the original ones:

\[
\|m^{(k)}_{l,t}\| \leq \|m^{(k-1)}_{l,t}\| \quad \text{and} \quad \|g^{(k)}_{l,t}\| \leq \|g^{(k-1)}_{l,t}\|. \quad \text{(15)}
\]

Using the fact that mutual projections of two vectors do not change the angle, we then have

\[
\|m^{(k)}_{l,t} - g^{(k)}_{l,t}\| \leq \|m^{(k-1)}_{l,t} - g^{(k-1)}_{l,t}\|, \quad \text{(16)}
\]

where the equality holds when \( m_{l,t} \) and \( g_{l,t} \) are on the same line and \( \xi \) can be ignored in (12)-(13).

For the extreme case that each neural layer has only one parameter (i.e., \( g_{l,t} \in \mathbb{R}, \forall l \in [L] \)), one can easily show that the projection operation has no effect. That is, \( (g^{(1)}_{l,t} - m^{(1)}_{l,t})^2 = (g_{l,t} - m_{l,t})^2 \) for all \( k \in [K] \) if \( \xi \) is ignored in (12)-(13). In this case, Aida reduces to AdaBelief.

From a statistic point of view, due to the functional homogeneity of the parameters within the \( l \)th layer, the elements of the squared difference \( (m^{(K)}_{l,t} - g^{(K)}_{l,t})^2 \) should follow a single distribution. That is, those elements are statistically similar. Therefore, from (6), (14) and (16), it is reasonable to argue that the layerwise average stepsizes and standard deviations of Aida are respectively larger and smaller than those of AdaBelief if the same common stepsize \( \eta_l \) is used in the two methods for training a DNN model.

Next we argue that the layerwise average stepsizes of Aida across all neural layers should be more compact than those of AdaBelief when training a DNN model. It is noted that the angles \( \{ \angle g_{l,t}, m_{l,t} \mid t \geq 0 \} \) affect not only the layerwise average stepsizes of the \( l \)th layer in AdaBelief but also the differences between \( \{ (g^{(K)}_{l,t} - m^{(K)}_{l,t})^2 \mid t \geq 0 \} \) in Aida and \( \{ (g_{l,t} - m_{l,t})^2 \mid t \geq 0 \} \) in AdaBelief. Consider optimising a DNN model by AdaBelief. Those neural layers who have smaller angles (or equivalently, accurate prediction of \( \{ g_{l,t} \mid t \geq 0 \} \) via \( \{ m_{l,t} \mid t \geq 0 \} \)) across iterations would have larger layerwise average stepsizes than other layers. If Aida is also used for training the same DNN model, the layerwise average stepsizes in the neural layers who have small angles would be roughly the same as those in the corresponding layers when applying AdaBelief. On the other hand, large angles in those neural layers would push the small layerwise average stepsizes of AdaBelief to be considerably large in Aida. That is, the angles \( \{ \angle g_{l,t}, m_{l,t} \mid l \in [L], t \geq 0 \} \) have a nonlinear impact on the vector projections, making the resulting layerwise average
stepsizes of Aida to be more compact.

Fig. 5 provides convincing empirical evidence to support our earlier argument. It is clear that Aida indeed has significantly smaller layerwise standard deviations than those of AdaBelief and Adam+. The layerwise average stepsizes of Aida are much more compact than those of the other two methods, which may be beneficial for training a DNN model. Consider the extreme case that the histogram of all the adaptive stepsizes is quite broad and flat over iterations. This would make the impact of the gradient descent operations significantly nonuniform in the parameter space. Certain neurons with very small adaptive stepsizes would receive quite low-magnitude updates in the whole training procedure, which is undesirable in the learning process.

III. EXPERIMENTS

We evaluate our new method Aida for image classification, natural language processing, and image generation. In all experiment, the additional parameter \( \xi \) in Aida in Alg. 3 was set to \( \xi = 1 e - 20 \). It is found that Aida performs either better or competitive in comparison to a number of popular adaptive optimisation methods.

A. Training of VGG11 and ResNet34 over CIFAR10 and CIFAR100

We adopted the open source\(^1\) for AdaBelief of \(^2\) due to the fact that Aida built upon AdaBelief. In total, thirteen methods were tested (see Table II) by following a similar experimental setups in \(^2\). In particular, the methods Aida, AdaBelief, AdaBeliefL, LAdaBelief, Adam, AdamL, LAdam, AdamW, and RAdam\(^3\) share the parameters \((\eta_0, \beta_1, \beta_2, \epsilon) = (0.001, 0.9, 0.999, 1e - 8)\). The parameters of Yogi \(^4\) and MSVAG \(^5\) were set to \((\eta_0, \beta_1, \beta_2, \epsilon) = (0.001, 0.9, 0.999, 0.001)\) and \((\eta_0, \beta_1, \beta_2, \epsilon) = (0.1, 0.9, 0.999, 1e - 8)\), respectively. The learning rate of Fromage \(^6\) was set to 0.01. As for SGD, the momentum and initial stepsize were set to 0.9 and 0.1, respectively. The weight decay in all methods except AdamW was set to 5e - 4, where in AdamW, it was set to 0.01. The batchsize and epoch were set to 128 and 200. The common stepsize \( \eta_t \) is decreased by multiplying 0.1 at 100 and 160 epoch. Three experimental repetitions were conducted for each optimiser to alleviate the effect of the randomness.

The validation performance is summarised in Table III. It is clear that the newly proposed five methods have comparable validation accuracies as AdaBelief and SGD.

B. On training a Transformer

In this experiment, we compare the performance of Aida, AdaBelief, and Adam in training a Transformer for WMT16: multimodal translation. To make a fair comparison, we adopted an existing open-source\(^7\) which produces reasonable validation performance using Adam. In the training process, we retained almost all of the default hyper-parameters provided in the open-source except the batchsize. Due to limited GPU memory, we changed the batchsize from 256 to 200. The parameters of Adam was set to \((\beta_1, \beta_2, \epsilon) = (0.9, 0.999, 1e - 9)\) while the parameters of Aida and AdaBelief were set to \((\beta_1, \beta_2, \epsilon) = (0.9, 0.999, 1e - 18)\). Three experimental repetitions were conducted for each optimiser to alleviate the effect of the randomness.

C. On training LSTMs

In this experiment, we investigate the performance of Aida and AdaBelief for training LSTMs over the Penn TreeBank dataset \(^8\), of which \((\beta_1, \beta_2) = (0.9, 0.999)\) and the weight decay was set to 1.2e - 6. In Aida, \((\eta_0, \epsilon)\) were set to \((0.001, 1e - 16)\) while in AdaBelief, \( \epsilon \in \{1e - 8, 1e - 12, 1e - 16\} \) and \( \eta_0 \in \{0.01, 0.001\} \) were searched to find the optimal configuration that produces the best validation performance.

Fig. 6 displays the obtained validation perplexity (lower is better) of the two methods for training 1, 2, 3-layer LSTMs. It is clear that Aida outperforms AdaBelief considerably in all three scenarios. One observes that in plot (a), the two curves cross with each other while in plot (b) and (c), there is no crossing between the two curves. This is because the optimal configurations of \((\epsilon, \eta_0)\) for AdaBelief are different between (a) and (b)-(c).

---

1 https://github.com/juntang-zhuang/AdaBelief-Optimizer
2 https://github.com/jadore801120/attention-is-all-you-need-pytorch
3 https://github.com/LadaBelief
4 https://github.com/yogi-y/SGD
5 https://github.com/MSVAG
6 https://github.com/Fromage
7 https://github.com/jadore801120/attention-is-all-you-need-pytorch
8 https://github.com/PennTreeBank
methods. We conduct analysis in two steps. Firstly, layerwise adaptive stepsizes are introduced to Adam and AdaBelief via either pre or post processing to obtain four new methods: LAdam, LAdaBelief, AdamL, and AdaBeliefL. It is noted that the first two methods LAdam and LAdaBelief are memory-efficient in comparison to the original ones due to the fact that only a scalar parameter per neural layer is required to be stored and updated for computing the layerwise stepsizes. Empirical study shows that the four new methods produce comparable performance as the original ones in a few typical image classification tasks, indicating that layerwise gradient statistics can indeed help with effectiveness of Adam-type methods.

In the second step, we propose Aida as an extension of AdaBelief with the objective that the layerwise standard deviations of individual stepsizes when running the new method are significantly smaller than those of AdaBelief. This is motivated by the fact that the parameters within the same neural layer are functionally homogeneous. Correspondingly, the individual adaptive stepsizes within the same layer should not be far away from each other. To achieve the above goal, Aida is designed to track the EMA of layerwise squared individual stepsizes \((\hat{g}_{l,t})^2\) of the \(K\)-th order projected vectors \((m_{l,t} - g_{l,t})^2\) as opposed to the EMA of \((m_{l,t} - g_{l,t})^2\) in AdaBelief. Experiments show that Aida performs either better or competitive with respect to a number of popular methods from literature.

### REFERENCES

[1] P. L. Combettes and J.-C. Pesquet, “Proximal splitting methods in signal processing. In: Fixed-Point Algorithms for Inverse Problems in Science and Engineering,” Springer, pp. 185–212, 2011.

[2] Y. LeCun, Y. Bengio, and G. Hinton, “Deep Learning,” Nature, vol. 521, pp. 436–444, 2015.

[3] H. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in International Conference on Machine Learning (ICML), 2013.

[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] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[6] A. C. Wilson, R. Roelofs, M. Stern, N. Srivastava, and B. Recht, “The Marginal Value of Adaptive Gradient Methods in Machine Learning,” in 31st Conference on Neural Information Processing Systems (NIPS), 2017.
Algorithm 1: AidaGrad

1: Input: $\beta_1$, $\beta_2$, $\eta$, $\epsilon > 0$, $\xi > 0$ 
2: Init.: $x_0 \in \mathbb{R}^d$, $m_0 = 0$, $v_0 \in \mathbb{R}^d$ 
3: for $t = 1, 2, \ldots, T$ do 
4: \hspace{1em} $g_t \leftarrow \nabla f(x_{t-1})$ 
5: \hspace{1em} $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)g_t$ 
6: \hspace{1em} for $l = 1, \ldots, L$ do 
7: \hspace{2em} $g_{l,t}^p \leftarrow \frac{(m_{l,t} \eta_k)^2}{\|m_{l,t} \eta_k\|^2 + \xi}$ 
8: \hspace{2em} $v_{l,t} \leftarrow \beta_2 v_{l,t-1} + (1 - \beta_2)(g_{l,t}^p)^2 + \epsilon$ 
9: \hspace{1em} end for 
10: \hspace{1em} $\tilde{m}_t \leftarrow \frac{m_t}{1 - \beta_1}$, $\tilde{v}_t \leftarrow \frac{v_t}{1 - \beta_2}$ 
11: \hspace{1em} $x_t \leftarrow x_{t-1} - \frac{\eta}{\sqrt{\tilde{v}_{l,t}}} \tilde{m}_t$ 
12: end for 
13: Output: $x_T$

Algorithm 2: Aida with Preprocessing Layerwise Stepsizes (Laida)

1: Input: $\beta_1$, $\beta_2$, $\eta$, $\epsilon > 0$, $\xi > 0$, $K \geq 1$ 
2: Init.: $x_0 \in \mathbb{R}^d$, $m_0 = 0$, $\{v_{l,0} \in \mathbb{R}\}_{l=1}^L$ 
3: for $t = 1, 2, \ldots, T$ do 
4: \hspace{1em} $g_t \leftarrow \nabla f(x_{t-1})$ 
5: \hspace{1em} $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)g_t$ 
6: \hspace{1em} for $l = 1, \ldots, L$ do 
7: \hspace{2em} $m_{l,t}^{(0)} = m_{l,t}$, $g_{l,t}^{(0)} = g_{l,t}$ 
8: \hspace{2em} for $k = 1, \ldots, K$ do 
9: \hspace{3em} $m_{l,t}^{(k)} = \frac{(m_{l,t}^{(k-1)} - g_{l,k-1})}{\|m_{l,t}^{(k-1)} - g_{l,k-1}\| + \epsilon}$ 
10: \hspace{3em} $g_{l,t}^{(k)} = \frac{\|m_{l,t}^{(k-1)} - g_{l,k-1}\|}{\|m_{l,t}^{(k-1)} - g_{l,k-1}\|^2 + \xi}$ 
11: \hspace{2em} end for 
12: \hspace{2em} $v_{l,t} \leftarrow \beta_2 v_{l,t-1} + (1 - \beta_2)\text{mean}(m_{l,t}^{(K)} - g_{l,t}^{(K)})^2 + \epsilon$ 
13: \hspace{2em} end for 
14: \hspace{1em} $\tilde{m}_t \leftarrow \frac{m_t}{1 - \beta_1}$, $\tilde{v}_t \leftarrow \frac{v_t}{1 - \beta_2}$ 
15: \hspace{1em} for $l = 1, \ldots, L$ do 
16: \hspace{2em} $x_{t,l} \leftarrow x_{t-1,l} - \frac{\eta}{\sqrt{\tilde{v}_{l,t}}} \tilde{m}_{l,t}$ 
17: \hspace{1em} end for 
18: end for 
19: Output: $x_T$