Computing, Telecommunication and Control, 2022, Vol. 15, No. 4, Pp. 86–97.
Информатика, телекоммуникации и управление. 2022. Том 15, № 4. С. 86–97.

Research article
DOI: https://doi.org/10.18721/JCSTCS.15407
UDC 004.852

FIXING 1-BIT ADAM AND 1-BIT LAMB ALGORITHMS

D.A. Tarasov[1] ✉ , *V.A. Ershov*[2]

[1] National Research University Higher School of Economics, St. Petersburg, Russian Federation;

[2] Yandex LLC, St. Petersburg, Russian Federation

✉ tarasov.denis.al@gmail.com

Abstract. Today, various neural network models are trained using distributed learning in order to reduce the time spent. The most common way of distributed learning today is the approach, in which the data are divided into parts and sent along with the model to different devices, each device calculates updates for the model, then the updates are aggregated on the server, the server updates the weights of the model and transfers their new version to the devices. Slow network communication between devices can significantly reduce distribution efficiency. Recent studies propose one-bit versions of the Adam and LAMB algorithms, which can significantly reduce the amount of transmitted information, thus improving the scalability of training. However, it turned out that these algorithms diverge in some neural network architectures. The goal of this work is an empirical study of these algorithms, to find the solution of the discovered divergence problem and propose new aspects of testing gradient descent algorithms.

Keywords: machine learning, deep learning, gradient descent, distributed training, optimization

Acknowledgements: We thank Andrey Kirilenko for help in solving different technical issues, which appeared during our research. We also thank Gleb Yengalych for comments on the manuscript.

Citation: Tarasov D.A., Ershov V.A. Fixing 1-bit Adam and 1-bit LAMB algorithms. Computing, Telecommunications and Control, 2022, Vol. 15, No. 4, Pp. 86–97. DOI: 10.18721/JCSTCS.15407

Научная статья
DOI: https://doi.org/10.18721/JCSTCS.15407
УДК 004.852

РЕШЕНИЕ ПРОБЛЕМ АЛГОРИТМОВ 1-BIT ADAM И 1-BIT LAMB

Д.А. Тарасов[1] ✉ , *В.А. Ершов*[2]

[1] Национальный исследовательский университет «Высшая школа экономики», Санкт-Петербургский филиал, Санкт-Петербург, Российская Федерация;

[2] «Яндекс», Санкт-Петербург, Российская Федерация

✉ tarasov.denis.al@gmail.com

Аннотация. На сегодняшний день различные нейросетевые модели учат с помощью распределенного обучения, чтобы снизить затрачиваемое время. Самым распространенным способом распределенного обучения является подход, при котором данные разбиваются на части и вместе с моделью отправляются на разные устройства, каждое устройство вычисляет обновления для модели, затем обновления аггрегируются на сервере, сервер обновляет веса модели и передает их новую версию на устройства. Медленное сетевое взаимодействие, связывающее устройства, на которых происходит обучение, может значительно снизить эффективность распределения. Недавние исследования предлагают однобитные версии алгоритмов Adam и LAMB, позволяющие сократить объём передаваемой информации в несколько раз, вследствие чего масштабируемость обучения улучшается. Однако на практике оказалось, что данные алгоритмы расходятся на некоторых архитектурах нейронных сетей. Цель статьи – эмпирическое исследование указанных алгоритмов, решение обнаруженной проблемы расходимости, а также рассмотрение новых аспектов для тестирования алгоритмов градиентного спуска.

Ключевые слова: машинное обучение, глубинное обучение, градиентный спуск, распределенное обучение, оптимизация

Благодарности: Благодарим Андрея Кириленко за помощь в решении технических проблем, возникших во время исследования. Также благодарим Глеба Енгалыча за комментарии к рукописи.

Для цитирования: Tarasov D.A., Ershov V.A. Fixing 1-bit Adam and 1-bit LAMB algorithms // Computing, Telecommunications and Control. 2022. Т. 15, № 4. С. 86–97. DOI: 10.18721/JCSTCS.15407

1. Introduction

The latest breakthroughs in deep learning bring about many challenges from areas such as natural language processing, computer vision, and more. The training of neural networks is usually performed on GPUs (Graphical Processing Units) or TPUs (Tensor Processing Units) and their power constantly grows, but the training speed-up is compensated by the fact that models become more complex and the amount of data used for training increases. Training a single model may require weeks, while people need to run many experiments to find a good set of hyperparameters or continuously train existing models using new available data. Therefore, there is a lot of research, the goal of which is to reduce training time while maintaining quality.

Optimization problems arising in training neural network models are solved using the stochastic gradient descent (SGD) method and its various modifications. The aim of several studies is to get an optimization algorithm with not only better convergence rate in theory, but one which provides a speed-up in practice as well: SGD with Nesterov momentum, RMSProp [1], Adagrad [2], Adadelta [3], Adam [4],

LARS [5], LAMB [6], Novograd [7], etc. At the moment, SGD with Nesterov momentum and Adam are the most used ones.

In terms of computation speed, one of the most effective ways to speed up training is to increase the batch size for each training step while reducing the number of steps. Using large batches can lead to worse performance of the trained model, so either some tricks, such as in [8], or suitable optimization methods, such as LARS and LAMB, are required. GPUs and TPUs have limited video memory, so the following approach has found widespread use: the data are divided into the required number of parts and training is performed on several devices, while all the calculated gradients are averaged before the gradient descent step, so that all processes have a model with the same weights. The described approach makes it possible to linearly speed up training if the network communication between devices has sufficient bandwidth and suitable methods are used. However, in practice, network bandwidth may turn out to be a bottleneck that does not allow obtaining a linear increase in the training rate [9].

In [10, 11] 1-bit Adam and 1-bit LAMB algorithms were presented, which propose compressing the transmitted data before synchronisation in order to reduce the amount of transmitted information and, accordingly, accelerate the learning process without a loss of quality. One-bit compression is the most compact among compression approaches, which preserves some useful information about each value. However, in practice, during our experiments, it turned out that these algorithms can lead to divergence for certain models.

Our contributions are as follows:

1. We empirically study 1-bit Adam and 1-bit LAMB algorithms convergence on different tasks.
2. We propose to consider two methods that, to our knowledge, have not been discussed during development of new SGD algorithms.
3. Using these techniques we discover weaknesses of 1-bit Adam and 1-bit LAMB algorithms, as well as a proposing a simple empirical solution to one of the critical problems associated with the original implementations.

2. Previous work

In this section, we will provide a short overview of 1-bit optimizers based on [10]. The idea of the one-bit modifications of Adam and LAMB comes from the idea of the one-bit SGD [12]. A single step of the SGD algorithm looks like this:

$$w_{t+1} = w_t - \alpha g_t = w_0 - \alpha \sum_{k=0}^{t} g_k,$$

where w_t is the model weights at step t; x_t is a training sample at step t; α is the parameter of the learning rate, and if $f_w(x)$ is the objective function, then $g_t = \nabla f_{w_t}(x_t)$, these notations are used further.

If we denote the compression operator by $C(x)$, then the SGD step with compression can be done as follows:

$$w_{t+1} = w_t - \alpha C(g_t) = w_t - \alpha(g_t - \delta_t) = w_0 - \alpha \sum_{k=0}^{t} g_k + \alpha \sum_{k=0}^{t} \delta_k$$

δ_k denotes the error introduced by compression after step k. As it is seen, error accumulates and without its compensation, there is no guarantee of convergence. Therefore, the update is slightly modified so that the error does not accumulate:

$$w_{t+1} = w_t - \alpha C(g_t + \delta_{t-1}) = w_t - \alpha(g_t - \delta_t + \delta_{t-1}) =$$

$$= w_0 - \alpha\sum_{k=0}^{t} g_k + \alpha\sum_{k=0}^{t}\left(\delta_t - \delta_{t-1}\right) = w_0 - \alpha\sum_{k=0}^{t} g_k + \alpha\delta_t.$$

If α depends on the step number, the error compensation must look like $C\left(g_t + \frac{\alpha_t}{\alpha_{t-1}}\delta_t\right)$ instead of $C\left(g_t + \delta_{t-1}\right)$.

1-bit Adam. 1-bit Adam is a modification of the widely used Adam algorithm. Adam weights update rules:

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

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

$$w_{t+1} = w_t - \alpha\frac{m_{t+1}}{\sqrt{v_{t+1}} + \varepsilon},$$

where β_1, β_2 are numbers between 0 and 1; ε is a small constant for numerical stability.

Because of the quadratic dependence of the second momentum $\left(v_k\right)$ on gradients, it is impossible to apply error-compensated gradient compression similar to SGD compression.

$$v_{t+1} = \beta_2 v_2 + \left(1-\beta_2\right)\left(C\left(g_t + \delta_{t-1}\right)\right)^2 =$$

$$= \beta_2 v_t + \left(1-\beta_2\right)\left(g_t + \delta_{t-1} - \delta_t\right)^2 =$$

$$= \beta_2 v_2 + \left(1-\beta_2\right)\left(\left(g_t\right)^2 + \left(\delta_{t-1} - \delta_t\right)^2 + 2g_t\left(\delta_{t-1} - \delta_t\right)\right)$$

$\left(\delta_{t-1} - \delta_t\right)^2$ from the equality above is not cancelled if the sum is written out, and as a result, the error accumulates.

If we consider $\frac{\alpha}{\sqrt{v_t} + \varepsilon}$ as a changing learning rate (individual for each parameter), then we are forced to apply compression to the first momentum $\left(m_t\right)$ in the following manner:

$$m_{t+1} = C\left(\beta_1 m_1 + \left(1-\beta_1\right) g_1 + \frac{\sqrt{v_{t-1}} + \varepsilon}{\sqrt{v_t} + \varepsilon}\delta_{t-1}\right) =$$

$$= \beta_2 v_t + \left(1-\beta_2\right)\left(\left(g_t\right)^2 + \left(\delta_{t-1} - \delta_t\right)^2 + 2g_t\left(\delta_{t-1} - \delta_t\right)\right).$$

But after the compression, we cannot express v_t and, accordingly, get the coefficient to compensate for the error.

In [10], the authors argued that after some training time, the second momentum stops changing much, and used this observation to build a one-bit version of Adam, see Algorithm 1. First, after several steps (the original work recommends 15–25 % of steps, if the amount for which the model should be trained is known) updates are made in accordance with the original Adam algorithm. After that, the steps begin at which one-bit compression of the first moment occurs, and the second moment remains unchanged. In addition to the compressed vector, its stretching coefficient is also transmitted between training nodes in order to preserve the magnitudes of the initial moments. A theoretical analysis of the algorithm and more details are available in [10].

Algorithm 1:

Initialize: $\alpha, \beta_1, \beta_2, \epsilon, w_0, m_0 = 0, v_0 = 0$, compression errors $\delta = 0$, number of training steps T, number of warm up steps $T_w, C_1(x) = \mathbb{1}_{[x\geq 0]}(x) - \mathbb{1}_{[x<0]}(x), n$ training nodes
Perform T_w steps of the original Adam algorithm, keep the last second momentum v_{T_w} and stop updating it
for $t = T_w, ..., T$ **do**
    (**On** i**-th node**)
    Sample a random subset of examples $x_t^{(i)}$
    $g_t^{(i)} = \nabla_w f_{w_t}(x_t^{(i)})$
    $m_t^{(i)} = \beta_1 m_{t-1} + (1-\beta_1) g_t^{(i)} + \delta_{t-1}^{(i)}$
    $\hat{m}_t^{(i)} = C_1(m_t^{(i)})$
    $\delta_t^{(i)} = m_t^{(i)} - \hat{m}_t^{(i)}$
    (**On server**)
    $\overline{m}_t' = \frac{1}{n}\sum_{i=0}^{n} \frac{\|m_t^{(i)}\|}{\|\hat{m}_t'^{(i)}\|} \hat{m}_t^{(i)} + \overline{\delta}_{t-1}$
    $\overline{m}_t = C_1(\overline{m}_t')$
    $\overline{\delta}_t = \overline{m}_t' - \overline{m}_t$
    (**On** i**-th node**)
    $m_t = \frac{\|\overline{m}_t'\|}{\|\overline{m}_t\|} \overline{m}_t$
    $w_t = w_{t-1} - \alpha \frac{m_t}{\sqrt{v_{T_w}} + \epsilon}$
end for

1-bit LAMB. Adam, as the research showed, gives poor results with large batch sizes. To solve the problem, the LAMB algorithm was proposed, which takes Adam as a basis and slightly modifies it with the following update rules:

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

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

$$u_{t+1}^{(l)} = \frac{m_{t+1}^{(l)}}{\sqrt{v_{t+1}^{(l)}} + \varepsilon},$$

$$c_{t+1}^{(l)} = clip\left(\frac{\left\|w_t^{(l)}\right\|}{\left\|u_{t+1}^{(l)}\right\|},\ c_{\min},\ c_{\max}\right),$$

$$w_{t+1}^{(l)} = w_t^{(l)} - \alpha c_{t+1}^{(l)} u_{t+1}^{(l)}$$

$c_{\min}, c_{\max}$ are new algorithm hyperparameters responsible for clipping, index (l) denotes that vector belongs to the l^{th} layer of the network, so coefficient $c_t^{(l)}$ is different for each layer.

In [11], the authors tried to apply the idea from 1-bit Adam without changes, i.e. also stop updating the second moment and use the one calculated at the last step before compression stages. However, this approach led to suboptimal solutions, so the algorithm was complicated to obtain quality results comparable to the original LAMB. We will not describe this algorithm in detail and provide only its pseudocode, see Algorithm 2.

Algorithm 2:

Initialize: $\alpha, \beta_1, \beta_2, \beta_3, \epsilon, r_{min}, r_{max}, r_{th}$, $w_0^{(l)}, m_0^{(l)} = 0, v_0^{(l)} = 0, c_{avg}^{(l)} = 0, r^{(l)} = 1$, compression errors $\delta = 0$, number of training steps T, number of warm up steps T_w, $C_1(x) = \mathbb{1}_{[x \geq 0]}(x) - \mathbb{1}_{[x<0]}(x)$, n training nodes

Perform T_w steps of the original LAMB algorithm, at each step update $c_{avg}^{(l)} = \beta_3 c_{avg}^{(l)} + (1 - \beta_3) c_t^{(l)}$. Keep the last second momentum $v_{T_w}^{(l)}$, stop updating $c_{avg}^{(l)}$.

for $t = T_w, ..., T$ **do**

  (**On** i**-th node**)

  Sample a random subset of examples $x_t^{(i)}$

  $g_t^{(i)} = \nabla_w f_{w_t}(x_t^{(i)})$

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

  $\hat{m}_t^{(i)} = C_1(m_t^{(i)})$

  $\delta_t^{(i)} = m_t^{(i)} - \hat{m}_t^{(i)}$

  (**On server**)

  $\overline{m}_t' = \frac{1}{n} \sum_{i=0}^{n} \frac{\|m_t^{(i)}\|}{\|\hat{m}_t'^{(i)}\|} \hat{m}_t^{(i)} + \overline{\delta}_{t-1}$

  $\overline{m}_t = C_1(\overline{m}_t')$

  $\overline{\delta}_t = \overline{m}_t' - \overline{m}_t$

  (**On** i**-th node**)

  $m_t = \frac{\|\overline{m}_t'\|}{\|\overline{m}_t\|} \overline{m}_t$

  for layer l **do**

    $g_t^{(l)} = \frac{m_t^{(l)} - \beta_1 m_{t-1}^{(l)}}{1 - \beta_1}$

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

    $r_t^{(l)} = \left\| \frac{v_{T_w}^{(l)}}{v_t^{(l)}} \right\|_\infty$

    $r_t^{(l)} = \text{clip}(r_t^{(l)}, (1 - r_{th}) r_{t-1}^{(l)}, (1 + r_{th}) r_{t-1}^{(l)})$

    $r_t^{(l)} = \text{clip}(r_t^{(l)}, r_{min}, r_{max})$

    $c_t^{(l)} = r_t^{(l)} c_{avg}^{(l)}$

    $w_t = w_{t-1} - \alpha c_t^{(l)} \frac{m_t^{(l)}}{\sqrt{v_{T_w}^{(l)}} + \epsilon}$

  end for

end for

3. Discovered effects

During our experiments with the considered algorithms, we discovered effects that are not mentioned in the corresponding works. For research, we used the original implementations provided in the DeepSpeed[1] library. Because 1-bit LAMB in the experiments has the same behaviour and shares the same problem as 1-bit Adam, we show only graphs with 1-bit Adam, in order to reduce the volume.

Divergence. The original work of 1-bit Adam provides the results of training ResNet [13] on ImageNet and CIFAR10 datasets, training and fine-tuning BERT [14], and training DCGAN [15] on CelebFaces dataset. 1-bit LAMB was considered only for the task of training and fine-tuning BERT.

We confirm that training ResNet[2] using one-bit algorithms on ImageNet and CIFAR10 goes without problems and even find the effect that sometimes the compression stage can be started earlier than recommended by the authors, without loss of quality, which is described below.

We attempted to train networks of the VGG[3] [16] family using one-bit algorithms for the classification problem on the CIFAR-10 and ImageNet datasets, Transformer[4] [17] for the task of translating from English into German on the WMT-14 dataset and Jasper[5] [18] for speech recognition task on LibriSpeach dataset. From the model listed above, only Jasper did not begin diverging after the onset of the compression stage. VGG divergence with 1-bit Adam is also observed in [19].

Optimization of simple functions. While trying to identify the divergence problem cause, we looked at the behaviour of algorithms when minimising simple functions, such as the sum of squares, and compared it with the behaviour of algorithms without compression.

It can be observed that compression can lead to both slower convergence when approaching the optimum of the function, but at the same time it can have better convergence up to a certain moment, see Fig. 1. This observation shows that one-bit versions of the algorithms require more optimization steps to get as close to the function optimum as the original algorithms.

Early compression without quality loss. For the considered algorithms, the authors recommend starting the compression stage after 15–25 % of the training steps. However, our experiments showed that when training ResNet on the CIFAR10 dataset, compression can be started even after only 3 optimization steps and the quality of the model does not suffer very much, and after 100 the quality is the same as that of the algorithm without compression, see. Fig. 2. In other settings (for example, when training ResNet on the ImageNet dataset), early compression was affecting the quality of the trained model, but this effect may be the subject of further research.

4. Divergence problem

The divergence problem described in Section 3 is critical to the use of the algorithms. Our goal was to identify the cause of this behaviour and find a solution.

Zero gradients are not the only problem. The DeepSpeed documentation, where the algorithm implementations were taken from, recommends that parameter groups known to contain zero gradients should be updated without compression , since one-bit compression encodes zero into a non-zero value, because of which there may be an update of parameters whose gradients are zero. However, we may not know in advance which parameters have a zero gradient. In addition, as our experiments have shown, such parameters are not the only problem. To test the hypothesis that the problem is in zero gradients, we replaced 1-bit compression, with 2-bit compression – positive values of gradients encoded as 1s, negative values encoded as –1s and zeros encoded as 0s. However, even with the 2-bit compression models diverged, which indicates that zero gradients are not the only problem with the considered algorithms.

[1] https://github.com/microsoft/DeepSpeed
[2] Implementations were taken from https://github.com/pytorch/vision and https://github.com/kuangliu/pytorch-cifar
[3] Implementations were taken from https://github.com/pytorch/vision and https://github.com/chengyangfu/pytorch-vgg-cifar10
[4] Implementation was taken from https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Machine-Translation
[5] Implementation was taken from https://github.com/NVIDIA/DeepLearningExamples

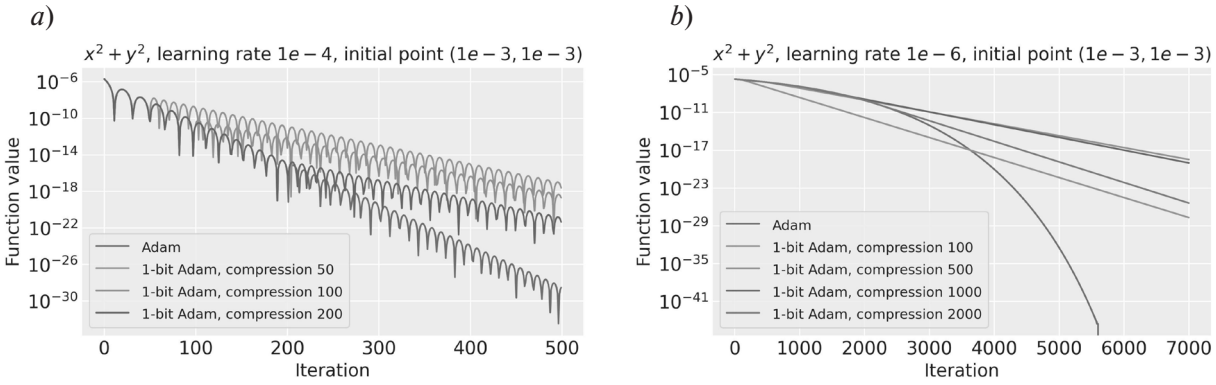

Fig. 1. Minimising $x^2 + y^2$ function with the original Adam and 1-bit Adam algorithms: *a* – optimization with learning rate 10^{-4} for 500 iterations; *b* – learning rate is 10^{-6} and the number of iterations is set to 7000; "compression *n*" denotes that compression stage started after *n* optimization steps

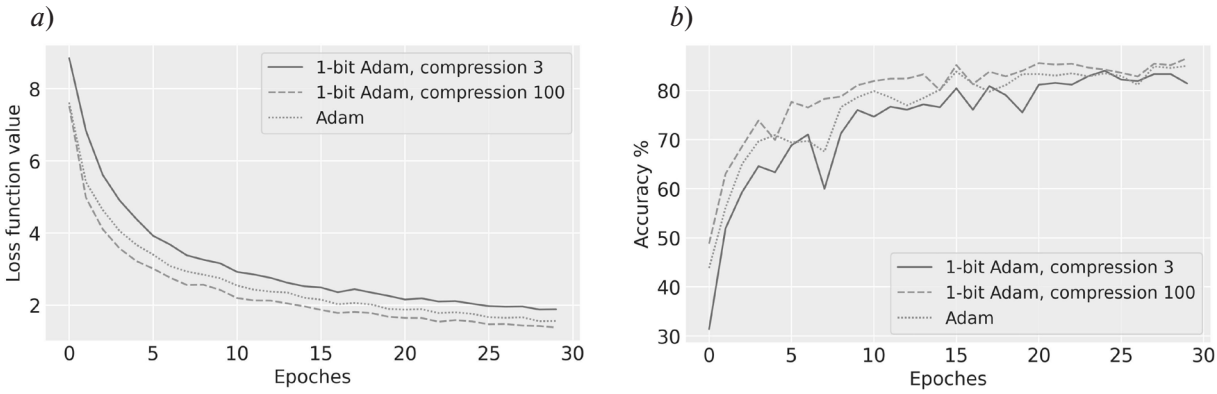

Fig. 2. Training ResNet on CIFAR10 dataset with usage of Adam and 1-bit Adam algorithms. Each epoch contains 250 optimization steps; "compression *n*" denotes that compression stage started after *n* optimization steps; *a* – loss function values, *b* – accuracy of prediction on the test set

Different magnitudes of gradients. To understand the possible reason, we decided to analyse the values of the gradients in different models. As it turned out, the ResNet gradients have values of close orders for different layers, the maximum difference in mean and median values is about 100 times, see Fig. 3. At the same time, for example, VGG16 gradients can be very different: the differences of mean and median values are of the 10^8 order, see Fig. 4. For the Transformer model, mean gradient values are not that representative but median values show that gradient values differ from each other a lot in this model too, see Fig. 5. From these observations, we concluded that this might be the problem.

If we apply the compression procedure used in Algorithm 1 and Algorithm 2 to the following vector of numbers

$$m := \left(10^{-4},\ 10^{-4},\ -10^{-3},\ -10^{-2},\ -10^{-6}\right),$$

as a result of compression, there will be a vector

$$\hat{m} := \left(1,\ 1,\ -1,\ -1,\ 1\right).$$

After multiplying by scaling coefficient, we end up with vector

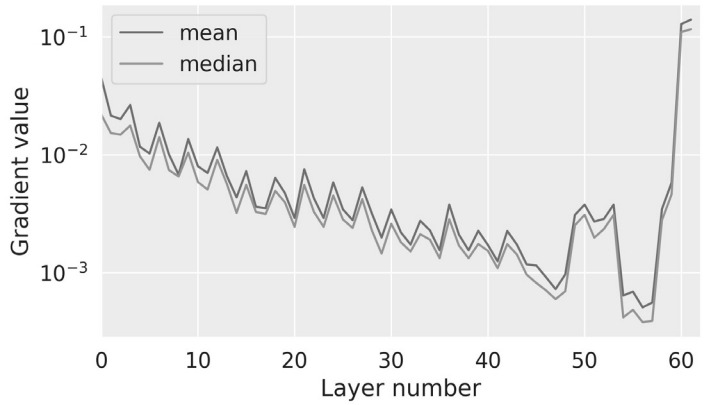

Fig. 3. Mean and median values of gradients from different layers of ResNet18 at 1000th training step with Adam optimizer on CIFAR10 dataset

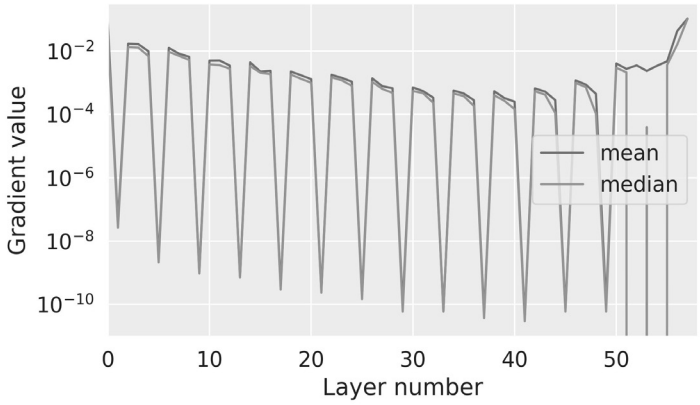

Fig. 4. Mean and median values of gradients from different layers of VGG16 at 1000th training step with Adam optimizer on CIFAR10 dataset

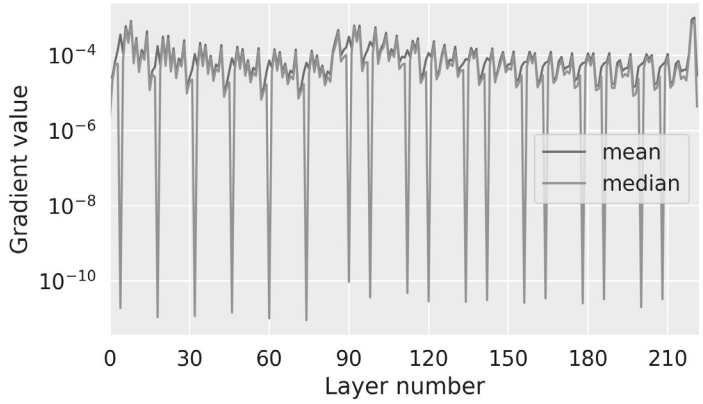

Fig. 5. Mean and median values of gradients from different layers of Transformer at 1000th training step with Adam optimizer on WMT-14 dataset

$$\frac{\|m\|}{\|\hat{m}\|}\hat{m} = 0.0045 \cdot (1,\ 1,\ -1,\ -1,\ 1).$$

Thus, it turns out that the last element of the restored vector with value of 0.0045 differs from the last element with value of 10^{-6} in the original vector by 4500 times. As it turns out in practice, such changes for the gradient may be enough for the model to start diverging even after one optimization step with compression. If in a vector with zero elements there is at least one nonzero, then after compression and restoration of the norm in the new vector there will be no zeroes. Considering the example with the sum of squares, we can also observe this effect: one-bit optimizers may start diverging if one of the arguments is an order of magnitude larger than the second, see Fig. 6.

Possible solution. As shown above, after the procedure for compressing the vector of the first momenta, some of the first momenta can become orders of magnitude larger than the true ones. Then in the formula of weights update

$$w_t = w_{t-1} - \alpha \frac{m_t}{\sqrt{v_{T_w}} + \varepsilon} \tag{1}$$

too large update step will be made for the corresponding parameters (the denominator is a constant), which may lead to a model divergence.

To solve the problem, we tried several approaches, for example, clip the first momentum at each training node after synchronisation so that the values do not exceed those obtained in the previous step, or restore the second momentum, as is done in line 18 of Algorithm 2, to use it for weights update. Nevertheless, none of this worked and led to a divergence, most likely due to the fact that the changes were not taken into account in the error compensation rules, or did not give a significant effect immediately – one step with problematic compression may be enough for the models to diverge, i.e. the problem must be solved immediately.

There is an option to change the value of the hyperparameter α to a smaller value or increase e after the start of compression. This method works, but it requires an additional search for the values, because we want the model trained in this way to show comparable results. In this regard, after starting training with compression, we propose to put the e under the square root in (1) so the update rule has the following form:

$$w_t = w_{t-1} - \alpha \frac{m_t}{\sqrt{v_{T_w}} + \varepsilon}. \tag{2}$$

This modification does not complicate the calculations in any way, and, as our experiments showed, the convergence of unproblematic models does not suffer in any way, and for divergent models, it helps to prevent divergence while saving performance. Fig. 7 shows how this change can help to train VGG16 which diverged with original implementations on the CIFAR10 dataset.

5. Conclusion and future work

In this work, we conducted an empirical study of 1-bit Adam and 1-bit LAMB algorithms and found a problem associated with the original implementation. We proposed a simple solution to the problem: it is computationally simple and quite effective in practice, while not requiring any additional tuning of the parameters. Further studies of the considered algorithms can be focused on their application to other neural network architectures, search for a non-empirical solution to a discovered problem or the effect that in some situations compression can be started almost from the very beginning of model training.

In addition, we hope that the techniques that we used in this work to find problems will be taken into account when creating new modifications of gradient descent where information about the gradients of

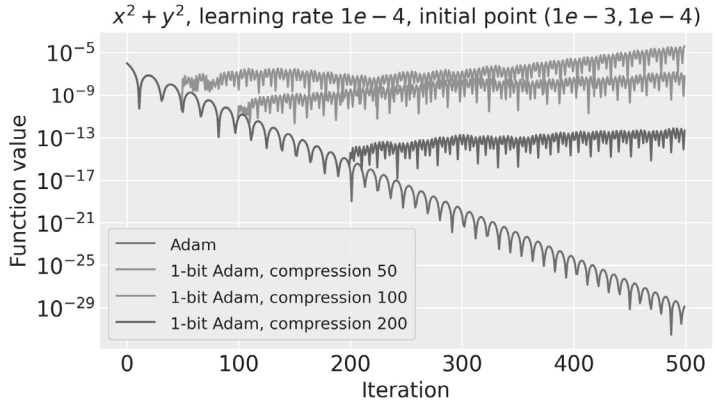

Fig. 6. Divergence of 1-bit Adam while minimising $x^2 + y^2$ function because of different magnitude orders of parameters

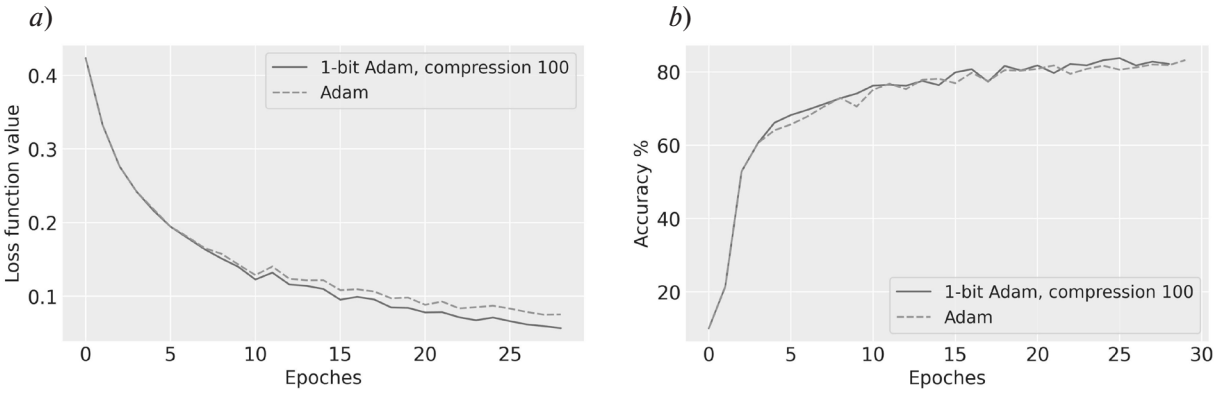

Fig. 7. VGG16 training on CIFAR10 dataset using Adam and 1-bit Adam modified according to (2). Each epoch contains 24 update steps; "compression n" means that compression started after n update steps; a – loss function value, b – prediction accuracy on the test set

one parameter used when updating other parameters. Using these techniques can help detect problems similar to those that we found in 1-bit Adam and 1-bit LAMB.

REFERENCES

1. **Hinton G., Srivastava N., Swersky K.** RMSProp: Divide the gradient by a running average of its recent magnitude. *Neural networks for machine learning*, Coursera lecture 6e, 2012, P. 13.

2. **Duchi J., Hazan E., Singer Y.** Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011, vol. 12 (7).

3. **Zeiler M.D.** Adadelta: An adaptive learning rate method. *arXiv preprint arXiv: 1212.5701, 2012.*

4. **Kingma D., Ba J.** Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014, 12.

5. **You Y., Gitman I., Ginsburg B.** Large batch training of convolutional networks. *arXiv preprint arXiv: 1708.03888, 2017.*

6. **Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, Cho-Jui Hsieh.** Large batch optimization for deep learning: Training Bert in 76 minutes. *arXiv preprint arXiv: 1904.00962, 2019.*

7. **Ginsburg B., Castonguay P., Hrinchuk O., Kuchaiev O., Lavrukhin V., Leary R., Li J., Huyen Nguyen, Yang Zhang, Cohen J.M.** Stochastic gradient methods with layer-wise adaptive moments for training of deep networks. *arXiv preprint arXiv: 1905.11286*, *2019*.

8. **Goyal P., Dollár P., Girshick R., Noordhuis P., Wesolowski L., Kyrola A., Tulloch A., Yangqing Jia, Kaiming He.** Accurate, large minibatch SGD: Training imagenet in 1 hour. *arXiv preprint arXiv: 1706.02677*, *2017*.

9. **Zhen Zhang, Chaokun Chang, Haibin Lin, Yida Wang, Raman Arora, Xin Jin.** Is network the bottleneck of distributed training? *Proceedings of the Workshop on Network Meets AI & ML*, 2020, Pp. 8–13.

10. **Hanlin Tang, Shaoduo Gan, Ammar Ahmad Awan, Samyam Rajbhandari, Conglong Li, Xiangru Lian, Ji Liu, Ce Zhang, Yuxiong He.** 1-bit Adam: Communication efficient large-scale training with Adam's convergence speed. *ICML*, 2021.

11. **Conglong Li, Ammar Ahmad Awan, Hanlin Tang, Samyam Rajbhandari, Yuxiong He.** 1-bit LAMB: Communication efficient large-scale large-batch training with LAMB's convergence speed. *ArXiv, abs/2104.06069*, *2021*.

12. **Frank Seide, Hao Fu, Jasha Droppo, Gang Li, Dong Yu.** 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. *15th Annual Conference of the International Speech Communication Association*. Citeseer, 2014.

13. **Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.** Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, Pp. 770–778.

14. **Devlin J., Ming-Wei Chang, Lee K., Toutanova K.** Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv: 1810.04805*, *2018*.

15. **Radford A., Metz L., Chintala S.** Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv: 1511.06434*, *2015*.

16. **Simonyan K., Zisserman A.** Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv: 1409.1556*, *2014*.

17. **Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., Polosukhin I.** Attention is all you need. *Advances in Neural Information Processing Systems*, 2017, Pp. 5998–6008.

18. **Li J., Lavrukhin V., Ginsburg B., Leary R., Kuchaiev O., Cohen J.M., Huyen Nguyen, Ravi Teja Gadde.** Jasper: An end-to-end convolutional neural acoustic model. *arXiv preprint arXiv: 1904.03288*, *2019*.

19. **Shaoduo Gan, Xiangru Lian, Rui Wang, Jianbin Chang, Chengjun Liu, Hongmei Shi, Shengzhuo Zhang, Xianghong Li, Tengxu Sun, Jiawei Jiang, Binhang Yuan, Sen Yang, Ji Liu, Ce Zhang.** Bagua: Scaling up distributed learning with system relaxations. *ArXiv, abs/2107.01499*, *2021*.

INFORMATION ABOUT AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Тарасов Денис Алексеевич
Denis A. Tarasov
E-mail: tarasov.denis.al@gmail.com

Ершов Василий Алексеевич
Vasily A. Ershov
E-mail: noxoomo@yandex-team.ru

Поступила: 29.11.2022; Одобрена: 26.12.2022; Принята: 12.01.2023.
Submitted: 29.11.2022; Approved: 26.12.2022; Accepted: 12.01.2023.