Communication-Efficient Adam-Type Algorithms for Distributed Data Mining

Wenhan Xian, Feihu Huang, Heng Huang
Department of Electrical and Computer Engineering, University of Pittsburgh
Pittsburgh, United States
wex37@pitt.edu, huangfeihu2018@gmail.com, heng.huang@pitt.edu

***Abstract*—Distributed data mining is an emerging research topic to effectively and efficiently address hard data mining tasks using big data, which are partitioned and computed on different worker nodes, instead of one centralized server. Nevertheless, distributed learning methods often suffer from the communication bottleneck when the network bandwidth is limited or the size of model is large. To solve this critical issue, many gradient compression methods have been proposed recently to reduce the communication cost for multiple optimization algorithms. However, the current applications of gradient compression to adaptive gradient method, which is widely adopted because of its excellent performance to train DNNs, do not achieve the same ideal compression rate or convergence rate as Sketched-SGD. To address this limitation, in this paper, we propose a class of novel distributed Adam-type algorithms (*i.e.*, SketchedAMSGrad) utilizing sketching, which is a promising compression technique that reduces the communication cost from $O(d)$ to $O(\log(d))$ where d is the parameter dimension. In our theoretical analysis, we prove that our new algorithm achieves a fast convergence rate of $O(\frac{1}{\sqrt{nT}} + \frac{1}{(k/d)^2 T})$ with the communication cost of $O(k \log(d))$ at each iteration. Compared with single-machine AMSGrad, our algorithm can achieve the linear speedup with respect to the number of workers n. The experimental result of training distributed DNN validates the performance of our algorithms.**

***Index Terms*—distributed data mining, adaptive gradient, gradient compression**

I. Introduction

Nowadays, as more and more data mining and machine learning applications take advantage of large-scale data, plenty of learning models are trained in a distributed fashion across many worker nodes [1]. Specifically, the problem of these tasks can be formulated as:

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\xi_i \sim D_i} F_i(x; \xi_i), \tag{1}$$

where $f_i(x) = \mathbb{E}_{\xi_i \sim D_i} F_i(x; \xi_i)$ is the local objective function on the i-th node that is generally smooth and possibly nonconvex, and n is the number of worker nodes. Here D_i denotes the data distribution on the i-th node, and different $\{D_i\}_{i=1}^n$ are probably non-identical.

Although distributed training has shown significant advantage, it still suffers from the communication bottleneck, especially when the network bandwidth is limited or the size of model is large. To address this critical issue, many methods have been presented to reduce the cost of communication. Among these methods, one of the most popular ways is to compress the transmitted message in each communication round, such as gradient quantization [2], [3] and gradient sparsification [4]–[6], on which are the focused by this paper. Other methods such as model compression [7] or decentralization [8] also alleviate the bottleneck issue.

Gradient quantization reduces the communication cost by lowering the float-point precision of gradients so that less amount of bits will be transmitted. 1-bit Stochastic Gradient Descent (1-bit SGD) [2] is a classic and primitive gradient quantization work which uses 1-bit quantization and dramatically enhances the communication efficiency. Quantized Stochastic Gradient Descent (QSGD) adopts stochastic randomized rounding to obtain an unbiased estimator after compression. SignSGD and its variant with momentum named Signum [9] only transmit the 1-bit gradient sign between workers and central node, which is convenient to implement.

Gradient sparsification is another widely-used strategy to decrease the communication cost which sparsifies the gradient instead of quantizing each element. The most popular way is to extract the top-k coordinates of local gradients and send them to the master node to estimate the overall mini-batch gradient. Some of these methods are also combined with other techniques such as momentum correction and error-feedback.

Recently, more variants of gradient compression with theoretical guarantees have been proposed, such as SGD with Error-Feedback (EF-SGD) [10], Distributed SGD with Error-Feedback (dist-EF-SGD) [11] and SGD with Error Reset (CSER) [12]. In some recent works like [11], [12], the aggregated gradient estimator is also compressed before sending back to workers. Some works also apply gradient compression to other optimizer such as Frank-Wolfe algorithm [13].

Besides, to solve problem (1), we also need an efficient optimizer to search for the optimal solution. Among existing popular optimization methods, adaptive gradient algorithms [14], [15] have become ones of the most important optimization algorithms to pursue higher efficiency or accuracy in a wide range of data mining and machine learning problems. In the family of adaptive gradient algorithms, Adam [16] is one of the most popular ones that combines momentum and adaptive learning rate. Though it achieves great success in practice, several technical issues in the analysis were pointed out [17] and in some cases the algorithm could diverge.

In [17], two variants of Adam, named as AMSGrad and Adamnc, were proposed to fix the theoretical issues in the

analysis of Adam. AMSGrad makes quantity $\Gamma_{t+1} = (\frac{\sqrt{V_{t+1}}}{\alpha_{t+1}} - \frac{\sqrt{V_t}}{\alpha_t})$ positive to ensure the convergence, while Adamnc adopts an increasing parameter $\beta_{2,t} = 1 - \frac{1}{t}$.

Despite of the success of gradient compression methods, it is hard to use them in distributed adaptive gradient method. So far the application of gradient compression to adaptive gradient algorithm with theoretical guarantee is still limited. Quantized Adam [18] combines gradient quantization with Adamnc, which keeps track of local momentum and variance terms on each worker node and uses quantization when averaging the parameter. Efficient-Adam [19] is similar to Quantized Adam where the gradient message sent back is also compressed. However, both Quantized Adam and Efficient-Adam are not proven to achieve linear speedup or convergence on non-iid data. APMSqueeze [20] and 1-bit Adam [21] are Adam-preconditioned momentum SGD algorithms with gradient compression. However, the variance term is fixed during the training process. Even though it is computed by Adam at the end of warmup step, technically APMSqueeze and 1-bit Adam are not a true adaptive gradient method.

Therefore, it is difficult to apply gradient compression to adaptive gradient methods and maintain the excellent performance of distributed Adam-type algorithms. The challenge is that the original adaptive learning rate is adjustable based on global information such as the aggregated gradient. Although the compressed message is a good estimation of local gradient or momentum, the adaptive learning rate calculated by these inexact messages could be far away from the original one.

To address the challenging high communication cost limitation in distributed adaptive gradient methods, we propose a class of novel distributed Adam-type algorithms (called as SketchedAMSGrad), based on the distributed version of AMSGrad [17] algorithm and the gradient sparsification technique named sketching [22], [23].

Our main contributions are summarized as follows:

(1) To efficiently address the communication bottleneck problem in distributed data mining, we propose a class of novel communication-efficient algorithms named SketchedAMSGrad with two averaging strategies: parameter averaging and gradient averaging. Our new methods can reduce the communication cost from $O(d)$ to $O(\log(d))$.
(2) We provide theoretical analysis based on mild assumptions to guarantee the convergence of our algorithms. Specifically, we prove that our SketchedAMSGrad algorithms have a convergence rate of $O(\frac{1}{\sqrt{nT}})$, which shows a linear speedup. Our theoretical analysis also allows the data distribution to be non-identical.
(3) To the best of our knowledge, our method is the first one to utilize the sketching technique to solve the communication bottleneck in distributed adaptive gradient methods. The experimental results on training various DNNs verify the performances of our algorithms, on both identical and non-identical distributed datasets.

II. Related Works

In the section, we review the related adaptive gradient algorithms with their compressed versions and introduce some preliminary background of sketching. The summary of properties of related methods is listed in Table I. Top-k is considered as the compressor in the result of convergence rate.

A. Quantized-Adam and Efficient-Adam

Quantized-Adam [18] is proposed to combine quantization scheme with distributed Adam algorithm to reduce the communication cost. Specifically, on each worker, it owns a local momentum term $m_t^{(i)}$ and a local variance term $v_t^{(i)}$. These two terms are updated by the exponential moving averaging used in Adam-type algorithms. Gradient quantization is used to compress the term $m_t^{(i)}/\sqrt{v_t^{(i)}}$.

Efficient-Adam [19] is a similar work to Quantized Adam. The only difference is that Efficient-Adam compresses the updating term rather than the parameter. Both of these two algorithms are parameter averaging since if there is no compression, they degenerate to an algorithm where each node is updated by Adam and then the model parameter is averaged. It is not mathematically equivalent to the typical distributed Adam algorithm where gradient averaging is used. Though in some cases parameter averaging is convenient to implement, it is likely to cause bad convergence or be detrimental to the model accuracy especially when the optimizer relies on past local gradient. Actually, Quantized-Adam and Efficient-Adam fail to achieve linear speedup on non-iid data.

B. APMSqueeze and 1-bit Adam Algorithms

APMSqueeze [20] and 1-bit Adam [21] are communication-efficient Adam-preconditioned momentum SGD algorithms. Since the definitions of these two algorithms are similar and 1-bit Adam is the later work, in this paper we will only discuss 1-bit Adam. In the warmup stage, it calculates a variance term v_{T_w}. During the training process, v_{T_w} is fixed and serves as the exponential moving averages term v_t in regular Adam-type algorithms. However, since v_{T_w} is a fixed variable, 1-bit Adam is not technically an adaptive gradient method. In our method, the variance term v_t is dynamic and computed by exponential moving averaging. Besides, we do not need the warmup stage with a separate communication-inefficient optimizer.

C. Sketching

Sketching [22] is a novel and promising gradient sparsicifation technique that compresses a gradient vector g into a sketch $S(g)$ of size $O(\log(d)\epsilon^{-1})$ such that $S(g)$ can approximately recover every coordinates by $\hat{g}_i^2 = g_i^2 \pm \epsilon\|g\|_2^2$. It is originated from a data structure used in data streaming named Count Sketch [24] which is designed to find large coordinates in a vector g defined by a sequence of updates $\{(i_j, w_j)\}_{j=1}^n$.

In [22], sketching serves as a compressor that will approximately recover the true top-k coordinates of mini-batch gradient $\frac{1}{n}\sum_{i=1}^n g_t^{(i)}$ where n is the number of workers. In [25], the authors explicitly treat it as a compressor and the sketching and unsketching operators are denoted by $\mathcal{S}$ and $\mathcal{U}$. For convenience, we also use these notations in this

TABLE I: Comparison of Related Algorithms with Compression

Name	Convergence rate	Linear speedup	Non-iid	Adaptive	Reference
Quantized-Adam	$O(\frac{1}{\sqrt{T}})$	×	×	√	[18]
Efficient-Adam	$O(\frac{1}{\sqrt{T}})$	×	×	√	[19]
APMSqueeze	$O(\frac{1}{\sqrt{nT}}+\frac{1}{(k/d)^{2/3}T^{2/3}})$	√	√	×	[20]
1-bit Adam	$O(\frac{1}{\sqrt{nT}}+\frac{1}{(k/d)^{2/3}T^{2/3}})$	√	√	×	[21]
SketchedAMSGrad (GA)	$O(\frac{1}{\sqrt{nT}}+\frac{1}{(k/d)^{2}T})$	√	√	√	this paper

paper. Sketching method reduces the communication cost to $O(\log(d))$ while gradient quantization only achieves a constant level reduction and the communication cost is still $O(d)$. The current best results for quantization method achieve an approximate $32\times$ compression rate [11], [26]. Compared with top-k method, one advantage of sketching is to recover the true top-k coordinates, where the gradient estimator is $v_1 \approx Top_k(\frac{1}{n}\sum_{i=1}^n g_t^{(i)})$. Although applying the method in [11] can avoid the $O(n)$ return communication cost mentioned in [22], the gradient estimator $v_2 = Top_k(\frac{1}{n}\sum_{i=1}^n Top_k(g_t^{(i)}))$ is still probably far away from the true top-k coordinates. This issue can be reflected by the second dominating term in the convergence rate. In [11], the second dominating term is $O(\frac{1}{(k/d)^{4/3}T^{2/3}})$ which is claimed to be the price to pay for two-way compression and linear speedup. In 1-bit Adam the step size is dependent on the compression ratio and this term becomes $O(\frac{1}{(k/d)^{2/3}T^{2/3}})$ as we have mentioned. However, in Sketched-SGD and our algorithms, the corresponding term is $O(\frac{1}{(k/d)^2 T})$, which is smaller when T is large.

III. Sketched Adam-type Algorithms

A. SketchedAMSGrad (Parameter Averaging)

In this subsection, we will propose the SketchedAMSGrad (PA) algorithm using parameter averaging, the description of which is shown in Algorithm 1.

In Algorithm 1, we use AMSGrad algorithm to update each worker node, based on local momentum term $m_t^{(i)}$ and exponential moving averages of squared past gradients $v_t^{(i)}$. α_t is the stepsize and $\beta_1, \beta_2 \in (0,1)$ are exponential moving average hyperparameters in Adam-type algorithm. $\epsilon > 0$ is the initial value of v_0 to avoid zero denominators. The multiplication, division and square operation between vectors are component-wise. We use sketching to improve communication efficiency and average the parameters. We also use error-feedback to further accelerate the convergence.

For convenience, we also use the notations $\mathcal{S}$ and $\mathcal{U}$ defined in [25] to represent the sketching operator and unsketching operator. They can be treated as a compressor that will approximately recover the true top-k coordinates. In practice, we use a second round communication which is also required in SketchedSGD [22]. After unsketching, we get an estimation of the aggregated mini-batch gradient which is denoted by $\mathcal{U}(S_t)$. Then we select the largest Pk coordinates to extract their exact values before sketching from each worker during the second round communication. Finally, we select the top-k coordinates among these Pk coordinates as Δ_t and send it back to each worker. $\Delta_t^{(i)}$ contains the corresponding k coordinates in $m_t^{(i)}/\sqrt{\hat{v}_t^{(i)}} + \frac{\alpha_{t-1}}{\alpha_t}e_{t-1}^{(i)}$ and automatically it satisfies $\Delta_t = \frac{1}{n}\sum_{i=1}^n \Delta_t^{(i)}$. Therefore, at each iteration, the total communication cost is $|S|+Pk+k$ and the compression rate is $2d/(|S|+Pk+k)$ where $|S|$ is the size of sketch.

Algorithm 1 SketchedAMSGrad (parameter averaging)

Input: initial value x_1, sketching operator $\mathcal{S}$ and unsketching operator $\mathcal{U}$
Set: $m_0^{(i)}=\mathbf{0}$, $v_0^{(i)}=\hat{v}_0^{(i)}=\epsilon$, $e_0^{(i)}=\mathbf{0}$ on i-th worker node
for $t=1$ **to** T **do**
  On i-th worker node:
    Estimate a stochastic gradient $g_t^{(i)}$;
    Compute $m_t^{(i)} = \beta_1 m_{t-1}^{(i)} + (1-\beta_1)g_t^{(i)}$;
    $v_t^{(i)} = \beta_2 v_{t-1}^{(i)} + (1-\beta_2)[g_t^{(i)}]^2$;
    $\hat{v}_t^{(i)} = \max\{\hat{v}_{t-1}^{(i)}, v_t^{(i)}\}$;
    Sketch $S_t^{(i)} = \mathcal{S}(m_t^{(i)}/\sqrt{\hat{v}_t^{(i)}} + \frac{\alpha_{t-1}}{\alpha_t}e_{t-1}^{(i)})$;
    Send $S_t^{(i)}$ to the master node;
    Send $\Delta_t^{(i)}$ to the master node after unsketching;
    Compute $e_t^{(i)} = m_t^{(i)}/\sqrt{\hat{v}_t^{(i)}} + \frac{\alpha_{t-1}}{\alpha_t}e_{t-1}^{(i)} - \Delta_t^{(i)}$;
    Receive Δ_t from the master node;
    Update $x_{t+1} = x_t - \alpha_t\Delta_t$.
  On the master node:
    Aggregate $S_t = \frac{1}{n}\sum_{i=1}^n S_t^{(i)}$;
    Unsketch $\Delta_t = \frac{1}{n}\sum_{i=1}^n \Delta_t^{(i)} = \text{Top-}k(\mathcal{U}(S_t))$;
    Send Δ_t back to each worker node;
    Update $x_{t+1} = x_t - \alpha_t\Delta_t$.
end for

B. SketchedAMSGrad (Gradient Averaging)

In the subsection, we propose the SketchedAMSGrad (GA) algorithm using gradient averaging, which is demonstrated in Algorithm 2.

In Algorithm 2, the meanings of hyperparameters α_t, β_1 and β_2 are the same as those in Algorithm 1. We also keep track of local momentum term $m_t^{(i)}$ on each node but the exponential moving averaging squared gradient v_t is defined on the master node. The index set $\mathcal{I}_t$ represents the coordinates updated at iteration t, which is obtained by the unsketching operator. Notation $h_t^{(i)} = (g_t^{(i)})_{\mathcal{I}_{t-1}}$ means for $\forall j \in \mathcal{I}_{t-1}$, $h_t^{(i)}$ maintains the j-th coordinate of $g_t^{(i)}$. Otherwise, if $j \notin \mathcal{I}_{t-1}$, the j-th coordinate of $h_t^{(i)}$ is 0. We define $\mathcal{I}_t$ in this way because we want to accumulate the coordinates of squared gradient which are just updated and we want to define an

Algorithm 2 SketchedAMSGrad (gradient averaging)

Input: initial value x_1, sketching operator $\mathcal{S}$ and unsketching operator $\mathcal{U}$
Set: $m_0^{(i)} = \mathbf{0}$, $e_0^{(i)} = \mathbf{0}$ on i-th worker node; $v_0 = \hat{v}_0$ on the master node; index set $\mathcal{I}_0 = \varnothing$
for $t = 1$ **to** T **do**
  On i-th worker node:
    Estimate a stochastic gradient $g_t^{(i)}$;
    Compute $m_t^{(i)} = \beta_1 m_{t-1}^{(i)} + (1-\beta_1) g_t^{(i)}$;
    Send $h_t^{(i)} = (g_t^{(i)})_{\mathcal{I}_{t-1}}$ to the master node;
    Sketch $S_t^{(i)} = \mathcal{S}(m_t^{(i)} + \frac{\alpha_{t-1}}{\alpha_t} e_{t-1}^{(i)})$;
    Send $S_t^{(i)}$ to the master node;
    Send $\Delta_t^{(i)}$ to the master node after unsketching;
    Compute $e_t^{(i)} = m_t^{(i)} + \frac{\alpha_{t-1}}{\alpha_t} e_{t-1}^{(i)} - \Delta_t^{(i)}$;
    Receive Δ_t from the master node;
    Update $x_{t+1} = x_t - \alpha_t \Delta_t$.
  On the master node:
    Aggregate $h_t = \frac{1}{n}\sum_{i=1}^n h_t^{(i)}$;
    Compute $v_t = \beta_2 v_{t-1} + (1-\beta_2) h_t^2$;
    $\hat{v}_t = \max\{\hat{v}_{t-1}, v_t\}$;
    Aggregate $S_t = \frac{1}{n}\sum_{i=1}^n S_t^{(i)}$;
    Unsketch $\Delta_t = \frac{1}{n}\sum_{i=1}^n \Delta_t^{(i)} = $Top-$k(\mathcal{U}(S_t, \hat{v}_t))$;
    Send Δ_t back to each worker node;
    Update $x_{t+1} = x_t - \alpha_t \Delta_t$.
end for

Algorithm 3 Unsketching Operator in Algorithm 2

Input: $r \times c$ sketch S, vector v, bucket hashes $\{h_j\}_{j=1}^r$, original unsketching operator $\mathcal{U}_0$
for $i = 1$ **to** d **do**
  for $j = 1$ **to** r **do**
    $S[j, h_j(i)] = S[j, h_j(i)]/\sqrt{v_i}$
  end for
end for
return $\mathcal{U}_0(S)$

auxiliary sequence that makes the convergence analysis more convenient. Algorithm 2 is a gradient averaging algorithm because if there is no compressor applied, this algorithm is degenerated to the common distributed AMSGrad optimizer. In Algorithm 2 the unsketching operator $\mathcal{U}$ requires a vector $\hat{v}_t$ as another input and is used to recover the top-k coordinates of term $\tilde{\Delta}_t$, which is defined as follows.

$$\tilde{\Delta}_t = \frac{1}{n}\sum_{i=1}^n \tilde{\Delta}_t^{(i)},\ \tilde{\Delta}_t^{(i)} = \hat{v}_t^{-1/2}(m_t^{(i)} + \frac{\alpha_{t-1}}{\alpha_t} e_{t-1}^{(i)}) \quad (2)$$

The index set of these k coordinates is denoted as $\mathcal{I}_t$. The implementation of $\mathcal{U}$ is shown in Algorithm 3, which is established on the original sketching and unsketching operator. According to the linear property of sketching $\mathcal{S}$, it is equivalent to compress $\tilde{\Delta}_t$ by $\mathcal{S}$ and then unsketch it by the normal unsketching operator. $\Delta_t^{(i)}$ contains the coordinates of $\tilde{\Delta}_t^{(i)}$ that belongs to index set $\mathcal{I}_t$ and $\Delta_t = \frac{1}{n}\sum_{i=1}^n \Delta_t^{(i)}$.

As the top-k coordinates of $m_t/\sqrt{\hat{v}_t}$ and m_t are likely to change a lot, it is hard to estimate the Adam updating term $m_t/\sqrt{\hat{v}_t}$ by the known vector $m_t^{(i)}$ on each node. However, the sketching technique makes it possible within the communication cost of $O(\log(d))$.

In Algorithm 2, thus, the total communication cost at each iteration is $|S|+Pk+2k$ and the compression rate is $2d/(|S|+Pk+2k)$ where $|S|$ is the size of sketch.

In fact, our SketchedAMSGrad (GA) algorithm is compatible with 1-bit Adam algorithm. We can also regard the v_{T_w} in the 1-bit Adam algorithm as the initial value of v_0 in Algorithm 2. The only difference is that in the theoretical analysis we need to replace the initial value ϵ with the v_{min} defined in the 1-bit Adam. Moreover, if we do not send h_t or update v_t, our algorithm is reduced to the 1-bit Adam with sketching compressor.

IV. Convergence Analysis

In the section, we provide the convergence analysis of our algorithms. Due to the space limit, we will only provide the conclusions of Theorem 1 and Theorem 2. We begin with giving some mild assumptions.

Assumption 1. *(Lipschitz Gradient) There is a constant L such that for $\forall x, y \in R^d$, $\|\nabla f(x) - \nabla f(y)\| \le L\|x - y\|$.*

Assumption 2. *(Lower Bound) Function $f(x)$ has the lower bound, i.e., $\inf_{x\in R^d} f(x) = f^* > -\infty$*

Assumption 3. *(Bounded Gradient) There is a constant G such that for $\forall i \in \{1, \cdots, n\}$, $\forall \xi_i \sim D_i$, we have $\|\nabla F_i(x; \xi_i)\|_\infty \le G$.*

These assumptions are commonly used in related works of Adam-type algorithms in nonconvex optimization [27]–[29].

A. SketchedAMSGrad (PA)

Theorem 1. *Assume that Assumption 1 to Assumption 3 are satisfied and data distribution $\{D_i\}_{i=1}^n$ are identical. In Algorithm 1, let $\beta_1 < 1$, $\beta_2 < 1$, $\epsilon > 0$ and $\alpha_t = \frac{\alpha}{\sqrt{1+T}}$, $\alpha > 0$. Then we have*

$$\frac{1}{T}\sum_{t=1}^T \mathbb{E}\|\nabla f(x_t)\|^2 \le \frac{C_1}{\sqrt{T}} + \frac{C_2}{T},$$

where constants C_1 and C_2 are independent of T.

B. SketchedAMSGrad (GA)

Theorem 2. *Assume that Assumptions 1-3 are satisfied. In Algorithm 2, let $\beta_1 < 1$, $\beta_2 < 1$, $\epsilon > 0$ and $\alpha_t = \frac{\alpha}{\sqrt{1+T/n}}$, $\alpha > 0$. Then we have*

$$\frac{1}{T}\sum_{t=1}^T \mathbb{E}\|\nabla f(x_t)\|^2 \le \frac{C_1}{\sqrt{nT}} + \frac{C_1 + C_2}{T},$$

where constants C_1 and C_2 are independent of T.

Corollary 1. *In Theorem 2, we can see the dominating term is $O(\frac{1}{\sqrt{nT}})$, which achieves a linear speedup compared with AMSGrad in nonconvex optimization [28].*

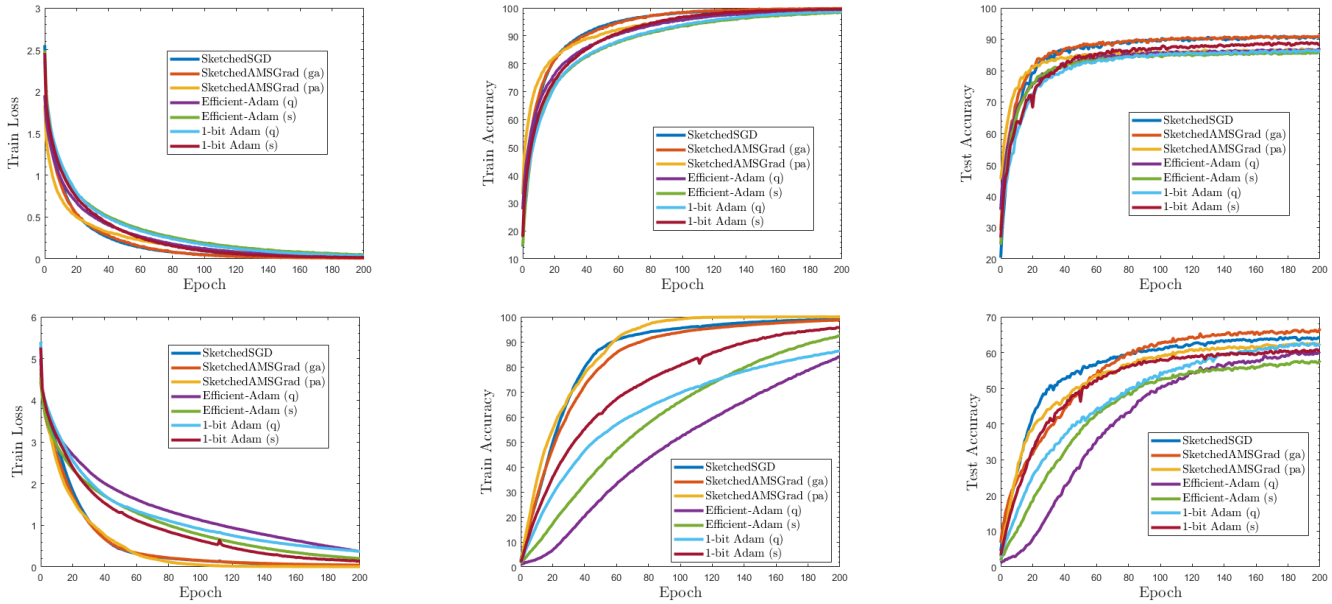

Fig. 1: The experimental results of training ResNet-50 on CIFAR10 and CIFAR100. Figures (a), (b) and (c) show the experimental results on CIFAR10. Figures (d), (e) and (f) show the experimental results on CIFAR100. Figures (a) and (d) show the train loss value. Figures (b) and (e) show the train accuracy. Figures (c) and (f) show the test accuracy.

Remark 1. *In Algorithm 2, the data distribution D_i's are allowed to be non-identical. In both Algorithm 1 and Algorithm 2, β_1 and β_2 are constants in $(0, 1)$, which is applicable to the common default settings that $\beta_1 = 0.9$ and $\beta_2 = 0.999$.*

C. Discussion on the Compression Rate

Now we will discuss the how the compression rate, *i.e.*, the choice of k influences the convergence rate. In both of Theorem 1 and Theorem 2, constant C_1 is independent on k. Hence the dominating term is not affected by the compression rate. This result is the same as many other gradient compression methods. According to the definitions of C_2 in Theorem 1 and Theorem 2, the second dominating term is affected by k with the form $O(\frac{1}{(k/d)^2 T})$ for both parameter averaging and gradient averaging SketchedAMSGrad algorithms.

V. Experiments

In this section we will show the experimental results of the distributed data mining task of image categorization to validate our methods. All experiments are run on a server with 64-core Intel Xeon E5-2683 v4 2.10GHz processor and 4 Nvidia P40 GPUs. We simulate the edge-based training environment on the GPU server where the root process represents the edge server, each process represents an IoT device and the dataset represents the captured data. The code is implemented by PyTorch 1.4.0 and CUDA 10.1.

A. ResNet on CIFAR

Our first task is to train ResNet-50 [30] using CIFAR10 and CIFAR100 datasets [31], which are benchmark datasets for image classification tasks. Both CIFAR10 and CIFAR100 contain 60,000 32×32 pixel images with RGB channels, 50,000 of which is regarded as training set and the other 10,000 of which is used for testing. The images are distributed evenly over 10 and 100 classes for CIFAR10 and CIFAR100 respectively. The ResNet-50 model has about 25M parameters. We use cross-entropy loss to train the neural network.

In our experiment, we compare our SketchedAMSGrad (PA) and SketchedAMSGrad (GA) with Sketched-SGD [22], Efficient-Adam [19] and 1-bit Adam [21]. For Efficient-Adam and 1-bit Adam, we consider both quantization and sparsification as compressor. For gradient quantization, we adopt the following scheme used in [11] which is a variant of SignSGD:

$$C(x) = \frac{\|x\|_1}{d} sign(x) \tag{3}$$

The number of workers in this task is set to be 16. The batch-size on each worker node is 32. Hence the total batch-size at each iteration is 512. We run 200 epochs in total. For each algorithm, we grid search the learning rate from $\{0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$ and ϵ from $\{1e-2, 1e-4, 1e-6\}$ and select the values that get the best training result. For Adam-type algorithms, β_1 and β_2 are set to be the common choices that $\beta_1 = 0.9$ and $\beta_2 = 0.999$. For 1-bit Adam, similar to [21], we run 13 epochs to compute the Adam-preconditioned vector v_{T_w}. For sketching methods, the sketch is set to have 100,000 columns and 10 rows. We set $k = 50,000$ and $P = 8$. For Efficient-Adam and 1-bit Adam with top-k compressor, we choose $k = 750,000$. Therefore, all algorithms implemented in this task are communication-efficient and approximately achieve the same compression rate (about $32\times$ reduction).

Figure 1 shows the experimental results of this image classification task. According to the result of train loss value, we can see the three sketching methods converge faster than other algorithms on both CIFAR10 and CIFAR100 dataset. When comparing the train accuracy, the sketching methods are still advantageous over other methods. Our parameter averaging

and gradient averaging SketchedAMSGrad and SketchedSGD approximately have the same performance. On CIFAR100, our parameter averaging SketchedAMSGrad is slightly better on the train accuracy results. According to the test accuracy results, our gradient averaging SketchedAMSGrad and SketchedSGD also outperform other algorithms on both dataset. On CIFAR100, our gradient averaging SketchedAMSGrad achieves the best performance on test accuracy. From this experiment we can see that although using compression on the returning message avoids the growing $O(n)$ communication cost issue of local top-k (mentioned in [22]), it probably encounters slow convergence since the estimator is too far away from the true top-k coordinates.

Theoretically, when the sketch size is larger, the probability of recovering top-k coordinates is higher. The sketch size used in this experiment is 1,000,000. On CIFAR10, the test accuracy of our SketchedAMSGrad (GA) is 91.04%. When we increase the sketch size to 2,000,000 and 3,000,000, the test accuracy is increased by 0.24% and 0.39% respectively. Thus, we can see the influence of sketch size. If the sketch size is larger, our algorithm will probably show better performance.

VI. Conclusion

In this paper, we propose a class of communication-efficient distributed adaptive gradient algorithm named SketchedAMSGrad based on two averaging strategies to tackle the high communication cost issue for distributed training. Specifically, the communication cost of our algorithm at each iteration is reduced to $O(\log(d))$ from $O(d)$. Moreover, we proved that our algorithm achieves a fast convergence rate of $\tilde{O}(\frac{1}{\sqrt{nT}})$, which achieves the linear speedup compared with single-machine AMSGrad. In particular, our analysis of gradient averaging SketchedAMSGrad works for both identical and non-identical data distribution. To the best of our knowledge, our algorithm is the first one to apply sketching technique to adaptive gradient methods.

Acknowledgment

This work was partially supported by NSF IIS 1838627, 1837956, 1956002, 2211492, CNS 2213701, CCF 2217003, DBI 2225775.

References

[1] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala, "Pytorch distributed: Experiences on accelerating data parallel training," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, 2020.

[2] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns," *International Speech Communication Association*, 2014.

[3] D. Alistarh, D. Grubic, J. Z. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Neural Information Processing Systems*, 2017.

[4] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *Conference on Empirical Methods in Natural Language Processing*, 2017.

[5] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," *Neural Information Processing Systems*, 2018.

[6] S. U. Stich, J. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," *Neural Information Processing Systems*, 2018.

[7] Y. Zhang, S. Gao, and H. Huang, "Exploration and estimation for model compression," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 487–496, October 2021.

[8] W. Xian, F. Huang, Y. Zhang, and H. Huang, "A faster decentralized algorithm for nonconvex minimax problems," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 25865–25877, 2021.

[9] J. Bernstein, Y. Wang, K. Azizzadenesheli, and A. Anandkumar, "Signsgd: Compressed optimization for nonconvex problems," *International Conference on Machine Learning*, 2018.

[10] S. P. Karimireddy, Q. Rebjock, S. U. Stich, and M. Jaggi, "Error feedback fixes signsgd and other gradient compression schemes," *International Conference on Machine Learning*, 2019.

[11] S. Zheng, Z. Huang, and J. T. Kwok, "Communication-efficient distributed blockwise momentum sgd with error-feedback," *Neural Information Processing Systems*, 2019.

[12] C. Xie, S. Zheng, S. Koyejo, I. Gupta, M. Li, and H. Lin, "Cser: Communication-efficient sgd with error reset," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 12593–12603, Curran Associates, Inc., 2020.

[13] W. Xian, F. Huang, and H. Huang, "Communication-efficient frank-wolfe algorithm for nonconvex decentralized distributed learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 10405–10413, May 2021.

[14] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, 2011.

[15] T. Tieleman and G. Hinton, "Rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, 2011.

[16] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2014.

[17] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *International Conference on Learning Representations*, 2018.

[18] C. Chen, L. Shen, H. Huang, Q. Wu, and W. Liu, "Quantized adam with error feedback," *arXiv:2004.14180*, 2020.

[19] C. Chen, L. Shen, H. Huang, W. Liu, and Z. Luo, "Efficient-adam: Communication-efficient distributed adam with complexity analysis," *openreview.net*, 2020.

[20] H. Tang, S. Gan, S. Rajbhandari, X. Lian, J. Liu, Y. He, and C. Zhang, "Apmsqueeze: A communication efficient adam-preconditioned momentum sgd algorithm," *arXiv:2008.11343*, 2020.

[21] H. Tang, S. Gan, A. A. Awan, S. Rajbhandari, C. Li, X. Lian, J. Liu, C. Zhang, and Y. He, "1-bit adam: Communication efficient large-scale training with adam's convergence speed," *arXiv preprint arXiv:2102.02888*, 2021.

[22] N. Ivkin, D. Rothchild, E. Ullah, V. Braverman, I. Stoica, and R. Arora, "Communication-efficient distributed sgd with sketching," *Neural Information Processing Systems*, 2019.

[23] T. Li, Z. Liu, V. Sekar, and V. Smith, "Privacy for free: Communication-efficient learning with differential privacy using sketches," *arXiv:1911.00972*, 2019.

[24] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *International Colloquium on Automata, Languages, and Programming*, 2002.

[25] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora, "Fetchsgd: Communication-efficient federated learning with sketching," *International Conference on Machine Learning*, 2020.

[26] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, "signsgd with majority vote is communication efficient and fault tolerant," *International Conference on Learning Representations*, 2019.

[27] D. Zhou, Y. Tang, Z. Yang, Y. Cao, and Q. Gu, "On the convergence of adaptive gradient methods for nonconvex optimization," *arXiv:1808.05671*, 2018.

[28] X. Chen, S. Liu, R. Sun, and M. Hong, "On the convergence of a class of adam-type algorithms for nonconvex optimization," *International Conference on Learning Representations*, 2019.

[29] A. Alacaoglu, Y. Malitsky, P. Mertikopoulos, and V. Cevher, "A new regret analysis for adam-type algorithms," *International Conference on Machine Learning*, 2020.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[31] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Technical Report TR-2009*, 2009.