# Learned Gradient Compression for Distributed Deep Learning

Lusine Abrahamyan, Yiming Chen, Giannis Bekoulis, and Nikos Deligiannis, *Member, IEEE*

***Abstract*—Training deep neural networks on large datasets containing high-dimensional data requires a large amount of computation. A solution to this problem is data-parallel distributed training, where a model is replicated into several computational nodes that have access to different chunks of the data. This approach, however, entails high communication rates and latency because of the computed gradients that need to be shared among nodes at every iteration. The problem becomes more pronounced in the case that there is wireless communication between the nodes (i.e., due to the limited network bandwidth). To address this problem, various compression methods have been proposed including sparsification, quantization, and entropy encoding of the gradients. Existing methods leverage the *intra-node* information redundancy, that is, they compress gradients at each node *independently*. In contrast, we advocate that the gradients across the nodes are correlated and propose methods to leverage this *inter-node* redundancy to improve compression efficiency. Depending on the node communication protocol (*parameter server* or *ring-allreduce*), we propose two instances of the LGC approach that we coin *Learned Gradient Compression* (LGC). Our methods exploit an autoencoder (i.e., trained during the first stages of the distributed training) to capture the common information that exists in the gradients of the distributed nodes. To constrain the nodes' computational complexity, the autoencoder is realized with a lightweight neural network. We have tested our LGC methods on the image classification and semantic segmentation tasks using different convolutional neural networks (ResNet50, ResNet101, PSPNet) and multiple datasets (ImageNet, Cifar10, CamVid). The ResNet101 model trained for image classification on Cifar10 achieved significant compression rate reductions with the accuracy of $93.57\%$, which is lower than the baseline distributed training with uncompressed gradients only by $0.18\%$. The rate of the model is reduced by 8095$\times$ and 8$\times$ compared to the baseline and the state-of-the-art deep gradient compression (DGC) method, respectively.**

***Index Terms*—Deep learning, data-parallel distributed training, gradient compression, autoencoders.**

## I. Introduction

Recent successful results in the field of artificial intelligence (AI) are achieved with deep learning models that contain a large number of parameters and are trained using a massive amount of data. For example, FixResNeXt-101 32x48d [1], a state-of-the-art model for image classification, contains approximately 800 million parameters, and BERT (Bidirectional Encoder Representations from Transformers) [2], a recent model for natural language processing, contains 110 million parameters. Training such deep networks in a single machine (given a fixed set of hyperparameters) can take weeks.

L. Abrahamyan, Y. Chen, G. Bekoulis, and N. Deligiannis are with Vrije Universiteit Brussel, Pleinlaan 2, B-1050 Brussels, Belgium and also with imec, Kapeldreef 75, B-3001 Leuven, Belgium. (e-mail: {alusine, ychen, gbekouli,ndeligia}@etrovub.be).

An answer to this problem is to perform the training of such networks in a number of computing nodes in parallel. Two parallelization approaches have been exploited in the literature: (1) *model-parallel* distributed training, where the different nodes train different parts of the model; and (2) *data-parallel* distributed training, where each node has a replica (i.e., a copy) of the model and access to a chunk of the data. In both distributed training approaches, there is a communication and latency overhead due to the transmission of information from the one node to the other. In the model-parallel distributed training approach, the data that need to be transferred consist of the activation values of a certain layer in the model. The transmission overhead in this case is typically small. In the data-parallel training approach, however, the calculated gradients of a model that need to be transferred can reach hundreds of megabytes (MBs) per iteration. In this paper, we focus on data-parallel distributed training and propose a new framework to reduce the overhead related to the gradient data transfer.

Most of the methods that aim to reduce the gradient communication bandwidth consider that the nodes exchange gradients in a synchronous manner. Nevertheless, the study in [3] proposed Downpour Stochastic Gradient Descent, where training is asynchronous in two ways: (*i)* model replicas (i.e., copies of the same model) update their gradients at different time instances (asynchronously), and (*ii)* the master-node gradient information is divided into shards, where each of the shards runs independently. Performing data-parallel distributed training asynchronously eliminates the need to synchronize the weight updates between the nodes; however, asynchronous training typically leads to a higher loss in the accuracy of the trained model. The authors of [4] proposed Qsparse-local-SGD, which can be applied to synchronous and hybrid (combining both synchronous and asynchronous training) scenarios; in the latter case, nodes are divided into groups. The gradient transfer within a specific group is synchronous and asynchronous within the different groups. The Qsparse-local-SGD method achieved over 20$\times$ reduction in terms of the total number of bits transmitted in the training of Resnet50 [5] on the ImageNet [6] dataset, with an approximate $1\%$ loss in the final accuracy.

Furthermore, a reduction in the total number of training iterations can also reduce the number of gradient transfers performed within the training and as a result, cause the decrease in the total amount of transferred data. Reduction in the number of the iterations can be reached by an increase in the batch size. Given a fixed amount of memory in the graphical processing unit (GPU), an increase of the batch is possible if the space allocated for the model is decreased. Such a reduction is possible by means of implying model compression methods.

A model can be compressed, employing quantization [7]–[12], pruning [13]–[16], and knowledge distillation [17], [18]. Jacob *et al.* [8] quantized the pre-trained FP32 network to a lower bit-depth precision using 8-bit integers for both weights and activations. Han *et al.* [13] proposed to prune redundant connections using a three-step method. First, they learn important connections within a network, prune unimportant connections, and finally retrain the network. The authors of [17] introduced the knowledge distillation method, where knowledge from a larger model is transferred to a smaller model. Typically, the downside of these methods is an inevitable degradation in performance. Moreover, model compression methods are usually applied on the already pretrained networks; hence, most of them cannot solve the problem of huge communication bandwidth needed for the distributed training.

Several approaches have been proposed to address the gradient communication problem, including gradient sparsification [4], [19], [20], quantization [4], [21], [22] and entropy coding [23], [24] of the gradient tensors; these approaches are proposed within the context of synchronous data-parallel distributed training. The core idea of gradient sparsification is to transfer a fraction of the gradient, depending on some importance metric. Deep Gradient Compression [20], for example, follows the gradient sparsification approaches, achieving up to $99.9\%$ gradient sparsification without loss in accuracy. In the gradient quantization approaches, the gradients are being quantized before transferring. The authors of [22], for example, trained the ResNet-152 [5] network to full accuracy on ImageNet [6] $1.8\times$ faster than the variant with full-presicion gradients. Another approach is to combine the aforementioned techniques, as it has been done in the method presented in [4].

Despite their efficiency, these approaches only explore the intra-node gradient redundancy by means of sparsification, quantization and entropy coding. In this work, we propose to exploit the correlation between the gradients of the distributed nodes in order to achieve further compression gains. It is worth mentioning that our approach can be combined with gradient sparsification and quantization. An attempt to explore the redundancies of gradients by different nodes was also made in [24], [25]. The method described in [24] is based on distributed source coding, realised by low-density parity check (LDPC) codes, which leads to an impractically high decoding latency and complexity. In contrast, our approach employs lightweight autoencoders for the compression of gradients, which substantially reduce the communication rate without compromising the encoding and decoding speed. In [25], the authors also utilized the correlation between the gradient tensors. They introduced a new Cyclic Local Top-k selection mechanism for the ring-allreduce communication pattern, where a set of indices for the gradient selection is the same for all the nodes. Compared with this approach, our LGC framework is able to provide higher compression ratios because of the further utilization of the correlation and introduction of autoencoder-based distributed compressors. In summary, the contributions of this work are:

- we study experimentally the statistical dependency among gradients of distributed nodes using information-theoretic metrics and show that there is a considerable rate reduction that can be achieved if these correlations are exploited.
- we propose a novel framework for performing distributed compression of the gradients, coined Learned Gradient Compression (LGC). Our framework uses lightweight autoencoder models to compress gradients by leveraging the correlation between them. We propose different instances of our framework for the *parameter server* and *ring-allreduce* distributed communication patterns. To the best of our knowledge, this is the first attempt to use autoencoders, which capture the correlation across distributed signals, for compressing gradients.
- we experimentally evaluate our method against different benchmarks–including uncompressed gradients (baseline method), and the state-of-the-art DGC [20] and ScaleCom [25] methods—and show that we systematically achieve significant rate reductions for different tasks, models and datasets.

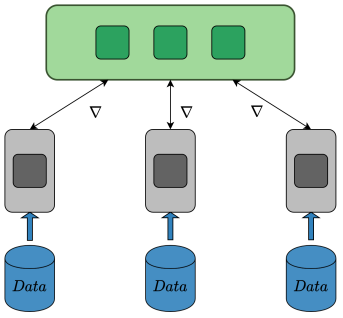



Fig. 1. Illustration of the parameter server communication pattern. $\nabla$ denotes a locally computed gradient tensor.

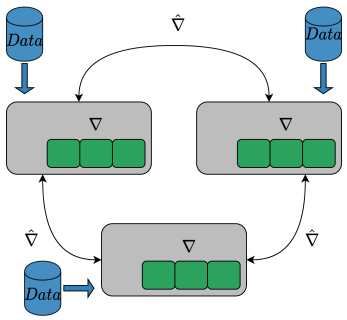



Fig. 2. Illustration of the ring-allreduce communication pattern. $\nabla$ is a locally computed gradient tensor and $\hat{\nabla}$ is a part of the gradient tensor, which is exchanged between the neighbors.

The remainder of the paper reads as follows: Section II presents different protocols for distributed training and discusses the related work. Section III analyses the correlation of gradient tensors in data-parallel distributed training through the lens of information theory. The result of the analysis motivates the use of the proposed gradient compression autoencoders, presented in Section IV. Section V describes how the autoencoder models are used within the proposed LGC framework. We present our experimental results in Section VI and conclude our work in Section VII.

## II. Setup and Related Work

### A. Setup

For the data-parallel distributed training, we consider two of the most well-established protocols of distributed communication, namely, the *parameter server* [26] and the *ring-allreduce* [27], [28] protocols. In the *parameter server* scenario (see its schema in Fig. 1), the nodes are divided into two types: the worker nodes, which contain a replica of the neural network to be trained and calculate the gradient tensor of it using the available data, and the master node[1], which receives the gradient tensors from the worker nodes, performs a reduction operation and sends back the updated gradient tensor to the worker nodes. In the *ring-allreduce* protocol (see its schema in Fig. 2), there is no master node and the calculation of the global gradient tensor is performed through the exchange of the local gradient tensors between neighboring nodes. In each node, the gradient tensor is divided into $K$ parts, where $K$ is the number of nodes. In the first phase of the communication, each node sends and receives a part of those gradient tensors. The received values are added to the corresponding values already available at the node. The transfer operation lasts for $K-1$ iterations. These $K-1$ information exchanges are organized in a way that, after each of them, each node holds a part of the final gradient tensor, e.g., the gradient values at the same indices across all nodes. Subsequently, for another $K-1$ iterations, the nodes exchange those parts of the final gradient tensor among them. In the *ring-allreduce* protocol, each of the $K$ nodes communicates with two of its neighbors $2 \times (K-1)$ times. The bottleneck in both approaches is the size of the gradient tensor, which is huge in the case of very deep models with a lot of parameters; hence, these gradient transfers can significantly decelerate the training.

### B. Related Work

Various solutions have been proposed to address the problem of the huge communication bandwidth required for distributed training. Bellow, we categorize the solutions into a few major themes.

**Sparsification:** The study in [29] proposed to sparsify the gradient tensor by replacing values below a certain threshold with zeros. The method resulted in a $99\%$ gradient sparsification during the training of a fully-connected deep neural network on MNIST [30], achieving an accuracy of $99.42\%$. Alternatively, Sparse Gradient Descent (Sparse GD) [19] applied top-k gradient selection to obtain sparse gradients. The study in [31] proposed a sparsification method that randomly drops out coordinates of the stochastic gradient vectors and amplifies the remaining coordinates to ensure that the sparsified gradient is unbiased.

**Quantization:** The authors of [22] proposed a family of algorithms for lossy gradient compression based on quantization, called QSGD. They trained the ResNet-152 [5] network to full accuracy on ImageNet [6], $1.8\times$ faster than the variant with full-precision gradients. Dithered quantization followed by adaptive arithmetic encoding of the quantized gradients was proposed in [23]. When training AlexNet [32] on Cifar10 [33], the method reduced the communication bits per node from $8531.5$ bits to $422.8$ bits, achieving $65.6\%$ accuracy, which is lower by $2.6\%$ compared with training using the original (a.k.a., uncompressed) gradients. In the context of federated learning, [34], [35] reported a combination of top-k gradient sparsification in the downstream communication (i.e., from the master to the workers) with ternary gradient quantization in the upstream communication, achieving $85.46\%$ accuracy on Cifar10 using a modified version of VGG [36] (coined VGG11). Furthermore, Amiri et al. [37] proposed to quantize the updates being sent from the central node to all of the devices by exploiting the knowledge of the last global model estimate available at the devices as side information.

**Error correction:** Error correction techniques, which compensate for the errors introduced by the compression of gradients, have also been proposed. The study in [38] introduced EF-SGD, an error-feedback (EF) mechanism applied to the training with SGD (on a single graphics processing units (GPU)). The idea was to combine residuals of the compression from the previous iteration with the current gradient before performing an update of the parameters. The authors of [39] applied the EF mechanism to data-parallel distributed training. The so-called Dist-EF-SGD method quantizes the gradient of each layer to 1 bit and combines it with the residual of the compressed gradient of the previous iteration. This way, it achieves a $32\times$ reduction of the communication cost, while retaining the same test accuracy on ResNet50 [5] trained on the ImageNet [6] dataset, and reducing the total time by $46\%$ compared to the case that uncompressed weights are used during training. Alternatively, the study in [21] proposed an EF mechanism where the gradients are quantized to 1 bit and the quantization errors are added to the gradients of the next batch. The combination of gradient compression via sparsification with error correction has been studied in [20], [40]. The study in [40] proposed MEM-SGD, a technique that keeps track of the accumulated errors due to top-k gradient sparsification and adds them back to the gradient estimator before each transmission. MEM-SGD converges at the same rate as SGD on convex problems, whilst reducing the communication overhead by a factor equal to the problem dimensionality. The authors of [20] proposed Deep Gradient Compression (DGC), in which only important gradients (i.e., based on the magnitude of the gradient) are sent per node at every iteration. The rest of the gradients are accumulated at each node using momentum correlation, a sort of error correction mechanism, and sent when they pass the threshold of importance. DGC achieves up to $99.9\%$ sparsification of gradients without loss of accuracy when training ResNet101 [5] on Cifar10 [33]. In the field of federated learning, the authors of [41] proposed a method called FetchSGD, where they moved momentum and error accumulation from clients to the central aggregator. This

[1]Depending on the application scenario, one of the worker nodes can take over the responsibility of the master node. This eliminates the required resource allocation for a standalone master node in the system.

transition of the accumulation operation was possible because of the use of the so-called Count Sketch randomized data structure. The advantage of the Count Sketch is that it can compress a vector by randomly projecting it several times to lower-dimensional spaces, such that high-magnitude elements can later be approximately recovered. Further, in [42], the authors proposed the Artemis framework in which, using an error correction mechanism, they compress both upstream and downstream communication.

**Gradients similarity:** Few works in the field of distributed training concentrate on the study of the correlation among the gradients of the different nodes. The authors of [25] proposed the Scalable Sparsified Gradient Compression (ScaleCom) method, where they explored the similarity between the gradient residuals within distributed training in combination with error-correction techniques. Considering the cosine distance between the gradient residuals at the different nodes, they showed that the distance decreases fast over the iterations, i.e., the local memory similarity increases fast and stays notable through most of the training process. Based on this similarity, the authors of [25] proposed a new Cyclic Local Top-k (CLT-k) compressor for the ring-allreduce distributed training. Their method works as follows: one of the workers sorts its error-feedback gradient, obtains its local top-k indices and further, all other workers follow the leading worker's top-k index selection for compressing their local error-feedback gradients. It is worth mentioning that a similar selection process is also used in our LGC framework for the ring-allreduce communication pattern. However, because of the proposed autoencoder, our framework can provide higher compression rates compared with ScaleCom. Leveraging the similarities in the gradient information transmitted by different workers has also been proposed by Abdi and Fekri [24]. Their compression method follows the distributed source coding [43]–[45] principles: The gradients of the different nodes are modeled as noisy versions of the true gradient, where the noise is assumed *i.i.d.* Gaussian (this refers to the CEO problem [45] in information theory). The technique applies asymmetric Wyner-Ziv coding [44], where the gradients from a group of nodes are intra-encoded and used as side information to decode the gradients from the rest of the nodes. For the latter group of nodes, compression is performed using nested scalar quantization followed by syndrome encoding [46] realized by low-density parity check (LDPC) codes. However, applying LDPC decoding of the gradients per training iteration induces a significant decoding complexity and latency, limiting the practical application of the method. In this work, we introduce a new compression framework that uses autoencoders to capture the correlation across gradients of distributed nodes. Unlike the method in [24], our approach (*i*) assumes that gradients share a common information component and differ by an innovation component; (*ii*) uses lightweight autoencoders that induce limited computational complexity at the encoder and decoder side.

To the best of our knowledge, our work is the first to propose the use of autoencoder models for the compression of gradients in distributed training, which also capture the gradient correlations across nodes. Autoencoder models have been proposed for image compression [47], [48] and image compressed sensing [49]. These models, however, use deep CNN and recurrent neural network (RNN) architectures for compression (as opposed to our lightweight models), which are not favorable in the distributed training setting where fast encoding and decoding is paramount. Furthermore, while the autoencoder architecture in [47] also considers the correlation across images from distributed cameras, the model architecture, the data at hand (images in [47] versus gradients in our case) and the setting (distributed camera communication in [47] versus distributed training in our work) are different than ours.

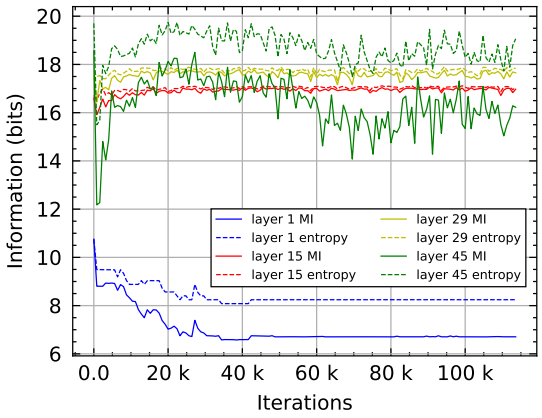
## III. Information Plane of Gradients within a Distributed Training

In this section we demonstrate the importance of leveraging the correlation (a.k.a., redundancy) of gradients in distributed nodes in the reduction of the compression rate from an information-theoretic point of view. To this end, we analyze the statistical dependencies among the gradient tensors produced by different computing nodes within distributed training using information-theoretic measures; namely, the marginal and conditional entropy, and the mutual information (MI). The calculation of these measures relies on the estimation of the underlying probability density function (pdf) of the observed variables. Different approaches can be used to calculate the marginal and conditional pdf of gradient tensors including histogram, parametric, and non-parametric estimation. In this study, we consider the basic method of calculating the histogram as we are merely interested in corroborating that gradients across distributed nodes are correlated.

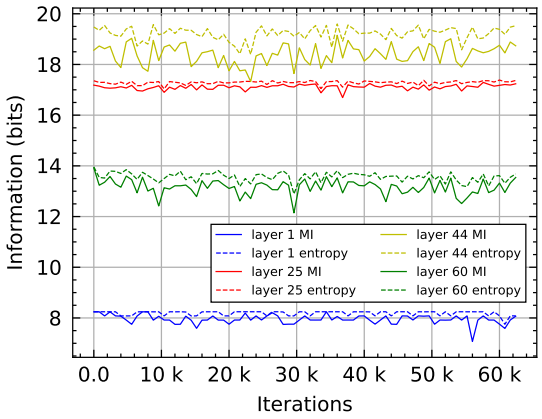
We conduct experiments using deep neural networks trained on two image transformation tasks, image classification, and semantic segmentation. For the image classification task, we train the Resnet50 [5] model on the Cifar10 [33] dataset and, for the semantic segmentation task, we train the PSPNet [50] model on the CamVid [51] dataset. In both cases, training is performed on two distributed nodes using synchronous SGD.

Let $g_{l,1}^{(i)}$ and $g_{l,2}^{(i)}$ denote gradient vectors (vectorized tensors) of distributed nodes, where $l = 1, 2, \ldots, L$, indexes the convolutional layer in the model, 1 or 2 identifies the computing node that calculates the gradient, and $i = 0, 1, \ldots, N$, denotes the training iteration. Each gradient is discretized using the uniform quantizer with $2^{32}$-level (32 bit) accuracy. The quantized gradients are then used to approximate the marginal and the joint densities, respectively, $p_{g_{l,2}}^{(i)}$ and $p_{g_{l,1},g_{l,2}}^{(i)}$, using the histogram method. Using the estimated densities, we estimate the marginal and conditional entropy, denoted by $H(g_{l,2}^{(i)})$ and $H(g_{l,2}^{(i)}|g_{l,1}^{(i)})$, respectively. The MI between the two gradient vectors, at iteration $i$, is then calculated as

$$
\begin{aligned}
&I(g_{l,1}^{(i)};g_{l,2}^{(i)}) \\
&\quad = H(g_{l,2}^{(i)}) - H(g_{l,2}^{(i)}|g_{l,1}^{(i)}) \\
&\quad = -\sum p_{g_{l,2}}^{(i)} \log_2 p_{g_{l,2}}^{(i)} + \sum p_{g_{l,1},g_{l,2}}^{(i)} \log_2 \frac{p_{g_{l,1},g_{l,2}}^{(i)}}{p_{g_{l,1}}^{(i)}}.
\end{aligned} \tag{1}
$$

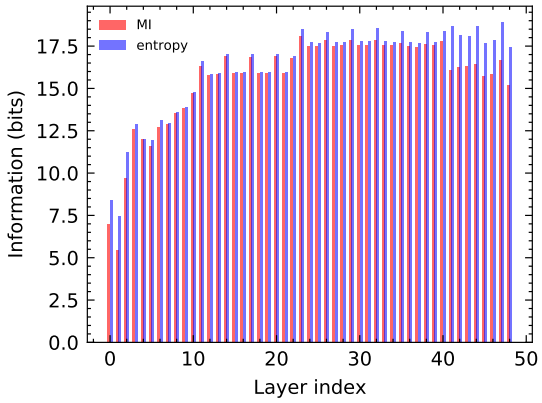



(a)

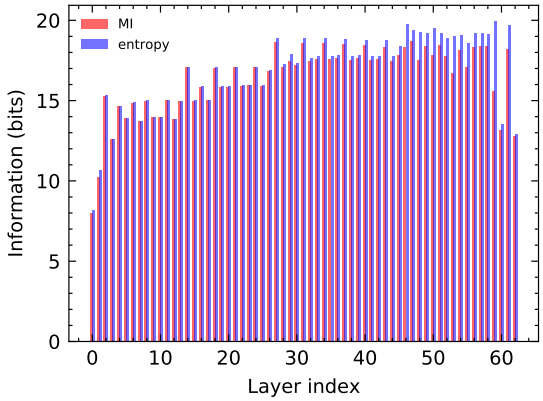



(b)

Fig. 3. The mutual information (solid lines) and the marginal entropy (dotted lines) between gradient tensors of the same layer on the different nodes, through the training iterations, for distributed training of (a) the Resnet50 and (b) the PSPNet models.




(a)




(b)

Fig. 4. The mean marginal entropy and the mean MI per layer, averaged over the iterations, for (a) the Resnet50 trained on Cifar10 and (b) the PSPNet trained on CamVid.

Figure 3 depicts the marginal entropy $H(g_{l,2}^{(i)})$ and the MI $I(g_{l,1}^{(i)}; g_{l,2}^{(i)})$ for different pairs of layers in the models throughout the training iterations. We observe that the MI values are high; specifically, approximately $80\%$ of the average information content (i.e., the entropy) contained in the layer's gradient tensor at every iteration is common for both nodes. This means that there is a significant amount of information that can be obtained from one gradient tensor about the other per iteration. We observe similar trends between the marginal entropy and the MI across the iterations, which indicates that this correlation can be leveraged independent of the changes in the information content of the gradient tensor. Methods that encode the gradients at each node independently (intra-node encoding) do not leverage this correlation. In that case, the lower rate bound is the sum of marginal entropies of the gradients of the nodes, which is higher than the minimum achievable rate bound for correlated sources [43], that is, the joint entropy: $H(g_{l,1}^{(i)}, g_{l,2}^{(i)}) = H(g_{l,1}^{(i)}) + H(g_{l,2}^{(i)}) - I(g_{l,1}^{(i)}; g_{l,2}^{(i)}) \leq H(g_{l,1}^{(i)}) + H(g_{l,2}^{(i)})$.

Figure 4 illustrates the mean marginal entropy $\left(\frac{1}{N}\sum_i^N H(g_{l,2}^{(i)})\right)$ and the mean MI $\left(\frac{1}{N}\sum_i^N I(g_{l,1}^{(i)}; g_{l,2}^{(i)})\right)$ for all layers $l = 1, 2, \ldots, L$, averaged over the iterations. One can observe that the mean marginal entropy increases with the layer index, which is due to the increase in the number of parameters in corresponding layers. Interestingly, the mean MI is systematically comparable with the mean marginal entropy for almost every layer. The lowest amounts of MI between the gradients of the two nodes–namely, the higher discrepancy with the entropy–were obtained in the first and the last layers. This is because the gradients of these layers are influenced the most by the input training samples and the ground truth labels in the mini-batch. We can also notice some systematic peaks in the values of the entropy and MI. These peaks can be explained based on the architecture of each model. Both models contain residual connections, where information between layers is combined through the element-wise summation operation. Thus, the amount of information should increase after the addition operation. This could be also confirmed from the results. Specifically, we observe that the layers with a higher amount of information are those that come after residual connections.

Our empirical results suggest that the gradient information per layer at each node consists of two parts, the *common information* shared across all gradient vectors and the *innovation information* that is specific for each node. Moreover, the *common information* holds a substantial amount of the gradients' information. In other words, in distributed training,

there is a significant amount of redundant information – currently being sent at each iteration from all the nodes– which if eliminated can further reduce the communication rates without affecting the performance of the trained model. Within the proposed framework, we assume that the innovation component can be captured by performing top-k selection (with a sparsity rate of $0.001\%$) over the gradient tensors of each node.

In what follows, we present two instances of our LGC framework that leverage this information in the *parameter server* and the *ring-allreduce* communication pattern.

## IV. Autoencoders for Gradient Compression

In this section, we introduce two autoencoder architectures for performing lossy (i.e., a reconstructed signal differs from the original one) distributed compression of gradients. We design our architectures based on our empirical observations in Section III. Specifically, our models aim at capturing the fact that gradients at the same layers of the different nodes are highly correlated. Two such autoencoder models are described, one for the parameter server (see Section IV-A) communication pattern and the other for the ring-allreduce (see Section IV-B).

### A. Autoencoder Based on Decoupling of the Gradients for the Parameter Server Communication Pattern

Consider a set of gradient tensors, each unfolded in the form of a vector $g_k$, $k = 1, ..., K$. These gradients correspond to the same training iteration of synchronous SGD across $K$ nodes, where the index of the iteration is omitted for simplicity. According to our modelling approach, each of the gradient vectors can be expressed as:

$$g_k = g_k^{cp} + g_k^I, \tag{2}$$

where $g_k^{cp}$ is the common part that is shared by the gradient vectors and $g_k^I$ is the innovation component that is unique to each gradient vector. The proposed autoencoder model for the compression and the reconstruction of the gradients consists of one encoder, $E_c$, and $K$ decoders, $D_c^k$, $k = 1, ..., K$. The encoder encodes one of the gradients $g_i$, with $i \in (1, ..., K)$, into a *compressed common representation*:

$$g^c = E_c(g_i). \tag{3}$$

Each of the $K$ decoders inputs the compressed common representation, $g^c$, and combines it with the innovation information of the $k$-th gradient, $g_k^I$, to obtain the corresponding reconstructed gradient vector,

$$g_k^{rec} = D_c^k(g^c, g_k^I). \tag{4}$$

During the training process of the autoencoder, in order to obtain the compressed common representation $g^c$ from the gradient vectors, all gradients are fed to the encoder $E_c$. The following similarity loss (i.e., the Euclidean distance between the compressed representations of the gradients) is minimized:

$$L_{sim} = \sum_{k=0}^{K} \sum_{m=0,\, m\neq k}^{K} ||(E_c(g_k) - E_c(g_m))||_2^2. \tag{5}$$

In order to reconstruct the original gradients from the compressed ones, the following reconstruction loss is also applied to the output of the decoders $D_c^k$:

$$L_{rec} = \sum_{k=0}^{K} ||(g_k - g_k^{rec})||_2^2. \tag{6}$$

The final loss function therefore consists of two terms, the reconstruction loss and the similarity loss, that is,

$$L = \lambda_1 L_{rec} + \lambda_2 L_{sim}. \tag{7}$$

Within the data-parallel distributed training, the training of the autoencoder [see Fig. 5(a)] is performed at the master node as follows. The master node receives the gradient vectors from the worker nodes and passes them sequentially to the encoder $E_c$. The encoded representations are used to calculate the similarity loss $L_{sim}$. Furthermore, one of the encoded representations (chosen randomly at each iteration) is combined with the innovation components, $g_k^I$, $k = 1, \ldots, K$, at the corresponding decoders $D_c^k$ to reconstruct the gradients and compute the $L_{rec}$ loss (see Fig. 6).

### B. Autoencoder Based on Aggregation of the Gradients for the Ring-Allreduce Communication Pattern

The final goal of one iteration of distributed training is to obtain an aggregated gradient from the gradients of distributed nodes and send it back to all nodes so that the parameters are further updated. To this end, we design an architecture where, as a first step, a separate compressed representation for each of the gradients is obtained and then, in a next step, the compressed representations are averaged. One aggregated gradient vector, which approximates the average of the gradients of distributed nodes, is then reconstructed from the averaged compressed representations. In this case, the autoencoder consists of one encoder $E_c$, which is responsible for the compression of the gradient vectors, and one decoder $D_c$ that is responsible for the final aggregated gradient vector. For the group of the gradients, $g_k$, $k = 1, ..., K$, the compressed representations, $g_k^c$, obtained by the encoder $E_c$ are given using:

$$g_k^c = E_c(g_k). \tag{8}$$

The intermediate representation $g^{avg}$, which is obtained as:

$$g^{avg} = \frac{1}{K} \sum_{k=1}^{K} E_c(g_k), \tag{9}$$

is fed to the decoder $D_c$, which reconstructs the final aggregated gradient vector as:

$$g^{rec} = D_c(g^{avg}). \tag{10}$$

In the training process, the following reconstruction loss is applied on the final aggregated gradient to minimize the distance between the output of the decoder and the average of the gradient vectors:

$$L_{rec} = ||(g^{rec} - \frac{1}{K} \sum_{k=1}^{K} g_k)||_2^2. \tag{11}$$

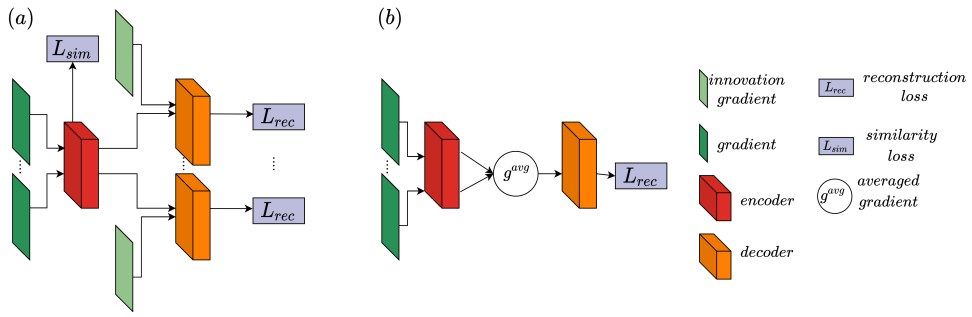



Fig. 5. The architecture of the proposed autoencoders for the two different distributed communication scenarios: (a) in the parameter server scenario, gradients are fed sequentially to the encoder and, at each decoder, the innovation gradients are concatenated with the intermediate output before the last convolutional layer; (b) in the ring-allreduce scenario, the vectors are fed to the encoder sequentially, and after that, the outputs of the encoder form the averaged gradient, which is passed to the decoder for the reconstruction. Note that the number of gradients, decoders and the reconstruction losses needs to be equal to the number of distributed nodes.

Within the data-parallel distributed training, the training of the autoencoder [see Fig. 5(b)] is performed as follows. One node obtains the gradient vectors, $g_k$, and sequentially feeds them into the encoder $E_c$, which in turn constructs the intermediate representation, $g^{avg}$, and passes this representation to the decoder $D_c$. The decoder reconstructs the gradient $g^{rec}$, which is used to calculate the loss $L_{rec}$ (see Fig. 7).

### C. Distributed Autoencoders Architecture

In our distributed compression approach, the proposed autoencoder networks (see Section IV-A and Section IV-B) consist of convolutional and deconvolutional layers. The kernels of the convolutional and deconvolutional layers are one-dimensional (1D) since the inputs of the networks are vectors. This approach reduces the number of network parameters compared with the conventional 2D kernels by approximately $60\%$.

Regarding the autoencoder architecture that is based on decoupling of the gradients (see Section IV-A), the encoder, $E_c$, which takes as input the vector $g_k$, comprises 5 convolutional layers with 1D kernels and a non-linearity in the form of the leaky-relu [52]. The decoder, $D_c$, takes as input the compressed representation produced by the encoder and performs an upsampling operation using 5 deconvolutional layers. Before the final convolutional layer, the intermediate representation is concatenated with the innovation component, $g_k^I$, which consists of the top-magnitude values of $g_k$ and zeros elsewhere; more details on how the innovation component is constructed are provided in Section V.

Regarding the autoencoder that is based on the aggregation of the gradients (see Section IV-B), the architecture of the encoder, $E_c$, and the decoder, $D_c$, are the same with the only difference that there is no concatenation operation and the input of the decoder is the average, $g^{avg}$, of the compressed gradients.

## V. Distributed Learning Framework

Assume a system with multiple GPUs, consisting of $K$ processing nodes. The goal is to train a model (with layers $l = 1, \ldots, L$) in a data-parallel distributed manner (where replicas of the model are located at each node) using synchronous SGD. Without loss of generality, it can be considered that the model is a fully convolutional neural network [32], [53]. At each training iteration, a gradient tensor $\nabla_{k,l}$ is produced for each layer $l = 1, \ldots, L$ of the neural network and for each node $k = 1, \ldots, K$. Each such gradient tensor is unfolded in the form of a vector $g_{k,l} \in \mathbb{R}^{n_l}$, where $n_l = k_l^h \cdot k_l^w \cdot f_{l-1} \cdot f_l$, with $k_l^h, k_l^w$ denoting respectively the kernel height and the kernel width at the layer $l$, $f_{l-1}$ the number of filters in the previous layer and $f_l$ the number of filters in the current layer.

### A. Gradient Sparsification Process

Within the proposed LGC framework, in order to reduce the amount of gradient information sent from each node, a certain amount of values are selected from each vector, $g_{k,l}$, in two different ways depending on the communication pattern. Specifically, under the *parameter server* communication pattern, the framework extracts the $\alpha\%$ of the values in $g_{k,l}$ with the highest magnitude and constructs the vector $\tilde{g}_{k,l} \in R^{\mu_l}$, where $\mu_l = \frac{\alpha}{100} \cdot n_l$. The top-magnitude gradient selection process is repeated for all layers and concatenates the $\tilde{g}_{k,l}$, $l = 1, \ldots, L$, vectors together to form the vector $\tilde{g}_k \in R^{\mu}$, with $\mu = \sum_{l=0}^{L} \mu_l$. This process is performed independently at each node $k = 1, \ldots, K$ with $\alpha$ fixed across the nodes (typically $\alpha = 0.1$). Alternatively, under the *ring-allreduce* communication pattern, our framework selects a node $k$ randomly for each iteration. This selected node in turn extracts the $\alpha\%$ of the values in $g_{k,l}$ with the highest magnitude and constructs the vector $\tilde{g}_{k,l} \in R^{\mu_l}$. As in the previous setting, a vector $\tilde{g}_k \in R^{\mu}$ is created by concatenation of the top-magnitude gradients of all layers. The selected node then shares the indices of the extracted gradient values to all remaining nodes in the network, which in turn construct the corresponding vectors $\tilde{g}_{k'} \in R^{\mu}$, where $k' = 1, \ldots, K$ and $k' \neq k$.

Hence, while in the *parameter server* communication pattern, each node is free to select independently the important gradients, in the *ring-allreduce* pattern all nodes select gradients in the same positions as indicated by the selected node for each iteration. The transferred indices are entropy encoded–using the DEFLATE compression method [54]–and their rate is taken into account in the total rate calculation (see Section VI).

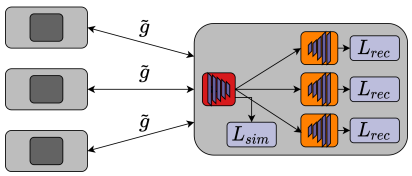



Fig. 6. Training of the proposed autoencoder for the parameter server communication pattern. $\tilde{g}$ is the gradient vector that is constructed using the top-magnitude gradient values. Note that in the training phase of the autoencoder, the innovation gradient vector $\tilde{g}^I$ is extracted on the master node from the $\tilde{g}$ gradient vector.

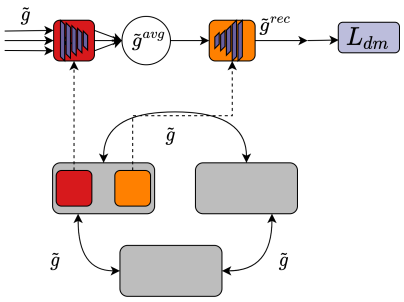



Fig. 7. Training of the proposed autoencoder for ring-allreduce communication pattern. Note that our framework selects randomly at each iteration the node that is responsible for extracting the values with the highest magnitude in order to construct the $\tilde{g}$ gradient vector.

In both patterns (i.e., parameter server and ring-allreduce), the selected gradients $\tilde{g}_k$ are passed to an encoder, the architecture of which is described in Section IV, which performs further compression. Finally, the remaining non-selected gradients are being accumulated using a momentum correlation similar to the method described in [20].

### *B. Distributed Training Process*

During the first iterations, the weights of a model change very aggressively and thus, the calculated gradients are being rapidly outdated. Any substitution or transformation of the gradients at this stage can be harmful to the performance of the model; this has also been stated in [20]. For this reason, we do not apply the gradient sparsification process and compression at the first iterations of the training.

After a number of iterations, the weights of the model are updated using the gradient vector $\tilde{g}_k$ (which has been constructed using the highest magnitude values, see Section V-A for more details). In parallel, the autoencoder network is trained using the gradient vectors at a particular node. This is either the master node in the *parameter server* communication pattern (see Fig. 6) or any selected node in the *ring-allreduce* pattern (see Fig. 7). Note that there can be various criteria for selecting the node that is responsible for training the compression network in the *ring-allreduce* pattern, such as energy, computational capacity, or bandwidth constraints. The training of the compression network lasts for a number of iterations (typically, 200-300 iterations depending on the task, see Section VI-G for more details) and then it can be used to compress the gradients. To this end, our approach treats the two communication patterns differently.

**Algorithm 1** Compression and reconstruction of the gradient with the LGC, for the parameter server communication pattern, on the node $k$ (which is produce compressed-common representation)

**Input :** $K$ nodes, minibatch size $b$, encoder $E_c$, decoder $D_c^k$, Loss function, $Loss$, optimizer *SGD*, # layers $L$

$g_{acc} \longleftarrow 0$ **for** *it = 0, 1, ...* **do**
  **for** $l$ *= 0, ...,L* **do**
    $g_l \longleftarrow \nabla Loss + g_{acc}$
    $threshold \longleftarrow min(top\ 0.1\%\ of\ abs(g_l))$
    $mask \longleftarrow abs(g_l) \geq threshold$
    $\tilde{g}_l \longleftarrow mask \odot g_l$
    $g_{acc} \longleftarrow g_{acc} + (\neg mask) \odot g_l$
    $threshold_{inv} \longleftarrow min(top\ 10\%\ of\ abs(\tilde{g}_l))$
    $mask_{inv} \longleftarrow abs(\tilde{g}_l) \geq threshold_{inv}$
    $\tilde{g}_l^I \longleftarrow \tilde{g}_l \odot mask_{inv}$
  **end**
  $\tilde{g}_k \longleftarrow concatenate(\tilde{g}_l)$
  $\tilde{g}_k^I \longleftarrow concatenate(\tilde{g}_l^I)$
  $\tilde{g}^c \longleftarrow E_c(\tilde{g}_k)$
  $\tilde{g}^{rec} \longleftarrow D_c^k(\tilde{g}^c, \tilde{g}_k^I)$
**end**

**Algorithm 2** Compression and reconstruction of the gradient with the LGC, for the ring-allreduce communication pattern, on the node $k$

**Input :** $K$ nodes, minibatch size $b$, encoder $E_c$, decoder $D_c$, Loss function, $Loss$, optimizer *SGD*, # layers $L$

$g_{acc} \longleftarrow 0$
**for** *it = 0, 1, ...* **do**
  **for** $l$ *= 0, ...,L* **do**
    $g_l \longleftarrow \nabla Loss + g_{acc}$
    $threshold \longleftarrow min(top\ 0.1\%\ of\ abs(g_l))$
    $mask \longleftarrow abs(g_l) \geq threshold$
    $\tilde{g}_l \longleftarrow mask \odot g_l$
    $g_{acc} \longleftarrow g_{acc} + (\neg mask) \odot g_l$
  **end**
  $\tilde{g}_k \longleftarrow concatenate(\tilde{g}_l)$
  $\tilde{g}_k^c \longleftarrow E_c(\tilde{g}_k)$
  $\tilde{g}^{avg} \longleftarrow \frac{1}{K}\sum_k \tilde{g}_k^c (k = 1, \ldots, K)$
  $\tilde{g}^{rec} \longleftarrow D_c(\tilde{g}^{avg})$
**end**

*1) LGC - Parameter Server Communication Pattern:* In the *parameter server* communication pattern, the weights of the learned encoder are transferred to one of the worker nodes. It is worth mentioning that in the *parameter server* communication pattern (see Algorithm 1 for more details), the trained encoder $E_c$ of the proposed LGC framework at one given worker node $k$ compresses its (top-magnitude) gradient vector $\tilde{g}_k$ to the representation $\tilde{g}_k^c$, which is in turn transmitted to the master node. In parallel, all worker nodes—including the node mentioned before—apply coarse gradient selection on gradient vectors $\tilde{g}_k$ with a very aggressive sparsification rate of

$0.001\%$, resulting in transmitting the vector $\tilde{g}_k^I \in \mathbb{R}^{0.00001*\mu}$, $k = 1, \ldots, K$. One can think of $\tilde{g}_k^I$ as the innovation part of $\tilde{g}_k$, which is specific to each worker node, and $\tilde{g}_k^c$ as the common-compressed information shared across all nodes. Therefore, by having only one worker node sharing this information, we can leverage our observations as detailed in Section III (i.e., the the common and the innovation gradient information, see the relevant section for more details).

At the master node, $\tilde{g}^c$ and $\tilde{g}_k^I$ are used to reconstruct the gradient $\tilde{g}_k^{rec}$ with the help of the decoder $D_c^k$ of the proposed autoencoder (see Fig. 8), that is,

$$\tilde{g}_k^{rec} = D_c(\tilde{g}^c, \tilde{g}_k^I). \tag{12}$$

The master node then obtains the aggregated gradient by averaging the reconstructed gradients:

$$\tilde{g}^{rec} = \frac{1}{K} \sum_{k=1}^{K} \tilde{g}_k^{rec}. \tag{13}$$

The weight update in the training process is performed in three consecutive stages. In the first stage, i.e., during the initial iterations, the weights are updated using the original gradients:

$$w_t^l = w_{t-1}^l + \lambda \nabla_t^l, \tag{14}$$

where $w_t^l$ is the weight of the layer $l$ at iteration $t$. In the second stage, during the training of the compression network, the weights are updated using the top-magnitude gradients:

$$w_t^l = w_{t-1}^l + \lambda \tilde{g}_t^l. \tag{15}$$

In the third stage, the weights are updated with the reconstructed aggregated gradients:

$$w_t^l = w_{t-1}^l + \lambda \tilde{g}_t^{l,rec}. \tag{16}$$

*2) LGC - Ring-Allreduce Communication Pattern:* In the *ring-allreduce* pattern, after the training of the autoencoder, the weights of the encoder and the decoder are sent to all other nodes; namely, the $K-1$ nodes, except for the node that the training of the compression network is performed. This communication takes place only once during training and the associated rate, which is negligible, is counted in the total rate (see Section VI).

In the *ring-allreduce* communication pattern (see Algorithm 2 for more details), the top-magnitude gradient vector $\tilde{g}_k$ at each node $k = 1, \ldots, K$, is passed to the encoder $E_c$ of the proposed autoencoder, and transformed to the compressed representation:

$$\tilde{g}_k^c = E_c(\tilde{g}_k). \tag{17}$$

These compressed representations are exchanged between the nodes, and after this exchange, each node obtains the averaged compressed representation:

$$\tilde{g}^{avg} = \frac{1}{K} \sum_{k=1}^{K} \tilde{g}_k^c. \tag{18}$$

Then, at each node, $\tilde{g}^{avg}$ is passed to the decoder part of the proposed autoencoder, delivering the reconstructed aggregated gradient:

$$\tilde{g}^{rec} = D_c(\tilde{g}^{avg}). \tag{19}$$

TABLE I
COMPARISON OF THE PROPOSED LGC FRAMEWORK WITH OTHER METHODS OF DATA-PARALLEL DISTRIBUTED TRAINING. DGC AND SPARSE GD REFER TO THE DEEP GRADIENT COMPRESSION AND SPARSE GD METHODS, RESPECTIVELY.

| | LGC | DGC | Sparse GD |
|---|---|---|---|
| Sparse Gradients | ✓ | ✓ | ✓ |
| Locally Accumulated Gradients | ✓ | ✓ | ✓ |
| Momentum Correlation | ✓ | ✓ | - |
| Same Hyperparameters for Distributed and Non-Distributed Training | ✓ | - | ✓ |
| Fixed Sparsification Amount during Training | ✓ | - | ✓ |
| Gradient Compression via Autoencoder | ✓ | - | - |

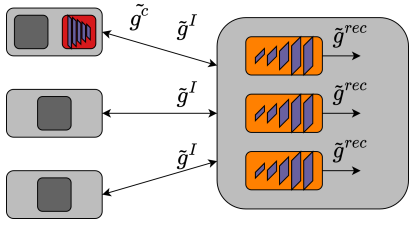



Fig. 8. Distributed training within the parameter server communication pattern: $\tilde{g}^c$ is compressed-common gradient vector, $\tilde{g}^I$ is an innovation gradient vector, and $\tilde{g}^{rec}$ is a reconstructed gradient.

Finally, the reconstructed aggregated gradients are used to update the weights of the model (see Fig. 9). In the *ring-allreduce* communication pattern, the weight updates in the training process are also performed in three stages as described by the equations (14), (15), and (16).

In Table I, we compare the proposed methodology LGC with other methods for data-parallel distributed training. As we observe, LGC relies on techniques, such as momentum correlation and local gradient accumulation to improve the performance. These techniques has also been used in other works (DGC [20] and Sparse GD [19]). Unlike the aforementioned architectures, our method exploits gradient compression of distributed nodes to reduce the communication rate. Furthermore, it is worth mentioning that, in case of LGC, switching from single-node training to distributed training is smooth, as the framework does not require any hyperparameter change in the training flow. Moreover, in case of DGC, the sparsification rate is changing during the warm-up iterations (i.e., the strategy of the warm-up is a hyper-parameter introduced by DGC). In contrast, in our case we adopt a fixed sparsification rate regardless of the task, that is, no sparsification at the warm up iterations and sparsification with the highest rate for the rest of the iterations.

## VI. EXPERIMENTS

In this section, we present the evaluation of our LGC framework on classification and image-to-image transformation tasks. All experiments are performed on a single machine with four GeForce RTX 2080 Ti GPUs and 128 GB of RAM, by emulating more than one node on each GPU. The LGC framework is built on top of Pytorch's [55] distributed package.

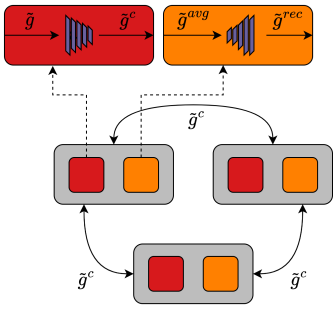



Fig. 9. Distributed training within the ring-allreduce communication pattern: $\tilde{g}^c$ is a compressed top-magnitude gradient vector, $\tilde{g}^{avg}$ is a compressed averaged gradient, and $\tilde{g}^{rec}$ is a final gradient vector.

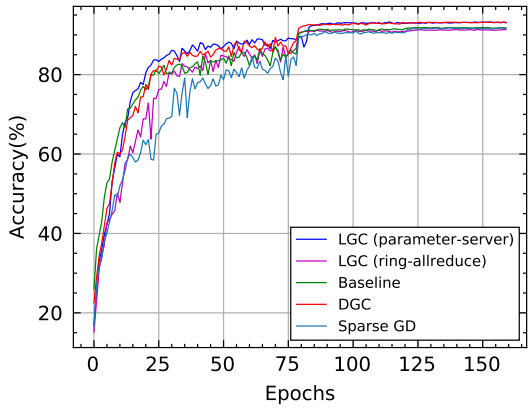



Fig. 10. Comparison of the learning curves in terms of the top1 accuracy of the Resnet50 model on the image classification task between the two versions of the LGC model (i.e., parameter server and ring-allreduce) and the rest of the examined architectures. Note that the baseline model refers to the distributed training with the uncompressed, non-modified gradients.

Furthermore, we want to stress that downlink communication is not the focus of this work (like in [20], [56]–[58]), but it can be inexpensive when the broadcast routine is implemented with the "tree" topology as in several MPI (Message Passing Interface) implementations [59].

### A. Experimental Setup

Regarding the image classification task, we consider the Resnet50 convolutional neural network trained on the Cifar10 [33] and ImageNet [6] datasets, and ResNet101 [5] on the Cifar10 [33] dataset. Regarding the image-to-image transformation task, we consider the PSPNet [50] model trained on the CamVid [51] semantic segmentation dataset.

The distributed training strategy within the LGC framework is as follows: the model is initially trained without any gradient modification for approximately 200 (depending on the task) iterations. This is because in the first iterations of training, the weights change fast and any transformation at the gradients can reduce the performance of the model (see Section VI-F). Then, the weights are updated using the top-magnitude values of the gradients, and at the same time, the autoencoder is being trained as described in Section V. This process lasts for another 200 iterations for the image-to-image transformation task and 300 iterations for the classification tasks. The autoencoder is trained with the SGD optimizer using a learning rate of 0.001 and a batch-size of 1. For the remaining iterations—that is, approximately, 89% and 83% of the total iterations for the classification and the image-to-image transformation tasks, respectively—the framework performs distributed training with the compressed top-magnitude values of the gradients using the trained autoencoder. Following other studies, e.g., [20], the aforementioned process is applied to all layers of the model except for the following two: *(i)* the first layer for which the update of the weights is performed using the original gradients; and *(ii)* the last layer of the network (fully-connected layer in the case of the classification tasks and convolutional layer in the case of image-to-image transformation task), where the top-magnitude values of the gradient are selected without further compression (i.e., the autoencoder is not used). In all layers, where the top-magnitude gradient values are selected, we set the sparsity level $\alpha$ to 0.1%.

Regarding the hyperparameters used in the trainings, we follow the exact same setup as in the original papers (see [5] for the ResNet model and [50] for the PSPNet model). For the experiments with DGC and Sparse GD, we follow the strategies that were described at the corresponding papers of DGC and Sparse GD. Specifically, for the DGC method, we adopt a warm-up approach for the learning rate as described in the work of Goyal *et al.* [60]. For the Sparse SGD method, we follow the original setup presented in the papers of each specific architecture (i.e., ResNet and PSPNet). This is because in the Sparse GD paper, the authors of the work do not explicitly specify the initialization and the learning rate. In all experiments, we report the compression ratio (CR) defined as,

$$\text{CR} = \text{size}\big(G_k^{\text{original}}\big)/\text{size}\big(G_k^{\text{compressed}}\big),$$

where $G_k^{\text{original}}$ and $G_k^{\text{compressed}}$ are the uncompressed and compressed gradients at the training node $k$, and the size$(\cdot)$ function computes the size of the gradient tensor in Megabytes. Two compression ratios are reported for the proposed LGC framework under the *parameter server* pattern. The first one refers to the worker node that shares the common-compressed and the innovation gradient component and the second one refers to all other nodes where only the innovation component is being sent. In what follows, the Baseline refers to performing distributed training of the model with the original, a.k.a., uncompressed gradients.

### B. ImageNet Classification

In the first set of experiments, we conduct distributed training of ResNet50 on ImageNet [6] and assess the achieved accuracy and speedup versus the gradient compression ratio.

ImageNet is a large-scale image classification dataset with over $1.2M$ training and $50K$ validation images belonging to 1000 classes. We used the following settings in our experiments on ImageNet: the SGD optimizer with a momentum of $0.9$, a weight decay of $1e-4$, and an initial learning rate of $0.1$ that decays by 10 every 30 epochs. Furthermore, we adopt Inception preprocessing with an image size of $224 \times 224$ pixels [61]

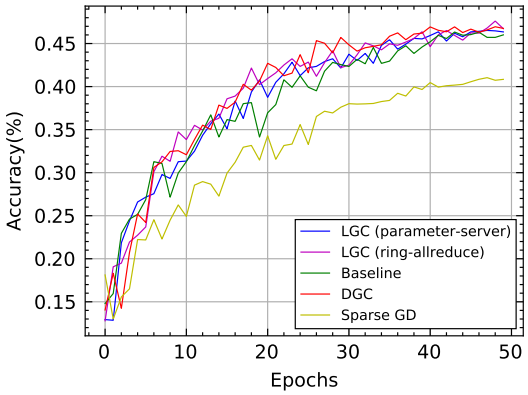



Fig. 11. Comparison of the learning curves in terms of the (pixel) accuracy of the PSPNet model on the semantic segmentation task (CamVid dataset) between the two versions of the LGC model (i.e., parameter server and ring-allreduce) and the rest of the examined architectures. Note that the baseline model refers to thedistributed training with the uncompressed, non-modified gradients.

TABLE II
TOP 1 ACCURACY VERSUS THE COMPRESSION RATIO IN DISTRIBUTED TRAINING OF RESNET50 ON IMAGENET. INFORMATION IS INDICATING THE TOTAL AMOUNT OF INFORMATION BEING SENT DURING THE WHOLE TRAINING. IN THE CASE OF PARAMETER SERVER, THE FIRST NUMBER IN THE COMPRESSION RATIO IS INDICATING THE METRICS OF THE NODE THAT IS SENDING BOTH THE COMPRESSED COMMON AND THE INNOVATION COMPONENT, AND THE SECOND NUMBER IS FOR THE REST OF THE NODES.

| Training Method | Top 1 Accuracy | Compression Ratio | Information |
|---|---|---|---|
| Baseline | 75.98% | 1× | 351TB |
| LGC (parameter server) | 75.88% | **386/2800×** | **0.4TB** |
| LGC (ring-allreduce) | 75.91% | 202× | 1.9TB |
| ScaleCom | 75.98% | 96× | 3.6TB |
| DGC | **76.01%** | 277× | 1.2TB |
| Sparse GD | 75.54% | 277× | 1.2TB |

and a batch size of 256. The reported results are single-crop performance evaluations on the ImageNet validation set.

For the experimental evaluation of our LGC framework, we performed distributed training on eight nodes (by simulating two nodes on each GPU). Table II depicts the classification accuracy versus compression ratio as well as the total amount of the gradient information (in TBs) sent from all nodes during the whole training for the LGC framework and alternative state-of-the-art distributed training methods, namely, DGC [20] (our implementation), ScaleCom [25] and Sparse GD [19] (our implementation). The results show that the proposed framework is able to achieve 386× compression of the gradients without loss of accuracy in the case of the parameter server communication pattern and 202× in the case of ring-allreduce, compared with the baseline of distributed training with the uncompressed, non-modified gradients. Moreover, in the case of the parameter server communication pattern our results outperform the state-of-the-art ScaleCom and DGC methods, where the achieved compressions are 96× and 277×, respectively. In Table II, the reported total amount of gradient information transferred during the entire training in the case of the LGC, includes updates with the original and the top-k gradients during the first two training phases, as described in Section V. The total amount of information sent is lower by 794× for the parameter server communication pattern and by 184× for the ring-allreduce compared with the baseline training. Even though the gradient compression ratio in the ring-allreduce pattern is lower than that in the parameter server pattern, the gain in the speedup is much higher. For the parameter server communication pattern, LGC achieved 1.7× speedup and for the ring-allreduce, 2.56×. This is due to that the encoder latency in ring-allreduce is lower than that of the parameter server communication pattern; the encoder in the latter setting performs a top-k selection for the construction of the innovation gradient. Specifically, the average inference time at the encoder is 0.007 ms and 0.01 ms for the ring-allreduce and the parameter server pattern, respectively, and the average inference time at the decoder is 1 ms in both patterns. Moreover in Table III we are presenting the duration of one iteration for each type of gradient update that we are using within a distributed training. Particularly, the latency for the ring-allreduce pattern in the stage of the updates with the top-k gradients and the stage of updates with the compressed gradients is lower by 1.7× and 1.5×, respectively.

TABLE III
DURATION (IN SEC) OF ONE ITERATION OF THE DISTRIBUTED TRAINING FOR EACH OF THE THREE PHASES OF GRADIENT UPDATES ON THE DISTRIBUTED TRAINING OF RESNET50 ON IMAGENET WITH EIGHT NODES. **FULL UPDATE** REFERS TO THE UPDATE WITH THE UNCOMPRESSED GRADIENTS. **TOP-K UPDATE** REFERS TO THE UPDATE WITH THE TOP-K VALUES OF THE GRADIENTS, AND ITERATIONS DURING WHICH THE AUTOENCODER IS TRAINED. **COMPRESSED UPDATE** REFERS TO THE UPDATES WITH THE PROPOSED AUTOENCODER.

| Phase | LGC parameter server | LGC ring-allreduce |
|---|---|---|
| Full update | 1 sec | 1 sec |
| Top-k update | 1.6 sec | 0.9 sec |
| Compressed update | 0.6 sec | 0.4 sec |

### C. Cifar10 Classification

The Cifar10 [33] dataset consists of 50,000 training and 10,000 validation images (with resolution 32 × 32 pixels) from 10 different classes. The distributed training of Resnet50 and Resnet101, on the Cifar10 dataset, is performed on 2 and 4 nodes, respectively. The training procedure and the hyperparameter selection for the models follows that in [5]. Figure 10 depicts the evolution of the classification accuracy with respect to the number of epochs for the ResNet50 model trained on Cifar10 using the proposed LGC framework. We compare the performance of LGC, under the parameter server and ring-allreduce communication patterns, against the baseline, DGC [20] and Sparse GD [19]. Furthermore, Table IV reports the Resnet50 top-1 classification accuracy, the total amount of gradient information sent from all nodes per iteration, and the compression ratio for each framework. The results show that our framework achieves comparable (for the ring-allreduce communication pattern) or even higher classification accuracy (for the parameter server communication pattern) compared to the baseline approach while reducing by 5709× and 3193×

the gradient information per iteration in the parameter server and the ring-allreduce communication patterns, respectively.

Furthermore, Table IV presents the performance of the proposed LGC framework (in the parameter server and in the ring-allreduce communication patterns) when training the ResNet101 model on the Cifar10 dataset using four nodes. Again, we compare LGC against the baseline, DGC [20] and Sparse GD [19]. The results show that our approach can drastically reduce the gradient rate compared to the baseline while incurring a small loss in the model performance. Specifically, in the parameter server setup, our approach reduces the size of the gradient information sent at each iteration and per node from 170MB to 0.021MB, while incurring a $0.18\%$ loss in accuracy compared to distributed training with the uncompressed, non-modified gradients. It is also worth noting that our approach reduces significantly, by almost an order of magnitude in the parameter server communication pattern, the gradient rate compared to the state-of-the-art DGC [20].

### *D. CamVid Semantic Segmentation*

CamVid [51] is a semantic segmentation dataset consisting of 32 different classes. It contains 701 images with a spatial resolution of $720 \times 960$ pixels. On the CamVid dataset, we performed training of the PSPNet 11 model on 2 nodes, with a batch size of 12, using momentum SGD. The network trained with random crops of size $473 \times 473$ pixels, on which augmentations in the form of scaling and rotation are applied.

Table IV reports the results obtained with training the PSPNet model on two nodes to address the semantic segmentation task. The results illustrate that our method reduces the size of the gradient information sent from each node at each iteration by a fraction of 450×–700×, depending on the communication setup, without any reduction in terms of performance (i.e., pixel accuracy). In effect, the proposed method leads to the same (for the parameter server setup) or even higher (for the ring-allreduce setup) pixel accuracy compared to the baseline method (i.e., a PSPNet model without distributed training). Moreover, the compression ratio—and the pixel accuracy (in case of the ring-allreduce setup)—of our LGC model is higher than that obtained with DGC and Sparse GD. Fig. 11 depicts the learning curves of the PSPNet for the various distributed training methods (and the baseline). We notice that the learning curves obtained with LGC and DGC are on par with that of the baseline, whereas the learning curve obtained with the Sparse GD method [19] is significantly lower.

### *E. Information Plane in the Large Scale Distributed Training*

In order to extend the findings presented in Section III (i.e., about the statistical dependencies among the gradient tensors in the distributed training methodology), we conduct two additional experiments in the more complex case that more than two computing nodes are available. First, we analyze the dependencies among the gradient tensors within the distributed training context of the VGG11 model when 16 nodes are available, and in the custom ConvNet5 model when 22 nodes are available.

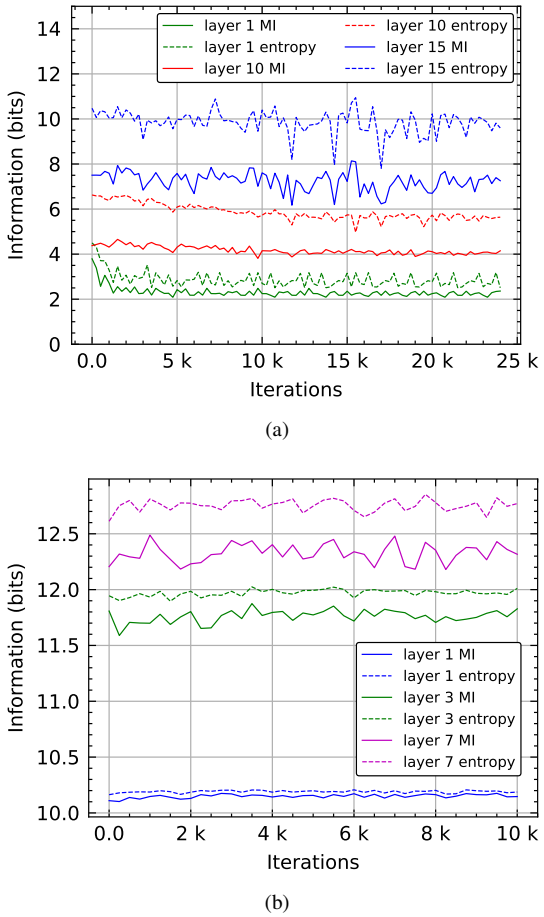



Fig. 12. The mutual information (solid lines) and the marginal entropy (dotted lines) between gradient tensors of the same layer on the (a) 3rd and 11th nodes of the VGG11 model, (b) 8th and 10th nodes of the ConvNet5 model, through the training iterations of distributed training.

**VGG11**: To conduct our additional distributed training experiment we use a modified version of the VGG16 [36] neural network, namely VGG11, comprising 11 convolutional layers. Each convolution is followed by a ReLU nonlinearity. Similar to the original implementation of VGG16, we also adopt a max-pooling operation as a method of spatial dimensionality reduction. The distributed training was performed on 16 nodes using the Food101 [62] dataset. Food-101 consists of $75,750$ training and $25,250$ testing images from $101$ different classes. We trained our model for $25$K iterations with a cumulative batch size of $128$, learning rate of $0.001$, using the SGD optimizer. To analyze the statistical dependencies between the gradient tensors, at each iteration of the training, we construct pairs of gradients of different nodes and calculate the mutual information and the entropy of each gradient pair. Figure 12(a) illustrates the mutual information between randomly picked pairs of gradients. As in the case of the ResNet50 (see Section III), we observe that a large part of the average information content (i.e., the entropy) contained in the gradient tensor of the layer at each iteration is common for both nodes.

**ConvNet5**: For our second experiment, we construct a convolutional neural network with five convolutional layers, namely ConvNet5. After each convolution, batch normalization

TABLE IV
PERFORMANCE COMPARISON OF THE LGC FRAMEWORK (FOR PARAMETERS SERVER (PS) AND RING-ALLREDUCE (RAR) COMMUNICATION PATTERNS) WITH THE OTHER METHODS OF DISTRIBUTED TRAINING. **INFO SIZE** REFERS TO THE SIZE OF INFORMATION BEING SENT PER FORWARD PASS. **RATIO** CORRESPONDS TO THE COMPRESSION RATIO. IN THE CASE OF PARAMETER SERVER, THE FIRST NUMBER IS INDICATING THE METRICS OF THE NODE SENDING BOTH COMPRESSED COMMON REPRESENTATION AND INNOVATION, AND THE SECOND NUMBER IS FOR THE REST OF THE NODES. **TOP1/PIXEL ACC.** REFERS TO THE TOP1 CLASSIFICATION ACCURACY AND THE PIXEL ACCURACY, CORRESPONDINGLY.

| Training Method | ResNet50 on Cifar10 | | | ResNet101 on Cifar10 | | | PSPNet on CamVid | | |
|---|---|---|---|---|---|---|---|---|---|
| | Top1 | Info size | Ratio | Top1 | Info size | Ratio | Pixel Acc. | Info size | Ratio |
| Baseline | 91.88% | 102.2MB | 1× | 93.75% | 170MB | 1× | 46.3% | 120MB | 1× |
| Sparse GD | 91.82 | 0.102MB | 1000× | 92.75 | 0.17MB | 1000× | 41% | 0.29MB | 413× |
| DGC | 93.2% | 0.102MB | 1000× | **93.87%** | 0.17MB | 1000× | 46.5% | 0.29MB | 413× |
| LGC (RAR) | 91.4% | 0.032MB | 3193× | 93.07% | 0.074MB | 2297× | **47.6%** | 0.261MB | 459× |
| LGC (PS) | **93.27** % | **0.017/ 0.012MB** | **5709/ 8616**× | 93.57% | **0.021/ 0.01MB** | **8095/ 17000**× | 46.3% | **0.17/ 0.16MB** | **693/ 722**× |

and ReLU nonlinearity are added. The model trained on the Tiny ImageNet [63] dataset. It consists of $100,000$ training and $10,000$ validation images from 200 classes. The distributed training is performed on 22 nodes for 14K iterations with the cumulative batch size of 128, a learning rate of $0.001$, and the SGD optimizer. To calculate the mutual information and the entropy, we followed the same procedure as for the VGG11 model. Empirical results [see Fig. 12(b)] suggest that also in this case the mutual information is high, indicating that there is a considerable amount of information that can be exploited among the gradient tensors. This is also the reason that explains the benefit of our architecture compared to the rest of the distributed training approaches.

### *F. Sparsification Strategy*

To empirically validate the advantage of our choice of sparsification strategy, presented in Section V, we conduct an experimental comparison between our method and the other methods widely used in distributed training. Specifically, we compare three approaches: (*i*) the method of exponential increase used in the DGC (e.g. exponentially increase the gradient sparsity from a relatively small value to the final value), (*ii*) the technique of applying fixed value sparsification within the whole training (e.g. a fixed value of gradient sparsity from the first iteration and till the last) used in [19], [22], [25], (*iii*) the sparsification with the warmup used in our method (e.g. no sparsification at the first iterations and fixed value gradient sparsification for the rest of the iterations). The experiments are performed on two types of neural networks: on the compact neural network ConvNet5 (see the description in Section VI-E) and a model with a large number of parameters, i.e, ResNet50. The experimental results presented in the Fig. 13 illustrate the advantage of our method. Notably, both in the case of a fixed value and exponential sparsification, we can notice that the loss is decreasing visibly slower, which can lead to the poor convergence of the model. In contrast, in our case, by performing updates with the original gradients within the first iterations and switching to the sparse updates afterwards, we can reach a faster decrease in the loss and better convergence. Our extensive experimental results suggest that independent of the model, 200 iterations for the updates with the original gradients are enough to avoid any further mislead in the optimization process caused by gradient sparsification.

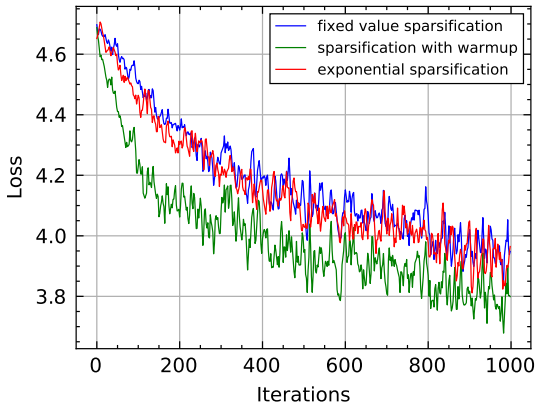
### *G. Convergence of the Autoencoders*

We also examined the convergence of our autoencoders (one for the case of parameter server communication pattern and one for the ring-allreduce) depending on the number of distributed nodes and the primary model (i.e., the model being trained in a distributed manner). We conduct experiments on the VGG16 [36] model trained on the Tiny ImageNet dataset [63] with 32 distributed nodes and on the ResNet50 [5] model on the Cifar100 [33] dataset with 24 nodes.

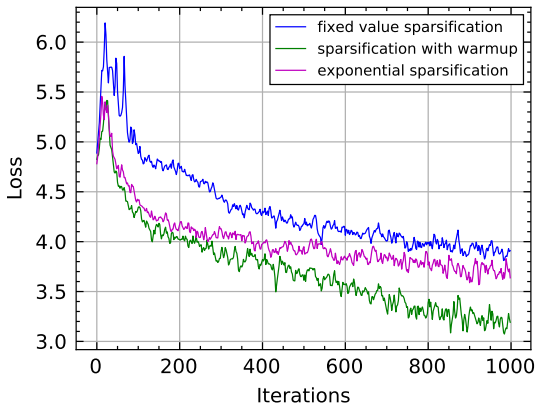
The results presented in Fig. 14 illustrate the behaviour of the reconstruction loss within the training iterations. For the parameter server scenario, we have randomly selected the reconstruction loss of the 5th and 32nd nodes in case of the ResNet50 and VGG16 training correspondingly. For all cases, we observe good convergence of the autoencoders. For the parameter server scenario we have also examined the impact of the similarity loss (see Section IV-A) on the reconstruction error. The results presented in Fig. 14 suggest that the similarity loss (i.e., $\lambda_2 = 0.5$) helps to reconstruct the gradients better.

## VII. CONCLUSION

In this paper, we have introduced a novel method for data-parallel distributed training of deep neural networks. It was shown empirically that the method is able to compress the gradients by $99.99\%$ on image classification tasks, without any reduction in terms of the accuracy or rate of convergence. This was made possible by exploring the correlations between the gradients of different nodes within the scope of distributed training and designing distributed autoencoders for gradient compression. To the best of our knowledge, the compression rate achieved on the image classification task is the highest ever reported among the other methods on the same network and the same dataset. Moreover, our LGC framework can provide up to $794\times$ reduction in the total number of bits, being transferred within a distributed training of ResNet50 on

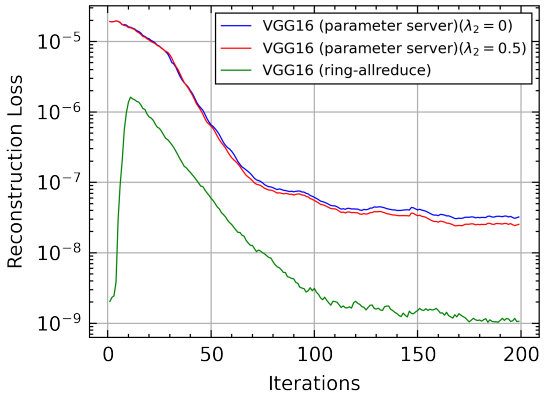



(a)

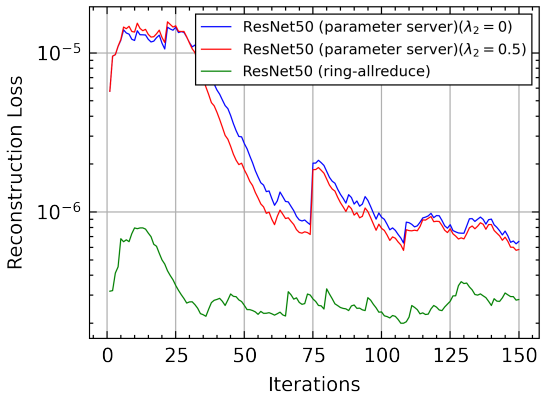



(b)

Fig. 13. The loss of the (a) ConvNet5 and (b) ResNet50 within a distributed training for the different types of sparsification strategies: (*i*) fixed value sparsification refers to the case when sparse updates are applied from the first iteration, (ii) exponential sparsification refers to the case when the percentage of sparsification is increasing exponentially during the first iterations, (iii) sparsification with warmup refers to the case when sparsification is being applied only after the first iterations (e.g. no sparsification at the beginning of the training).




(a)




(b)

Fig. 14. Reconstruction loss of the proposed autoencoders within a distributed training of (a) VGG16 and (b) ResNet50, where $\lambda_2$ is the coefficient of the similarity loss [see (7)]. $\lambda_2 = 0$ and $\lambda_2 = 0.5$ refer to the cases of training the autoencoder without and with the similarity loss.

ImageNet, compared with the baseline distributed training with original uncompressed gradients.

## REFERENCES


[1] D. Mahajan, R. Girshick, V. R. andK. He, M. Paluri, Y. Li, A. Bharambe, , and L. van der Maaten, "Exploring the limits of weakly supervised pretraining," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[3] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[4] D. Basu, D. Data, C. Karakus, and S. Diggavi, "Qsparse-local-SGD: Distributed sgd with quantization, sparsification, and local computations," 2019. [Online]. Available: https://arxiv.org/pdf/1906.02367.pdf

[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)*, 2016.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei., "ImageNet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[7] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[8] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[9] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[10] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[11] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[12] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," 2017.

[13] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 1135–1143. [Online]. Available: http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf

[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[15] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep

networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1586–1595.

[16] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *Stat*, vol. 1050, p. 9, 2015.

[18] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 535–541.

[19] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[20] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[21] 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," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[22] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. s 1707–1718.

[23] A. Abdi and F. Fekri, "Nested dithered quantization for communication reduction in distributed training," 2019. [Online]. Available: https://arxiv.org/abs/1904.01197

[24] ——, "Reducing communication overhead via CEO in distributed training," in *IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2019, pp. 1–5.

[25] C.-Y. Chen, J. Ni, S. Lu, X. Cui, P.-Y. Chen, X. Sun, N. Wang, S. Venkataramani, V. V. Srinivasan, W. Zhang *et al.*, "Scalecom: Scalable sparsified gradient compression for communication-efficient distributed training," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[26] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[27] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, p. 117–124, 2009.

[28] R. Rabenseifner, "Optimization of collective reduction operations," *ICCS*, 2004.

[29] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[31] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *Advances in Neural Information Processing Systems*, 2018, pp. 1299–1309.

[32] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[33] A. S. Krizhevsky, "Learning multiple layers of features from tiny images," *Technical Report*, 2009.

[34] Google, "Federated learning: Collaborative machine learning without centralized training data," in *Google AI Blog*, 2017. [Online]. Available: https://research.googleblog.com/2017/04/federated-learning-collaborative.html

[35] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[37] M. M. Amiri, D. Gunduz, S. R. Kulkarni, and H. V. Poor, "Federated learning with quantized global model updates," *arXiv preprint arXiv:2006.10672*, 2020.

[38] S. P. Karimireddy, Q. Rebjock, S. U. Stich, and M. Jaggi, "Error feedback fixes signsgd and other gradient compression schemes," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[39] S. Zheng, Z. Huang, and J. Kwok, "Communication-efficient distributed blockwise momentum sgd with error-feedback," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 446–11 456.

[40] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," in *Advances in Neural Information Processing Systems*, 2018, pp. 4447–4458.

[41] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora, "FetchSGD: Communication-efficient federated learning with sketching," in *Proceedings of the 37th International Conference on Machine Learning*, 2020, pp. 8253–8265.

[42] C. Philippenko and A. Dieuleveut, "Artemis: tight convergence guarantees for bidirectional compression in federated learning," *arXiv preprint arXiv:2006.14591*, 2020.

[43] D. Slepian and J. Wolf, "Noiseless coding of correlated information sources," *IEEE Transactions on Information Theory*, vol. 19, no. 4, pp. 471–480, 1973.

[44] A. Wyner and J. Ziv, "The rate-distortion function for source coding with side information at the decoder," *IEEE Transactions on Information Theory*, vol. 22, no. 1, pp. 1–10, 1976.

[45] T. Berger, Z. Zhang, and H. Viswanathan, "The ceo problem [multiterminal source coding]," *IEEE Transactions on Information Theory*, vol. 42, no. 3, pp. 887–902, 1996.

[46] S. S. Pradhan and K. Ramchandran, "Distributed source coding using syndromes (DISCUS): Design and construction," *IEEE Transactions on Information Theory*, vol. 49, no. 3, pp. 626–643, 2003.

[47] E. Diao, J. Ding, and V. Tarokh, "Drasic: Distributed recurrent autoencoder for scalable image compression," *arXiv preprint arXiv:1903.09887*, 2019.

[48] R. Zarcone, D. Paiton, A. Anderson, J. Engel, H. P. Wong, and B. Olshausen, "Joint source-channel coding with neural networks for analog data compression and storage," in *2018 Data Compression Conference*. IEEE, 2018, pp. 147–156.

[49] Y. Wu, M. Rosca, and T. Lillicrap, "Deep compressed sensing," in *International Conference on Machine Learning (ICML)*, 2019.

[50] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[51] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A highdefinition ground truth database," *Pattern Recognition Letters*, 2009.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers:surpassing human-level performance on imagenet classification," in *International Conference on Computational Vision (ICCV)*, 2015.

[53] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, 1989.

[54] P. W. Katz, "String searcher, and compressor using same," *US Patent*, no. 5051745, 1991.

[55] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[56] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in neural information processing systems*, 2017, pp. 1509–1519.

[57] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," in *Advances in Neural Information Processing Systems*, 2018, pp. 5973–5983.

[58] D. Basu, D. Data, C. Karakus, and S. Diggavi, "Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 695–14 706.

[59] M. J. Rashti, J. Green, P. Balaji, A. Afsahi, and W. Gropp, "Multi-core and network aware mpi topology functions," in *European MPI Users' Group Meeting*. Springer, 2011, pp. 50–60.

[60] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd:training imagenet in 1 hour," in *arXiv:1706.02677*, 2017.

[61] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2017.

[62] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101 – mining discriminative components with random forests," in *European Conference on Computer Vision*, 2014.

[63] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," 2015.