Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data

Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, *Member, IEEE*, and Wojciech Samek, *Member, IEEE*

***Abstract*— Federated learning allows multiple parties to jointly train a deep learning model on their combined data, without any of the participants having to reveal their local data to a centralized server. This form of privacy-preserving collaborative learning, however, comes at the cost of a significant communication overhead during training. To address this problem, several compression methods have been proposed in the distributed training literature that can reduce the amount of required communication by up to three orders of magnitude. These existing methods, however, are only of limited utility in the federated learning setting, as they either only compress the upstream communication from the clients to the server (leaving the downstream communication uncompressed) or only perform well under idealized conditions, such as i.i.d. distribution of the client data, which typically cannot be found in federated learning. In this article, we propose sparse ternary compression (STC), a new compression framework that is specifically designed to meet the requirements of the federated learning environment. STC extends the existing compression technique of top-*k* gradient sparsification with a novel mechanism to enable downstream compression as well as ternarization and optimal Golomb encoding of the weight updates. Our experiments on four different learning tasks demonstrate that STC distinctively outperforms federated averaging in common federated learning scenarios. These results advocate for a paradigm shift in federated optimization toward high-frequency low-bitwidth communication, in particular in the bandwidth-constrained learning environments.**

***Index Terms*— Deep learning, distributed learning, efficient communication, federated learning, privacy-preserving machine learning.**

Manuscript received March 6, 2019; revised June 28, 2019; accepted September 25, 2019. Date of publication November 1, 2019; date of current version September 1, 2020. This work was supported in part by the Fraunhofer Society through the MPI-FhG Collaboration Project "Theory & Practice for Reduced Learning Machines," in part by the German Ministry for Education and Research as Berlin Big Data Center under Grant 01IS14013A, in part by the Berlin Center for Machine Learning under Grant 01IS18037I, in part by DFG under Grant EXC 2046/1 and Grant 390685689, and in part by the Information & Communications Technology Planning & Evaluation (IITP) Grant funded by the Korea Government under Grant 2017-0-00451. *(Corresponding authors: Klaus-Robert Müller; Wojciech Samek.)*

F. Sattler, S. Wiedemann, and W. Samek are with the Fraunhofer Heinrich Hertz Institute, 10587 Berlin, Germany (e-mail: wojciech.samek@hhi.fraunhofer.de).

K.-R. Müller is with the Technische Universität Berlin, 10587 Berlin, Germany, with the Max Planck Institute for Informatics, 66123 Saarbrücken, Germany, and also with the Department of Brain and Cognitive Engineering, Korea University, Seoul 136-713, South Korea (e-mail: klaus-robert.mueller@tu-berlin.de).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2944481

I. Introduction

THREE major developments are currently transforming the ways how data are created and processed: First of all, with the advent of the Internet of Things (IoT), the number of intelligent devices in the world has rapidly grown in the last couple of years. Many of these devices are equipped with various sensors and increasingly potent hardware that allow them to collect and process data at unprecedented scales [1]–[3].

In a concurrent development, deep learning has revolutionized the ways that information can be extracted from data resources with groundbreaking successes in areas such as computer vision, natural language processing, or voice recognition, among many others [4]–[9]. Deep learning scales well with growing amounts of data and its astounding successes in recent times can be at least partly attributed to the availability of very large data sets for training. Therefore, there lays huge potential in harnessing the rich data provided by IoT devices for the training and improving deep learning models [10].

At the same time, data privacy has become a growing concern for many users. Multiple cases of data leakage and misuse in recent times have demonstrated that the centralized processing of data comes at high risk for the end users privacy. As IoT devices usually collect data in private environments, often even without explicit awareness of the users, these concerns hold particularly strong. It is, therefore, generally not an option to share this data with a centralized entity that could conduct training of a deep learning model. In other situations, local processing of the data might be desirable for other reasons such as increased autonomy of the local agent.

This leaves us facing the following dilemma: How are we going to make use of the rich combined data of millions of IoT devices for training deep learning models if this data cannot be stored at a centralized location?

Federated learning resolves this issue as it allows multiple parties to jointly train a deep learning model on their combined data, without any of the participants having to reveal their data to a centralized server [10]. This form of privacy-preserving collaborative learning is achieved by following a simple three-step protocol illustrated in Fig. 1. In the first step, all participating clients download the latest master model $\mathcal{W}$ from the server. Next, the clients improve the downloaded model, based on their local training data using stochastic gradient descent (SGD). Finally, all participating clients upload their locally improved models $\mathcal{W}_i$ back to the server, where they are gathered and aggregated to form a new master model

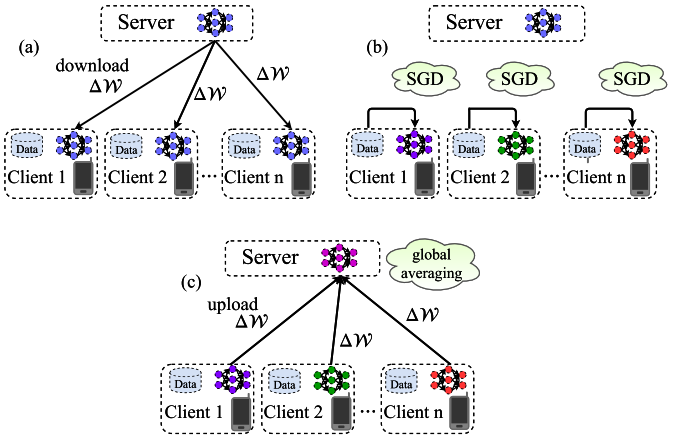

Fig. 1. Federated learning with a parameter server. Illustrated is one communication round of distributed SGD. (a) Clients synchronize with the server. (b) Clients compute a weight update independently based on their local data. (c) Clients upload their local weight updates to the server, where they are averaged to produce the new master model.

(in practice, weight updates $\Delta\mathcal{W} = \mathcal{W}^{\text{new}} - \mathcal{W}^{\text{old}}$ can be communicated instead of full models $\mathcal{W}$, which is equivalent as long as all clients remain synchronized). These steps are repeated until a certain convergence criterion is satisfied. Observe that when following this protocol, training data never leave the local devices as only model updates are communicated. Although it has been shown that in adversarial settings information about the training data can still be inferred from these updates [11], additional mechanisms, such as homomorphic encryption of the updates [12], [13] or differentially private training [14], can be applied to fully conceal any information about the local data.

A major issue in federated learning is the massive communication overhead that arises from sending around the model updates. When naively following the protocol described earlier, every participating client has to communicate a full model update during every training iteration. Every such update is of the same size as the trained model, which can be in the range of gigabytes for modern architectures with millions of parameters [15], [16]. Over the course of multiple hundred thousands of training iterations on big data sets, the total communication for every client can easily grow to more than a *petabyte* [17]. Consequently, if communication bandwidth is limited or communication is costly (naive), federated learning can become unproductive or even completely unfeasible.

The total amount of bits that have to be uploaded and downloaded by every client during training is given by

$$\mathrm{b}^{\text{up/down}} \in \mathcal{O}(\underbrace{N_{\text{iter}} \times f}_{\text{\# updates}} \times \underbrace{|\mathcal{W}| \times (H(\Delta\mathcal{W}^{\text{up/down}}) + \eta)}_{\text{update size}}) \tag{1}$$

where N_{iter} is the total number of training iterations (forward–backward passes) performed by every client, f is the communication frequency, $|\mathcal{W}|$ is the size of the model, $H(\Delta\mathcal{W}^{\text{up/down}})$ is the entropy of the weight updates exchanged during upload and download, respectively, and η is the inefficiency of the encoding, i.e., the difference between the true update size and the minimal update size (which is given by the entropy). If we assume the size of the model and number of training iterations to be fixed (e.g., because we want to achieve a certain accuracy on a given task), this leaves us with three options to reduce communication: 1) we can reduce the communication frequency f; 2) reduce the entropy of the weight updates $H(\Delta\mathcal{W}^{\text{up/down}})$ via lossy compression schemes; and/or 3) use more efficient encodings to communicate the weight updates, thus reducing η.

II. Challenges of the Federated Learning Environment

Before we can consider ways to reduce the amount of communication, we first have to take into account the unique characteristics, which distinguish federated learning from other distributed training settings such as parallel training (compare also with [10]). In federated learning, the distribution of both training data and computational resources is a fundamental and fixed property of the learning environment. This entails the following challenges.

1) *Unbalanced and non-i.i.d. data:* As the training data present on the individual clients is collected by the clients themselves based on their local environment and usage pattern, both the size and the distribution of the local data sets will typically vary heavily between different clients.
2) *Large number of clients:* Federated learning environments may constitute of multiple millions of participants [18]. Furthermore, as the quality of the collaboratively learned model is determined by the combined available data of all clients, collaborative learning environments will have a natural tendency to grow.
3) *Parameter server:* Once the number of clients grows beyond a certain threshold, direct communication of weight updates becomes unfeasible because the workload for both communication and aggregation of updates grows linearly with the number of clients. In federated learning, it is, therefore, unavoidable to communicate via an intermediate parameter server. This reduces the amount of communication per client and communication rounds to one single upload of a local weight update to and one download of the aggregated update from the server and moves the workload of aggregation away from the clients. Communicating via a parameter server, however, introduces an additional challenge to communication-efficient distributed training, as now both the upload to the server and the download from the server need to be compressed in order to reduce communication time and energy consumption.
4) *Partial participation:* In the general federated learning for IoT setting, it can generally not be guaranteed that all clients participate in every communication round. Devices might lose their connection, run out of battery or seize to contribute to the collaborative training for other reasons.
5) *Limited battery and memory:* Mobile and embedded devices often are not connected to a power grid.

TABLE I
DIFFERENT METHODS FOR COMMUNICATION-EFFICIENT DISTRIBUTED DEEP LEARNING PROPOSED IN THE LITERATURE. NONE OF THE EXISTING METHODS SATISFIES ALL REQUIREMENTS **(R1)**–**(R3)** OF THE FEDERATED LEARNING ENVIRONMENT. WE CALL A METHOD "ROBUST TO NON-I.I.D. DATA" IF THE FEDERATED TRAINING CONVERGES INDEPENDENT OF THE LOCAL DISTRIBUTION OF CLIENT DATA. WE CALL COMPRESSION RATES GREATER THAN ×32 "STRONG" AND THOSE SMALLER OR EQUAL TO ×32 "WEAK"

Method	Downstream Compression	Compression Rate	Robust to NON-IID Data
TernGrad [19], QSGD [20], ATOMO [21]	NO	WEAK	NO
signSGD [22]	YES	WEAK	NO
Gradient Dropping [23], Variance based [24], DGC [25], Strom [26]	NO	STRONG	YES
Federated Averaging [10]	YES	STRONG	NO
Sparse Ternary Compression (ours)	YES	STRONG	YES

Instead, their capacity to run computations is limited by a finite battery. Performing iterations of SGD is notoriously expensive for deep neural networks. It is, therefore, necessary to keep the number of gradient evaluations per client as small as possible. Mobile and embedded devices also typically have only very limited memory. As the memory footprint of SGD grows linearly with the batch size, this might force the devices to train on very small batch sizes.

Based on the above-mentioned characterization of the federated learning environment, we conclude that a communication-efficient distributed training algorithm for federated learning needs to fulfil the following requirements.

(R1): It should compress both upstream and downstream communications.

(R2): It should be robust to non-i.i.d., small batch sizes, and unbalanced data.

(R3): It should be robust to large numbers of clients and partial client participation.

III. CONTRIBUTION

In this article, we will demonstrate that none of the existing methods proposed for communication-efficient federated learning satisfies all of these requirements (see Table I). More concretely, we will show that the methods that are able to compress both upstream and downstream communications are very sensitive to non-i.i.d. data distributions, while the methods that are more robust to this type of data do not compress the downstream (see Section V). We will then proceed to construct a new efficient communication protocol for federated learning that resolves these issues and meets all requirements **(R1)**–**(R3)**. We provide a convergence analysis of our method as well as extensive empirical results on four different neural network architectures and data sets that demonstrate that the sparse ternary compression (STC) protocol is superior to the existing compression schemes in that it requires both fewer gradient evaluations and communicated bits to converge to a given target accuracy (see Section IX). These results also extend to the i.i.d. regime.

IV. RELATED WORK

In the broader realm of communication-efficient distributed deep learning, a wide variety of methods has been proposed to reduce the amount of communication during the training process. Using (1) as a reference, we can organize the substantial existing research body on communication-efficient distributed deep learning into three different groups.

1) **Communication delay** methods reduce the communication frequency f. McMahan *et al.* [10] propose federated averaging where instead of communicating after every iteration, every client performs multiple iterations of SGD to compute a weight update. McMahan *et al.* observe that on different convolutional and recurrent neural network architectures, communication can be delayed for up to 100 iterations without significantly affecting the convergence speed as long as the data are distributed among the clients in an i.i.d. manner. The amount of communication can be reduced even further with longer delay periods; however, this comes at the cost of an increased number of gradient evaluations. In a follow-up work, Konečnỳ *et al.* [27] combine this communication delay with random sparsification and probabilistic quantization. They restrict the clients to learn random sparse weight updates or force random sparsity on them afterward ("structured" versus "sketched" updates) and combine this sparsification with probabilistic quantization. Their method, however, significantly slows down convergence speed in terms of SGD iterations. Communication delay methods automatically reduce both upstream and downstream communication and are proven to work with large numbers of clients and partial client participation.
2) **Sparsification** methods reduce the entropy $H(\Delta W)$ of the updates by restricting changes to only a small subset of the parameters. Strom [24] presents an approach (later modified by [26]) in which only gradients with a magnitude greater than a certain predefined threshold are sent to the server. All other gradients are accumulated in a residual. This method is shown to achieve upstream compression rates of up to three orders of magnitude on an acoustic modeling task. In practice, however, it is hard to choose appropriate values for the threshold, as it may vary a lot for different architectures and even different layers. To overcome this issue, Aji and Heafield [23] instead fix the sparsity *rate* and only communicate the fraction p entries with the biggest magnitude of each gradient while also collecting all other gradients in a residual. At a sparsity rate of $p = 0.001$, their method only slightly degrades the convergence speed and final accuracy of the trained model. Lin *et al.* [25] present minor modifications to the work of Aji and Heafield [23] that even close this small

performance gap. Sparsification methods have been proposed primarily with the intention to speed up parallel training in the data center. Their convergence properties in the much more challenging federated learning environments have not yet been investigated. Sparsification methods (in their existing form) primarily compress the upstream communication, as the sparsity patterns on the updates from different clients will generally differ. If the number of participating clients is greater than the inverse sparsity rate, which can easily be the case in federated learning, the downstream update will not even be compressed at all.

3) **Dense quantization** methods reduce the entropy of the weight updates by restricting all updates to a reduced set of values. Bernstein *et al.* [22] propose signSGD, a compression method with theoretical convergence guarantees on i.i.d. data that quantizes every gradient update to its binary sign, thus reducing the bit size per update by a factor of $\times 32$. signSGD also incorporates download compression by aggregating the binary updates from all clients by means of a majority vote. Other authors propose to stochastically quantize the gradients during upload in an unbiased way (TernGrad [19], quantized stochastic gradient descent (QSGD) [20], ATOMO [21]). These methods are theoretically appealing, as they inherit the convergence properties of regular SGD under relatively mild assumptions. However, their empirical performance and compression rates do not match those of sparsification methods.

Out of all the above-listed methods, only *federated averaging* and *signSGD* compress both the upstream and downstream communications. All other methods are of limited utility in the federated learning setting defined in Section II, as they leave the communication from the server to the clients uncompressed.

Notation: In the following, calligraphic $\mathcal{W}$ will refer to the entirety of parameters of a neural network, while regular uppercase W refers to one specific tensor of parameters within $\mathcal{W}$ and lowercase w refers to one single scalar parameter of the network. Arithmetic operations between the neural network parameters are to be understood elementwise.

V. Limitations of Existing Compression Methods

The related work on efficient distributed deep learning almost exclusively considers i.i.d. data distributions among the clients, i.e., they assume unbiasedness of the local gradients with respect to the full-batch gradient according to

$$\mathbb{E}_{x\sim p_i}[\nabla_{\mathcal{W}} l(x, \mathcal{W})] = \nabla_{\mathcal{W}} R(\mathcal{W}) \quad \forall i = 1, .., n \tag{2}$$

where p_i is the distribution of data on the ith client and $R(\mathcal{W})$ is the empirical risk function over the combined training data.

While this assumption is reasonable for parallel training where the distribution of data among the clients is chosen by the practitioner, it is typically not valid in the federated learning setting where we can generally only hope for unbiasedness in the mean

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{x^i\sim p_i}[\nabla_{\mathcal{W}} l(x^i, \mathcal{W})] = \nabla_{\mathcal{W}} R(\mathcal{W}) \tag{3}$$

while the individual client's gradients will be biased toward the local data set according to

$$\mathbb{E}_{x\sim p_i}[\nabla_{\mathcal{W}} l(x, \mathcal{W})] = \nabla_{\mathcal{W}} R_i(\mathcal{W}) \neq \nabla_{\mathcal{W}} R(\mathcal{W}) \quad \forall i = 1, .., n. \tag{4}$$

As it violates assumption (2), a non-i.i.d. distribution of the local data renders existing convergence guarantees, as formulated in [19]–[21] and [29], inapplicability and has dramatic effects on the practical performance of communication-efficient distributed training algorithms as we will demonstrate in the following experiments.

A. Preliminary Experiments

We run preliminary experiments with a simplified version of the well-studied 11-layer VGG11 network [28], which we train on the CIFAR-10 [30] data set in a federated learning setup using ten clients. For the i.i.d. setting, we split the training data randomly into equally sized shards and assign one shard to every one of the clients. For the "non-i.i.d. (m)" setting, we assign every client samples from exactly m classes of the data set. The data splits are nonoverlapping and balanced, such that every client ends up with the same number of data points. The detailed procedure that generates the split of data is described in Section B of the Appendix in the Supplementary Material. We also perform experiments with a simple logistic regression classifier, which we train on the MNIST data set [31] under the same setup of the federated learning environment. Both models are trained using momentum SGD. To make the results comparable, all compression methods use the same learning rate and batch size.

B. Results

Fig. 2 shows the convergence speed in terms of gradient evaluations for the two models when trained using different methods for communication-efficient federated learning. We observe that while all compression methods achieve comparably fast convergence in terms of gradient evaluations on i.i.d. data, closely matching the uncompressed baseline (black line), they suffer considerably in the non-i.i.d. training settings. As this trend can be observed also for the logistic regression model, we can conclude that the underlying phenomenon is not unique to deep neural networks and also carries over to convex objectives. We will now analyze these results in detail for the different compression methods.

1) Federated Averaging: Most noticeably, federated averaging [10] (see orange line in Fig. 2), although specifically proposed for the federated learning setting, suffers considerably from non-i.i.d. data. This observation is consistent with Zhao *et al.* [32] who demonstrated that model accuracy can drop by up to 55% in non-i.i.d. learning environments

[0]We denote by VGG11* a simplified version of the original VGG11 architecture described in [28], where all dropout and batch normalization layers are removed and the number of convolutional filters and size of all fully connected layers is reduced by a factor of 2.

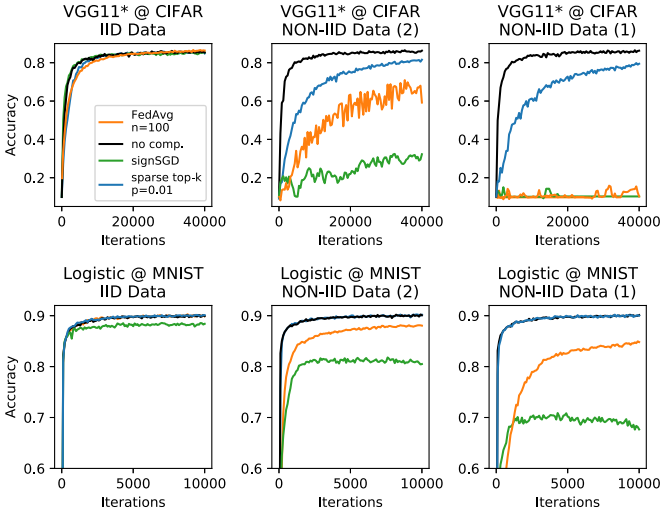

Fig. 2. Convergence speed when using different compression methods during the training of VGG11*[2] on CIFAR-10 and logistic regression on MNIST and Fashion-MNIST in a distributed setting with ten clients for i.i.d. and non-i.i.d. data. In the non-i.i.d. cases, every client only holds examples from exactly two respectively one of the ten classes in the data set. All compression methods suffer from degraded convergence speed in the non-i.i.d. situation, but sparse top-k is affected by far the least.

compared to the i.i.d. ones. They attribute the loss in accuracy to the increased weight divergence between the clients and propose to side-step the problem by assigning a shared public i.i.d. data set to all clients. While this approach can indeed create more accurate models, it also has multiple shortcomings, the most crucial one being that we generally cannot assume the availability of such a public data set. If a public data set were to exist, one could use it to pretrain a model at the server, which is not consistent with the assumptions typically made in federated learning. Furthermore, if all clients share (part of) the same public data set, overfitting to this shared data can become a serious issue. This effect will be particularly severe in highly distributed settings where the number of data points on every client is small. Finally, even when sharing a relatively large data set between the clients, the original accuracy achieved in the i.i.d. situation cannot be fully restored. For these reasons, we believe that the data-sharing strategy proposed by [32] is an insufficient workaround to the fundamental problem of federated averaging having convergence issues on non-i.i.d. data.

2) SignSGD: The quantization method signSGD [29] (see green line in Fig. 2) suffers from even worse stability issues in the non-i.i.d. learning environment. The method completely fails to converge on the CIFAR benchmark, and even for the convex logistic regression objective, the training plateaus at a substantially degraded accuracy.

To understand the reasons for these convergence issues, we have to investigate how likely it is for a single batch gradient to have the "correct" sign. Let

$$g_w^k = \frac{1}{k}\sum_{i=1}^{k} \nabla_w l(x_i, \mathcal{W}) \tag{5}$$

be the batch gradient over a specific minibatch of data $D^k = \{x_1, \ldots, x_k\} \subset D$ of size k at parameter w. Let, further, g_w be the gradient over the entire training data D. Then, we can define this probability by

$$\alpha_w(k) = \mathbb{P}\big[\text{sign}(g_w^k) = \text{sign}(g_w)\big]. \tag{6}$$

We can also compute the mean statistic

$$\alpha(k) = \frac{1}{|\mathcal{W}|}\sum_{w\in\mathcal{W}} \alpha_w(k) \tag{7}$$

to estimate the average congruence over all parameters of the network.

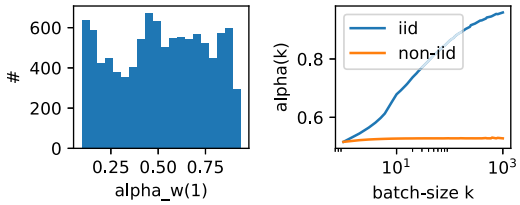

Fig. 3. Left: distribution of values for $\alpha_w(1)$ for the weight layer of logistic regression over the MNIST data set. Right: development of $\alpha(k)$ for increasing batch sizes. In the i.i.d. case, the batches are sampled randomly from the training data, while in the non-i.i.d. case, every batch contains samples from only exactly one class. For i.i.d. batches, the gradient sign becomes increasingly accurate with growing batch sizes. For non-i.i.d. batches of data, this is not the case. The gradient signs remain highly incongruent with the full-batch gradient, no matter how large the size of the batch.

Fig. 3 (left) exemplary shows the distribution of values for $\alpha_w(1)$ within the weights of logistic regression on MNIST at the beginning of training. As we can see, at a batch size of 1, g_w^1 is a very bad predictor of the true gradient sign with a very high variance and an average congruence of $\alpha(1) = 0.51$ just slightly higher than random. The sensitivity of signSGD to non-i.i.d. data becomes apparent once we inspect the development of the gradient sign congruence for increasing batch sizes. Fig. 3 (right) shows this development for batches of increasing size sampled from an i.i.d. and non-i.i.d. distribution. For the latter one, every sampled batch only contains data from exactly one class. As we can see, for i.i.d. data, α quickly grows with increasing batch size, resulting in increasingly accurate updates. For non-i.i.d. data, however, the congruence stays low, independent of the size of the batch. This means that if clients hold highly non-i.i.d. subsets of data, signSGD updates will only weakly correlate with the direction of steepest descent, no matter how large of a batch size is chosen for training.

3) Top-k Sparsification: Out of all existing compression methods, top-k sparsification (see blue line in Fig. 2) suffers least from non-i.i.d. data. For VGG11 on CIFAR the training still converges reliably even if every client only holds data from exactly one class, and for the logistic regression classifier trained on MNIST, the convergence does not slow down at all. We hypothesize that this robustness to non-i.i.d. data is due to mainly two reasons. First of all, the frequent communication of weight updates between the clients prevents them from diverging too far from one another, and hence, top-k sparsification does not suffer from weight divergence [32] as it is the case for federated averaging. Second, sparsification does not destabilize the training nearly as much as signSGD does

since the noise in the stochastic gradients is not amplified by quantization. Although top-k sparsification shows promising performance on non-i.i.d. data, its utility is limited in the federated learning setting as it only directly compresses the upstream communication.

Table I summarizes our findings. None of the existing compression methods supports both download compression and properly works with non-i.i.d. data.

VI. Sparse Ternary Compression

Top-k sparsification shows the most promising performance in distributed learning environments with non-i.i.d. client data. We will use this observation as a starting point to construct an efficient communication protocol for federated learning. To arrive at this protocol, we will solve three open problems that prevent the direct application of top-k sparsification to federated learning.

1) We will further increase the efficiency of our method by employing quantization and optimal lossless coding of the weight updates.
2) We will incorporate downstream compression into the method to allow for efficient communication from server to clients.
3) We will implement a caching mechanism to keep the clients synchronized in case of partial client participation.

A. Ternarizing Weight Updates

Regular top-k sparsification, as proposed in [23] and [25], communicates the fraction of largest elements at full precision, while all other elements are not communicated at all. In our previous work (Sattler *et al.* [17]), we already demonstrated that this imbalance in update precision is wasteful in the distributed training setting and that higher compression gains can be achieved when sparsification is combined with quantization of the nonzero elements.

We adopt the method described in [17] to the federated learning setting and quantize the remaining top-k elements of the sparsified updates to the mean population magnitude, leaving us with a ternary tensor containing values $\{-\mu, 0, \mu\}$. The quantization method is formalized in Algorithm 1.

Algorithm 1 STC

input: flattened tensor $T \in \mathbb{R}^n$, sparsity p
output: sparse ternary tensor $T^* \in \{-\mu, 0, \mu\}^n$
$k \leftarrow \max(np, 1)$
$v \leftarrow \text{top}_k(|T|)$
$\text{mask} \leftarrow (|T| \geq v) \in \{0, 1\}^n$
$T^{masked} \leftarrow \text{mask} \odot T$
$\mu \leftarrow \frac{1}{k} \sum_{i=1}^{n} |T_i^{masked}|$
return $T^* \leftarrow \mu \times \text{sign}(T^{masked})$

This ternarization step reduces the entropy of the update from

$$H_{\text{sparse}} = -p \log_2(p) - (1-p) \log_2(p) + 32p \tag{8}$$

to

$$H_{\text{STC}} = -p \log_2(p) - (1-p) \log_2(p) + p \tag{9}$$

when compared to the regular sparsification. At a sparsity rate of $p = 0.01$, the additional compression achieved by ternarization is $H_{\text{sparse}}/H_{\text{STC}} = 4.414$. In order to achieve the same compression gains by pure sparsification, one would have to increase the sparsity rate by approximately the same factor.

Using a theoretical framework developed by Stich *et al.* [33], we can prove the convergence of STC under standard assumptions on the loss function. The proof relies on bounding the impact of the perturbation caused by the compression operator. This is formalized in the following definition.

Definition 1 (k-Contraction) [33]: For a parameter $0 < k \leq d$, a k-contraction is an operator $\text{comp} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ that satisfies the contraction property

$$\mathbb{E}\|x - \text{comp}(x)\|^2 \leq \left(1 - \frac{k}{d}\right) \|x\|^2 \quad \forall x \in \mathbb{R}^d. \tag{10}$$

We can show that STC indeed is a k-contraction.

Lemma 2: STC_k as defined in Algorithm 1 is a $\tilde{k}$-contraction, with

$$0 < \tilde{k} = \frac{\|\text{top}_k(x)\|_1^2}{k\|x\|_2^2} d \leq d. \tag{11}$$

The proof can be found in Appendix E in the Supplementary Material. It then directly follows from [33, Th. 2.4] that for any L-smooth, μ-strongly convex objective function f with bounded gradients $\mathbb{E}\|\Delta \mathcal{W}\|^2 \leq G^2$, the update rule

$$\mathcal{W}^{(t+1)} := \mathcal{W}^{(t)} - \text{STC}_k\left(\mathcal{A}^{(t)} + \eta \Delta \mathcal{W}_{i_t}^{(t)}\right) \tag{12}$$

$$\mathcal{A}^{(t+1)} := \mathcal{A}^{(t)} + \Delta \mathcal{W}_{i_t}^{(t+1)} - \text{STC}_k\left(\Delta \mathcal{W}_{i_t}{}^{(t+1)}\right) \tag{13}$$

converges according to

$$\mathbb{E}[f(\overline{\mathcal{W}_T})] - f^* \leq \mathcal{O}\left(\frac{G^2}{\mu T}\right) + \mathcal{O}\left(\frac{\frac{d^2}{\tilde{k}^2} G^2 \frac{L}{\mu}}{\mu T^2}\right) + \mathcal{O}\left(\frac{\frac{d^3}{\tilde{k}^3} G^2}{\mu T^3}\right). \tag{14}$$

This means that for $T \in \mathcal{O}((d/\tilde{k})((L/\mu))^{1/2})$, STC converges at rate $\mathcal{O}((G^2/\mu T))$, which is the same as for regular SGD!

Preliminary experiments are in line with our theoretical findings. Fig. 4 shows the final accuracy of the VGG11* model when trained at different sparsity levels with and without ternarization. As we can see, additional ternarization does only have a negligible effect on the convergence speed and sometimes does even increase the final accuracy of the trained model. It seems evident that a combination of sparsity and quantization makes more efficient use of the communication budged than pure sparsification.

B. Extending to Downstream Compression

Existing compression frameworks that were proposed for distributed training (see [19], [20], [23], [25]) only compress the communication from clients to the server, which is sufficient for applications where aggregation can be achieved via

an all-reduce operation. However, in the federated learning setting, where the clients have to download the aggregated weight-updates from the server, this approach is not feasible, as it will lead to a communication bottleneck.

To illustrate this point, let $\mathrm{STC}_k : \mathbb{R}^n \to \mathbb{R}^n$, $\Delta\mathcal{W} \mapsto \Delta\tilde{\mathcal{W}}$ be the compression operator that maps a (flattened) weight update $\Delta\mathcal{W}$ to a sparsified and ternarized weight update $\Delta\tilde{\mathcal{W}}$ according to Algorithm 1. For local weight updates $\Delta\mathcal{W}_i^{(t)}$, the update rule for STC can then be written as

$$\Delta\mathcal{W}^{(t+1)} = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\mathrm{STC}_k\left(\Delta\mathcal{W}_i^{(t+1)} + A_i^{(t)}\right)}_{\Delta\tilde{\mathcal{W}}_i^{(t+1)}} \tag{15}$$

$$A_i^{(t+1)} = A_i^{(t)} + \Delta\mathcal{W}_i^{(t+1)} - \Delta\tilde{\mathcal{W}}_i^{(t+1)} \tag{16}$$

starting with an empty residual $A_i^{(0)} = 0 \in \mathbb{R}^n$ on all clients. While the updates $\Delta\tilde{\mathcal{W}}_i^{(t+1)}$ that are sent from clients to the server are always sparse, the number of nonzero elements in the update $\Delta\mathcal{W}^{(t+1)}$ that is sent downstream grows linearly with the amount of participating clients in the worst case. If the participation rate exceeds the inverse sparsity $1/p$, the update $\Delta\mathcal{W}^{(t+1)}$ essentially becomes dense.

To resolve this issue, we propose to apply the same compression mechanism that is used on the clients *also at the server side* to compress the downstream communication. This modifies the update rule to

$$\Delta\tilde{\mathcal{W}}^{(t+1)} = \mathrm{STC}_k\left(\frac{1}{n}\sum_{i=1}^{n} \underbrace{\mathrm{STC}_k\left(\Delta\mathcal{W}_i^{(t+1)} + A_i^{(t)}\right)}_{\Delta\tilde{\mathcal{W}}_i^{(t+1)}} + A^{(t)}\right) \tag{17}$$

with a client-side and a server-side residual updates

$$A_i^{(t+1)} = A_i^{(t)} + \Delta\mathcal{W}_i^{(t+1)} - \Delta\tilde{\mathcal{W}}_i^{(t+1)} \tag{18}$$

$$A^{(t+1)} = A^{(t)} + \Delta\mathcal{W}^{(t+1)} - \Delta\tilde{\mathcal{W}}^{(t+1)}. \tag{19}$$

We can express this new update rule for both upload and download compression (17) as a special case of pure upload compression (15) with generalized filter masks. Let M_i, $i = 1, .., n$ be the sparsifying filter masks used by the respective clients during the upload and M be the one used during the download by the server. Then, we could arrive at the same sparse update $\Delta\tilde{\mathcal{W}}^{(t+1)}$ if all clients use filter masks $\tilde{M}_i = M_i \odot M$, where $\odot$ is the Hadamard product. We, thus, predict that training models using this new update rule should behave similar to regular upstream-only sparsification but with a slightly increased sparsity rate. We experimentally verify this prediction:

Fig. 5 shows the accuracies achieved by VGG11 on CIFAR10, when trained in a federated learning environment with five clients for 10 000 iterations at different rates of upload and download compression. As we can see, for as long as download and upload sparsity are of the same order, sparsifying the download is not very harmful to the convergence and decreases the accuracy by at most 2% in both the i.i.d. and the non-i.i.d. case.

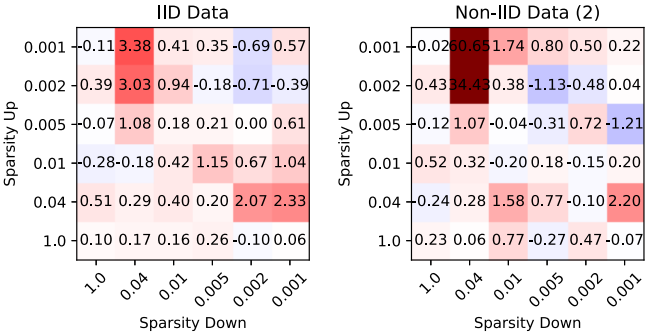

Fig. 4. Effects of ternarization at different levels of upload and download sparsities. Displayed is the difference in final accuracy in % between a model trained with sparse updates and a model trained with sparse binarized updates. Positive numbers indicate better performance of the model trained with pure sparsity. VGG11 trained on CIFAR10 for 16 000 iterations with five clients holding i.i.d. and non-i.i.d. data.

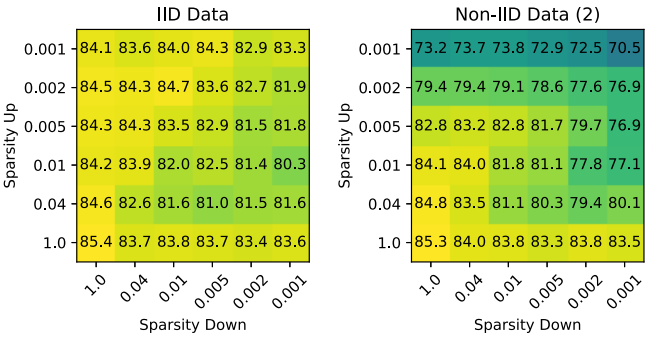

Fig. 5. Accuracy achieved by VGG11* when trained on CIFAR in a distributed setting with five clients for 16 000 iterations at different levels of upload and download sparsity. Sparsifying the updates for downstream communication reduces the final accuracy by at most 3% when compared to using only upload sparsity.

C. Weight Update Caching for Partial Client Participation

This far we have only been looking at scenarios in which all of the clients participate throughout the entire training process. However, as elaborated in Section II, in federated learning, typically only a fraction of the entire client population will participate in any particular communication round. As clients do not download the full model $\mathcal{W}^{(t)}$, but only compressed model updates $\Delta\tilde{\mathcal{W}}^{(t)}$; this introduces new challenges when it comes to keeping all clients synchronized.

To solve the synchronization problem and reduce the workload for the clients, we propose to use a caching mechanism on the server. Assume that the last τ communication rounds have produced the updates $\{\Delta\tilde{\mathcal{W}}^{(t)} | t = T-1, \ldots, T-\tau\}$. The server can cache all partial sums of these updates up until a certain point $\{P^{(s)} = \sum_{t=1}^{s} \Delta\tilde{\mathcal{W}}^{(T-t)} | s = 1, .., \tau\}$ together with the global model $\mathcal{W}^{(T)} = \mathcal{W}^{(T-\tau-1)} + \sum_{t=1}^{\tau} \Delta\tilde{\mathcal{W}}^{(T-t)}$. Every client that wants to participate in the next communication round then has to first synchronize itself with the server by either downloading $P^{(s)}$ or $\mathcal{W}^{(T)}$, depending on how many previous communication rounds it has skipped. For general sparse updates, the bound on the entropy

$$H(P^{(\tau)}) \leq \tau H(P^{(1)}) = \tau H(\Delta\tilde{\mathcal{W}}^{(T-1)}) \tag{20}$$

can be attained. This means that the size of the download will grow linearly with the number of rounds a client has skipped training. The average number of skipped rounds is equal to the inverse participation fraction $1/\eta$. This is usually tolerable as the downlink typically is cheaper and has far higher bandwidth than the uplink, as already noted in [10] and [19]. Essentially, all compression methods that communicate only parameter updates instead of full models suffer from this same problem. This is also the case for signSGD although here the size of the downstream update only grows logarithmically with the delay period according to

$$H(P^{(\tau)}_{signSGD}) \leq \log_2(2\tau + 1). \tag{21}$$

Partial client participation also has effects on the convergence speed of federated training, both with delayed and sparsified updates. We will investigate these effects in detail in Section VII-C.

D. Lossless Encoding

To communicate a set of sparse ternary tensors produced by STC, we only need to transfer the positions of the nonzero elements in the flattened tensors, along with one bit per nonzero update to indicate the mean sign μ or $-\mu$. Instead of communicating the absolute positions of the nonzero elements, it is favorable to communicate the distances between them. Assuming a random sparsity pattern we know that for big values of $|W|$ and $k = p|W|$, the distances are approximately geometrically distributed with success probability equal to the sparsity rate p. Therefore, we can optimally encode the distances using the Golomb code [34]. The Golomb encoding reduces the average number of position bits to

$$\bar{b}_{\text{pos}} = \mathbf{b}^* + \frac{1}{1 - (1-p)^{2^{\mathbf{b}^*}}} \tag{22}$$

with $\mathbf{b}^* = 1 + \lfloor \log_2((\log(\phi - 1)/\log(1-p))) \rfloor$ and $\phi = (\sqrt{5} + 1/2)$ being the golden ratio. For a sparsity rate of e.g., $p = 0.01$, we get $\bar{b}_{\text{pos}} = 8.38$, which translates to $\times 1.9$ compression, compared to a naive distance encoding with 16 fixed bits. Both the encoding and the decoding scheme can be found in Section A of the Appendix (Algorithms A1 and A2) in the Supplementary Material. The updates are encoded both before upload and before download.

The complete compression framework that features upstream and downstream compression via sparsification, ternarization, and optimal encoding of the updates is described in Algorithm 2.

Algorithm 2 Efficient Federated Learning With Parameter Server Via STC

input: initial parameters $\mathcal{W}$
output: improved parameters $\mathcal{W}$
init: all clients C_i, $i = 1, .., [\text{Number of Clients}]$ are initialized with the same parameters $\mathcal{W}_i \leftarrow \mathcal{W}$. Every Client holds a different data set D_i, with $|\{y : (x, y) \in D_i\}| = [\text{Classes per Client}]$ of size $|D_i| = \varphi_i |\cup_j D_j|$. The residuals are initialized to zero $\Delta\mathcal{W}, \mathcal{R}_i, \mathcal{R} \leftarrow 0$.
for $t = 1, .., T$ **do**
  for $i \in I_t \subseteq \{1, .., [\text{Number of Clients}]\}$ ***in parallel*** **do**
    Client C_i does:
    · $\text{msg} \leftarrow \text{download}_{S \rightarrow C_i}(\text{msg})$
    · $\Delta\mathcal{W} \leftarrow \text{decode}(\text{msg})$
    · $\mathcal{W}_i \leftarrow \mathcal{W}_i + \Delta\mathcal{W}$
    · $\Delta\mathcal{W}_i \leftarrow \mathcal{R}_i + \text{SGD}(\mathcal{W}_i, D_i, b) - \mathcal{W}_i$
    · $\Delta\tilde{\mathcal{W}}_i \leftarrow \text{STC}_{p_{up}}(\Delta\mathcal{W}_i)$
    · $\mathcal{R}_i \leftarrow \Delta\mathcal{W}_i - \Delta\tilde{\mathcal{W}}_i$
    · $\text{msg}_i \leftarrow \text{encode}(\Delta\tilde{\mathcal{W}}_i)$
    · $\text{upload}_{C_i \rightarrow S}(\text{msg}_i)$
  end
  Server S does:
  · $\text{gather}_{C_i \rightarrow S}(\Delta\tilde{\mathcal{W}}_i),\ i \in I_t$
  · $\Delta\mathcal{W} \leftarrow \mathcal{R} + \frac{1}{|I_t|}\sum_{i \in I_t} \Delta\tilde{\mathcal{W}}_i$
  · $\Delta\tilde{\mathcal{W}} \leftarrow \text{STC}_{p_{down}}(\Delta\mathcal{W})$
  · $\mathcal{R} \leftarrow \Delta\mathcal{W} - \Delta\tilde{\mathcal{W}}$
  · $\mathcal{W} \leftarrow \mathcal{W} + \Delta\tilde{\mathcal{W}}$
  · $\text{msg} \leftarrow \text{encode}(\Delta\tilde{\mathcal{W}})$
  · $\text{broadcast}_{S \rightarrow C_i}(\text{msg}),\ i = 1, .., M$
end
return $\mathcal{W}$

VII. Experiments

We evaluate our proposed communication protocol on four different learning tasks and compare its performance to federated averaging and signSGD in a wide a variety of different federated learning environments.

Models and Data Sets: To cover a broad spectrum of learning problems, we evaluate on differently sized convolutional and recurrent neural networks for the relevant federated learning tasks of image classification and speech recognition:

VGG11 on CIFAR*: We train a modified version of the popular 11-layer VGG11 network [28] on the CIFAR [30] data set. We simplify the VGG11 architecture by reducing the number of convolutional filters to [32, 64, 128, 128, 128, 128, 128, 128] in the respective convolutional layers and reducing the size of the hidden fully-connected layers to 128. We also remove all dropout layers and batch-normalization layers as the regularization is no longer required. Batch normalization has been observed to perform very poorly with both small batch sizes and non-i.i.d. data [35], and we do not want this effect to obscure the investigated behavior. The resulting VGG11* network still achieves 85.46% accuracy on the validation set after 20 000 iterations of training with a constant learning rate of 0.16 and contains 865 482 parameters.

CNN on KWS: We train the four-layer convolutional neural network (CNN) from [27] on the speech commands data set [36]. The speech commands data set consists of 51 088 different speech samples of specific keywords. There are 30 different keywords in total, and every speech sample is of 1-s duration. Like [32], we restrict us to the subset of the ten most common keywords. For every speech command, we extract the Mel spectrogram from the short-time Fourier transform,

TABLE II

MODELS AND HYPERPARAMETERS. THE LEARNING RATE IS KEPT CONSTANT THROUGHOUT TRAINING

Task	VGG11* @ CIFAR-10	CNN @ KWS	LSTM@ Fashion-MNIST	Logistic @ MNIST
Iterations	20000	10000	20000	5000
Learning Rate	0.016	0.1	0.1	0.04
Momentum	0.9	0.0	0.9	0.0
Base Accuracy	85.46%	91.23%	90.21%	92.31%
Parameters	865482	876938	216330	7850

which results in a 32×32 feature map. The CNN architecture achieves 89.12% accuracy after 10 000 training iterations and has 876 938 parameters in total.

LSTM on Fashion-MNIST: We also train an Long Short-Term Memory (LSTM) network with two hidden layers of size 128 on the Fashion-MNIST data set [37]. The Fashion-MNIST data set contains 60 000 train and 10 000 validation greyscale images of ten different fashion items. Every 28×28 image is treated as a sequence of 28 features of dimensionality 28 and fed as such in the many-to-one LSTM network. After 20 000 training iterations with a learning rate of 0.04, the LSTM model achieves 90.21% accuracy on the validation set. The model contains 216 330 parameters.

Logistic Regression on MNIST: Finally, we also train a simple logistic regression classifier on the MNIST [31] data set. The MNIST data set contains 60 000 training and 10 000 test greyscale images of handwritten digits of size 28×28. The trained logistic regression classifier achieves 92.31% accuracy on the test set and contains 7850 parameters.

The different learning tasks are summarized in Table II. In the following, we will primarily discuss the results for VGG11* trained on CIFAR; however, the described phenomena carry over to all other benchmarks and the supporting experimental results can be found in the Appendix in the Supplementary Material.

Compression Methods: We compare our proposed STC method at a sparsity rate of $p = 1/400$ with federated averaging at an "equivalent" delay period of $n = 400$ iterations and signSGD with a coordinatewise step size of $\delta = 0.0002$. At a sparsity rate of $p = 1/400$, STC compresses updates both during upload and download by roughly a factor of $\times 1050$. A delay period of $n = 400$ iterations for federated averaging results in a slightly smaller compression rate of $\times 400$. Further analysis on the effects of the sparsity rate p and delay period n on the convergence speed of STC and federated averaging can be found in Section C of the Appendix in the Supplementary Material. During our experiments, we keep all training related hyperparameters constant for the different compression methods. To be able to compare the different methods in a fair way, all methods are given the same budged of training iterations in the following experiments (one communication round of federated averaging uses up n iterations, where n is the number of local iterations).

Learning Environment: The federated learning environment described in Algorithm 2 can be fully characterized by five parameters. For the base configuration, we set the number of clients to 100, the participation ratio to 10%, and the local batch size to 20 and assign every client an equally sized subset of the training data containing samples from ten different classes. In the following experiments, if not explicitly signified otherwise, all hyperparameters will default to this base configuration summarized in Table III. We will use the short notations "Clients: $\eta N/N$" and "Classes: c" to refer to a setup of the federated learning environment in which a random subset of ηN out of a total of N clients participates in every communication round and every client is holding data from exactly c different classes.

TABLE III

BASE CONFIGURATION OF THE FEDERATED LEARNING ENVIRONMENT IN OUR EXPERIMENTS

Parameter	Number of Clients	Participation per Round	Classes per Client	Batch-Size	Balanced-ness
Value	$N = 100$	$\eta = 0.1$	$c = 10$	$b = 20$	$\gamma = 1.0$

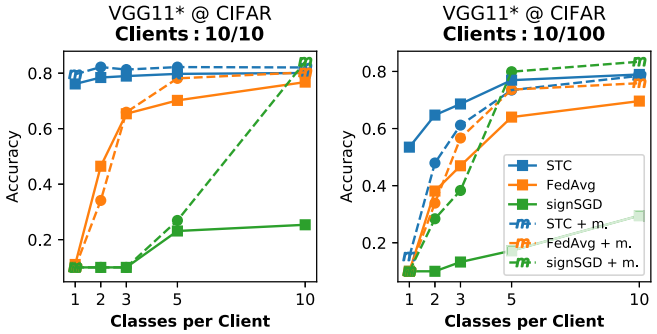

Fig. 6. Robustness of different compression methods to the non-i.i.d.-ness of client data on four different benchmarks. VGG11* trained on CIFAR. STC distinctively outperforms federated averaging on non-i.i.d. data. The learning environment is configured as described in Table III. Dashed lines signify that a momentum of $m = 0.9$ was used.

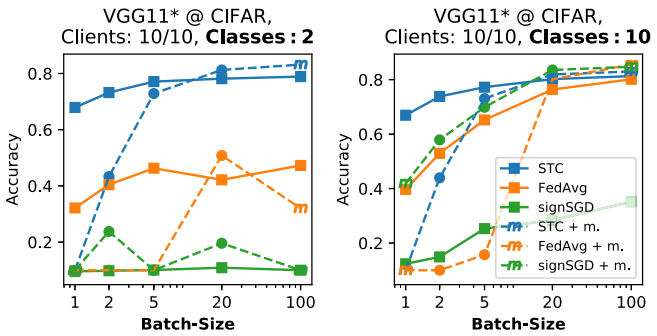

Fig. 7. Maximum accuracy achieved by the different compression methods when training VGG11* on CIFAR for 20 000 iterations at varying batch sizes in a federated learning environment with ten clients and full participation. Left: Every client holds data from exactly two different classes. Right: Every client holds an i.i.d. subset of data.

A. Momentum in Federated Optimization

We start out by investigating the effects of momentum optimization on the convergence behavior of the different compression methods. Figs. 6–9 show the final accuracy achieved by federated averaging ($n = 400$), STC ($p = 1/400$), and signSGD after 20 000 training iterations in a variety of different federated learning environments. In Figs. 6–9, dashed

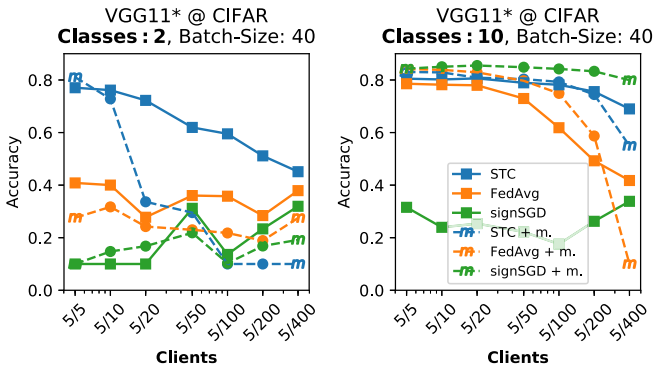

Fig. 8. Validation accuracy achieved by VGG11* on CIFAR after 20 000 iterations of communication-efficient federated training with different compression methods. The relative client participation fraction is varied between 100% (5/5) and 5% (5/100). Left: Every client holds data from exactly two different classes. Right: Every client holds an i.i.d. subset of data.

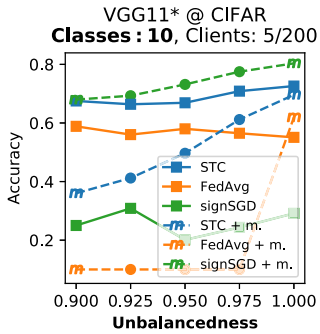

Fig. 9. Validation accuracy achieved by VGG11* on CIFAR after 20 000 iterations of communication-efficient federated training with different compression methods. The training data are split among the client at different degrees of unbalancedness with γ varying between 0.9 and 1.0.

lines refer to experiments where the momentum of $m = 0.9$ was used during training, while solid lines signify that classical SGD was used. As we can see, momentum has a significant influence on the convergence behavior of the different methods. While signSGD always performs distinctively better if momentum is turned on during the optimization, the picture is less clear for STC and federated averaging. We can make out three different parameters of the learning environment that determine whether momentum is beneficial or harmful to the performance of STC. If the participation rate is high and the batch size used during training is sufficiently large (see Fig. 7 left), momentum improves the performance of STC. Conversely, momentum will deteriorate the training performance in situations where training is carried out on small batches and with low client participation. The latter effect is increasingly strong if clients hold non-i.i.d. subsets of data [see Fig. 6 (right)]. These results are not surprising, as the issues with stale momentum described in [25] are enhanced in these situations. Similar relationships can be observed for federated averaging where again the size (see Fig. 7) and the heterogeneity (see Fig. 6) of the local minibatches determine whether the momentum will have a positive effect on the training performance or not.

When we compare federated averaging, signSGD and STC in the following, we will ignore whichever version of these methods (momentum "on" or "off") performs worse.

B. Non-i.i.d.-ness of the Data

Our preliminary experiments in Section V have already demonstrated that the convergence behavior of both federated averaging and signSGD is very sensitive to the degree of i.i.d.-ness of the local client data, whereas sparse communication seems to be more robust. We will now investigate this behavior in some more detail. Fig. 6 shows the maximum achieved generalization accuracy after a fixed number of iterations for VGG11* trained on CIFAR at different levels of non-i.i.d.-ness. Additional results on all other benchmarks can be found in Fig. A2 in the Appendix in the Supplementary Material. Both at full (left plot) and partial (right plot) client participations, STC outperforms federated averaging across all levels of i.i.d.-ness. The most distinct difference can be observed in the non-i.i.d. regime, where the individual clients hold less than five different classes. Here, STC (without momentum) outperforms both federated averaging and signSGD by a wide margin. In the extreme case where every client only holds data from exactly one class, STC still achieves 79.5% and 53.2% accuracy at full and partial client participations, respectively, while both federated averaging and signSGD fail to converge at all.

C. Robustness to Other Parameters of the Learning Environment

We will now proceed to investigate the effects of other parameters of the learning environment on the convergence behavior of the different compression methods. Figs. 7–9 show the maximum achieved accuracy after training VGG11* on CIFAR for 20 000 iterations in different federated learning environments. Additional results on the three other benchmarks can be found in Section D in the Appendix in the Supplementary Material.

We observe that STC (without momentum) consistently dominates federated averaging on all benchmarks and learning environments.

1) Local Batch Size: The memory capacity of mobile and IoT devices is typically very limited. As the memory footprint of SGD is proportional to the batch size used during training, clients might be restricted to train on small minibatches only. Fig. 7 shows the influence of the local batch size on the performance of different communication-efficient federated learning techniques exemplary for VGG11* trained on CIFAR. First of all, we notice that using momentum significantly slows down the convergence speed of both STC and federated averaging at batch sizes smaller than 20 independent of the distribution of data among the clients. As we can see, even if the training data is distributed among the clients in an i.i.d. manner (see Fig. 7 right) and all clients participate in every training iteration, federated averaging suffers considerably from small batch sizes. STC, on the other hand, demonstrates to be far more robust to this type of constraint. At an extreme batch size of one, the model trained with STC still achieves an accuracy of 63.8%, while the federated averaging model only reaches 39.2% after 20 000 training iterations.

2) Client Participation Fraction: Fig. 8 shows the convergence speed of VGG11* trained on CIFAR10 in a federated

learning environment with different degrees of client participation. To isolate the effects of reduced participation, we keep the absolute number of participating clients and the local batch sizes at constant values of 5 and 40, respectively, throughout all experiments and vary only the total number of clients (and thus the relative participation η). As we can see, reducing the participation rate has negative effects on both federated averaging and STC. The causes for these negative effects, however, are different. In federated averaging, the participation rate is proportional to the effective amount of data that the training is conducted on in any individual communication round. If a nonrepresentative subset of clients is selected to participate in a particular communication round of federated averaging, this can steer the optimization process away from the minimum and might even cause catastrophic forgetting [38] of previously learned concepts. On the other hand, partial participation reduces the convergence speed of STC by causing the clients residuals to go out sync and increasing the gradient staleness [25]. The more rounds a client has to wait before it is selected to participate during training again, the more outdated its accumulated gradients become. We can observe this behavior for STC most strongly in the non-i.i.d. situation (see Fig. 8 left), where the accuracy steadily decreases with the participation rate. However, even in the extreme case where only 5 out of 400 clients participate in every round of training, STC still achieves higher accuracy than federated averaging and signSGD. If the clients hold i.i.d. data (see Fig. 8 right), STC suffers much less from a reduced participation rate than federated averaging. If only 5 out of 400 clients participate in every round, STC (without momentum) still manages to achieve an accuracy of 68.2% while federated averaging stagnates at 42.3% accuracy. signSGD is affected the least by reduced participation, which is unsurprising, as only the absolute number of participating clients would have a direct influence on its performance. Similar behavior can be observed on all other benchmarks, and the results can be found in Fig. A3 in the Appendix in the Supplementary Material. It is noteworthy that in federated learning, it is usually possible for the server to exercise some control over the rate of client participation. For instance, it is typically possible to increase the participation ratio at the cost of a long waiting time for all clients to finish.

3) Unbalancedness: Up until now, all experiments were performed with a balanced split of data in which every client was assigned the same amount of data points. In practice, however, the data sets on different clients will typically vary heavily in size. To simulate different degrees of unbalancedness, we split the data among the clients in a way such that the ith out of n clients is assigned a fraction

$$\varphi_i(\alpha, \gamma) = \frac{\alpha}{n} + (1-\alpha)\frac{\gamma^i}{\sum_{j=1}^{n} \gamma^j} \tag{23}$$

of the total data. The parameter α controls the minimum amount of data on every client, while the parameter γ controls the concentration of data. We fix $\alpha = 0.1$ and vary γ between 0.9 and 1.0 in our experiments. To amplify the effects of unbalanced client data, we also set the client participation to a low value of only 5 out of 200 clients. Fig. 9 shows the final accuracy achieved after 20 000 iterations for different values of γ. Interestingly, the unbalancedness of the data does not seem to have a significant effect on the performance of either of the compression methods. Even if the data are highly concentrated on a few clients (as is the case for $\gamma = 0.9$), all methods converge reliably, and for federated averaging, the accuracy even slightly goes down with increased balancedness. Apparently, the rare participation of large clients can balance out several communication rounds with much smaller clients. These results also carry over to all other benchmarks (see Fig. A5 in the Appendix in the Supplementary Material).

D. Communication Efficiency

Finally, we compare the different compression methods with respect to the number of iterations and communicated bits they require to achieve a certain target accuracy on a federated learning task. As we saw in Section V, both federated averaging and signSGD perform considerably worse if clients hold non-i.i.d. data or use small batch sizes. To still have a meaningful comparison, we, therefore, choose to evaluate this time on an i.i.d. environment where every client holds ten different classes and uses a moderate batch size of 20 during training. This setup favors federated averaging and signSGD to the maximum degree possible! All other parameters of the learning environment are set to the base configuration given in Table III. We train until the target accuracy is achieved or a maximum amount of iterations is exceeded and measure the amount of communicated bits both for upload and download. Fig. 10 shows the results for VGG11* trained on CIFAR, CNN trained on keyword spotting (KWS), and the LSTM model trained on Fashion-MNIST. We can see that even if all clients hold i.i.d. data, STC still manages to achieve the desired target accuracy within the smallest communication budget out of all methods. STC also converges faster in terms of training iterations than the versions of federated averaging with comparable compression rate. Unsurprisingly, we see that both for federated averaging and STC, we face a tradeoff between the number of training iterations ("computation") and the number of communicated bits ("communication"). On all investigated benchmarks, however, STC is Pareto-superior to federated averaging in the sense for any fixed iteration complexity, it achieves a lower (upload) communication complexity.

Table IV shows the amount of upstream and downstream communications required to achieve the target accuracy for the different methods in megabytes. On the CIFAR learning task, STC at a sparsity rate of $p = 0.0025$ only communicates 183.9 MB worth of data, which is a reduction in communication by a factor of $\times 199.5$ as compared to the baseline with requires 36696 MB and federated averaging ($n = 100$), which still requires 1606 MB. Federated averaging with a delay period of 1000 steps does not achieve the target accuracy within the given iteration budget.

VIII. Lessons Learned

We will now summarize the findings of this article and give general suggestions on how to approach communication-constrained federated learning problems (see our summarizing Fig. 11).

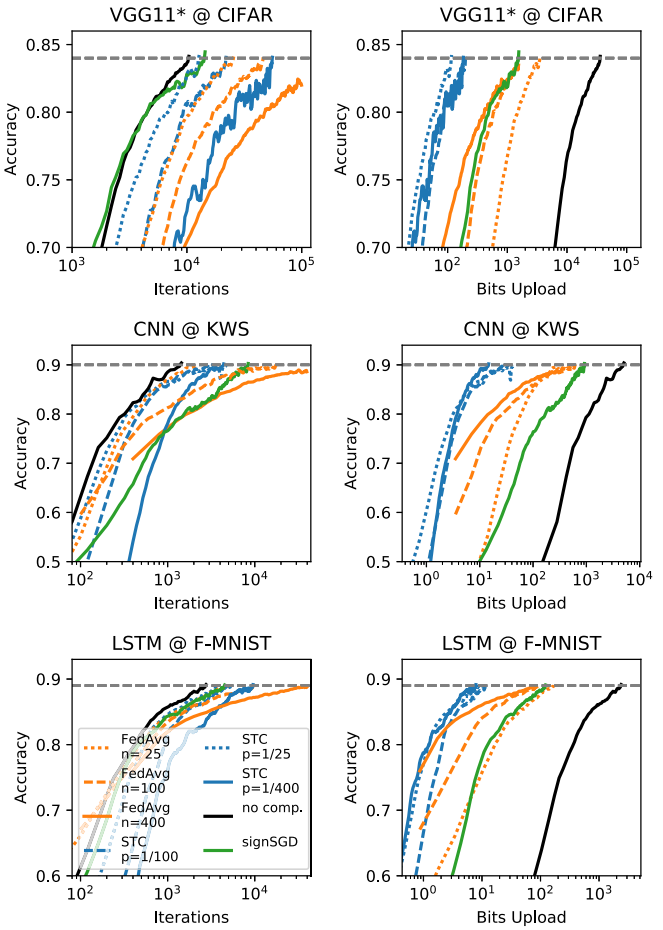

Fig. 10. Convergence speed of federated learning with compressed communication in terms of training iterations (left) and uploaded bits (right) on three different benchmarks (top to bottom) in an i.i.d. federated learning environment with 100 clients and 10% participation fraction. For better readability, the validation error curves are average-smoothed with a step size of five. On all benchmarks, STC requires the least amount of bits to converge to the target accuracy.

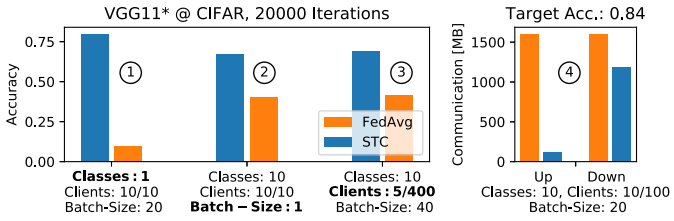

Fig. 11. Left: accuracy achieved by VGG11* on CIFAR after 20 000 iterations of federated training with federated averaging and STC for three different configurations of the learning environment. Right: upstream and downstream communication necessary to achieve a validation accuracy of 84% with federated averaging and STC on the CIFAR benchmark under i.i.d. data and a moderate batch-size.

1) If clients hold non-i.i.d. data, sparse communication protocols such as STC distinctively outperform federated averaging across all federated learning environments [see Figs. 6, 7 (left), and 8 (left)].
2) The same holds true if clients are forced to train on small minibatches (e.g., because the hardware is memory constrained). In these situations, STC outperforms federated averaging even if the client's data are i.i.d. [see Fig. 7 (right)].

TABLE IV

BITS REQUIRED FOR *Upload and/ Download* TO ACHIEVE A CERTAIN TARGET ACCURACY ON DIFFERENT LEARNING TASKS IN AN I.I.D. LEARNING ENVIRONMENT. A VALUE OF "n.a." IN THE TABLE SIGNIFIES THAT THE METHOD HAS NOT ACHIEVED THE TARGET ACCURACY WITHIN THE ITERATION BUDGET. THE LEARNING ENVIRONMENT IS CONFIGURED AS DESCRIBED IN TABLE III

Compression Method	VGG11*@CIFAR Acc. = 0.84	CNN@KWS Acc. = 0.9	LSTM@F-MNIST Acc. = 0.89
Baseline	36696 MB / 36696 MB	5191 MB / 5191 MB	2422 MB / 2422 MB
signSGD	1579.5 MB / 6937.6 MB	925.17 MB / 4063.6 MB	123.31 MB / 541.6 MB
FedAvg $n = 25$	3572.7 MB / 3572.7 MB	301.67 MB / 301.67 MB	174.79 MB / 174.79 MB
FedAvg $n = 100$	1606.3 MB / 1606.3 MB	617.3 MB / 617.3 MB	83.94 MB / 83.94 MB
FedAvg $n = 400$	n.a.	350.78 MB / 350.78 MB	86.53 MB / 86.53 MB
STC $p = 1/25$	**118.43 MB / 1184.3 MB**	43.57 MB / 435.7 MB	8.84 MB / 88.4 MB
STC $p = 1/100$	202.2 MB / 2022 MB	31.0 MB / 310 MB	12.1 MB / 121 MB
STC $p = 1/400$	183.9 MB / 1839 MB	**14.8 MB / 148 MB**	**7.9 MB / 79 MB**

3) STC should also be preferred over federated averaging if the client participation rate is expected to be low, as it converges more stable and quickly in both the i.i.d. and non-i.i.d. regime [see Fig. 8 (right)].
4) STC is generally most advantageous in situations where the communication is bandwidth-constrained or costly (metered network, limited battery), as it does achieve a certain target accuracy within the minimum amount of communicated bits even on i.i.d. data (see Fig. 10 and Table IV).
5) Federated averaging in return should be used if the communication is latency-constrained or if the client participation is expected to be very low (and 1–3 do not hold).
6) Momentum optimization should be avoided in federated learning whenever either clients are training with small batch sizes or the client data are non-i.i.d. and the participation rate is low (see Figs. 6–8).

IX. CONCLUSION

Federated learning for mobile and IoT applications is a challenging task, as generally little to no control can be exerted over the properties of the learning environment.

In this article, we demonstrated that the convergence behavior of current methods for communication-efficient federated learning is very sensitive to these properties. On a variety of different data sets and model architectures, we observe that the convergence speed of federated averaging drastically decreases in learning environments where the clients either hold non-i.i.d. subsets of data are forced to train on small minibatches or where only a small fraction of clients participates in every communication round. To address these issues, we propose STC, a communication protocol that compresses both the upstream and downstream communications via sparsification, ternarization, error

accumulation, and optimal Golomb encoding. Our experiments show that STC is far more robust to the above-mentioned peculiarities of the learning environment than federated averaging. Moreover, STC converges faster than federated averaging both with respect to the number of training iterations and the amount of communicated bits even if the clients hold i.i.d. data and use moderate batch sizes during training.

Our approach can be understood as an alternative paradigm for communication-efficient federated optimization that relies on high-frequent low-volume instead of low-frequent high-volume communication. As such, it is particularly well suited for federated learning environments that are characterized by low latency and low bandwidth channels between clients and server.

REFERENCES

[1] R. Taylor, D. Baron, and D. Schmidt, "The world in 2025: 8 Predictions for the next 10 years," in *Proc. 10th Int. Microsyst., Packag., Assembly Circuits Technol. Conf. (IMPACT)*, 2015, pp. 192–195.

[2] S. Wiedemann, K.-R. Müller, and W. Samek, "Compact and computationally efficient representation of deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published. doi: 10.1109/TNNLS.2019.2910073.

[3] S. Wiedemann, A. Marban, K.-R. Müller, and W. Samek, "Entropy-constrained training of deep neural networks," in *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, 2019, pp. 1–8.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[5] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2015, pp. 3128–3137.

[6] S. Bosse, D. Maniry, K.-R. Müller, T. Wiegand, and W. Samek, "Deep neural networks for no-reference and full-reference image quality assessment," *IEEE Trans. Image Process.*, vol. 27, no. 1, pp. 206–219, Jan. 2018.

[7] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2014, pp. 1725–1732.

[8] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[9] W. Samek, T. Wiegand, and K.-R. Müller, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models," *ITU J., ICT Discoveries*, vol. 1, no. 1, pp. 39–48, 2018.

[10] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016, *arXiv:1602.05629*. [Online]. Available: https://arxiv.org/abs/1602.05629

[11] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," 2018, *arXiv:1807.00459*. [Online]. Available: https://arxiv.org/abs/1807.00459

[12] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.

[13] S. Hardy *et al.*, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," 2017, *arXiv:1711.10677*. [Online]. Available: https://arxiv.org/abs/1711.10677

[14] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[16] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proc. IEEE CVPR*, vol. 1, Jun. 2017, no. 2, p. 3.

[17] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Sparse binary compression: Towards distributed deep learning with minimal communication," in *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, 2019, pp. 1–8.

[18] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," 2019, *arXiv:1902.01046*. [Online]. Available: https://arxiv.org/abs/1902.01046

[19] W. Wen *et al.*, "TernGrad: Ternary gradients to reduce communication in distributed deep learning," 2017, *arXiv:1705.07878*. [Online]. Available: https://arxiv.org/abs/1705.07878

[20] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1707–1718.

[21] H. Wang, S. Sievert, Z. Charles, D. Papailiopoulos, S. Liu, and S. Wright, "ATOMO: Communication-efficient learning via atomic sparsification," 2018, *arXiv:1806.04090*. [Online]. Available: https://arxiv.org/abs/1806.04090

[22] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signSGD: Compressed optimisation for non-convex problems," 2018, *arXiv:1802.04434*. [Online]. Available: https://arxiv.org/abs/1802.04434

[23] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," 2017, *arXiv:1704.05021*. [Online]. Available: https://arxiv.org/abs/1704.05021

[24] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Proc. 16th Annu. Conf. Int. Speech Commun. Assoc.*, 2015, pp. 1488–1492.

[25] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," 2017, *arXiv:1712.01887*. [Online]. Available: https://arxiv.org/abs/1712.01887

[26] Y. Tsuzuku, H. Imachi, and T. Akiba, "Variance-based gradient compression for efficient distributed deep learning," 2018, *arXiv:1802.06058*. [Online]. Available: https://arxiv.org/abs/1802.06058

[27] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*. [Online]. Available: https://arxiv.org/abs/1610.05492

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: https://arxiv.org/abs/1409.1556

[29] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, "signSGD with majority vote is communication efficient and byzantine fault tolerant," 2018, *arXiv:1810.05291*. [Online]. Available: https://arxiv.org/abs/1810.05291

[30] A. Krizhevsky, V. Nair, and G. Hinton. (2014). *The CIFAR-10 Dataset*. [Online]. Available: http://www.cs.toronto.edu/kriz/cifar.html

[31] Y. LeCun. (1998). *The MNIST Database of Handwritten Digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[32] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-IID data," 2018, *arXiv:1806.00582*. [Online]. Available: https://arxiv.org/abs/1806.00582

[33] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 4447–4458.

[34] S. Golomb, "Run-length encodings (corresp.)," *IEEE Trans. Inf. Theory*, vol. 12, no. 3, pp. 399–401, Jul. 1966.

[35] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1945–1953.

[36] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018, *arXiv:1804.03209*. [Online]. Available: https://arxiv.org/abs/1804.03209

[37] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*. [Online]. Available: https://arxiv.org/abs/1708.07747

[38] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," 2013, *arXiv:1312.6211*. [Online]. Available: https://arxiv.org/abs/1312.6211

Felix Sattler received the B.Sc. degree in mathematics, the M.Sc. degree in computer science, and the M.Sc. degree in applied mathematics from the Technische Universität Berlin, Berlin, Germany, in 2016, 2018, and 2018, respectively.

He is currently with the Machine Learning Group, Fraunhofer Heinrich Hertz Institute, Berlin. His current research interests include distributed machine learning, neural networks, and multitask learning.

Simon Wiedemann received the M.Sc. degree in applied mathematics from the Technische Universität Berlin, Berlin, Germany, in 2017.

He is currently with the Machine Learning Group, Fraunhofer Heinrich Hertz Institute, Berlin. His current research interests include machine learning, neural networks, and information theory.

Klaus-Robert Müller (M'12) received the Ph.D. degree in computer science from the Technische Universität Karlsruhe, Karlsruhe, Germany, in 1992, where he studied physics from 1984 to 1989.

He has been a Professor of computer science with the Technische Universität Berlin, Berlin, Germany, since 2006, where he is currently co-directing the Berlin Big Data Center. After completing a postdoctoral position at GMD FIRST, Berlin, he was a Research Fellow with The University of Tokyo, Tokyo, Japan, from 1994 to 1995. In 1995, he founded the Intelligent Data Analysis Group, GMD-FIRST (later Fraunhofer FIRST), and directed it until 2008. From 1999 to 2006, he was a Professor with the University of Potsdam, Potsdam, Germany. His current research interests include intelligent data analysis, machine learning, signal processing, and brain–computer interfaces.

Dr. Müller was elected to be a member of the German National Academy of Sciences-Leopoldina in 2012, the Berlin Brandenburg Academy of sciences in 2017, and an External Scientific Member of the Max Planck Society in 2017. He received the 1999 Olympus Prize by the German Pattern Recognition Society, DAGM. He received the SEL Alcatel Communication Award in 2006, the Science Prize of Berlin awarded by the Governing Mayor of Berlin in 2014, and the Vodafone Innovation Award in 2017.

Wojciech Samek (M'13) received the Diploma degree in computer science from the Humboldt University of Berlin, Berlin, Germany, in 2010, and the Ph.D. degree in machine learning from the Technische Universität Berlin, Berlin, in 2014.

In 2014, he founded the Machine Learning Group, Fraunhofer Heinrich Hertz Institute, where he is currently the Director. He was a Scholar of the German National Academic Foundation and a Ph.D. Fellow with the Bernstein Center for Computational Neuroscience Berlin, Berlin, where he is also with the Berlin Big Data Center. He was visiting with Heriot-Watt University, Edinburgh, U.K., and The University of Edinburgh, Edinburgh, from 2007 to 2008. In 2009, he was with the Intelligent Robotics Group, NASA Ames Research Center, Mountain View, CA, USA. His current research interests include interpretable machine learning, neural networks, federated learning, and computer vision.