Structured Sparse Ternary Compression for Convolutional Layers in Federated Learning

Structured Sparse Ternary Compression for Convolutional Layers in Federated Learning

Alessio Mora
University of Bologna
Bologna, Italy
alessio.mora@unibo.it

Luca Foschini
University of Bologna
Bologna, Italy
luca.foschini@unibo.it

Paolo Bellavista
University of Bologna
Bologna, Italy
paolo.bellavista@unibo.it

***Abstract*—In Cross-device Federated Learning, communication efficiency is of paramount importance. Sparse Ternary Compression (STC) is one of the most effective techniques for considerably reducing the per-round communication cost of Federated Learning (FL) without significantly degrading the accuracy of the global model, by using ternary quantization in series to top$_k$ sparsification. In this paper, we propose an original variant of STC that is specifically designed and implemented for convolutional layers. Our variant is originally based on the experimental evidence that a pattern exists in the distribution of client updates, namely, the difference between the received global model and the locally trained model. In particular, we have experimentally found that the largest (in absolute value) updates for convolutional layers tend to form clusters in a kernel-wise fashion. Therefore, our primary novel idea is to a-priori restrict the elements of STC updates to lay on such a structured pattern, thus allowing us to further reduce the STC communication cost. We have designed, implemented, and evaluated our novel technique, called Structured Sparse Ternary Compression (SSTC). Reported experimental results show that SSTC shrinks compressed updates by a factor of x3 with respect to traditional STC and with a reduction up to x104 with respect to uncompressed FedAvg, at the expense of negligible degradation of the global model accuracy.**

***Index Terms*—Federated Learning, Compression**

I. Introduction

Federated Learning (FL) has emerged as a solution to decouple the ability to train Deep Learning (DL) models from the necessity to directly access the raw data, by exploiting periodic broadcast of model weights and local computation of weight updates. Only ephemeral, locally processed information needs to be disclosed by the entities or users that participate to the collaborative learning process, and this favours the enforcing of privacy guarantees to a certain extent [1].

Although FL is primarily motivated by the urge of privacy while distilling collective knowledge, having on-device ML/DL models can meet low-latency inference requirements as well as be a greener technology with respect to cloud-based model training [2]. Furthermore, the design of FL can be useful both in Cross-silo federations, i.e. the learners involved are entities such as healthcare institutions [3], banks, companies, or in Cross-device federations, i.e. edge devices such as smartphones [4] or IoT equipment. Real-world examples of the latter setting are, for instance, smart apps that learn user habits without exporting sensitive raw data to companies' servers (e.g., [5]).

The Cross-device scenario exhibits peculiar characteristics. In fact, the federation includes a large number of participants that hold not Independently and Identically Distributed (non-IID) data, i.e., training examples on a specific device cannot be assumed to be representative of the global distribution. In addition, data are unbalanced and massively distributed (e.g., learners may hold very different amounts of training examples). A further significant concern about the feasibility of large scale Cross-device FL is the massive communication overhead implied by periodic broadcast of uncompressed model weights and iterative harvesting of locally-computed model updates. This, on the one hand, can hamper the participation of edge devices with limited bandwidth, and, on the other hand, may overwhelm the internet infrastructure with FL-related payloads [1].

Several communication-efficient strategies have been proposed to reduce the amount of information exchanged. One of the most effective is Sparse Ternary Compression (STC) [6], that can extremely shrink the payload size of model parameters and updates without significantly degrading the global model accuracy. In this paper, we propose a novel variant of STC, namely Structured Sparse Ternary Compression (SSTC), that is specifically designed and optimized for updates of convolutional layers in neural networks.

In this paper we make the following contributions:

- We demonstrate that the largest (in absolute value) weight updates in convolutional layers tend to form clusters along the kernel dimension (i.e., the majority of the largest updates comes from a subset of kernels).
- Based on such an empirical evidence, we propose to restrict the range of indexes in which an update element can lay to obtain a more aggressive compression.
- For the sake of result reproducibilty and to foster related lines of work, we provide a reference to our repository that contains the simulation code we used for experimental results (https://github.com/alessiomora/SSTC).

II. Related Work

Some communication-efficient strategies have been recently designed to reduce the per-round cost of Cross-device FL. Sparsification, quantization and encoding, proposed in different flavors [6]–[10], represent the main line of work to

significantly lower the bandwidth requirements of the decentralised learning process. Also federated pruning [11] and federated dropout [12] result in reduced communication cost for learners, but alone cannot achieve extreme compression as sparsification- and quantization-based compression.

In [6], the authors propose STC, see Sec. III-A for detail, and demonstrate that their ternary compression can reach target global model accuracy with extremely lower communication budget with respect to uncompressed Federated Averaging [13], also when clients hold heterogeneous data distribution.

To the best of our knowledge, FedSCR [14] is the only work that analyses and exploits patterns in the distribution of convolutional updates for enhancing compression. They focus on convolutional architecture, as we do in our SSTC. However, the main similarity between our work and FedSCR stands in the attempt of exploiting update patterns for compression scope, while ours and their proposed techniques are deeply different from several other perspectives: in our proposal we try to embed such empirical observations in STC for a more efficient encoding and we act at the kernel level, while in FedSCR they consider channels and filters for their structure-wise identification of more significant update components; furthermore, in FedSCR, learners accumulate updates deemed insignificant, eventually synchronizing them with the server; conversely, SSTC does not assume that the same client will participate more than once in the FL process.

III. Federated Learning

In this paper, we consider Federated Averaging (FedAvg) as the baseline for Federated Learning [13]. In FedAvg, collaborative learning proceeds in synchronous rounds by leveraging a client-server paradigm. At the beginning of each round, the server (or aggregator) broadcasts the current parameters of the global model to a fraction of available clients (i.e., the participants). Each learner locally trains the received model parameters on its private data, and sends back an update to the server (e.g. the difference between the received and the locally tuned model parameters). The server collects the updates from the federation, and uses a given strategy (weighted average, according to the amount of local examples held by clients, in the case of FedAvg) to aggregate the gathered contributions. The aggregated updates are then applied to the global model. At this point, a new round of FL can start by distributing the novel version of the global model [13]. Our proposed SSTC compression technique applies to client updates, implemented as the difference between the last received parameters of the global model and the locally calculated model parameters.

A. Sparse Ternary Compression

STC [6] is a lossy compression scheme able to extremely reduce the per-round communication cost of FL iterations. STC uses, in series, top_p sparsification and ternary quantization. Firstly, all but the p larger absolute values[1] in the input tensor (either model weights or weight updates) are zeroed out. Then, the survived fraction of elements is binary quantized to $\{\mu, -\mu\}$, with positive values substituted with μ, negative values with $-\mu$, and μ being the mean, in absolute value, of the non-zero elements. Therefore, the algorithm outputs a ternary tensor with values $\{-\mu, 0, \mu\}$.

The ternarization process of STC reduces the entropy of the tensor to be communicated, and favors an efficient encoding. Only the mean value, μ, and the indexes of non-zero elements has to be transmitted, instead of all the values in the original tensor. The decoder assumes that elements corresponding to non-communicated indexes are filled with 0 value.

It is worth noting that indexes can be losslessly compressed by transmitting the distances between consecutive indexes instead of their absolute positions in the tensor. Then distances can be optimally encoded with Golomb code [6], by reducing the bits necessary to represent them[2]. Let us highlight that this can be applied as well in our SSTC proposal; however, we do not consider the possibility of using Golomb encoding for indexes either in STC and SSTC in the comparison that follows in order to better point out the advantages deriving from the only exploitation of the SSTC approach.

Algorithm 1: SSTC algorithm.
$l \in [1, L]$ with L the number of convolutional layers in the neural network, sparsity p, fraction of kernels k.

Input : list ΔW of per-layer update tensors $\Delta W_l \in \mathbb{R}^{K \times K \times C_l \times F_l}$, p, k
Output: list ΔW^{sstc} of per-layer SSTC update tensors $\Delta W_l^{sstc} \in \{-\mu, 0, \mu\}^{K \times K \times C_l \times F_l}$

$\Delta W \leftarrow$ Reshaping ΔW_l to 2-d tensors
$T \leftarrow$ Concatting reshaped ΔW_l to one 2-d tensor
$T^{top_k\_kernels}$, $indexes \leftarrow$ Selecting top$_k$ kernels on T
$T^{top_k\_kernels} \leftarrow$ Flattening $T^{top_k\_kernels}$
$T^{stc} \leftarrow$ STC on $T^{top_k\_kernels}$ with sparsity p
$T^{sstc} \leftarrow$ Scattering T^{stc} column to the original shape of T by means of kernel $indexes$
$\Delta W^{sstc} \leftarrow$ Slicing and reshaping T^{sstc}
return ΔW^{sstc}

IV. Our SSTC original proposal

As also demonstrated in [14], for convolutional layers, the distribution of client updates is not uniformly distributed. A specific pattern has been demonstrated to emerge: the largest (in absolute value) updates form clusters on a subset of kernels. Fig. 1 shows the output of top$_{0.1\%}$ sparsification on updates (in absolute value) for a convolutional layer at three different rounds of the federated training.

On the other hand, in traditional STC the communication payload is almost entirely due to indexes. In our original SSTC we propose to exploit the presence of kernel-wise pattern in the distribution of large updates for convolutional layers to reduce

[1] p can be expressed as a percentage of the number of elements in the input tensor. $top_{x\%}$ sparsification retains the $x\%$ larger values.

[2] In [6] the authors assume a random sparsity pattern, and that the distances among consecutive indexes can be approximated by a geometric distribution.

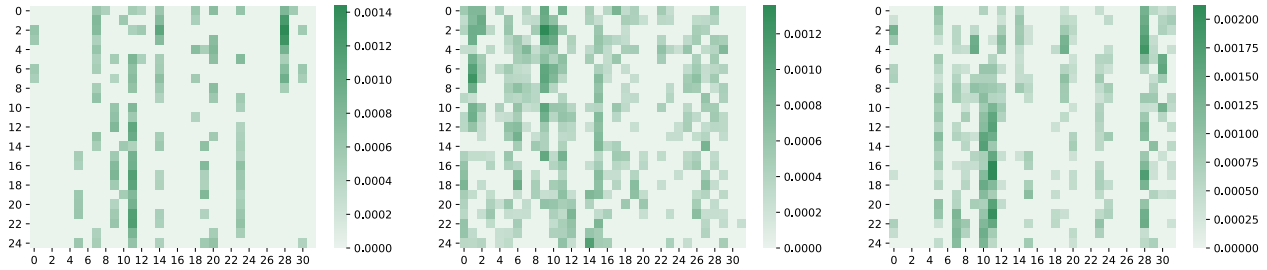


Fig. 1: Distribution of convolutional weight updates (in absolute value) after $top_{0.1\%}$ sparsification at three different rounds on three randomly sampled clients. Each column in the heat maps represents a kernel update. The reported results are obtained using the neural architecture and settings described in Sec. V, and refer to the first layer of the network.

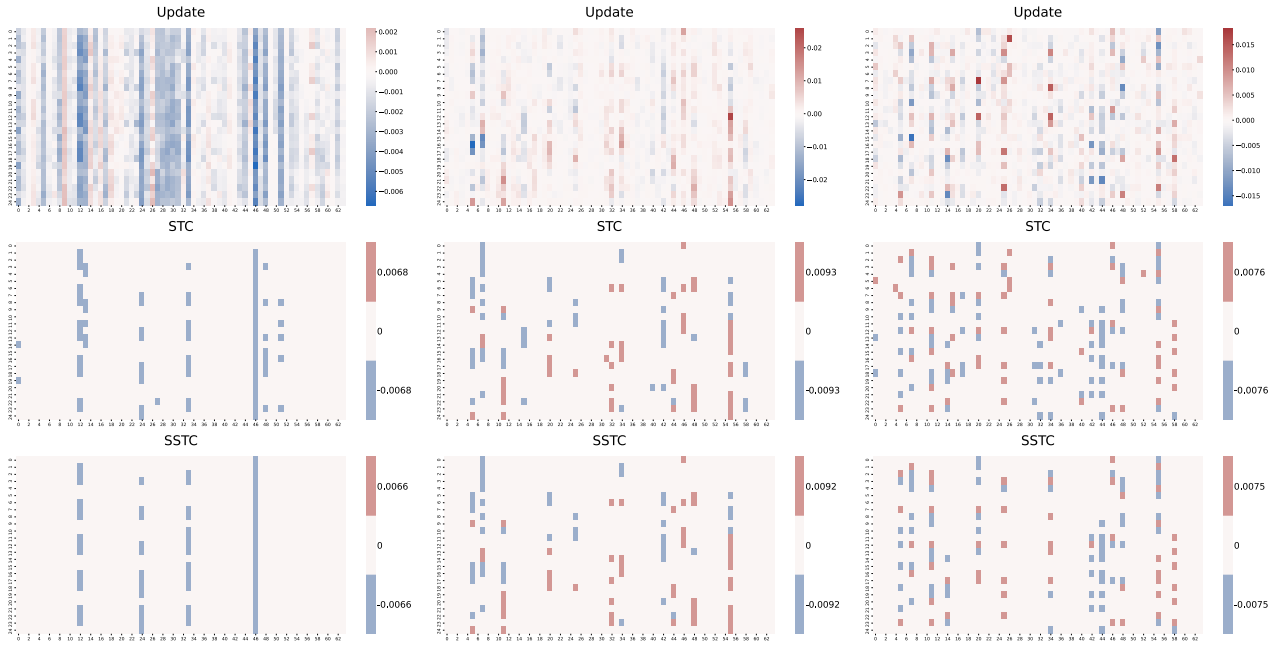


Fig. 2: The figure reports a comparison among STC-compressed and SSTC-compressed updates at three different rounds (round 1, 500 and 1000). Each column of three heat maps refers to a round. From top to bottom, the heat maps represent the uncompressed updates, the updates after STC application and the updates after SSTC application for that round on a random client. The reported results are obtained using the neural architecture and settings described in Sec. V, and refer to the second layer of the network, with STC sparsity equal to 1% and kernel fraction for SSTC equal to 12.5%. Each column in the heat maps visualizes kernel update.

the communication cost of STC. In a practical perspective, the principle is that knowing a priori that the largest elements of the updates will lay on a restricted subset of the convolutional kernels makes possible a more efficient encoding of indexes.

SSTC supposes to find the majority of the largest updates on a subset of the kernels in the convolutional layers, and restricts the search space for the top$_p$ elements by considering only a fixed number of kernels among all the convolutional layers (i.e., the top$_k$ kernels, identified according to the average of the absolute values of the elements in kernels updates – the columns in the heat maps in Fig. 1 and Fig. 2).

Algorithm 1 formalizes our compression method. For the sake of clarity, Alg. 1 considers a neural network composed by only convolutional layers with the same kernel size. This simplification does not preclude the application of SSTC to neural networks that also have, e.g., fully connected layers, as it is shown also in the reported experimental results.

Lastly, we note that SSTC exactly matches STC when there is no pre-selection of kernels, i.e., when the fraction of kernels to search for larger elements is equal to 1.

Fig. 2 compares the effect of STC and SSTC on the same convolutional weight updates. Different aspects emerge: (i) The distribution of the non-zero elements of STC exhibits a column-wise pattern; (ii) The SSTC approximation of STC is quite accurate even though we are considering, in the depicted results, only the 12.5% of the columns. This is due to the fact

that STC's top$_p$ sparsification zeroes out entire columns while SSTC does not consider such columns in the first place. (iii) SSTC tends to have more dense kernel updates with respect to STC since it considers a reduced subset of them; (iv) SSTC inevitably ignores and zeroes out large absolute values that lay on kernels with average lower than the top$_k$ while it may include elements that would have been excluded by STC.

A. Lossless Encoding

To transmit the produced structured sparse ternary tensor, the indexes of the top$_k$ kernels and a ternary map of the values' sign in each kernel are communicated. The ternary maps have the size of the kernel. The values in the maps are $\{-1, 0, 1\}$, i.e, 0 for sparsified elements, -1 or 1 to signify the sign of the non-zero elements. The zeros within retained kernels are communicated since we expect the non-zero values to be dense in the selected kernels. Intuitively, the gain stands in indexing groups of consecutive values, instead of indexing values one by one.

Let us note that the above SSTC encoding results to be convenient if compared with STC when it holds that:

$$\frac{b_p * p}{b_k * k + 2 * K * K * k} > 1 \tag{1}$$

Where the nominator and denominator in Eq. 1 respectively refer to the communication cost due to indexes in STC and SSTC, and $b_k = 1 + \lfloor \log_2(C * F) \rfloor$ is the bitsize to represent each of the kernels in SSTC with F being the number of filters and C the number of channels, $b_p = 1 + \lfloor \log_2(K * K * C * F) \rfloor$ is the bitsize to represent each of the non-zero elements in STC with $K \times K$ being the size of convolutional kernels, k is the number of the communicated kernels in SSTC and p is the number of non-zero values in STC. Eq. 1 can be rewritten as:

$$k < p * \frac{b_p}{2 * K * K + b_k} \tag{2}$$

It is worth noting that Eq. 1 and Eq. 2 refer to the application of STC and SSTC on convolutional weights only. Hence, if we consider a neural network with only convolutional layers, $p = sparsity * W$ with W being the total amount of weights. Conversely, if we consider a neural network with, for example, both convolutional and fully connected layers, p will most probably vary round by round, since sparsification is applied considering the whole model parameters, and non-zeroed elements can be distributed among all the layers.

V. Experimental Results

A. Experimental Setup

We consider FedAvg as the baseline and we compare SSTC vs. STC in terms of top-1 accuracy of the global model and of compression ratio. We use the same hyperparameter tuning for every strategy: the local epochs of each client are fixed to 5; Stochastic Gradient Descent (SGD) is used as local optimizer with 0.1 learning rate; the client batch size is fixed to 16. The simulations are run for 1000 rounds with 50 clients (out of 3,400 total learners) randomly selected per round. The sparsity is equal to 0.01 (i.e. 1%) both for STC and SSTC. We implement the simulation using TensorFlow (TF) and TensorFlow Federated (TFF).

For the experiments, we use the LEAF version of the Federated EMNIST dataset (FEMNIST) available in TFF for 62-class image classification. Each learner holds her/his own handwritten characters to reproduce data heterogeneity, and the labels are unbalanced in number and not uniformly distributed among clients. FEMNIST has a total of 671,585 examples for training, distributed among 3,400 participants, and 77,483 examples for testing. For the FEMNIST classification task, we use a Convolutional Neural Network (CNN) composed by two 5x5 convolution layers, 32 and 64 filters respectively, each followed by a max pooling layer, a fully connected dense layer with 512 units, and a final softmax output layer.

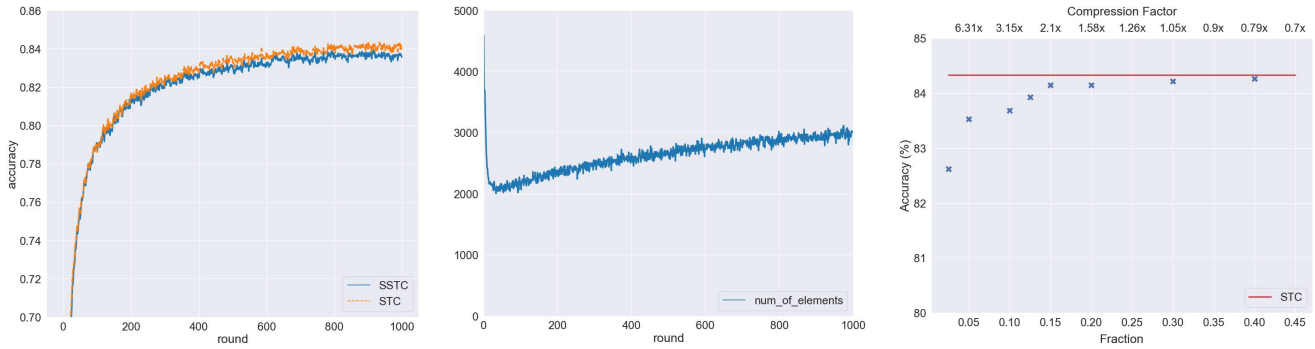
B. Measured Performance Results and Related Discussion

The first two convolutional layers of the neural network that we considered for the experiments have 52,000 weights in total, excluding biases that are not subject to compression (the first layer has 32 filters with 5x5 kernels size, the second layer has 32 channels with 64 filters with 5x5 kernel size). The neural network also has two fully connected layers for classification. As we explained in Alg. 1, the algorithm performs a pre-selection of the kernels with larger element mean, in absolute value, and then consider only such a subset of convolutional weights when sparsification is applied to the whole neural network weights (parameters belonging to both convolutional and fully connected layers).

To compare the compression gain introduced by SSTC with respect to STC, as we noted in Sec. IV-A, we monitored how many non-zeroed elements belonging to the convolutional layers are communicated round by round in STC (Fig. 3b). In this way, we have a range of values for the term p in Eq. 1; in the experiments, p ranges among 2,000 and 3,000 (approximately). K is fixed to 5, since kernels have size of 5x5. In STC, to represent the maximum index (i.e., 51,999) 16 bits are needed (i.e., b_p = 16), also considering the worst case when encoding using distances. In SSTC the maximum index for kernel depends on the tuning of k. For example, if we select only the 12,5% of the kernels, we have $k = 260$ for the considered neural network, and 11 bits needed to represent them (i.e., b_k = 11). So, applying Eq. 1, the gain of SSTC with respect to STC considering the convolutional part of the considered neural network ranges, approximately, between 2x and 3x.

The gain introduced by SSTC is more evident if compared with uncompressed FedAvg, as reported in Table I.

Fig. 3c depicts the maximum accuracy reached by different options for k tuning. The global model accuracy of SSTC converges to the one of STC when k approaches the 40% of the kernels. However, the proposed encoding is efficient for very dense (i.e., containing few zero values) sign maps, hence with low k value. In fact, in the considered deployment environment, when the fraction of top$_k$ kernels is over (approximately) the 30% the encoding proposed in Sec. IV-A



(a) (b) (c)

Fig. 3: (a) Top-1 accuracy of global model on test set for STC and SSTC, with sparsity equal to 0.01 and the fraction of kernel equal to 0.125. (b) Number of elements in STC updates that belong to the convolutional layers during experiments in (a). (c) Accuracy reached with different tuning of k, expressed as a fraction, in SSTC. The line represents the maximum accuracy of STC. The top x-axis reports the compression factor, averaged within 1000 rounds, of SSTC with respect to STC, i.e. the average gain introduced by the encoding proposed in Sec. IV-A with respect to communicate indexes one by one.

becomes less efficient than directly sending indexes one by one (Fig. 3c reports, in the top x-axis, the compression factor of SSTC with respect to STC).

TABLE I: Compression factors of STC and SSTC for updates coming from the conv layers with respect to uncompressed (32-bit float) FedAvg. Sparsity = 1%, and k = 12.5% for SSTC.

	compression (≈)	max acc.
FedAvg + STC	41x	84.33%
FedAvg + SSTC (ours)	104x	83.94%

VI. Conclusions and Future Work

In this paper we presented an original compression technique that builds on top of STC, and leverages a specific pattern in the distribution of convolutional weight updates to shrink the STC encoding.

While this work may be seen as a specialization of STC, SSTC can be more generally thought as an attempt to exploit – and benefit from – evident patterns in FL updates, a research line that is still largely unexplored in the existing literature and that calls for additional future work by the Federated Learning community. Assumptions on the distribution of client updates can enhance compression techniques, as well as strategies that analyse updates (e.g., identifying malicious updates leveraging autoencoders).

The SSTC technique originally presented here only considers convolutional weight updates; our current research work includes the analysis of the distribution of fully connected or recurrent weight updates in search of meaningful patterns. In addition, we plan to extend the experimental results on deeper CNNs and more complex datasets.

References

[1] Paolo Bellavista, Luca Foschini, and Alessio Mora. Decentralised learning in federated deployment environments: A system-level survey. *ACM Computing Surveys (CSUR)*, 54(1):1–38, 2021.

[2] Xinchi Qiu, Titouan Parcollet, Daniel Beutel, Taner Topal, Akhil Mathur, and Nicholas Lane. Can federated learning save the planet? In *NeurIPS-Tackling Climate Change with Machine Learning*, 2020.

[3] Jie Xu, Benjamin S Glicksberg, Chang Su, Peter Walker, Jiang Bian, and Fei Wang. Federated learning for healthcare informatics. *Journal of Healthcare Informatics Research*, 5(1):1–19, 2021.

[4] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[5] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329*, 2019.

[6] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, 2019.

[7] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*, 2018.

[8] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Sparse binary compression: Towards distributed deep learning with minimal communication. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[9] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[10] Jinjin Xu, Wenli Du, Yaochu Jin, Wangli He, and Ran Cheng. Ternary compression for communication-efficient federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[11] Wenyuan Xu, Weiwei Fang, Yi Ding, Meixia Zou, and Naixue Xiong. Accelerating federated learning for iot in big data analytics with pruning, quantization and selective updating. *IEEE Access*, 9:38457–38466, 2021.

[12] Paolo Bellavista, Luca Foschini, and Alessio Mora. Communication-efficient heterogeneous federated dropout in cross-device settings. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2021.

[13] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[14] Xueyu Wu, Xin Yao, and Cho-Li Wang. Fedscr: Structure-based communication reduction for federated learning. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1565–1577, 2020.