ZEROFL: EFFICIENT ON-DEVICE TRAINING FOR FEDERATED LEARNING WITH LOCAL SPARSITY

Anonymous authors
Paper under double-blind review

ABSTRACT

When the available hardware cannot meet the memory and compute requirements to efficiently train high performing machine learning models, a compromise in either the training quality or the model complexity is needed. In Federated Learning (FL), nodes are orders of magnitude more constrained than traditional server-grade hardware and are often battery powered, severely limiting the sophistication of models that can be trained under this paradigm. While most research has focused on designing better aggregation strategies to improve convergence rates and in alleviating the communication costs of FL, fewer efforts have been devoted to accelerating on-device training. Such stage, which repeats hundreds of times (i.e. every round) and can involve thousands of devices, accounts for the majority of the time required to train federated models and, the totality of the energy consumption at the client side. In this work, we present the first study on the unique aspects that arise when introducing sparsity at training time in FL workloads. We then propose *ZeroFL*, a framework that relies on highly sparse operations to accelerate on-device training. Models trained with ZeroFL and 95% sparsity achieve up to 3% higher accuracy compared to competitive baselines obtained from adapting a state-of-the-art sparse training framework to the FL setting.

1 INTRODUCTION

Despite its being a relatively new subfield of machine learning (ML), Federated Learning (FL) (McMahan et al., 2017; Reddi et al., 2020; Horvath et al., 2021) has become an indispensable tool to enable privacy-preserving collaboratively learning, as well as to deliver personalised models tailored to the end-user's local data and context (Arivazhagan et al., 2019; Hilmkil et al., 2021; Cheng et al., 2021). For example: next-word prediction (Hard et al., 2018), physical activity detection (Doherty et al., 2017), keyword spotting (Hard et al., 2020), among others.

Unlike standard centralised training, which normally takes place on the Cloud and makes use of powerful hardware (Hazelwood et al., 2018), FL is envisioned to run on commodity devices such as smartphones or IoT devices often running of batteries, which are orders of magnitude more restricted in terms of compute, memory and power consumption. This triplet of factors drastically limits the complexity of the ML models that can be trained on-device in a federated manner, ceiling their usefulness for the aforementioned applications as a result. In order to adjust the memory and compute footprints of complex ML model to the FL setting, the research community has presented a number of approaches including: the use of distillation (Hinton et al., 2015) to enable the aggregation on the server side of heterogeneous model architectures (e.g. based on the compute capabilities of each device) that collaboratively train a single global model (Lin et al., 2020; Zhu et al., 2021); federated dropout, by which clients perform local training on a sub-model of the global model (Caldas et al., 2019), translates into lower overall communication costs and, enables better support for heterogeneous pools of clients regardless of their compute capabilities (Horvath et al., 2021); and, more generally, better aggregation strategies that enable faster convergence (Li et al., 2018; Wang et al., 2020; Reddi et al., 2020), reducing in this way overall device utilization (e.g. fewer local epochs) and number of communication rounds. Other optimization techniques such as quantization and sparsity have been used in the context of FL but mostly as a way to reduce communication costs (Liu et al., 2021; Amiri et al., 2020; Shahid et al., 2021) but not to accelerate on-device training.

The use of sparse operations (e.g. convolutions) at training time has recently been shown to be an effective technique to accelerate training in centralised settings (Sun et al., 2017; Goli & Aamodt, 2020; Raihan & Aamodt, 2020). The resulting models are as good or close to their densely-trained counterparts despite reducing by up to 90% their FLOPs budget and, resulting in an overall up to $3.3\times$ training speedup. Acceleration is achieved by performing sparse convolutions during the forward and/or backward pass, which requires at least one of the operands (i.e. inputs, weights, gradients) to be sufficiently sparse and, software and hardware support for such operations. However, it is unclear how the different FL-specific challenges (i.e. data imbalance, stateless clients, periodic aggregation) will restrict the quality of the global model.

This work considers the challenges and opportunities of inducing high levels of sparsity to accelerate training on-device for FL workloads, and provides the following contributions:

- The first framework for Federated Learning that leverages sparsity as a mechanism to accelerate on-device training by inducing up to 95% sparse weights and activations. This is achieved without the server imposing any restrictions on the form or elements that get zeroed out, preserving privacy and enabling each client leveraging its own data independently as a result.
- A study on the unique aspects that arise when introducing sparsity at training time in FL: the degree of overlap between non-zero values decreases with layer-depth index and, the locations of zero-valued weights in the global model remain constant throughout most of the training rounds. Our discussion sets the foundations for future research in this area.
- A technique that alleviates the accuracy degradation when applying a state-of-the-art off-the-shelf sparsification method to the FL domain. *ZeroFL* achieves +2.1% and +3.7% higher accuracy than baselines when inducing 90% and 95% sparsity respectively. In addition, ZeroFL also leverages sparsity when transferring the local models to the central server reducing communication costs by $3.0\times$ while still outperforming competitive baselines.

2 Related Work

Pruning neural networks involves discarding parts of the model (e.g. individual weights or entire channels) that are *irrelevant* for solving the task at hand. This procedure generally produces a lightweight model representation more suitable for deployment on constrained devices with limited memory and compute budgets. In this section we detail how different forms of pruning or sparsification have been used to accelerate inference and, to a lesser extent, training. We also discuss how these have been introduced to reduce communication costs in distributed and federated learning.

Unstructured pruning. Frameworks relying on unstructured pruning (Han et al., 2015a;b; Guo et al., 2016; Molchanov et al., 2017) often achieve higher compression ratios at the expense of inference stages being as compute intensive in practice as those of the original model. This is because, assuming pruning has been homogeneously applied on the model, sparse operations can only be efficiently accelerated on supported hardware, such as modern GPUs (Wang, 2020; Zachariadis et al., 2020; Hong et al., 2018) or custom accelerators (Zhang et al., 2016; Lu et al., 2019; Srivastava et al., 2020), for a sufficiently high sparsity ratio. The lower the ratio, the less likely sparse operations would translate into measurable speedups. In the case of CPUs, speedups due to sparse operations where one operand is unstructurally sparse are often only feasible at 90% sparsity ratios or higher (Hong et al., 2019; Wang, 2021).

Structured pruning. Methods that apply structured pruning (He et al., 2018; 2017; Jian-Hao Luo & Lin, 2017; Yu et al., 2018; Molchanov et al., 2019; Wang et al., 2017), on the other hand, trade compression for acceleration potential. These approaches modify the underlying computational graph by discarding entire channels, resulting in smaller but still dense convolution operations, or by removing the nodes all together if an entire layer is set to be removed by the chosen pruning strategy. As a result, structured pruning frameworks are the preferred option when aiming to accelerate inference on general purpose hardware. A body of work across structured and unstructured pruning methods, attempts to induce structure in otherwise randomly sparse networks S. Gray & Kingma (2017); Ren et al. (2018); Wen et al. (2020); Verelst & Tuytelaars (2020). This is often referred to as *block sparsity* and consists in subdividing the matrix representations of inputs or weights into tiles (e.g. 16×16 tiles), and restrict the training in such a way that some tiles contain only zeros while

the rest remain dense and real-valued. Matrix-matrix multiplications following such a pattern can be accelerated at lower global sparsity ratios compared to those following unstructured sparsity Hoefler et al. (2021). Other forms of constraining how sparsity occurs have been proposed, for example, a cache-aware reordering on the sparsity pattern of the weights Elsen et al. (2020). This can be used to ensure high cache reuse on Cortex-A mobile CPUs, resulting in $2.4\times$ acceleration of MobileNets.

Sparse training. Until recently the majority of works making use of sparsity were envisioned for either model compression or to accelerate inference. Only recently, sparse operations have been considered to accelerate training. The work of Sun et al. (2017) presented a mechanism to induce high levels of sparsity in the gradients during backpropagation and, demonstrated large speedups when training MLP-only models. More recently, Goli & Aamodt (2020) build upon the observation that gradients from consecutive batches are near identical. They present a framework to reuse a random sample of previously computed gradients and their thresholded difference w.r.t gradients from the current batch, resulting in a sparse tensor. Their framework, ReSprop, accelerates training of CNNs by performing sparse convolutions during the backward pass at the cost of pre-computing partial gradients during forward pass. Closer to our work is SWAT (Raihan & Aamodt, 2020), a framework that relies on sparsified weights during inference and sparsified weights and activations for backward propagation. Compared to the previous two frameworks, SWAT, which does not directly sparsify the gradients, achieves superior performance in terms of training acceleration and accuracy retention at high sparsity ratios.

Compression on communication. Konečnỳ et al. (2016) proposed to restricts the updates of weight matrices to have a pre-specified structure in order to reduce the total communication cost. The structure can either be random or low-rank structure. ATOMO (Wang et al., 2018) introduced a generalised gradient decomposition and sparsification technique, aiming to reduce the gradient sizes communicated upstream. Han et al. (2020) proposed a different way of aggregation in the server, which instead of aggregating model weights, it aggregates the sparsified gradients after every local update step. However, since the method requires to aggregate sparsified gradient after every step, it cannot benefit from multiple local updates. Hence it might require extra communication rounds to reach the target performance.

Nevertheless, none of the aforementioned works explored the challenges of extending state-of-the-art sparsification methods to federated learning as a way to accelerate on-device training. With ZeroFL, a framework specifically tailored to the FL setting, achieves better accuracy retention than with existing methods that remain exclusive to the centralised training paradigm.

3 Background

This section describes the state-of-the-art sparse training method SWAT (Raihan & Aamodt, 2020); the way we adapt it to the FL contexts; and the related challenges that would need be addressed.

3.1 Sparse Weights and Activations Training

The SWAT framework embodies two strategies in the training process. During each forward pass, the weights are partitioned into active weights and non-active weights by a `top-K` (in magnitude) operator and only the active weights are used. For the lth layer in the model, the layer maps the input activations a_{l-1} onto feature maps o_l using function f_l: $o_l = f_l(a_{l-1}, w_l)$. In this work we consider f_l being the 3×3 convolution in the l-th layer. In the backward pass, the gradient of input activations ($\bigtriangledown a_{l-1}$) and the gradient of weights ($\bigtriangledown w_l$) are calculated represented by functions G_l and H_l, as shown below:

$$\bigtriangledown a_{l-1} = G_l(\bigtriangledown a_l, w_l) \tag{1}$$

$$\bigtriangledown w_l = H_l(\bigtriangledown a_l, a_{l-1}) \tag{2}$$

Then in the backward pass, the retained layer inputs a_{l-1} are also partitioned into active and non-active by using the same `top-K` procedure. This results in full gradients and active weights being used in Eq. 1, while full gradients and active activations are used in Eq. 2. It is worth noticing that even weights and activations and sparsified in the forward and backward pass, the gradients generated through the training process are dense. Therefore, the resulting model is a dense. The compute cost of updating weights w_l given a dense $\bigtriangledown w_l$ tensor is negligible compare to the savings due to performing the underlying convolutions in Eq.1&2, as this is essentially a weighted sum.

3.2 FROM CENTRALISED TO FEDERATED SPARSE TRAINING

A direct adaptation of the SWAT framework to the FL setting could be done by framing each local training stage on a client as an instance of centralised training. However, one major difference between centralized training and FL is the fact how often data are presented during training. In a centralised scenario, each example in the training set is seen multiple times, once per epoch, allowing for the model to converge to a stable distribution of weights. This scenario is more suitable for sparcification.

On the other hand, in a typical *cross-device* scenario, client's availability is low and new data points are continuously being presented to the system as new clients participate in training rounds. Such training behaviour inevitably leads to distributions of weights that change over time, difficulting the application of sparsisty inducing methods as a result.

4 SPARSE TRAINING FOR FEDERATED LEARNING

As a first step, we conduct preliminaries investigations and measure SWAT's performance when directly applied to FL without any adaptation. This section describes the experimental protocol (Section 4.1), the obtained baseline results (Section 4.2) and a sparsification effect analysis to highlight the weaknesses of this approach in Section 4.3.

4.1 EXPERIMENTAL SETUP

While SWAT is used across the experiments as the standard sparsification methodology, results also depend on various FL-specific hyper-parameters. Federated learning is simulated with the Virtual Client Engine (VCE) of the Flower toolkit (Beutel et al., 2020) enabling us to scale to a large number of clients within a single machine of compute cluster. Considered datasets and hyper-parameters are detailed thereafter.

Dataset. Estimations are conducted on the CIFAR-10 dataset (Krizhevsky et al., 2009), arguable the most extensive image classification dataset considered in the FL literature. The task at hand is to classify 32×32 RGB images into one of 10 possible classes.

Data partitioning. Data distribution strongly impacts FL performance, it is therefore crucial to investigate different scenarios. We simulate two scenarios: a uniform partition (IID) where each client has approximately the same proportion of each class from the original dataset, and a heterogeneous partition (non-IID) for which each client has an unbalanced and different proportion of each class. We follow the latent Dirichlet allocation (LDA) partition method (Reddi et al., 2020; Yurochkin et al., 2019; Hsu et al., 2019) ensuring that each client is allocated the same number of training samples. The level of heterogeneity is governed by the parameter α. As $\alpha \to \infty$, partitions become more uniform (IID), and as $\alpha \to 0$, partitions tend to be more heterogeneous. Our experimental evaluation considers $\alpha = 1.0$ and $\alpha = 1000$.

Model Architecture. Following the convention for CIFAR-10, a ResNet-18 (He et al., 2016) architecture is instantiated in the client side and aggregated on the server. All 3×3 convolutional layers implement sparse training as discussed in Sec. 3.2. As it is common with other optimizations, we leave the input layer unchanged (i.e. performing standard dense training). The models are trained with SGD, and all experiments imply one local epoch (*i.e.* client epoch). An exponential decay defined as $\eta_t = \eta_{start} \exp(\frac{t}{T} \log(\eta_{\text{start}}/\eta_{\text{end}}))$ with η_{start}, η_{end} the starting and last learning rates respectively is applied at training time. T represents the total number of communication rounds.

Client partitioning. Following previous works, we propose to compose a pool of 100 clients with 10 active clients training concurrently in a given round (McMahan et al., 2017).

Sparsity Ratios. This work considers accelerating the convolutions involved during forward and backward propagation following a `Top-K` sparsity inducing mechanism at the weight level. As a result, the expected sparse pattern would be unstructured, which can only be accelerated if tensors are sufficiently sparse. While *sufficient* is mostly hardware-specific, for the target platforms often considered in FL (e.g. mobile CPUs and GPUs) we set a minimum sparsity ratio (sp) of 90%, above which acceleration can be achieved (Wang, 2021). We include in our study $sp \in [0.7, 0.9, 0.95]$ in

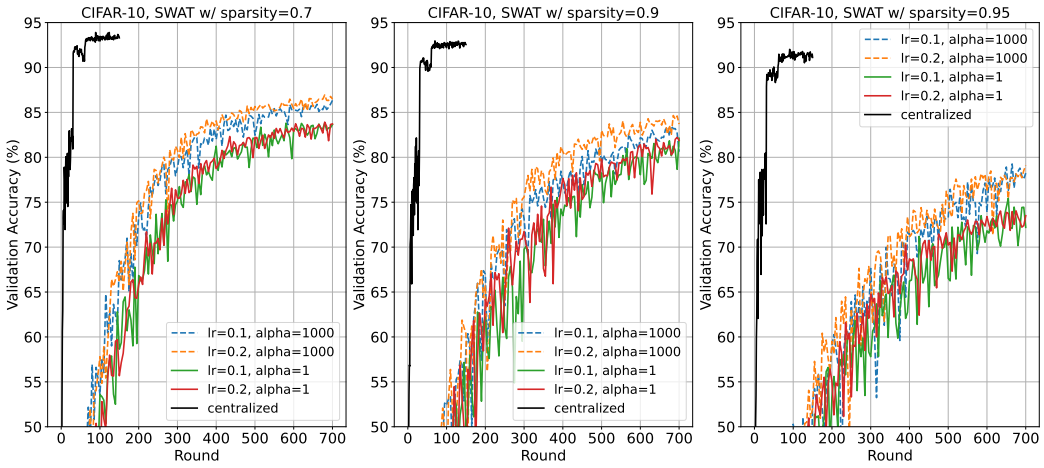

Figure 1: Comparison of validation accuracies in percentage of both centralised learning and FL on the CIFAR10 dataset with different sparsity and non-IID ratios. While centralised training suffers from minimal degradation at very high sparsity ratios (95%), the opposite happens for FL: we observe a 10% accuracy drop.

our initial evaluation. For $sp = 0.7$ we would expect minimal accuracy degradation for both IID and non-IID settings.

4.2 Baselines Results

We begin by studying the effect of applying SWAT in both centralised learning and FL with the CIFAR10 dataset, and the results are given in Fig. 1. Applying SWAT in centralised training does not impact the validation accuracy much despite the high level of sparsity level, which is equivalent to the results from the original paper (Raihan & Aamodt, 2020) with a validation accuracy reaching 91.21% with a sparsity level of 95% against 93.32% for 70%.

For FL, we first realise that the sparse training is particularly sensitive to the learning rate and its scheduler. In particular, exponential decay annealing is crucial to reach relatively good performance. It is also clear from the curves that a higher learning rate of 0.2 reaches better accuracies in general than 0.1. As expected, however, FL offers lower levels of performance across all setups compared to centralised training.

In addition, and conversely to centralised training, plots show consistent drops in the validation accuracy with the increase of the sparsity level. It is worth noticing that the validation accuracy decreases by 4.60% and 2.78% while the sparsity level increases from 90% to 95% and 70% to 90% respectively for IID ($\alpha = 1000$) settings. Then the validation accuracy drops by 8.3% and 1.84% when sparsity levels increase from 90% to 95% and 70% to 90% respectively for non-IID ($\alpha = 1$) settings. This highlights an important degradation of performance in non-IID settings with very high levels of sparsity.

4.3 Sparsification Effect Analysis

As shown in Fig. 1, high levels of sparsity with FL (*e.g.* $\geq 90\%$) induce an significant drop of accuracy that is more noticeable than for centralised training. We aim at understanding this phenomenon to further reduce the gap between centralised and FL training by properly adapting sparsification. As a first step, we propose to investigate the behaviour of the neural weights from clients to clients under SWAT and FL with a sparsity ratio equals to 90%.

Indeed, an undocumented effect of SWAT occurs at inference time and may motivate an extension of the technique to work properly with FL. During training, SWAT partitions the weights in two sets: active and non-active. The former set is used as a sparsity map during any forward propagation (*i.e.* both at training and inference times). In fact, a part of the weights are simply dropped from the neural network. This implies that the neural network must remain sparsified when evaluating and inferring

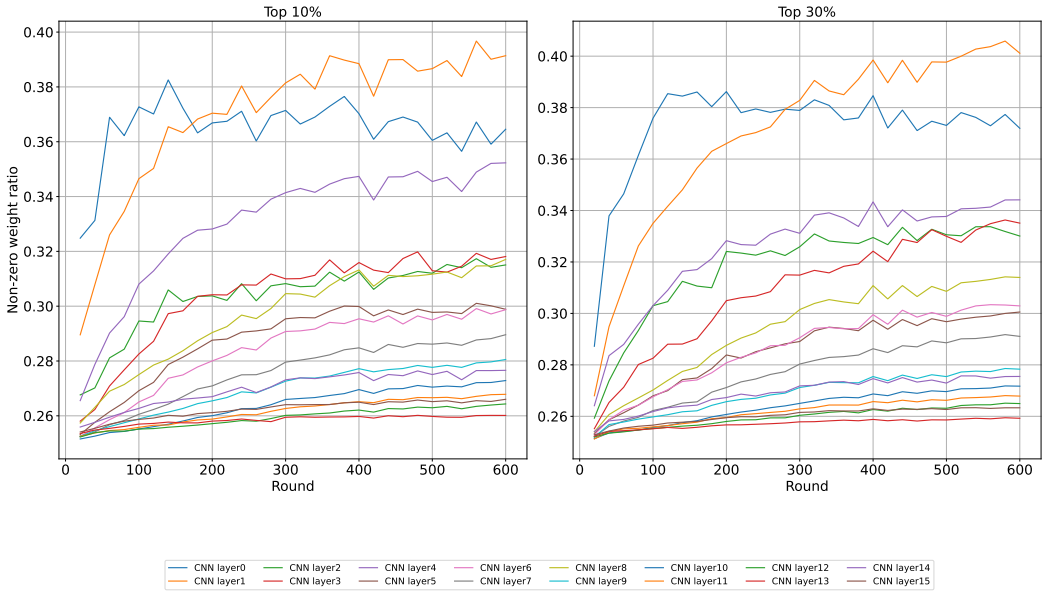

Figure 2: Evolution of the non-zero weights ratio after server aggregation (*i.e.* number of weights that are non-zero divided by the total number of parameter in that layer) of all CNN layers of a ResNet-18 trained on CIFAR10 with FL. Each of the 100 clients either send the `top-10` or `top-30` (*i.e.* weights with the highest norm) to the server.

or the accuracy will drastically drops. For instance, a FL trained system on CIFAR10 that reaches an accuracy of 82% on the validation set will drop to 10% if the inference is done without sparsification. It empirically validates that sparse training has an impact on the internal representation of a neural network making it dependent on the sparsification strategy during inference.

In practice, the latter behaviour is explained by the fact that, during centralised training, sparsified weights tend to be always the same *i.e.* some parameters are simply discarded. For FL, however, one may intuitively hypothesize that different clients will lead to different sparsification maps during the local training; preventing the creation of a global federated sparsification strategy. To investigate this, we propose to analyse the variations observed on the most active weights after aggregation on the server over the communication rounds.

Let us define the `top-K` weights as being the per-layer $K\%$ neural parameters with the highest norm. During each communication round, clients only send their `top-K` weights to the server while the remaining ones are set to zero. After aggregation on the server side, the resulting weight matrix informs us on the level of overlapping non-zero parameters observed on the clients. Indeed, every non-zero value obtained at a specific position will likely result in a non-zero value at the corresponding position in the aggregated weight matrix. For instance, if the number of non-sparse elements from the aggregated matrix is equivalent to the one of the clients ($K\%$) it means that all clients have the exact same `top-K` weights. With that in mind, we can define a non-zero ratio that is the number of non-zero parameters after aggregation divided by the the total number of elements in this layer. Thus, the higher this ratio is, the more different are the `top-K` weights sent from the clients. Fig. 2 depicts this ratio for different CNN layers over the communication round of a CIFAR10 training FL procedure with $K \in \{10, 30\}\%$. The non-IID parameter α is set to 1 as this setup is most likely to occur in realistic conditions.

First, it is clear that an significant overlap exists between clients across all CNN layers. Indeed, the non-zero ratio almost never exceeds 0.40, meaning that at least 60% of the weights are not comprised in the `top-K` weights of the clients. This advocates for the fact that the most important weights for the current task tend to be the same across clients. Then, we can see that the non-zero ratio does not seem to be hugely impacted by changing K from 10% to 30% as it only slightly increases for most CNN layers. This is explained by the fact that while clients may have different `top-10` weights, they tend to have the same `top-30` parameters *e.g.* a specific non-zero weight that is reported as

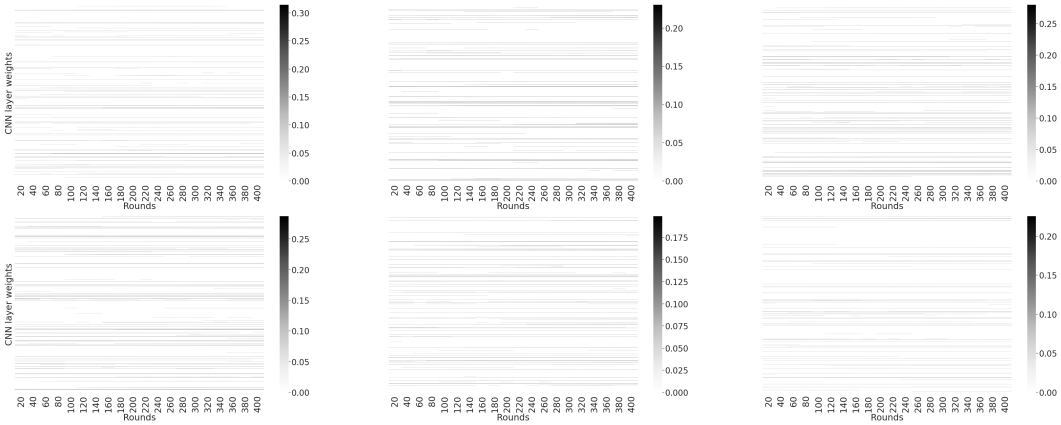

Figure 3: Heatmaps of 6 CNN layers (layer 4-9) in ResNet-18 when trained on CIFAR10 with 100 clients by only keeping the top 10% of weights. The weights are recorded every 20 communication rounds and *flatten* along the y-axis. The consistency across rounds (x-axis) indicates that, for the most part, the locations of non-zero weights remains constant.

a `top-10` element from a single client out of the selected ones is most likely to be reported as non-zero as well by more of them when we increase to `top-30`. In short, `top-30` will gather more information about weights that are considered as being *important* for the task while keeping the same level of sparsity.

Second, we propose to examine the exact positions of the `top-10` weights for certain CNN layers across a pool of 100 selected clients to better evaluate the overlap. Fig. 3 shows the weight matrix recorded every 20 communication rounds after aggregation on the server. Observations validate our intuition that most of zero and non-zero weights remain the same during the whole training.

Based on this analysis, we hypothesis that the degradation of performance observed with high levels of sparsity for FL is due to the dilution of important information during the aggregation process. For instance, a weight that would be only sent by a single client as part of its `top-10` parameters would not be *diluted* with the noise of all the others clients. Conversely, this very same weight may be completely corrupted if we send the entire dense model for aggregation.

5 ZEROFL: LOCAL SPARSIFICATION OF UPLINK COMMUNICATION

Motivated by our previous analysis which suggests that not all weights are necessary to be transferred to the central server for aggregation, we propose *ZeroFL*: a method that applies local sparsification before uplink communication. More precisely, we provide three strategies for local sparsification to improve the performance of sparse training while reducing the communication cost at the same time. Indeed, by leveraging local sparsification, we reduce the amount of information sent from the selected clients, hence reducing the noise aggregated on the central server observed in Section 4.3. In particular, if some updates are only sent by few clients, we force others clients to send zero in that particular positions of updates following our three strategies. After aggregation, the magnitude of these particular updates will be averaged with the uploaded values instead of being completely corrupted by the entire set of clients. All three methods are summarized in Algorithm 1.

Top-K-Weights. As shown in the previous section, only `top-K` active weights are involved in the validation and inference stages, and an important part of these weights tend not to change during training. Therefore, the first local sparsification method is to sparsify the `top-K` weights on the client-side before sending them back to the central server for aggregation. Then, and even though the positions of `top-K` weights largely overlap between each clients, only sending exactly a number of weights corresponding to the sparsity level might be too restrictive and prevents natural variations in the weights. Hence, we propose to introduce an other parameter denoted as the mask ratio r_{mask}, which indicates the additional amount of weights that the selected clients will choose to send to the central server after the local training.

Algorithm 1 *ZeroFL*: Let us consider a cluster of N total client with n local data set and each with a learning rate η_t at round t with T the total number of communication rounds. The client has the data set n_k. The number of local epoch is E and the number of clients participating in each round is denoted as K. w_t represent all the weights aggregated at round t and d_t the difference of weights.

Central server does:

for $t = 0, ..., T-1$ **do**
  Server randomly selects K devices.
  for all k in K **do**
    Perform TrainLocally(k, w_t)
  Aggregation:
  $w_{t+1} \leftarrow \sum_{k=0}^{K} \frac{n_k}{n} w_{t+1}^k$ ▷ Top-K-Weights
  $w_{t+1} \leftarrow w_t + \sum_{k=0}^{K} \frac{n_k}{n} d_{t+1}^k$ ▷ Diff on Top-K-Weights
  $w_{t+1} \leftarrow w_t + \sum_{k=0}^{K} \frac{n_k}{n} d_{t+1}^k$ ▷ Top-K Diff

TrainLocally(k, w_t)**:**

for $e = 1, ..., E$ **do**
  Do local model training via SWAT with sparsity level sp.
  $w_e \leftarrow w_{e-1} - \eta_t \bigtriangledown F(w_{e-1})$
Determine which weights to send for aggregation:
  Clients send top $1 - sp + r_{mask}$ weights to the central server. ▷ Top-K-Weights
  $d_{t+1}^k \leftarrow w_E - w_t$
  Clients send d_{t+1}^k of top $1 - sp + r_{mask}$ weights. ▷ Diff on Top-K Weights
  Clients send top $1 - sp + r_{mask}$ of d_{t+1}^k to the central server. ▷ Top-K-Weights Diff
return w_e or d_{t+1}^k

Let sp be the sparsity level of the local training. For instance, with $sp = 0.9$, and after the local training process, each selected client sparsifies their model by only keeping the top $(1 - sp + r_{mask})$ weights w.r.t their magnitude, while setting the rest to zero. In particular, if $sp = 0.9$ and $r_{mask} = 0.1$, the selected clients will send the `top-20%` to the central server for aggregation. The model produced after aggregation is not guaranteed to be sparse. If mask ratio equals to the sparsity level, the algorithm is degenerated to vanilla SWAT without local sparsification. However, and as shown in Fig.2, the resulting model is most likely to be sparse. Here, upling communications are saved as only $(1 - sp + r_{mask})$ weights are sent as dense values.

Diff on Top-K-Weights. The idea behind this method is to send local weight-updates for the top-k weights, rather than sending the top-k weights themselves. Given the sparsity level sp and the mask ratio r_{mask}, we first identify the weights that are the largest in magnitude by selecting the top $(1 - sp + r_{mask})$. The selected clients now send only the difference d of these top $(1 - sp + r_{mask})$ weights with respect to the originally received model w_t as $d_{t+1} = w_E - w_t$ with w_E the weights obtained after local training. The remaining differences are set to zero. In this way, after the aggregation in the central server, the weights that are not in the top $(1 - sp + r_{mask})$ part will remain the same as during the previous round, while only the top $(1 - sp + r_{mask})$ part of the weights will be updated.

Top-K-Weights Diff. Conversely to Diff on Top-K-Weights, this strategy proposed to first compute all the weights differences d and then only send the top $(1 - sp + r_{mask})$ of them to the server. With this method, only highly moving weights will be considered.

All the aforementioned local sparsification methods lead to substantial reductions in uplink communication costs. More precisely, total communications will be reduced by a factor of $(r_{mask} - sp)/2$.

6 ZEROFL: EXPERIMENTAL RESULTS

We conducted extensive experiments CIFAR10 following the same experimental protocol as for baselines experiments. The three local sparsification strategies are compared with various mask ratio $r_{mask} = \{0, 0.1, 0.2\}$ and to vanilla SWAT without ZeroFL. Similarly to results obtained in

Table 1: Experimental results with ZeroFL on CIFAR10 with non-IID data partition ($\alpha = 1.0$) and 100 clients. The table reports the highest test accuracy within 700 communication rounds. We report the size (in MB) of the artifact to be transmited to the server for aggregation. When models transmit a sparsified model, an efficient sparse representation (CSR) significantly reduces file size. ZeroFL improves the performance while reducing the uplink communication cost up to a factor of $7.4\times$ compared to vanilla SWAT. Best accuracy and communication saving numbers for each level of sparsity are reported in bold.

Sparsity Level	Mask Ratio	Test Accuracy (%) Dense	Top-K-W.	Diff. Top-K-W.	Top-K-W. Diff	File Size	Comms. Savings
	—	80.54				43.7	$1\times$
90 %	0.0		73.64	72.70	75.18	10.1	**4.3**$\times$
	0.1		81.08	80.70	80.80	18.7	$2.3\times$
	0.2		81.56	**82.64**	81.54	27.3	$1.6\times$
	—	73.59				43.7	$1\times$
95 %	0.0		65.80	65.72	65.84	5.9	**7.4**$\times$
	0.1		75.36	76.04	74.48	14.4	$3.0\times$
	0.2		**77.24**	75.96	76.94	23.0	$1.9\times$

Section 4.2, sparse training performs better with exponential learning rate decay. Hence, all setups are investigated with this scheduler with starting learning rates of 0.1 and 0.2. Table 1 reports the highest test accuracies achieved within 700 communication rounds.

First, it is worth noticing that all three local sparsification methods with mask ratios higher than 0.1 perform better than vanilla SWAT. The biggest improvement for 90% sparsity is achieved with the *Diff on Top-K-Weights* method with a mask ratio of 0.2, which increases the test accuracy by 2.1%. The largest improvement for 95% sparsity is achieved with the *Top-K-Weights* method with a mask ratio of 0.2, and the it increases the test accuracy by 3.74%.

Overall, performance improve with mask ratios comprised between 0 and 0.2, which indicates that there exist an optimal interval level. Indeed, a mask ratio equal to the sparsity level degenerates the system to the vanilla SWAT, which obtains worse results. It also implies that there is a trade-off between communication cost and performance. By using mask ratio of 0.1, each selected client performs local sparsification with an effective sparsity level of 80%, hence sending more data that for lower mask ratios values.

According to the results, ZeroFL enables us to reduce the performance degradation observed with high levels of sparsity with FL while not completely alleviating it. Then, during communications, the models are stored in the CSR sparse format using SciPy (Virtanen et al., 2020). Table 1 shows the level of compression and reduction in communication for different level of mask ratios. Compared to the original 43MB dense model, uplink communications are reduced by factors of $7.5\times$, $3.0\times$ and $1.9\times$ for mask ratios of 0.0, 0.1 and 0.2 respectively, with a sparsfication ratio of 95%.

7 Conclusion

In this work, we consider the challenges of inducing high level of sparsity to accelerate on-device training for federated learning. We provide the first framework to leverage sparsity as a mechanism to accelerate on-device training without the server imposing any restrictions. We study on the unique perspective that arise when introducing sparsity at training time, hence motivating us to propose innovative state-of-the-art off-the-shelve sparsification methods to the FL domain *ZeroFL*. The method achieves +2.1% and +3.7% higher accuracy while reducing $3.0\times$ uplink communication costs compared with competitive baselines. Our findings call for further investigations on the device-oriented optimisation of federated learning to motivate realistic deployments of this training methodology.

References

Mohammad Mohammadi Amiri, Deniz Gunduz, Sanjeev R. Kulkarni, and H. Vincent Poor. Federated learning with quantized global model updates, 2020.

Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. Federated learning with personalization layers, 2019.

Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2020.

Sebastian Caldas, Jakub Konečny, H. Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements, 2019.

Gary Cheng, Karan Chadha, and John Duchi. Fine-tuning is fine in federated learning, 2021.

Aiden Doherty, Dan Jackson, Nils Hammerla, Thomas Plötz, Patrick Olivier, Malcolm H Granat, Tom White, Vincent T Van Hees, Michael I Trenell, Christoper G Owen, et al. Large scale population assessment of physical activity using wrist worn accelerometers: The uk biobank study. *PloS one*, 12(2):e0169649, 2017.

Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Negar Goli and Tor M Aamodt. Resprop: Reuse sparsified backpropagation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1548–1558, 2020.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 1387–1395, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Pengchao Han, Shiqiang Wang, and Kin K Leung. Adaptive gradient sparsification for efficient federated learning: An online learning approach. *arXiv preprint arXiv:2001.04756*, 2020.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015a.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pp. 1135–1143, Cambridge, MA, USA, 2015b. MIT Press.

Andrew Hard, Chloé M Kiddon, Daniel Ramage, Francoise Beaufays, Hubert Eichner, Kanishka Rao, Rajiv Mathews, and Sean Augenstein. Federated learning for mobile keyboard prediction, 2018. URL `https://arxiv.org/abs/1811.03604`.

Andrew Hard, Kurt Partridge, Cameron Nguyen, Niranjan Subrahmanya, Aishanee Shah, Pai Zhu, Ignacio Lopez Moreno, and Rajiv Mathews. Training keyword spotting models on non-iid data with federated learning, 2020.

Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 620–629, 2018. doi: 10.1109/HPCA.2018.00059.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pp. 2234–2240. AAAI Press, 2018. ISBN 9780999241127.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Agrin Hilmkil, Sebastian Callh, Matteo Barbieri, Leon René Sütfeld, Edvin Listo Zec, and Olof Mogren. Scaling federated learning for fine-tuning of large language models, 2021.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, 2021.

Changwan Hong, Aravind Sukumaran-Rajam, Bortik Bandyopadhyay, Jinsung Kim, Süreyya Emre Kurt, Israt Nisa, Shivani Sabhlok, Ümit V. Çatalyürek, Srinivasan Parthasarathy, and P. Sadayappan. Efficient sparse-matrix multi-vector product on gpus. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, pp. 66–79, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357852. doi: 10.1145/3208040.3208062. URL https://doi.org/10.1145/3208040.3208062.

Changwan Hong, Aravind Sukumaran-Rajam, Israt Nisa, Kunal Singh, and P. Sadayappan. Adaptive sparse tiling for sparse matrix multiplication. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, PPoPP '19, pp. 300–314, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362252. doi: 10.1145/3293883.3295712. URL https://doi.org/10.1145/3293883.3295712.

Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos I Venieris, and Nicholas D Lane. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *arXiv preprint arXiv:2102.13451*, 2021.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

Jianxin Wu Jian-Hao Luo and Weiyao Lin. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. In *International Conference on Computer Vision (ICCV)*, October 2017.

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.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 2351–2363. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/18df51b97ccd68128e994804f3eccc87-Paper.pdf.

Lumin Liu, Jun Zhang, Shenghui Song, and Khaled B. Letaief. Hierarchical quantized federated learning: Convergence analysis and system design, 2021.

L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang. An Efficient Hardware Accelerator for Sparse Convolutional Neural Networks on FPGAs. In *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 17–25, 2019.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2498–2507. PMLR, 06–11 Aug 2017. URL http://proceedings.mlr.press/v70/molchanov17a.html.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance Estimation for Neural Network Pruning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Md Aamir Raihan and Tor M Aamodt. Sparse weight activation training. *arXiv preprint arXiv:2001.01969*, 2020.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. Sbnet: Sparse blocks network for fast inference, 2018.

A. Radford S. Gray and D. P. Kingma. Block-sparse gpu kernels, 2017. URL `https://blog.openai.com/block-sparse-gpu-kernels/`.

Osama Shahid, Seyedamin Pouriyeh, Reza M. Parizi, Quan Z. Sheng, Gautam Srivastava, and Liang Zhao. Communication efficiency in federated learning: Achievements and challenges, 2021.

N. Srivastava, H. Jin, S. Smith, H. Rong, D. Albonesi, and Z. Zhang. Tensaurus: A Versatile Accelerator for Mixed Sparse-Dense Tensor Computations. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 689–702, 2020.

Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. meProp: Sparsified back propagation for accelerated deep learning with reduced overfitting. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3299–3308, International Convention Centre, Sydney, Australia, 2017.

Thomas Verelst and Tinne Tuytelaars. Segblocks: Block-based dynamic resolution networks for real-time segmentation, 2020.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

Hongyi Wang, Scott Sievert, Zachary Charles, Shengchao Liu, Stephen Wright, and Dimitris Papailiopoulos. Atomo: Communication-efficient learning via atomic sparsification. *arXiv preprint arXiv:1806.04090*, 2018.

Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BkluqlSFDS`.

Yunhe Wang, Chang Xu, Chao Xu, and Dacheng Tao. Beyond Filters: Compact Feature Map for Portable Deep Model. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 3703–3711, 2017.

Ziheng Wang. Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, PACT '20, pp. 31–42, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380751. doi: 10.1145/3410463.3414654. URL `https://doi.org/10.1145/3410463.3414654`.

Ziheng Wang. Sparsednn: Fast sparse deep learning inference on cpus, 2021.

N. Wen, R. Guo, B. He, Yong Fan, and Ding Ma. Block-sparse cnn: towards a fast and memory-efficient framework for convolutional neural networks. *Applied Intelligence*, 51:441–452, 2020.

R. Yu, A. Li, C. Chen, J. Lai, V. I. Morariu, X. Han, M. Gao, C. Lin, and L. S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9194–9203, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society. doi: 10.1109/CVPR.2018.00958. URL `https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00958`.

Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *International Conference on Machine Learning*, pp. 7252–7261. PMLR, 2019.

Orestis Zachariadis, Nitin Satpute, Juan Gómez-Luna, and Joaquín Olivares. Accelerating sparse matrix–matrix multiplication with gpu tensor cores. *Computers Electrical Engineering*, 88: 106848, Dec 2020. ISSN 0045-7906. doi: 10.1016/j.compeleceng.2020.106848. URL http://dx.doi.org/10.1016/j.compeleceng.2020.106848.

S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen. Cambricon-X: An Accelerator for Sparse Neural Networks. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, 2016.

Zhuangdi Zhu, Junyuan Hong, and Jiayu Zhou. Data-free knowledge distillation for heterogeneous federated learning, 2021.