École Polytechnique Fédérale de Lausanne

Master's Thesis

Secure Aggregation on Sparse Models in Decentralized Learning Systems

presented by

Milos Vujasinovic

Supervisors:

Prof. Dr. Anne-Marie Kermarrec

Rishi Sharma

Dr. Rafael Pereira Pires

January 2023

"The fantastic advances in the field of electronic communication constitute a greater danger to the privacy of the individual."

– EARL WARREN

Contents

Abstract

Decentralized learning (DL) has shown promising potential for performing machine learning (ML) on sensitive distributed data. Compression techniques such as sparsification make DL communication-efficient, and hence, practical. Recent discoveries of privacy attacks against ML models have inspired work on mechanisms that enable privacy-preserving model aggregation in DL, with secure aggregation being one of the practical approaches. Unfortunately, sparsification and secure aggregation cannot be trivially combined together in DL settings.

In this paper, we introduce Communication-Efficient Secure AggRegation (CESAR), a DL system that performs secure aggregation of model parameters compatible with any sparsification technique. For a given topology, the nodes in CESAR agree on pairwise masks and the parameters to share each round with their neighbors. These masks cancel out during model aggregation, resulting in each node having the aggregated model, but can never isolate individual models. CESAR is safe against honest-but-curious adversaries. We empirically show that CESAR provides the benefits of secure aggregation with as little as 30% communication overhead over the standard DL algorithms using the same sparsification technique in a given network topology, while achieving similar or better accuracy.

Introduction

We have recently seen several breakthroughs in ML [1, 2, 3] aided by ingenious ways of utilizing large collections of publicly available data that greatly benefit such models [4]. The scope of these applications is however rather limited to a small set of fields where data is available. This can be overcome by combining data owned by multiple parties, but an issue arises when movement of such data is limited. It can be restricted by the data being either confidential, sensitive, or by various regulations such as Health Insurance Portability and Accountability Act (HIPAA) [5] and General Data Protection Regulation (GDPR) [6]. Majority of data for health applications, as well as that generated by our cellphones and wearable devices [7] is sensitive in nature. Hence, improving human lives through it, both in health, as well as convenience terms, requires keeping the data protected while extracting utility benefits out of it.

Federated learning (FL) [8] has been proposed as one of solutions for training ML models on sensitive data without moving the data to models, but rather moving models to the data. This gives reassurance that the data won't be directly seen by any other participant in the training. As such FL has seen use in variety of applications including emoji [9] and text prediction [10, 11], human activity recognition [12], predicting hospitalizations for cardiac events [13] and mortality chance [14] as well as many more.

Unfortunately, recent research on privacy of ML models point out to there being artifacts of training data left in the models during training that can be

later used to leak information about it [15, 16]. Variety of mechanisms have been proposed to prevent local models in FL being seen in plaintext, with secure aggregation [17] being a promising one. Secure aggregation relies heavily on obfuscating model data, removing any patterns increasing its entropy which makes usage of compression extremely hard. This is a serious issue on cellphones that are often on mobile data where every additional MB makes a difference. Sparsification presents itself as a natural solution in this case as it aims to select parts of model to share rather than focus on the way this parts are shared. Nonetheless, sparsification is still incompatible with popular secure aggregation protocols, so in this thesis we focus on creating a secure aggregation protocol that works with any form of sparsification.

On top of all this, FL relies heavily on the central server in every step of training which makes it a single point of failure, hence targeted attacks against the server lead to the service going down. For this reason, we focus on DL which works in a similar setting where all nodes are equal and it has been shown capable of outperforming FL [18].

[19] and [20] explored this topic in FL setting and their works share similarities with our own. The solution of [19] involves introducing a new distributed sparsification technique that is fixed and therefore the protocol doesn't work with general forms of sparsification. On the other hand, [20] proposes a solution that focuses primarily at TopK sparsification, but has potential to be applied with other techniques as well. Their solution relies on all clients sharing the union of indices selected by each, hence forcing every client to be either extremely selective in their sparsification making underlying sparsification technique less important, or otherwise risking sending nearly all indices after the union and mitigating the purpose of sparsification altogether. On the other hand, our solution relies on pairwise intersection of indices between nodes and guarantees that a node won't send anymore indices other than those proposed for sending by itself.

Preliminaries

3.1 Decentralized learning

Decentralized learning (DL) allows a set of nodes $\mathcal{N} = \{n_1, n_2, \ldots, n_k\}$ to train a ML model on their joint data without sharing it and with no presence of a central server.

Each node n_i owns a private dataset $D_i = \{(x_i, y_i)\}_i$ drawn from a hidden distributions ξ_i and has access only to its own data. Combining the private datasets would produce the global dataset D with distribution ξ. The data is called non-IID if local distributions ξ_i are skewed in comparison to the global distribution ξ, and IID otherwise.

The training aims to find parameters θ^* for a ML model that minimize the expected loss on the global dataset D:

$$\theta^* = \arg\min_{\theta} \frac{1}{|\mathcal{N}|} \sum_{n_i \in \mathcal{N}} \underbrace{\mathbb{E}_{s_i \sim D_i}\big[\mathcal{L}(\theta; s_i)\big]}_{\mathcal{L}_i} \tag{3.1}$$

The training consists of several rounds, each divided in three distinct stages. At the beginning of each round every node runs several iterations of mini-batch stochastic gradient descent [21] with the local data on their view of the model (or *local model* for short). In the second stage nodes share their local views of the model with other nodes in the network. The view is most often shared only with neighbors, either all (e.g. D-SGD [22]) or a subset of them (e.g. random

model walk (RMW) [23]). Finally, upon receiving all expected views or after a specified timeout, a node aggregates the received models. The aggregation is performed by doing weighted or unweighted averaging of the received views and the node replaces the local view of the model with the result.

3.2 Privacy attacks and secure aggregation protocol

The parameters of a ML model depend on the data the model has been trained on. Recent research on the privacy of collaborative learning shows that these models are vulnerable against variety of attacks capable of recovering training data [24, 25, 26] or inferring the presence of data in the private dataset [15, 16]. However, it is important to notice that an honest node in DL doesn't need any specific model of another node, but rather the aggregated value of models belonging to multiple nodes. Even though the aggregated model may be vulnerable against these attacks, the benefits of not revealing specific models are twofold:

- The attacks cannot be applied to specific local models, hence the adversary cannot directly infer the exact location of the data

- Multiple nodes contribute to the aggregated model therefore minimizing signal of each individual node and reducing the attack surface on each individual data

Secure aggregation is a group of protocols that calculate the average value of multiple inputs, each submitted by a different node, without revealing any specific input to other nodes in the protocol under certain set of assumptions.

3.3 Sparsification

In DL every round a node must send its local view of the model to multiple other nodes in the network, as well as receive their views. Transmitted models are usually large in size, and this causes the communication speed to become the bottleneck of the training. This can be eliminated by either designing algorithms that exchange less messages or reducing the size of each message.

Sparsification presents a way of reducing the size of exchanged models. It is a lossy compression technique usually used on gradients in FL representing them as a subset of their values selected using a specific criteria. However, FL has a single global model and working with gradients in this setting is effective. On the other hand, in DL every node has its own view of the model and views of different nodes are in different positions in the parameter space. Hence, averaging gradients does not produce as good results as it does in FL. Nevertheless, sparsification can be analogously used on model parameters. It is often used to improve communication efficiency of DL algorithm: instead of communicating full model, clients exchange the selected indices. Because some information of a model is lost in sparsification, the resulting models trained with sparsification in DL often have lower utility compared to models trained without it. Furthermore, in order to recover the selected parts of the original model by a receiving node, besides the values of selected parameters, their positions must be provided as well.

Random subsampling. Parameters can be chosen at random, and this approach is called *random subsampling* [27]. Various distributions can be used to sample parameters, but in this thesis we implicitly assume the use of uniform distribution due to its unbiased nature. The choice of distribution is often considered public and same for all nodes. Hence, if the same pseudo-random number generator is used at all nodes, they can benefit by exchanging only seeds used for generating their random masks, rather than positions of all selected pa-

rameters.

TopK sparsification. Other sparsification techniques usually rank parameters and select a specified percentage of the highest-ranked ones. For example, the parameters with the highest magnitude can be selected. However, this approach often leads to sharing the same or a similar set of parameters in every round. Therefore, more often parameters with the highest magnitude of change since the previous round are selected which we refer to as TopKsparsification.

Incompatibility with secure aggregation. Sparsification is performed separately on every node without communicating with others. This results in different indices being selected for sharing by different nodes. On the other hand, secure aggregation often relies on pairs of nodes agreeing on masks that cancel out, but if the pair selected different sets of indices for sharing they can't apply masks on indices without a pair, because they won't cancel out. However, if the pair applies the mask only on indices they both share they risk that some indices will be left unmasked and the privacy of the unmasked parameters might be endangered. This creates an issue of using sparsification with secure aggregation and it's the main motivation behind this thesis.

Protocol

In this section we introduce a basic version of the protocol that offers privacy against honest-but-curious adversary.

4.1 System

In our setting there is a network with a set of processes (or *nodes* used interchangeably) $\mathcal{N}$. We assume there already exists an underlying mechanism enabling the processes to agree on a common graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ between them. We call this graph *aggregation graph*. In its simplest form, an aggregation graph can be fixed and loaded during the initialization of the process. The purpose of aggregation graph is defining neighborhoods in which models are aggregated, and it should be strictly distinguished from the graph defining available communication channels between nodes. We assume there to be a bi-directional communication channel between every pair of nodes, but also later discuss how this can be relaxed. Furthermore, every process in $\mathcal{N}$ must, at minimum, have the knowledge of a subgraph containing all nodes in $\mathcal{G}$ at distance ≤ 2 from itself.

4.2 Threat models

In privacy analysis later we consider *honest-but-curious adversary*: nodes do not deviate from the protocol, but they will attempt to infer as much as possible from information available to them. We assume no collusion between the nodes. In the real world malicious nodes may deviate from the protocol and they are called *Byzantine nodes*, however, in this work we do not address this problem.

4.3 Algorithm

In Algorithm 1 we show the pseudo-code of the protocol. The protocol consists of two distinct steps.

In the first step, or *prestep*, every pair of nodes with a common neighbor exchanges indices they intended to share, but also send their contribution to a pairwise additive mask between them. The mask is stored at each node to be later applied on the intersection of the indices of the pair. For the sake of simplicity nodes agree on a mask by each generating its own mask over the given intersection of indices and sending it to the other node. The final mask is obtained by subtracting the received mask from that generated by the given node. This way each mask is an additive inverse of the other mask in the pair. However, creating masks in this way poses a privacy risk as one node can wait to receive the mask of the other and craft its own mask to weaken the overall mask of the pair. The simplest way to do this is selecting the same mask as the other node and therefore totally eliminating both shares of the pairwise mask. Hence, in [17] we see Diffie–Hellman key exchange being used to agree on a common seed for a pairwise additive mask, and, ideally, it would be used here too.

In the second step, every node i goes over its neighbors and for each neighbor k it adds masks it agreed upon with other neighbors of k to its model. Then, i selects all parameters that have at least one mask applied on them and sends

Algorithm 1: CESAR running in parallel on every Node i

Input: Accepts list of indices selected in sparsification I_i, all parameters of the local X, and aggregation graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ in which aggregation takes place

Output: The aggregated model X' of the node's neighborhood

for $j \in \mathcal{N} : i \neq j$ **do**
  if $\exists k \in \mathcal{N} : \{i,k\} \in \mathcal{E} \wedge \{j,k\} \in \mathcal{E}$ **then**
    Generate pseudo-random number $seed_{ij}$
    Send $(I_i, seed_{ij})$ to node j
for $j \in \mathcal{N} : i \neq j$ **do**
  if $\exists k \in \mathcal{N} : \{i,k\} \in \mathcal{E} \wedge \{j,k\} \in \mathcal{E}$ **then**
    $(I_j, seed_{ji}) \leftarrow$ Receive from node j
    $I_{ij} \leftarrow I_i \cap I_j$
    $M_{ij} \leftarrow$ Random mask over I_{ij} with seed $seed_{ij}$
    $M_{ji} \leftarrow$ Random mask over I_{ij} with seed $seed_{ji}$
    $M'_{ij} \leftarrow M_{ij} - M_{ji}$
for $k \in \mathcal{N} : \{i,k\} \in \mathcal{E}$ **do**
  $X_{ik} \leftarrow X$
  $C \leftarrow [0]$ of same dimension as I
  for $j \in \mathcal{N} : i \neq j \wedge \{i,k\} \in \mathcal{E} \wedge \{j,k\} \in \mathcal{E}$ **do**
    $X_{ik}[I_{ij}] \leftarrow X_{ik}[I_{ij}] + M'_{ij}$
    $C[I_{ij}] \leftarrow 1$
  $I'_{ik} \leftarrow p \in I : C[p] = 1$
  $Z_{ik} \leftarrow X_{ik}[I'_{ik}]$
  Send (I'_{ik}, Z_{ik}) to node k
$X' \leftarrow X$
$c \leftarrow 1$
for $k \in \mathcal{N} : \{i,k\} \in \mathcal{E}$ **do**
  $(I'_{ki}, Z_{ki}) \leftarrow$ Receive from node k
  $X'_k \leftarrow X$
  $X'_k[I'_{ki}] \leftarrow Z_{ki}$
  $X' \leftarrow X' + X'_k$
  $c \leftarrow c + 1$
$X' \leftarrow \frac{X'}{c}$

them to k. Finally, nodes calculate the mean of models they receive together with their own and continue training with the given result. For each additive mask sent to a node, there is the inverse pair of the mask among the other neighbors of the same node and by performing summation, required to calculate the mean, all masks cancel out.

Notice that parameters sent by node i to node k are dependant not only on indices i selected for sharing, but also on indices selected by other neighbors of k. Therefore, the same node can send different parameters to each of its neighbors.

Moreover, the masks between each pair of nodes are selected independently meaning that the value of a same parameter sent to two nodes can be different if their neighbors don't fully overlap.

Ideally, we have a channel between every pair of nodes so they can freely communicate with each other. However, in some cases this may not be true, and communication channels may exist only between nodes connected in the aggregation graph. Nodes at distance 2 are very close in the graph and hence the paths between them are simple, they have only their common neighbors directly in between. These common neighbors can be therefore used to relay messages between these nodes if there is no direct communication channel between them. It is important to emphasize that this doubles the number of messages in the prestep and therefore it isn't recommended unless absolutely necessary.

4.4 Intersection analysis

The introduced protocol has a property that it doesn't share all indices a node running it intended to share, but a subset of the indices. Rather than observing this selection process as a black-box it is beneficial to understand how parameters sent to other nodes in CESAR depend on the parameters of the system. In this section we analyze the probabilities that govern this behaviour.

The protocol aims to be used on top of any sparsification technique, and therefore it is expected that they may behave vastly different. For the sake of analysis, we impose a soft requirement that parameter selection process between nodes must be mutually independent, and that given a set of selected parameters every parameter of the model on which the selection was performed had uniform expected probability to be selected. In practice these requirements are very rarely fully satisfied, but some sparsification techniques behave similarly to the requirement, and the closer the behaviour is, more applicable results of the analysis are for the given technique. For instance, Random subsampling satisfies the requirement. Also, our evaluation in Chapter 5, as well as [19] show that

TopK gradients sparsification for non-IID data behaves similarly.

Let d be the total number of parameters in a model and assume that every node is expected to intend to share αd parameters ($\alpha \in [0, 1]$), for simplicity we will refer to this as α parameters being shared.

We look into an aggregation graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ and a specific node $n \in \mathcal{N}$ with its neighbors $N(n) = \{m_0, m_1, \ldots, m_k\} \subseteq \mathcal{N} \setminus \{n\}$ preparing to send n their masked models. Without loss of generality we analyze what percentage of parameters neighbor $m_i \in N(n)$ is expected to send in this case. We let $A_{m_i}(x)$ be 1 if neighbor m_i designated parameter x for sharing before the protocol, and 0 otherwise. Assuming uniform probability of sharing, every parameter has α probability to be selected for sharing before being passed to the protocol. If we assume all nodes in $N(n)$ intend to share the same percentage of parameters α, we can write:

$$P(A_m(x) = 1) = \alpha, \forall m \in N(n). \tag{4.1}$$

By the formulation of the protocol, m_i will share a parameter with n if the parameter is designated for sharing by m_i and at least one of other neighbors of n. Therefore, the he probability of parameter x being shared by m_i with n can be expressed as:

$$P\Big(S_{m_i,n}(x; \mathcal{G}) = 1\Big) = P\Big(A_{m_i}(x) = 1\Big) \cdot P\Big(\bigcup_{m \in N(n): m \neq m_i} (A_m(x) = 1)\Big). \tag{4.2}$$

Using basic probability transformations we calculate:

$$P\Big(\bigcup_{m\in N(n):m\neq m_i}(A_m(x)=1)\Big) = 1 - P\Big(\bigcap_{m\in N(n):m\neq m_i}(A_m(x)=0)\Big) \tag{4.3}$$

$$= 1 - \prod_{m\in N(n):m\neq m_i} P(A_m(x)=0) \tag{4.4}$$

$$= 1 - \prod_{m\in N(n):m\neq m_i} (1 - P(A_m(x)=1)) \tag{4.5}$$

$$\overset{(4.1)}{=} 1 - \prod_{m\in N(n):m\neq m_i} (1-\alpha) \tag{4.6}$$

$$= 1 - \Big(1-\alpha\Big)^{|N(n)|-1}. \tag{4.7}$$

Finally, by combining (4.1) and (4.7) into (4.2) we obtain:

$$P\Big(S_{m_i,n}(x;\mathcal{G})=1\Big) = \alpha\cdot\Big(1-\big(1-\alpha\big)^{|N(n)|-1}\Big). \tag{4.8}$$

Because we assumed uniform and independent selection of parameters, the overall percentage of parameters being sent by m_i to n is equal to the probability of a single parameter being sent. Because we consider every node to share the same percentage of parameters we annotate the percentage of parameters received by n from each of its neighbors with β:

$$\beta(\alpha, deg) = \alpha\cdot\Big(1-\big(1-\alpha\big)^{deg-1}\Big) \tag{4.9}$$

where $deg = |N(n)|$.

We see that the percentage depends on the intended percentage of parameters to be sent before the protocol, as well as the degree of a node the masked model is sent to. Using simple analysis we can observe changes in β for α and deg:

$$\frac{\partial\beta(\alpha,deg)}{\partial\alpha} = 1-\big(1-\alpha\big)^{deg-1} + \alpha(deg-1)(1-\alpha)^{deg-2} \tag{4.10}$$

$$\frac{\partial\beta(\alpha,deg)}{\partial\,\mathrm{deg}} = -\frac{\alpha}{1-\alpha}(1-\alpha)^{deg}\ln(1-\alpha) \tag{4.11}$$

The protocol makes sense only for values $\alpha \in [0, 1]$ and $deg \in \{2, 3, 4, \dots\}$ and in this interval both (4.10) and (4.11) are non-negative. Hence, the percentage of parameters that ends up being sent between nodes in our setting can be increased by either increasing the degree of the node receiving parameters or by increasing intended percentage of parameters for sharing at each sending node. Figure 4.1 visualizes this behavior of β. It shows that for higher α the relative initial shared percentage loss is lower compared to smaller values. Furthermore, higher α results in needing less neighbors to start converging towards the initial value.

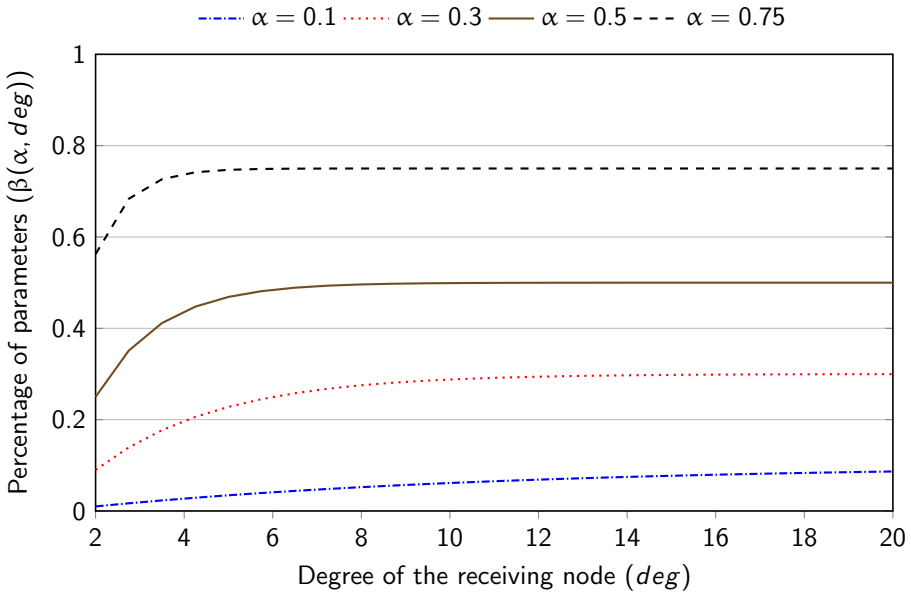

Figure 4.1: Theoretical dependence of the percentage of parameters being sent to another node in the protocol as the function of the intended percentage of parameters to be sent before the protocol (α) and the degree of the receiving node

4.5 Perturbing indices

In the first step of the protocol selected indices are shared among nodes in the non-obfuscated form. Often, however, sparsification process utilizes information about parameter values to select indices. For instance, TopK approach selects

indices whose parameters have the highest magnitude. As these values are dependant on the training data it has potential to leak sensitive information about.

The authors of [20] introduce the idea of adding indices that haven't been selected in sparsification and adding them to the selected ones with random probability. This way, every shared indice has < 1 probability to have been selected in sparsification, which reduces the attack surface of possible inference attacks. Formally, they consider a model with d parameters, and k out of d parameters being selected for sharing in sparsification. Out of $d-k$ non-selected indices they select μk at random where μ is a parameter specified by the user. The probability that a shared indice has been selected in sparsification therefore is $\frac{k}{k(1+\mu)} = \frac{1}{1+\mu}$ which converges towards 0 for large μ.

Beside privacy, this idea provides additional benefits. Selecting additional indices uniformly at random increases the number of intersections in an unbiased way among indices that wouldn't have an intersection pair otherwise. Furthermore, uniform indice selection brings this composite sparsification technique closer to the theoretical model introduced in section 4.4.

Unlike [20], we formulate this idea differently. We look at a model with d parameters and a node $m \in \mathcal{N}$ selects $\lceil \alpha d \rceil$ of them in a sparsification technique that satisfies the requirements of our theoretical model, i.e. indices are selected with uniform probability and independently among nodes. We are interested in percentage γ of additional indices that should be selected uniformly at random in order for a neighbor $n \in \mathcal{N}$ with degree deg to receive $\lceil \beta d \rceil$ indices from m. Furthermore, we assume that all neighbors of n also initially select $\lceil \alpha d \rceil$ and add the same number of non-selected indices uniformly at random.

After combining perturbated with initially selected indices we have the total percentage of indices being

$$\alpha' = \alpha + \gamma. \tag{4.12}$$

Dependence between α' and β can be expressed with (4.8):

$$\beta = \alpha' \cdot \Big(1 - \big(1 - \alpha'\big)^{deg-1}\Big). \tag{4.13}$$

Next, we do substitution $\alpha' = 1 - \delta$:

$$\beta = (1 - \delta) \cdot (1 - \delta^{deg-1}). \tag{4.14}$$

This can be transformed into a polynomial for δ:

$$\delta^{deg} - \delta^{deg-1} - \delta + (1 - \beta) = 0. \tag{4.15}$$

We are looking for real roots of this polynomial with $\delta \in [0, 1]$ that can be used to calculate $\alpha' = 1 - \delta$. If there is no such solution it is impossible to select α' percentage of parameters before the protocol in order to send β percentage of parameters to the neighbor. Even though finding the roots of an arbitrary polynom is a hard task, often programming tools offer functionality to find or very least approximate them, as is example of function `numpy.roots` that belongs in Python package `numpy` used in our implementation.

When we know to calculate α' we can obtain γ as

$$\gamma = \alpha' - \alpha. \tag{4.16}$$

However, we know that $\lceil \gamma d \rceil$ of remaining parameters should be selected, yet we do not know with which probability we should sample each remaining parameter to obtain that many. Because we sample with uniform probability every remaining parameter should be sampled with equal probability γ' and therefore we can divide the number of parameters needed to be selected with the

remaining number of parameters:

$$\gamma' = \frac{\gamma d}{d - \alpha d} \tag{4.17}$$

$$= \frac{d(\alpha' - \alpha)}{d(1 - \alpha)} \tag{4.18}$$

$$= \frac{\alpha' - \alpha}{1 - \alpha}. \tag{4.19}$$

4.6 Privacy guarantees

The purpose of using secure aggregation is to protect against exposing local models to adversaries during collaborative training. CESAR provides resistance against honest-but-curious adversary as it is shown in Theorem 1.

Theorem 1 (Resistance against honest-but-curious adversary)**.** *In Algorithm 1 no honest-but-curious adversary can learn the exact value of a parameter of a local model of another node.*

Proof. Let $n \in \mathcal{N}$ and $\{m_0, m_1, \ldots, m_k\} \subseteq \mathcal{N} \setminus \{n\}$ be its neighbors. During the protocol, node n only receives masked models of its neighbors. Without loss of generality we look at a parameter at an arbitrary position p.

If n didn't received parameter p from any neighbor, it trivially doesn't learn anything.

If n received parameter p from neighbor m_{a_0} it means that m_{a_0} must have applied at least one mask at parameter p. By the algorithm if m_{a_0} has applied at least one mask there must be a non-empty exhaustive set of the neighbors $\{m_{a_1}, \ldots, m_{a_l}\}$ that share pairwise masks with m_{a_0} for p. Hence, each node of set $\mathcal{A} = \{m_{a_0}, m_{a_1}, \ldots, m_{a_l}\}$ ($|\mathcal{A}| \geq 2$) must have intended to share p. Furthermore from the algorithm, every pair $\{i, j\}$ of nodes from $\mathcal{A}$ must have agreed on a random pairwise mask $M_{ij}^{'(p)} = -M_{ji}^{'(p)}$ for p. For example, if $v_m^{(p)}$ is a value of parameter p at m before the protocol, and $w_m^{(p)}$ is the value received by n from

$m \in \mathcal{A}$ for parameter p, we know that

$$w_m^{(p)} = v_m^{(p)} + \sum_{o \in \mathcal{A} \setminus \{m\}} M_{mo}^{'(p)}.$$

Because masks are chosen independently from each other, and their values are private to the pair that generated it, the only way for node n to retrieve $v_m^{(p)}$ from $w_m^{(p)}$ is to cancel out all masks in $w_m^{(p)}$. In order to do this n must add to $w_m^{(p)}$ all values for p received from other nodes that share a pairwise mask with m in equal proportion. Therefore, the most node n can infer from the information available to it is the sum of values

$$\sum_{m \in \mathcal{A}} w_m^{(p)} = \sum_{m \in \mathcal{A}} v_m^{(p)}.$$

From just the sum no specific contributing to it can be inferred.

□

Remark 1 (Following Theorem 1)**.** *A node n won't send any parameter to a node m if n is the only neighbor of m.*

4.7 Limitations

CESAR focuses on commnicationally efficient and privacy preserving aggregation of ML models in decentralized learning setting, however there are several additional challenges to be overcome for a similar protocol to be usable in practice.

Failing nodes. Nodes often fail and this may happen during the execution of the protocol. In such situation the specification given in Algorithm 1 would get stuck and wait for results forever. Majority of similar work in FL inspired by [17] uses Shamir [28] secret sharing to recover secrets of failed nodes and later pairwise masks using them. However, [17] show that the runtime of the server increases quadratically with the dropout rate of clients due to expensive process

of recovery. In DL setting where there is no big servers, this is especially costly. Because the number of neighbors each node has is considered to be relatively small compared to the size of the network, we consider repeating the protocol when a node fails a more suitable way of handling failures until a better solution is found.

Byzantine nodes. The presence of Byzantine nodes is another big concern not addressed by CESAR. A particular danger are poisoning attacks [29, 30] that aim to adjust weights of local models such that little can be learnt by the final model. [31] proposes usage of two servers to make Byzantine robust secure aggregation, but we are not aware of any work trying to address this concern in a fully decentralized setting.

Asynchronous aggregation. Secure aggregation is often considered in synchronous sense: every node must wait for others to finish their aggregation to proceed further. This often makes sense in FL where everything is closely interconnected. In such setting struggling clients might be considered failed due to their slow speed. On the other hand, in DL this synchrony may force a node at one side of the network to have to wait for a slow node at the opposite side to finish. Some decentralized learning algorithms, such as D-SGD address this issue, however, secure aggregation requires precise cooperation between nodes which adds a layer of complexity that makes creation of asynchronous protocols extremely hard.

Collusion. This thesis considers only a passive adversary working alone, but in the real world multiple adversaries might be working together to learn more. In fact, our algorithm can be modified to provide verifiable protection against colluding adversaries given only one receives model parameters: instead of discarding parameters that don't have any mask applied, for each parameter count how many masks have been applied on top of it and discard those where the count is lower than a specified threshold. If k masks are required to be applied

on a parameter, it is provable that at least k adversaries have to collude in order to reveal the parameter given that no more than one adversary receives parameter values. However, each adversary participates in their own aggregation and obtains aggregated models like any other node. Hence, in some configurations of network it may happen that 2 colluding adversaries have a single node m as difference in their neighborhoods, for example neighbors of one are $\{n_1, n_2, \ldots, n_k, m\}$, while of the are $\{n_1, n_2, \ldots, n_k\}$. In such setting one adversary can easily derive the sum of models of $\{n_1, n_2, \ldots, n_k, m\}$ from the final result, and the other of $\{n_1, n_2, \ldots, n_k\}$. By subtracting the two sums, the model of m is revealed. This can be generalized to more adversaries in less straightforward cases, and it shows that even though secure aggregation may require collusion of k adversaries to reveal local models performing it, it is enough to wait for it to finish in order to use the results to leak models with as little as 2 colluding nodes. This opens up an argument on the effectiveness of using secure aggregation in settings without a single global model, as well as of possibilities of restricting network structure such that these attacks can be prevented.

Accumulation. Variety of sparsification techniques rely on tracking changes of parameters over communications rounds and carrying these results later into training to assure optimal parameter selection. For example, TopK sparsifications may track how much each parameter changed since last being sent rather than since the last comunication round. This works when the same set of model parameters is sent to every neighbor like D-SGD does. However, set of parameters sent by CESAR to two neighbors in the same communication round may differ, hence keeping a single track of parameters being sent by a node doesn't work. On the other hand, keeping different statistics for all neighbors results in different sets of indices being selected for each in sparsification. This provides additional information to adversary which can be used to increase the success of inference attacks that reveal properties of parameters of another node.

Evaluation

CESAR is in essence a privacy preserving counterpart of D-SGD. However, the inner-workings of CESAR change the set of shared parameters and this may cause the performance of the two algorithms to deviate from each other. In this chapter we evaluate the two algorithms in same settings for random subsampling and TopK sparsification to see how comparable their performance is. Furthermore, we compare empirical results of CESAR with values and formulas derived in Section 4.4.

CESAR has been implemented as a fork of decentralized and federated learning library `decentralizepy` in Python which is available at `https://gitlab.epfl.ch/sacs/decentralizepy`. The implementation of CESAR is intended to be made available under the given repository.

5.1 Setting

We run tests on a simulated 96 nodes network. Every test is run 5 times with a different random seed. The network has 4-regular topology generated at random based on the starting seed of the test. We use a cluster of 3 machines, each running 32 nodes and designating a single CPU core to each node. Each machine is equipped with 126 GBs of random access memory and 32 virtual core Intel(R) Xeon(R) CPU E5-2630 v3 clocked at 2.40GHz.

5.1.1 Datasets and models

CESAR is evaluated on two different datasets, each with a different ML model. In this subsection we describe these datasets and models in more details. Furthermore, in Table 5.1 we provide a quick overview of their respective sizes.

CIFAR-10. CIFAR-10 [32] is a dataset containing RGB images of size 32×32. In total there are 60 000 images, of which 50 000 belongs to the training dataset and the rest is in the test dataset. The images are divided in 10 classes of animals and vehicles: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. Each class contains 6000 images, and exactly 1000 images of each class were at random selected to be in the test set.

We train LeNet [33] on CIFAR-10 data. It starts with 3 convolutional layers, each followed by group normalization [34] with 2 groups, rectified linear unit (ReLU) activation and max pooling with kernel size 2 and stride 2 in the given order. Every convolutional layer kernel size 5, keeps the size of the image the same and they have 32, 32 and 64 output channels respectively starting from the beginning. Finally, the output is flattened and passed towards one of 10 classes through a fully connected layer. In total, the model has 89 834 parameters.

CelebA. CelebFaces Attributes Dataset (CelebA) [35] is a dataset containing RGB images of celebrity faces of size 218×178. In total there are 202 599 images of 10 177 different celebrities. Each images has a celebrity identifier and 40 binary attributes associated with it. These attributes contain information such as whether the person is bald, chubby, smiling, young, has brown hair etc. We keep images of celebrities that have at least 5 appearances in the trainset, which results in 63 741 images of 2361 celebrities in the trainset, and 7097 images in the testset. All images are resized into 84×84 format.

The model used for CelebA is a binary classifier predicting whether the person in an image is smiling or not. The model accepts inputs of size $84 \times 84 \times 3$ followed by 4 convolutional layers, each with 32 output channels, kernel size 3,

stride 1 and keeping the same image format. Max pooling with kernel size 2 and stride 2 is applied after every convolutional layer performing ReLU activation. Finally, the parameters are flattened before being passed to one of the two output classes through a fully connected layer. In total, the model has 30 242 parameters.

Dataset	Trainset size	Testset size	# parameters
CIFAR-10	50 000	10 000	89 834
CelebA	63 741	7097	30 242

Table 5.1: Size of the used datasets and the number of parameters of models trained on them

Data partition. The training data for both datasets is split among nodes in the network such that no two nodes get the same training sample. The testsets are avaialble to all nodes and they are fully used to evaluate performance of models obtained at different points during training at all nodes.

The trainset of CelebA contains mappings between nodes and samples they get for training. On the hand, CIFAR-10 is split in non-IID fashion among nodes: if n is the number of nodes in a network all samples are sorted by label and split into $4n$ equally sized chunks which are at random partitioned among the nodes such that every node gets 4 chunks.

5.1.2 Training and evaluation

Metrics. For each test we evaluate the accuracy of the trained models over time, as well as the amount of data transferred per each node during the training. Furthermore, we look how the percentage of shared parameters trained with CESAR changes over time in comparison to equation (4.9).

Training and hyperparameters. The training is happening in the same fashion at all nodes. First, every node performs several iterations of mini-batch stochastic gradient descent (SGD) with a fixed batch size. After a specified number of SGD iterations a node shares and averages its view of the model

with its neighbors before going back to the SGD step with the new view of the model and repeating this cycle over many times. We refer to each such cycle as a round. Every several rounds make an epoch, number of rounds per epoch is handpicked for each dataset. Models are evaluated at the beginning of each epoch. No learning rate schedule is used.

Across the same dataset all algorithms and nodes use the same values of training hyperparameters regardless of the percentage of weights they try to share. These values are shown in Table 5.2. The values, except the number of rounds in an epoch, have been obtained by performing grid search for D-SGD with 100% parameter sharing. During the search, each combination of parameters is run for a fixed number of iterations regardless of whether the algorithm converges or not. Because of the nature of the search, we expect that given hyperparameters would be more suitable towards D-SGD compared to other algorithms.

Dataset	**Learning rate**	**Batch size**	**SGD iterations per aggregation**	**Rounds per epoch**
CIFAR-10	10^{-2}	8	20	21
CelebA	10^{-3}	8	10	10

Table 5.2: Hyperparameters used for models trained on different datasets

Compression. CESAR uses pairwise masks to obfuscate transferred parameter values. In an ideal case these masks would be selected such that the masked value is no different than uniformly at random selected noise, hence maximizing entropy and removing any patterns that would make this data easily compressible. For this reason and to make all results comparable, no compression is used on parameter values. In the real world, algorithms such as D-SGD would use all available forms of compression, however secure aggregation is often a requirement rather than an option and all forms of it limit available compression techniques similarly to CESAR. Therefore, D-SGD is used only as a baseline of utility in our tests.

On the other hand, all indices are transferred in plaintext and we compress

them using Elias gamma compression [36].

5.2 Random subsampling

The properties that make random subsampling predictable has been discussed in details. In this section we test how these properties behave in practice, as well as compare achieved accuracy and the amount of transferred data against D-SGD with the same sparsification technique.

Both CESAR and D-SGD are run with 30% parameters selected in sparsification. However, it is important to remember that some parameters selected in sparsification are discarded during CESAR, and based on equation (4.9) we expect that CESAR will share 19.71 % of parameters in the given setting if 30 % is selected in sparsification. Furthermore, solving equation (4.15) for the given setting tells us that 38.878 % parameters should be selected in sparsification in order to have 30 % of them sent to other nodes during CESAR. Therefore, CESAR is also run with 38.878 % parameters selected in sparsification.

Table 5.3 shows how empirical measurements of the percentage of parameters selected in sparsification and shared in CESAR compare to theoretical values prediction by equation (4.9). The values match almost perfectly with a minimal error on both datasets which verifies the correctness of our theoretical findings.

Dataset	**% selected in sparsification**	**Expected %**	**Measured %**
CIFAR-10	30	19.71	$19.71 \pm 7 \cdot 10^{-5}$
	38.878	30	$30 \pm 8 \cdot 10^{-5}$
CelebA	30	19.71	$19.711 \pm 2 \cdot 10^{-4}$
	38.878	30	$30 \pm 2 \cdot 10^{-4}$

Table 5.3: Comparison of theoretically expected versus actually measured % of parameters sent to neighbors during CESAR with random subsampling given different % of parameters selected in sparsification. All values are rounded to 3 decimal places

In Figure 5.1 and Table 5.4 we see the performance and the required amount of data to train a model in each of three settings we test. All three runs produce relatively similarly performing models with higher accuracy for higher percent-

age of parameters shared between nodes. Hence, D-SGD for 30% and CESAR for 38.878% perform almost identically with CESAR for 30% slightly behind. The amount of data required to transfer parameter values follows the same pattern, with D-SGD for 30% and CESAR for 38.878% transferring the most data, and CESAR for 30% behind them. When using CESAR with random subsampling in the first step nodes do not have to send the list of indices they selected, but due to properties of random subsampling they can send only the seed used to generate this list, hence the overhead of this step is negligible as it can be seen. On the other hand, upon receiving these seeds nodes need to perform intersections between received masks and their own in order to apply the pairwise masks on the given intersection. Furthermore, the final set of parameters a node sends to another is obtained by discarding parameters that haven't been masked. Consequently, this set of indices cannot be easily expressed as a random mask generated using a specific seed anymore. In comparison, D-SGD can share just the starting seed used for generating the mask, which requires constant amount of data to represent and explains why CESAR has much higher metadata compared to D-SGD.

	Maximum average accuracy (%)		Average cumulative data transferred (GiB)	
	CIFAR-10	CelebA	CIFAR-10	CelebA
D-SGD 30%	58.75	93.25	2.62 ± 10^{-5}	$0.39 \pm 4 \cdot 10^{-6}$
CESAR 30%	56.24	92.97	$1.95 \pm 4 \cdot 10^{-5}$	$0.30 \pm 2 \cdot 10^{-5}$
CESAR 38.878%	58.18	93.24	$2.89 \pm 4 \cdot 10^{-5}$	$0.43 \pm 2 \cdot 10^{-5}$

Table 5.4: Maximum average accuracy of trained models and average cumulative amount of data transferred per node during training of D-SGD and CESAR with random subsampling

5.3 TopK sparsification

While testing random subsampling in section 4.4 we see that comparing CESAR and D-SGD for same percentage of parameters selected in sparsification makes little sense. On the other hand comparing CESAR with 38.878% selected pa-

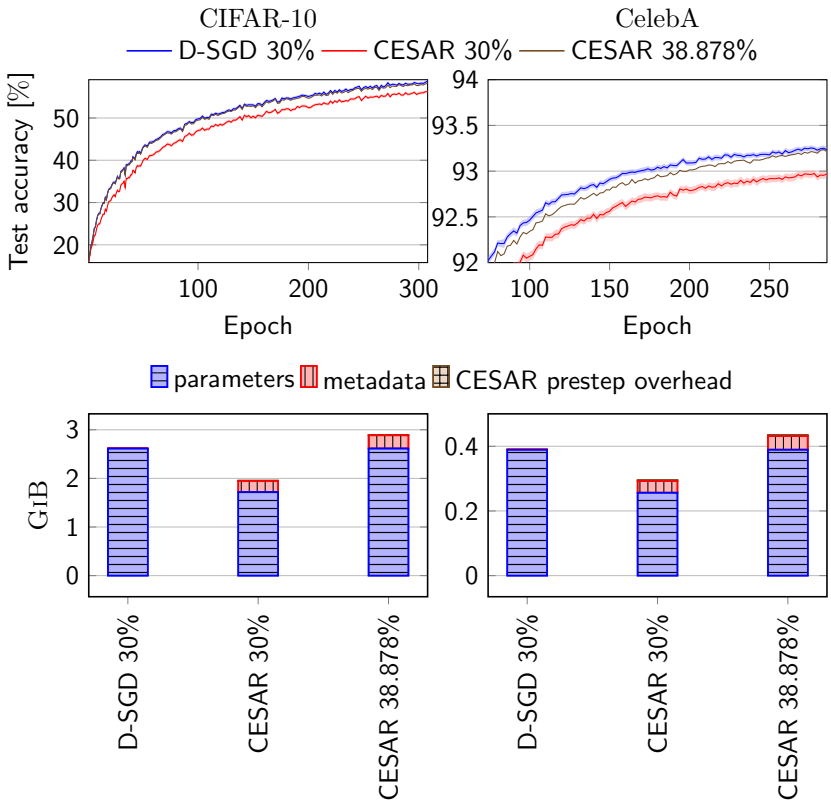

Figure 5.1: Average accuracy and average cumulative amount of data transferred per node for D-SGD and CESAR using random subsampling as sparsification technique (Percentages given in CESAR indicate the percentage of parameters selected using sparsification rather than the percentage of parameters sent to other nodes)

rameters produces quite comparable results as D-SGD with 30% selected parameters. Therefore, we don't test 30% sparsification for CESAR in this section. Additionally, in section 4.5 we discussed the benefits of adding indices not selected in sparsification and it is valuable to see how this affects the protocol, as it fundamentally changes the way indices that enter the protocol are selected. We evaluate how including a percentage of indices not selected in sparsification changes results. In total we run the following tests:

1. D-SGD with 30% of parameters selected in TopK sparsification
2. CESAR with 38.878% of parameters selected in TopK sparsification
3. CESAR with 30% of parameters selected in TopK sparsification and before the start of the protocol including additional 8.878% of total parameters that haven't been selected chosen uniformly at random, leading to total 38.878% of parameters (for simplicity we will refer to this test as 'CESAR 30+8.878%')

First, in Figure 5.2 we look how the percentage of parameters shared with other nodes in the protocol changes when some percentage of parameters selected in sparsification is replaced with the same percentage of randomly selected remaining parameters. We see in all tests during the whole training that the percentage of parameters sent to other nodes is higher than expected value. If we remind ourselves, the theoretical model assumes that sparsification process selects parameters with uniform probability and that it is independent on all nodes, however TopK doesn't satisfy these requirements. The reason for this is that parameters selected in TopK are dependant on values of these parameters and there are at least two confounders that create bias in the selection: all nodes use the same network architecture and hence the values of certain parameters may behave similarly, and the datasets used by all nodes is generated by the same natural process. For example, all faces of people in CelebA are similar and have patterns they share, as well as animals and vehicles present in CIFAR-10. It is also noticeable looking at the scale of y that the difference between em-

pirical and theoretical value is smaller for CIFAR-10 dataset which was split to be non-IID, hence the data differs more between the nodes minimizing its confounding influence. The same phenomenon was encountered by [19]. Finally, we see that adding indices that were not selected in sparsification brings the overall behaviour closer to what is theoretically expected.

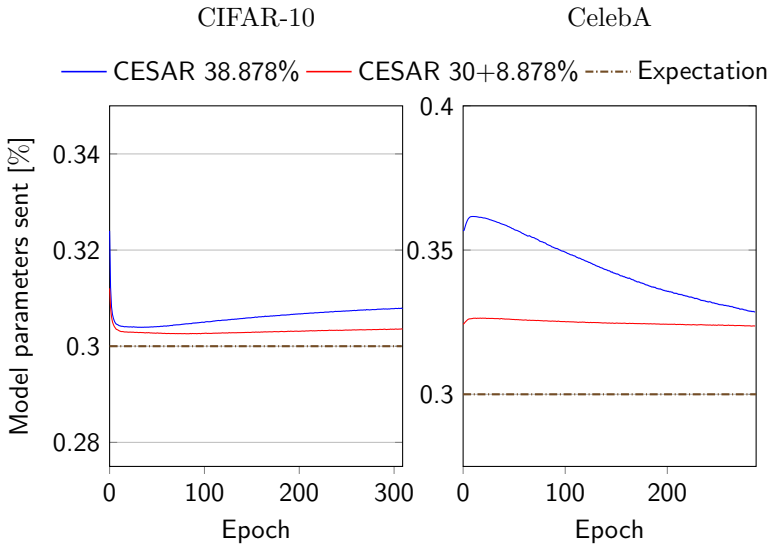

Figure 5.2: Comparison of theoretically expected percentage of parameters shared in CESAR with empirically measured values

D-SGD and both variations of CESAR send similar percentage of parameters to other nodes, and all three send the full list of selected indices as metadata. Therefore, excluding overhead caused by the first step of CESAR we expect all three to transfer similar amount of data which is confirmed by Figure 5.3. It also shows that the overhead of CESAR is similar in both variations and it is significant in comparison to what we saw for random subsampling. The figure together with Table 5.5 shows that accuracy achieved by D-SGD for 30% and CESAR for 30% with 8.878% additional indices perform similarly with CESAR coming slightly ahead on both datasets. However, CESAR for 38.878% shows a drop in performance which is surprising considering that nodes using it share the highest percentage of parameters with their neighbors. The reason it shares more data compared to CESAR with addition of random non-selected indices

is that it is further from the theoretical model, hence it is biased and some indices are selected often resulting in others being having to be selected less. For this reason, over many iterations neighbors send more parameters, but still see smaller portions of total local models of their neighbors. Nevertheless, we are only hypothesizing, yet this is a complex question that requires more research put into it to understand the behaviour and optimal way of selecting indices in this setting.

	Maximum average accuracy (%)		Average cumulative data transferred (GiB)	
	CIFAR-10	CelebA	CIFAR-10	CelebA
D-SGD 30%	55	93.19	$2.82 \pm 3 \cdot 10^{-4}$	$0.42 \pm 5 \cdot 10^{-5}$
CESAR 38.878%	50.48	93	$3.6 \pm 4 \cdot 10^{-3}$	$0.58 \pm 7 \cdot 10^{-4}$
CESAR 30+8.878%	57.16	93.26	$3.64 \pm 4 \cdot 10^{-3}$	$0.57 \pm 6 \cdot 10^{-4}$

Table 5.5: Maximum average accuracy of trained models and average cumulative amount of data transferred per node during training of D-SGD and CESAR with TopK sparsification

5.4 Fixed budget

For our final test we look how the best performing models seen in the two previous sections perform when given a fixed budget. We look what accuracy each model achieves for different amount of memory transferred. In this test we look at D-SGD and CESAR with random subsampling for 30% and 38.878% parameters selected in sparsification respectively. We refer to these two models as D-SGD$_{\text{RS}}$ 30% and CESAR$_{\text{RS}}$ 38.878%. For TopK sparsification we look at D-SGD with 30% parameters selected and CESAR with 30% parameters selected plus 8.878% non-selected parameters being added. These models are referred to as D-SGD$_{\text{TopK}}$ 30% and CESAR$_{\text{TopK}}$ 30 + 8.878%.

Figure 5.4 shows that variations of CESAR achieve very close accuracy to D-SGD with the same sparsification scheme for fixed budget. The difference on CIFAR-10 which is non-IID appears smaller compared to CelebA indicating that CESAR may work well for non-IID data. Interestingly both TopK algorithms

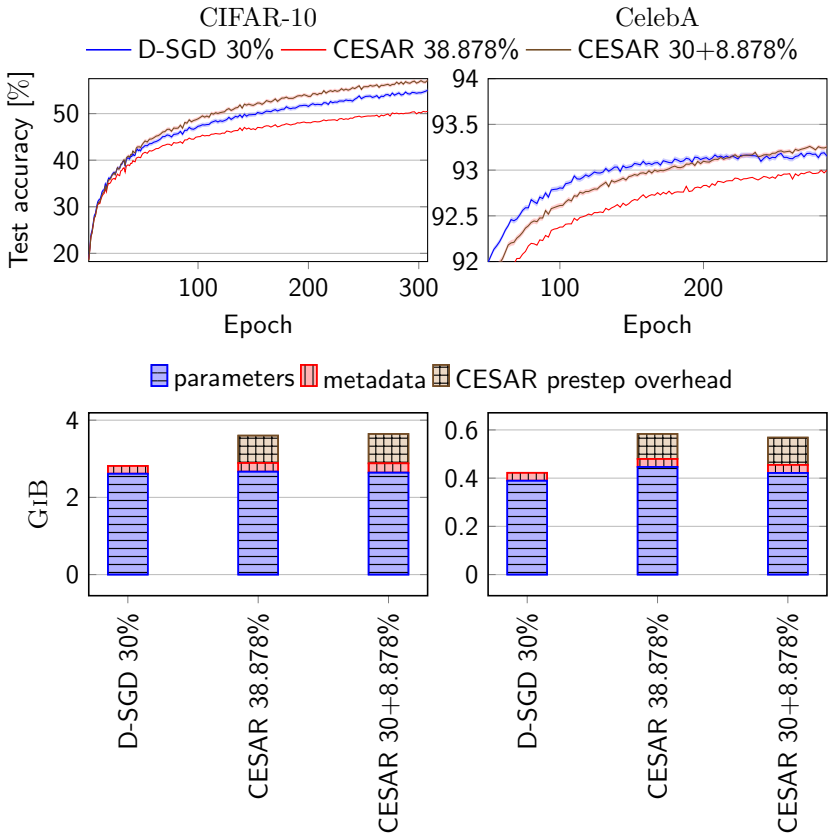

Figure 5.3: Average accuracy and cumulative amount of data transferred per node for D-SGD and CESAR using TopK (Percentages given in CESAR indicate the percentage of parameters selected before the start of the protocol rather than the percentage of parameters sent to other nodes)

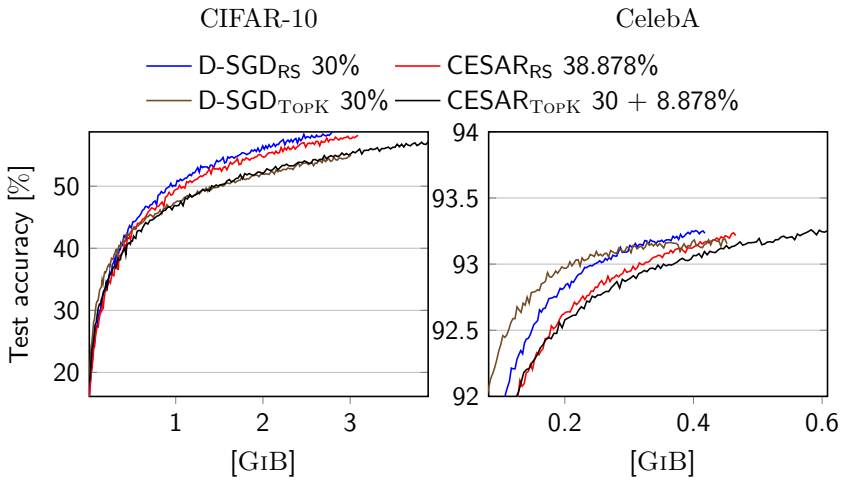

Figure 5.4: Average model accuracy achieved for fixed cumulative budget per node for different variations of D-SGD and CESAR run with TopK sparsification and random subsampling

start rising at the same rate as random subsampling for CIFAR-10, but then get overtaken while random subsampling continues rising at higher rate. Overall, these results show that the overhead of CESAR is minimal for networks where the number of neighbors each node has is significantly smaller than the size of the network like it's the case in our setting.

Related work

Various authors have been interested in exploring ways in which sensitive data can be used to train ML models, usually with focus on FL setting.

Differential privacy and blockchains. Since early work, there have been different examples of using differential privacy [37, 38] with varying levels of success. While majority of this works point out the privacy-utility trade-off [39] associated with the use of differential privacy, others claim that the decrease in utility can be overcome by using large datasets [40]. [41] leverages the power of differential privacy together with blockchains to provide private and secure way to perform DL. [42] tries to address the issue of poisoned models, which are another privacy concern, in DL using tangles, a structure that resembles blockchains.

Homomorphic encryption. In contrast to differential privacy, other work proposes HE as a way of providing private collaborative learning [43, 44]. This line of work is focused on keeping models encrypted at all times and performing training on models in this form. They achieve this through the use of highly advanced cryptosystems, such as Brakerski-FanVercauteren (BFV) [45] and Cheon-Kim-Kim-Song (CKKS) [46], by calculating and applying gradient calculations directly on encrypted models. However, performing operations on encrypted data is both computationally and communicationally expensive,

hence making use of HE difficult in practice.

Secure aggregation. The seminal work [17] lays the foundation for secure aggregation in FL setting and introduces many ideas later adopted by other authors: pairwise additive masks, node specific masks and the usage of Shamir [28] secret sharing for recovering secrets of failed nodes. [31] proposed a similar variation to provide resistance against byzantine nodes on top of secure aggregation using multiple servers and additive secret sharing. On other hand, [19] and [20] look into ways of incorporating sparsification in secure aggregation protocols.

Conclusion

This thesis introduces a novel secure aggregation protocol called CESAR that aims to provide privacy benefits of secure aggregation together with efficient communication with the use of sparsification. It is sparsification agnostic and supports any form of sparsification.

Our tests on two datasets show that the protocol produces models that achieve similar accuracy as models trained using D-SGD with the same sparsification technique. Furthermore, in settings where the number of neighbors is small CESAR comes at little to no cost while not restricting the choice of sparsification. This means that privacy against honest-but-curious adversary can be achieved almost for free using CESAR.

However, CESAR is only a first step in providing a practical way to perform privacy preserving aggregation in DL setting. Additional challenges include assuring that the protocol works with failing nodes, Byzantine nodes, and only with such improvements can it be used in practice.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[2] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, no. 4, pp. 681–694, 2020.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805

[4] X. Zhu, C. Vondrick, C. C. Fowlkes, and D. Ramanan, "Do we need more training data?" *International Journal of Computer Vision*, vol. 119, no. 1, pp. 76–92, 2016.

[5] "Health information privacy," https://www.hhs.gov/hipaa/index.html, accessed: 2023-01-19.

[6] "General data protection regulation," https://gdpr-info.eu/, accessed: 2023-01-19.

[7] J. Edwards, "Medicine on the move: Wearable devices supply health-care providers with the data and insights necessary to diagnose medical issues and create optimal treatment plans [special reports]," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 8–11, 2019.

[8] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[9] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard," 2019. [Online]. Available: https://arxiv.org/abs/1906.04329

[10] M. Chen, R. Mathews, T. Ouyang, and F. Beaufays, "Federated learning of out-of-vocabulary words," 2019. [Online]. Available: https://arxiv.org/abs/1903.10635

[11] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," 2018. [Online]. Available: https://arxiv.org/abs/1812.02903

[12] K. Sozinov, V. Vlassov, and S. Girdzijauskas, "Human activity recognition using federated learning," in *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, 2018, pp. 1103–1111.

[13] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated electronic health records," *International Journal of Medical Informatics*, vol. 112, pp. 59–67, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138650561830008X

[14] L. Huang, Y. Yin, Z. Fu, S. Zhang, H. Deng, and D. Liu, "Loadaboost: loss-based adaboost federated machine learning with

reduced computational complexity on iid and non-iid intensive care data," 2018. [Online]. Available: https://arxiv.org/abs/1811.12629

[15] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 691–706.

[16] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 739–753.

[17] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[18] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[19] I. Ergun, H. U. Sami, and B. Guler, "Sparsified secure aggregation for privacy-preserving federated learning," *arXiv preprint arXiv:2112.12872*, 2021.

[20] S. Lu, R. Li, W. Liu, C. Guan, and X. Yang, "Top-k sparsification with secure aggregation for privacy-preserving federated learning," *Computers & Security*, vol. 124, p. 102993, 2023.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[22] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *NIPS'17*, vol. 30, Long Beach, California, USA, 2017, pp. 5336–5346. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/f75526659f31040afeb61cb7133e4e6d-Paper.pdf

[23] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, 2013.

[24] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 603–618.

[25] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2512–2520.

[26] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *Advances in neural information processing systems*, vol. 32, 2019.

[27] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[28] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[29] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *European Symposium on Research in Computer Security*. Springer, 2020, pp. 480–501.

[30] G. Sun, Y. Cong, J. Dong, Q. Wang, L. Lyu, and J. Liu, "Data poisoning attacks on federated machine learning," *IEEE Internet of Things Journal*, 2021.

[31] L. He, S. P. Karimireddy, and M. Jaggi, "Secure byzantine-robust machine learning," *arXiv preprint arXiv:2006.04747*, 2020.

[32] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

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

[34] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[35] Z. Liu, P. Luo, X. Wang, and X. Tang, "Large-scale celebfaces attributes (celeba) dataset," *Retrieved August*, vol. 15, no. 2018, p. 11, 2018.

[36] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Transactions on Information Theory*, vol. 21, no. 2, pp. 194–203, 1975.

[37] W. Li, F. Milletarì, D. Xu, N. Rieke, J. Hancox, W. Zhu, M. Baust, Y. Cheng, S. Ourselin, M. J. Cardoso *et al.*, "Privacy-preserving federated brain tumour segmentation," in *International workshop on machine learning in medical imaging*. Springer, 2019, pp. 133–141.

[38] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1895–1912.

[39] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.

[40] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," *arXiv preprint arXiv:1710.06963*, 2017.

[41] X. Chen, J. Ji, C. Luo, W. Liao, and P. Li, "When machine learning meets blockchain: A decentralized, privacy-preserving and secure design," in *2018 IEEE international conference on big data (big data)*. IEEE, 2018, pp. 1178–1187.

[42] R. Schmid, B. Pfitzner, J. Beilharz, B. Arnrich, and A. Polze, "Tangle ledger for decentralized learning," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2020, pp. 852–859.

[43] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux, "Poseidon: Privacy-preserving federated neural network learning," *arXiv preprint arXiv:2009.00349*, 2020.

[44] G. Xu, G. Li, S. Guo, T. Zhang, and H. Li, "Privacy-preserving decentralized deep learning with multiparty homomorphic encryption," *arXiv preprint arXiv:2207.04604*, 2022.

[45] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[46] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International conference on the theory and application of cryptology and information security*. Springer, 2017, pp. 409–437.