Private Collaborative Neural Network Learning

Melissa Chase[1], Ran Gilad-Bachrach[1], Kim Laine[1], Kristin Lauter[1], and Peter Rindal[2]

[1] Microsoft Research, Redmond, WA
[2] Oregon State University, Corvallis, OR

Abstract. Machine learning algorithms, such as neural networks, create better predictive models when having access to larger datasets. In many domains, such as medicine and finance, each institute has only access to limited amounts of data, and creating larger datasets typically requires collaboration. However, there are privacy related constraints on these collaborations for legal, ethical, and competitive reasons. In this work, we present a feasible protocol for learning neural networks in a collaborative way while preserving the privacy of each record. This is achieved by combining Differential Privacy and Secure Multi-Party Computation with Machine Learning.

Keywords: Deep Learning, Neural Networks, Differential Privacy, Secure Multi-Party Computation

1 Introduction

Recent advances in neural network learning showed that significant improvements in accuracies can be achieved by having access to large datasets Krizhevsky et al. [2012]. Collecting large datasets may be challenging in domains such as medicine and finance, where privacy is a major concern. For example, the Health Insurance Portability and Accountability Act (HIPAA) in the United States establishes rules to protect patient health records, and puts constraints on the ability of health care providers to share patient data. Similarly, the Financial Privacy Rule in the Gramm–Leach–Bliley Act puts constraints on the sharing of financial information. In other cases, the data may be distributed between competing companies. For example, credit card companies may be interested in collaborating to create a model to detect fraud, but at the same time they are reluctant to share their data since it may leak trade secrets. Therefore, creating large datasets is, in many cases, a lengthy and costly legal process.

Secure Multi-Party Computation (MPC) refers to a set of cryptographic technologies designed to enable computation over data distributed between different parties so that only the result of the computation is revealed to the participants, but no other information is shared Yao [1982]. For example, two companies that are considering merging their businesses may be interested in computing how large is the intersection of their current customer bases without revealing their customer lists to each other.

MPC has been studied for several decades, but it has only recently become feasible for complicated computations due to newly discovered implementation optimizations, the availability of faster hardware with extremely fast implementations of the AES block cipher, and the availability of fast networking. It is now possible to evaluate circuits at the speed of up to 7 billion gates per second, assuming the computation is represented as a Boolean circuit Araki et al. [2016]. Nevertheless, MPC suffers from two main drawbacks for the purposes discussed in this study. First, it is still infeasible to run generic MPC protocols over many remote parties especially due to high communication costs. For example, Mohassel and Zhang Mohassel and Zhang [2017] have recently introduced a system

that can train neural networks using MPC and secret sharing. Their work uses clever techniques to speedup computation but nevertheless, it still requires large amounts of communication and therefore may be too slow for many applications. Second drawback of MPC is that while MPC secures any intermediate result, it does not provide any privacy in the end result. That is, a trained model might leak information about the dataset that was used to train it.

Differential Privacy (DP) Dwork et al. [2006b] addresses the issue of leakage of information from the result of a computed function. The goal is to reveal only results such that even in the presence of auxiliary information, it is impossible to tell if a certain record was, or was not, in the dataset that the function was computed on. Consider for example a function that computes the number of people in a certain hospital suffering from a particular disease. If this number is zero then it is possible to conclude that no person who is currently in that hospital has the disease. Hence, even innocent looking functions might leak private information.

To achieve differential privacy, it is necessary to add random noise to the result Dwork et al. [2014]. For example, when counting the number of patients with a certain disease, it is possible to achieve privacy by adding a Laplace-distributed random number to the computed number of patients, and publish only the perturbed result.

While it has been showed that neural network training can be made private using differential privacy tools Song et al. [2013], Abadi et al. [2016], DP does not provide a solution to the collaboration problem discussed here. If the data is distributed between multiple parties, then each one of them must add noise to the data it shares. This will result in excessive amounts of noise added during the training process, which will result in a poor model. The most severe situation is when there are many parties, but each one of them holds just small amounts of data. For example, imagine that the goal is to train a neural network over genetic information. It may be the case that each party has only a single record to share: their own genome. In this case, the noise would be much larger than the amplitude of the information, and the perturbed signal would be of limited use. In this case, the amount of noise each party adds compared to the amount of information would be so large that the shared information would not have any practical use.

Therefore, DP provides a solution to privacy issues, but does not handle the collaboration part, while MPC addresses the collaboration issue, but does not handle the privacy part. Hence, in this work we marry these two technologies to support collaborative privacy, especially in the case of training neural networks.

Training neural networks is typically done by some form of stochastic gradient descent Kelley [1960]. In each step of this iterative process, the average of the gradients of a mini-batch of samples is computed with respect the current weight vector. This is used as a proxy to the gradient of the loss function, and a step is taken in the inverse direction of the gradient to minimize the loss. Some algorithms also collect higher-order information to further accelerate the training process Kingma and Ba [2014].

In creating a privacy preserving training process we follow the approach of Abadi et al. [2016] and make each average gradient of a mini-batch private by adding random noise so that the perturbed gradient becomes differentially private. We introduce an MPC protocol that allows random noise to be added only once per mini-batch regardless of the number of parties involved in the computation. We take special care in making the MPC protocol computationally efficient, and communication efficient.

Our main contribution is the introduction of a private collaborative neural network training algorithm with theoretical and empirical evaluations. The methods we employ can be used also in other situations, for example, for training different types of models. Our solution assumes the semi-honest security model, where the parties are assumed to follow the designed protocol, but may attempt to learn private information by analyzing their view of the protocol execution. Furthermore, we assume

that the parties are non-colluding in the sense that they share only share the information instructed to share by the protocol. We conjecture that these are realistic assumptions in many real-world applications.

2 Background

To provide a solution to the problem of private collaborative neural network learning, we combine several technologies including MPC, DP, and secret sharing. In this section, we provide a brief introduction to these technologies.

2.1 Differential Privacy (DP)

The goal of differential privacy is to define a notion of privacy for evaluating functions on a dataset, and to construct mechanisms that can be mathematically proved to provide privacy in this sense Dwork et al. [2006b, 2014]. In differential privacy, the privacy guarantee is parametrized by two parameters: $\epsilon, \delta \geq 0$. A function $f : X^* \to Y$ is said to be (ϵ, δ)-DP if for every $S \subseteq Y$, and every pair of datasets $D, D' \in X^*$ that differ at a single record, it holds that

$$\Pr\left[f\left(D\right) \in S\right] \leq e^{\epsilon} \Pr\left[f\left(D'\right) \in S\right] + \delta\,,$$

where the probabilities are with respect to the internal randomness of f. The idea behind this definition is that differentially private functions reveal very little information about individual records in the datasets they were computed on.

One of the ways to turn a non-private function f into a private one is by adding random noise. This typically requires that f is a Lipschitz-smooth function with respect to some norm, i.e.

$$\left\|f\left(D\right) - f\left(D'\right)\right\| \leq \Delta \left|D - D'\right|\,.$$

In this case, it is possible to find a distribution such that the function $D \mapsto f(D) + \text{Rand}$ has the desired privacy, when Rand is sampled from the distribution.

When using DP, it is important to keep track of the aggregated information leak if more than one function is computed. Namely, assume a set of functions $f_1, f_2, \ldots$ is computed. Even if each one of them is (ϵ, δ)-DP, their combination might not have the same properties. In this case, composition bounds are use to analyze the privacy properties of the entire computation Kairouz et al. [2017]. In these settings a distinction is being made between the non-adaptive case, in which all the functions to be computed are announced in advance, and the adaptive case, in which the functions to be computed may be altered based on the results of previous computations.

2.2 Secure Multi-Party Computation (MPC)

MPC is a set of cryptographic tools aimed at allowing collaboration in computation. Assume that there are k parties such that each party i holds a private value x_i. The goal is to evaluate a function $f\left(x_1, \ldots, x_k\right)$ so that each party will learn the result, but no information about each other's inputs x_i beyond what can be inferred from the result. Some MPC techniques are generic in the sense that they can be applied to a broad range of functions (such as Boolean circuits) Yao [1982], Araki et al. [2016], while other techniques may be designed for a specific set of functions. It has already been shown that combining MPC and DP can be useful Dwork et al. [2006a].

2.3 Secret Sharing

Secret sharing is designed to allow storing data in a distributed way so that each individual share does not reveal any information about the data, but when combined, the full data can be recovered Shamir [1979], Beimel [2011]. For example, assume x is a number in the range $[-C, C)$. We can pick a random number r uniformly sampled from the same range, and create a secret share such that in one location r is recorded, and in another location $x + r$ smod C is recorded.[3] Since both r and $x + r$ are uniformly distributed in $[-C, C)$, each one of them in isolation does not reveal any information about x. However, having access to both allows computing the secret x. It is easy to see that two such secret shares can be added to create a secret share of the sum of the secrets. Therefore, the type of secret sharing presented here provides an efficient method to implement an MPC protocol for the addition function.

3 Algorithms and Analysis

In this section, we describe the algorithms proposed for solving the distributed private neural network learning problem, and we also analyze these algorithms. We start by discussing the issue of introducing privacy to gradient descent type algorithms, after which we discuss the collaborative part.

3.1 Private Gradient Descent

Many machine learning algorithms use variants of Gradient Descent (GD) to optimize the parameters of the model. In a nutshell, these algorithms begin with a random initial assignment w_0 to the parameters of the model. In some cases, an auxiliary variable a_0 is initialized as well.[4]

In the i'th iteration a subset of the training data B is selected, and a gradient

$$g_i = \sum_{z \in B_i} F'(w_{i-1}, z) \tag{1}$$

is computed, where F is the function to be optimized. At this point an update function ϕ is used to update the parameters of the problem

$$(w_i, a_i) = \phi(w_{i-1}, a_{i-1}, g_i)$$

where w_i is a new assignment to the parameters of the problem, and a_i is a new assignment to the auxiliary variables. This procedure is applied several times, and the final weight vector is used as the output of the learning process.

To ensure that the model does not leak private information, we use DP tools. These tools require bounding the sensitivity of the output with respect to changing a single input record. However, as observed by Abadi et al. [2016], since the update function ϕ might be complex, and since F' might be complex as well, it is easier to make sure that g_i is private, and use composition bounds Dwork et al. [2014] to make sure that the final model has the desired DP properties. The method we describe here is a generalization of the methods presented in Abadi et al. [2016] and Song et al. [2013].

[3] smod is the symmetric mode operation formally defined as $x \text{ smod } C = ((x + C) \mod 2C) - C$

[4] For example, some algorithms use auxiliary variables to track an approximation of the Hessian of the function to be optimized.

The first step in making g_i private is to make sure that each term $F'(w_{i-1}, z)$ is bounded, and then add some random noise. Following Abadi et al. [2016], we define a function Clip that scales each gradient to be no larger than the required size:

$$\text{Clip}(C, x) = \min\left(1, \frac{C}{\|x\|}\right) x\,,$$

where the exact norm used (in this paper either L_1, L_2 or L_∞ norm) will be discussed later, and $C > 0$ is the desired size. Therefore, we replace (1) with

$$g_i = \sum_{z \in B_i} \text{Clip}\left(C, F'(w_{i-1}, z)\right)\,. \tag{2}$$

We note that in neural networks it sometimes makes sense to split the "budget" term C between the layers, since convolution layers tend to have fewer parameters than dense layers, which leads to different step sizes taken in each of the layers. In these cases, the vector x has a natural breakdown into a set of vectors $x^1, \ldots, x^l$. We generate a latent budget $\tilde{C}$ such that if $\forall i,\ \left\|x^i\right\| \leq \tilde{C}$, then $\|x\| \leq C$. For the L_∞-norm, $\tilde{C} = C$, for the L_1-norm $\tilde{C} = C/l$, and for the L_2-norm $\tilde{C} = C/\sqrt{l}$. In these cases Clip(C, x) is performed by applying Clip$(\tilde{C}, x^i)$ for every i.

The g_i in (2) is C-sensitive in the sense that if $\tilde{B}_i$ differs from B_i only on a single record, and $\tilde{g}_i$ is computed from $\tilde{B}_i$, then $\|g_i - \tilde{g}_i\| \leq C$. Such functions can be made DP by adding random noise sampled from an appropriate distribution Dwork et al. [2014]. We denote by $b = b(\epsilon, \delta, C, l, d)$ the parameter of the distribution of the noise, which is a function of the desired privacy parameters ϵ and δ, the clipping parameter C, the number of layers l, and the dimension of the gradient vector d. Therefore, the private version of the gradient will be:

$$g_i = \sum_{z \in B_i} \text{Clip}\left(C, F'(w_{i-1}, z)\right) + \text{Rand}(b)\,, \tag{3}$$

where Rand(b) is a vector of the same dimensionality as the gradient, such that each component is an i.i.d. sample from the noise distribution. For the L_1 case, the Laplace distribution with parameter $b = C/\epsilon$ will make g_i $(\epsilon, 0)$-DP Dwork et al. [2014]. In the L_2 case, the Gaussian distribution with parameter $b = \sigma = C\sqrt{2\ln(1.25/\delta)}/\epsilon$ will make g_i (ϵ, δ)-DP Dwork et al. [2014]. For the L_∞ case, the Laplace distribution with

$$b = \frac{C\sqrt{d}\left(\sqrt{\log\frac{1}{\delta}} + \sqrt{\log\frac{1}{\delta} + 2\epsilon}\right)}{\epsilon\sqrt{2}}$$

will result in (ϵ, δ)-DP, which follows from composition theorems for non-adaptive queries. We present the proof in the Appendix.

The gradient in (3) already has DP properties, but during the learning process many gradient steps are taken, and we need to make sure that the compounded information leak throughout the entire process is controlled. This is achieved using so-called composition theorems. Note that the common practice in machine learning is to break the training process into epochs. In each epoch the entire training set is broken into disjoint mini-batches. For each such mini-batch, a gradient is computed, and the model is updated. In terms of privacy, since the mini-batches are disjoint, each record appears in at most one mini-batch. Therefore, as long as each gradient is (ϵ, δ)-DP, then the update done in the entire epoch has the same privacy properties. However, in the training process multiple epochs are performed. Since the gradient computed on a later epoch depends on the weight

Algorithm 1 Private Gradient Descent (PGD)

input $C > 0$, a bound on the size of the gradients
input $\epsilon^*, \delta* > 0$, differential privacy parameters
input T, the number of epochs
input w_0 an initial weight vector for the neural network (can be random)
input $m > 0$ the size of the mini-batches
output $\hat{w}$, a private approximate to $\arg\min \mathbb{E}_z[F(w, z)]$
  function PGD
    compute ϵ, δ based on (4)
    compute b based on ϵ, δ, and the norm used
    compute $n \leftarrow$ dataset size divided by m
    for $t = 1, \ldots, nT$ **do**
      let Z_t be the next mini-batch of size m
      compute $g_t = \sum_{z \in Z_t} \text{Clip}\left(C, F'\left(w_{t-1}, z\right)\right) + \text{Rand}\left(b\right)$
      compute $(w_t, a_t) \leftarrow \phi\left(w_{t-1}, a_{t-1}, g_t\right)$
    end for
  end function

vector w at that time, and since this weight vector depends on gradients computed in earlier epochs, it is necessary to make sure that this adaptive nature does not result in too much information being leaked. Theorem 3.4 in Kairouz et al. [2017] showed that if each gradient is (ϵ, δ)-DP and T epochs are performed, then the overall computation is (ϵ^*, δ^*)-DP with

$$\begin{aligned} \delta^* &= 1 - \left(1 - \tilde{\delta}\right)(1 - \delta)^T \\ \epsilon^* &= \min\left(T\epsilon, \frac{T\epsilon\left(e^\epsilon - 1\right)}{e^\epsilon + 1} + \sqrt{2T\epsilon^2 \log\left(e + \frac{\sqrt{T\epsilon^2}}{\tilde{\delta}}\right)}, \frac{T\epsilon\left(e^\epsilon - 1\right)}{e^\epsilon + 1} + \sqrt{2T\epsilon^2 \log\frac{1}{\tilde{\delta}}}\right), \end{aligned} \tag{4}$$

where $1 \geq \tilde{\delta} > 0$. Therefore, given a requirement for (ϵ^*, δ^*), a value for δ, and the number of epochs T, it is possible to search for values for $\tilde{\delta}$ and ϵ such that when used in (4) will result in the required privacy levels (ϵ^*, δ^*) for the entire process. The resulting algorithm is the Private Gradient Descent (PGD) algorithm (Algorithm 1). In our implementation of this algorithm, we use the ADAM algorithm to perform the gradient step $(w_t, a_t) \leftarrow \phi\left(w_{t-1}, a_{t-1}, g_t\right)$, since it is fast to converge Kingma and Ba [2014].

Analysis of the PGD algorithm (Algorithm 1) reveals interesting properties. Although in general the functions we use may be non-convex, we limit the analysis to the convex case. Following Theorem 5 by Dekel et al. [2012], we obtain the following result about the optimality gap of the algorithm, where the optimality gap is the difference in terms of loss between $F\left(\frac{1}{T}\sum_{t=2}^{T+1} w_t\right)$ and the minimum of F and $F = \mathbb{E}_z\left[f\left(w, z\right)\right]$.

Theorem 1. *Let W be a compact convex space, and assume that $F : W \times Z \to \mathbb{R}$ is L-smooth and convex for every z. Furthermore, assume that $\|F'\| \leq C$.*

Assume that

$$\forall w \in W, \;\; \sigma^2 \geq \mathbb{E}_z\left[\|\nabla_w f\left(w, z\right) - \nabla_w F\left(w\right)\|^2\right],$$

denote $D = \sqrt{\max_{u,v \in W} \|u - v\|_2^2 / 2}$, and M the size of the dataset times the number of epochs.

The expected optimality gap (when ϕ represents the dual averaging algorithm with an appropriate learning rate) is bounded by

$$\frac{2mD^2L}{M} + \frac{2D\sigma}{\sqrt{M}} + \frac{2D\tilde{\sigma}}{\sqrt{Mm}}\,, \tag{5}$$

where $\tilde{\sigma}^2$ is the variance of Rand (b).

The proof follows from Theorem 5 of Dekel et al. [2012] by observing that the variance of g_t is bounded by $\sigma^2/m + \tilde{\sigma}^2/m^2$. The significance of Theorem 1 is that it shows that the choice of the mini-batch size is not trivial in this case. Note that $\sigma < \tilde{\sigma}$, so if the batch size m is small, the right-most term in (5) dominates, and the optimum would be when the batch size is not too small. However, if $m = O\left(M^{1/3}\right)$, then the left term and the right term are of the same order of magnitude, and further increasing m will result in degraded performance. This also suggests that it may not be optimal to keep the batch size fixed during the learning process.

3.2 Collaborative Gradient Computation

In Section 3.1 we introduced the PGD algorithm, which is a privacy-preserving version of the gradient descent algorithm. The key to achieving privacy was adding random noise to the gradient computed over a mini-batch. However, in the collaborative setting, the records in each mini-batch may be distributed between different parties. Therefore, in this section we introduce a protocol that allows computing this perturbed gradient over the distributed records without leaking any intermediate results.

We introduce the Collaborative Private Gradient Descent algorithm (CPGD). The algorithm differs from the PGD algorithm (Algorithm 1) only in the way it computes the perturbed gradient, and therefore we describe here only this step.

Let Z be a batch that is distributed between k parties such $Z = \cup Z^i$, and the i'th party holds Z^i but would like to keep every $z \in Z^i$ private from any other party that is involved (or not involved) in the learning process. One possible way of implementing this step is through generic MPC protocols. The problem with this approach is both in the computational and communication complexity, especially when the batch size is large and/or the number of parties is large. Another possible approach is to have each party compute the gradients of the part of the batch it holds, and share it with other parties after adding random noise to make it differentially private. This approach is computationally efficient, but may result in poor models since the amount of noise added will be large.

To overcome these limitations, we introduce a novel protocol that is efficient, secure, and has high utility. The Collaborative Gradient Computation (CGC) algorithm (Algorithm 2) combines several techniques. First, each party does the gradient computation on the data it holds. Next, two hosts H_1 and H_2 are nominated, and the computed gradients are shared with these hosts using secret shares. This means sharing a random vector with H_2, and the gradient perturbed by this random vector with H_1. Therefore, each one of these hosts sees a random vector, but when these two vectors are put together the true gradient is revealed.

After each party has shared its secret shares with the two hosts, the second stage of the computation begins. First, each host sums the parts of the secret shares received from the different parties. These sums result in a secret share of the sum of the gradients over the entire mini-batch. At this point H_1 and H_2 use a secure computation protocol Yao [1982], Araki et al. [2016] to decipher the information in the secret share, and add the random noise needed for differential privacy. In terms of computation, this engagement is simple since it is independent of the size of the mini-batch, and of the number of parties, and it is straight-forward to parallelize. Moreover, if the different parties

Algorithm 2 Collaborative Gradient Computation

input k parties, each one with a dataset Z^i
input $C > 0$, a bound on the size of the gradients
input b, a parameter for the random noise
input w, the current weight vector
input m, the batch size
output $g = \sum_i \sum_{z \in Z^i} \text{Clip}\left(mC, F'\left(w_t, z\right)\right) + \text{Rand}\left(b\right)$

Stage I:

Each party performs:

compute $g^i = \sum_{z \in Z^i} \text{Clip}\left(C, F'\left(w_t, z\right)\right)$
compute $r^i \leftarrow \text{Uniform}\left[-mC, mC\right)$, a random vector with the same dimension as g^i
send $g^i + r^i$ to host H_1
send r^i to host H_2

Stage II:

H_1:

compute $\tilde{g}_1 = \sum_i \left(g_i + r_i\right) \text{ smod } mC$
compute s_1 a seed for the random number generator

H_2:

compute $\tilde{g}_2 = \sum_i r_i \text{ smod } mC$
compute s_2, a seed for the random number generator

Host 1 and host 2 use the garbled circuits protocol to compute $\left(\left(\tilde{g}_1 - \tilde{g}_2\right) \text{ smod } mC\right) + \text{Rand}_{s_1 \oplus s_2}\left(b\right)$

Recall that $x \text{ smod } C = \left(\left(x + C\right) \text{ mod } 2C\right) - C$.

and hosts do not collude, no information leaks during the entire computation until the final result is revealed. Therefore, there is no need to add excessive amounts of noise to guarantee privacy.

In the following Lemmas we prove the correctness and the privacy properties of this interaction.

Lemma 1. *If all the parties follow the protocol, then the MPC protocol computes*

$$g = \sum_i \sum_{z \in Z^i} \textit{Clip}\left(mC, F'\left(w_t, z\right)\right) + \textit{Rand}\left(b\right) \quad .$$

Proof. H_1 receives $\left(g_i + r_i\right) \text{ smod } mC$, and computes the sum smod mC of these variables. From the properties of the smod function it follows that (see the Appendix for proof):

$$\begin{aligned}
&\sum_i \left(\left(g_i + r_i\right) \text{ smod } mC\right) \text{ smod } mC \\
&= \sum_i \left(g_i + r_i\right) \text{ smod } mC \\
&= \left[\left(\sum_i g_i \text{ smod } mC\right) + \left(\sum_i r_i \text{ smod } mC\right)\right] \text{ smod } mC \\
&= \tilde{g}_1 \, .
\end{aligned}$$

At the same time, H_2 receives r_k and computes $\tilde{g}_2 = \sum r_k \text{ smod } mC$. Hence,

$$\left(\tilde{g}_1 - \tilde{g}_2\right) \text{ smod } mC = \sum_i g_i \text{ smod } mC = \sum_i g_i \, ,$$

where the last equality follows since $\left|\sum g_k\right| \leq \sum \left|g_k\right| < mC$, and the smod function is such that if $\left|x\right| \leq mC$ then $x \text{ smod } mC = x$ (see the Appendix for proof).

The random seed has the property that $s_1 \oplus s_2$ is uniformly distributed and independent of any parties view. Therefore, the selection of Rand (b) is independent as well. Hence, we conclude that the algorithm will compute $\sum_i g_i + \text{Rand}\,(b)$. □

Lemma 2. *Assume that all the parties and hosts follow the Collaborative Gradient Computation protocol and that neither of the parties or the hosts collude. The only new information revealed in this process is* $g = \sum_i \sum_{z \in Z^i} \textit{Clip}\,(mC, F'\,(w_t, z)) + \textit{Rand}\,(b)$.

Proof. We evaluate the information leaking to three types of participants: external to the process, parties contributing gradients to the process, and hosts of the computation.

External viewers and parties contributing to the process will see the end result

$$g = \sum_i \sum_{z \in Z^i} \text{Clip}\,(mC, F'\,(w_t, z)) + \text{Rand}\,(b)\ ,$$

as well as any information they contributed, and nothing else. The hosts H_1 and H_2 have access to the secret shares, and we need to show that no information is leaking while processing them.

Note that s_i is uniformly distributed even when conditioned on $s_1 \oplus s_2$, and therefore does not leak any information. H_2 sees uniformly distributed variables r_i, and therefore will not learn any additional information. Host H_1 sees $\{g_i + r_i \text{ smod } mC\}$, but since r_i is uniform in $[-mC, mC)$ then $g_i + r_i$ smod mC is uniformly distributed too. If there exists a mechanism M such that given g and $\{g_i + r_i \text{ smod } mC\}$ predicts a property p of $g_1, \ldots, g_k$, then

$$\begin{aligned}
&\Pr\left[M\left(g, \{g_i + r_i \text{ smod } mC\}\right) = p\left(g_1, \ldots, g_k\right)\right] \\
=\ &\Pr\left[M\left(g, \{u_i\}\right) = p\left(g_1, \ldots, g_k\right) \middle| u_i = g_i + r_i \text{ smod } mC\right] \\
=\ &\Pr\left[M\left(g, \{u_i\}\right) = p\left(g_1, \ldots, g_k\right) \middle| u_i \sim \text{Uniform}\left[-mC, mC\right]\right].
\end{aligned}$$

Therefore, any mechanism, even having access to the information available to one of the hosts, cannot reveal more information than any party which observes only g. □

3.3 Collaborative Private Gradient Descent

Combining the Private Gradient Descent algorithm (Algorithm 1) and the collaborative protocol for computing the gradient through the Collaborative Gradient Computation (Algorithm 2) results in a Collaborative Private Gradient Descent algorithm. This algorithm performs privacy-preserving gradient descent on distributed datasets.

Theorem 2. *Assuming all parties follow the protocol (semi-honest security model), and do not collude, the Collaborative Private Gradient Descent algorithm is* (ϵ, δ)*-DP.*

Proof. Lemma 2 shows that each mini-batch gradient computation does not leak any information beyond $\sum_{z \in Z_t} \text{Clip}\,(C, F'\,(w_{t-1}, z)) + \text{Rand}\,(b)$, where Z_t is the current mini-batch. The parameter b is selected so that even after computing all gradients over all mini-batches the entire process is (ϵ, δ)-DP. □

4 Complexity Analysis

In this section we analyze the computational complexity and the communication complexity of the Collaborative Gradient Computation algorithm. When studying computational complexity, we make a distinction between computation that is performed on plaintext and on computation performed using cryptographic tools, since the latter can be orders of magnitude slower Araki et al. [2016].

The plaintext computation performed by the Private Gradient Descent algorithm (Algorithm 1) is very similar to the computation performed by other distributed learning algorithms Dekel et al. [2012]. Thus, on large datasets, it may be faster than serial algorithms that perform all computation on a single box Dekel et al. [2012].

Cryptographic tools are needed to perform Stage II of the Collaborative Gradient Computation (Algorithm 2). First, each host sums the values in its part of the secret shares. If there are k parties, and the gradient is of d dimensions, then there are $O(kd)$ such operations, each requiring a single addition. After making all the additions, a single modular reduction is performed. Therefore, these operations are very efficient even though they use cryptographic tools.

Next, a garbled circuits protocol is applied between the two nominated hosts, and its complexity depends only on the dimension d of the gradient vector. Moreover, this operation is trivial to run in parallel, since the performed computation can be done independently for each coordinate of the vector. Regardless, it is independent of the size of the dataset and the number of parties involved. Thus, the complexity of the cryptographic computation is[5] $O(dk)+O(d)$. We engage in this process for every mini-batch, and therefore this complexity should be multiplied by the number of gradient steps performed.

We measured the time it takes to perform the garbled circuit part of this protocol. We have found that even when the hosts are simple laptops, it takes about 0.2 ms for each of the hosts to perform its part of the computation with 32 bits of precision. The time grows to 0.6 ms for 64 bits of precision. Note that it is easy to run this process in parallel so that all of the dimensions of the gradient vector are treated at the same time. This can be done on a single box or multiple boxes, making the latency of the entire process on the order of just a few milliseconds

Next, we look at the communication complexity of the protocol. In every round, every party sends a single gradient vector regardless of the size of the data it hosts. However, since we use secret sharing, for a d dimensional gradient vector, every party has to send $2d$ data points, i.e. $O(d)$ bits. Each party receives the aggregated d-dimensional gradient, which is $O(d)$ bits of information as well.

Each host receives $O(kd)$ data items from the two parties, and broadcasts the perturbed gradient at the end of the computation, requiring $O(d)$ bits of communication. On top of that, the two hosts engage in the MPC protocol to compute the perturb gradient. Each party contributes to this computation $O(d)$ bits, since it shares d data items. When we evaluate empirically the size of the data, we find that for 32 bits of precision the circuit required uses 2480 AND gates, and for 64 bits of precision 5561 AND gates. Therefore, the size of the garbled circuit is 79360 bytes for 32 bits of precision, which grows to 174 KB for the 64-bit case. We note, however, that these sizes can be significantly reduced by using other MPC schemes. For example, the size of the circuit can go down to 930 bytes when using the protocol of Araki et al. [2016].

Since the communication complexity is independent of the size of the dataset, for very large datasets the described protocol may even be more efficient than sending the data to a single location.

[5] Since the overhead of the garbled circuits protocol is much larger than the overhead of the computation with secret shares, we write the two terms separately even though they can be written as $O(dk)$.

5 Empirical Results

We tested the algorithm on the MNIST dataset. The MNIST dataset contains 60000 training examples of hand-written digits, and 10000 testing examples. Each image consists of 28×28 grayscale pixels. We conducted two experiments, one with a small network with 13940 tunable parameters in 3 layers, and another experiment with a large network with 127540 tunable parameters in 4 layers. When these models are trained without any privacy constraints the large models achieve 98.9% accuracy, while the smaller one achieves 98.5% accuracy.

5.1 Small Network

For this experiment a 3-layer network of the following structure was used:

1. a 5×5 convolution layer with 5 outputs per window, a $2 - 2$ stride, ReLU activation function, and padding;
2. a $1 \times 5 \times 5$ convolution layer with 10 outputs per window, a $1 - 2 - 2$ stride, no weight sharing on the first axis, and ReLU activation function;
3. a dense layer with 10 outputs.

This model has 13940 tunable parameters. Each classifier was trained using 50 epochs over the training data with different targets for the differential privacy ϵ parameter ($\delta = 10^{-3}$ was used in all experiments). A parameter sweep was used to find the optimal set of parameters. The following parameters were optimized:

- initial mini-batch size: $m = 600,\ 6000,\ 60000$;
- rate of mini-batch size increase: no-increase, $2x$ every 3 epochs;
- the learning rate in the ADAM algorithm $\eta = 0.001,\ 0.01$ (default parameters $\beta_1 = 0.9, \beta_2 = 0.999$ were used);
- gradient clip $C = 0.01,\ 0.1,\ 1$.

Table 1 shows the accuracies for different selections of norm and privacy levels. The best accuracies were achieved when the L_1 norm was used (together with the Laplace mechanism for differential privacy), where accuracies of 89%, 92%, and 94%, for $\epsilon = 0.5,\ 2,\ 8$ respectively, were obtained. We do not have a clear intuition to explain why the L_1 norm performs better than the L_2 norm, and substantially better than the L_∞ norm.

Table 1. Accuracies achieved for different selections of privacy parameters and norms for the small network.

	L_∞	L_1	L_2
$\epsilon = 0.5$	58.5%	89.1%	82.3%
$\epsilon = 2$	73.7%	91.8%	87.8%
$\epsilon = 8$	88.1%	94.2%	92.1%

As we conjectured from Theorem 1, the mini-batch size has substantial influence on the performance. For the L_1 norm, best performance was achieved when the mini-batch size was 6000. Table 2 shows how the accuracy changes when the mini-batch size is varied. Larger batch sizes may be less influenced by the random noise injected by the DP mechanism. However, when larger batch sizes are used, fewer gradient steps are made which results in worse accuracies. While it is true that it is possible to increase the number of gradient steps by making more epochs, doing that will require increasing the amount of noise injected.

Table 2. Accuracies for different mini-batch sizes (L_1 norm and $\epsilon = 8$) for the small network.

Initial batch size	Increment step	Accuracy
600	1x	89.7%
600	2x	93.7%
6000	1x	94.2%
6000	2x	93.7%
60000	1x	91.8%

5.2 Large Network

For this experiment a 4 layer network of the following structure was used:

1. a 5×5 convolution layer with 5 outputs per window, a $2 - 2$ stride, ReLU activation function, and padding;
2. a $1 \times 5 \times 5$ convolution layer with 10 outputs per window, a $1 - 2 - 2$ stride, no weight sharing on the first axis, and ReLU activation function;
3. a dense layer with 100 outputs, and ReLU activation function;
4. a dense layer with 10 outputs.

The additional dense layer increases the number of parameters in this network to 127540, which is 9x more parameters compared to the smaller network. The accuracies for this network are presented in Table 3. Similar to the smaller network, the L_1 norm delivers the best accuracies. In all but one case the smaller network provided better accuracies, despite the fact that when these networks are used without privacy the larger network has better accuracy. This is to be expected since the larger number of parameters results in larger amount of noise injected by the DP mechanism. However, given that the amount of noise does not grow when larger datasets are used, and given the relatively small differences in performance, especially in the L_1 case, we conjecture that with larger datasets the larger network would outperform the smaller one.

Table 3. The accuracies achieved for different selections of privacy parameters and norms for the large network.

	L_∞	L_1	L_2
$\epsilon = 0.5$	24.0%	84.7%	51.8%
$\epsilon = 2$	61.1%	90.3%	88.4%
$\epsilon = 8$	80.3%	92.9%	90.4%

6 Discussion

In this work, we studied the problem of collaborative training of neural networks with privacy constraints. We showed that by combining learning techniques with cryptographic tools, it is possible to provide a feasible solution to this problem. Special attention is made to the communication complexity which is of major concern since it induces large latencies. The amount of data communicated is independent of the size of the datasets. Moreover, it scales nicely with respect to the number of parties involved.

In the design of the algorithms we combine techniques from learning, differential privacy and secure multi-party computation. We show that each one of these techniques cannot achieve the designated goals. The approach we presented may be useful in other problems, for example, training different learning algorithms.

While our results are encouraging, there are still important problems that should be addressed. We saw in our experiments that accuracies drop when the number of parameters in the network is large. In the era of deep learning it is important to find better ways to handle large networks. Another issue to consider is situations where the dataset is split in different ways between the parties. In the current work, we assumed that each party holds some records and all parties use the same schema. However, in different settings it could be that each party holds some features on all participants. For instance, in a municipality, the health department has health information, while the tax department holds information about income. Handling these situations efficiently is a subject for future research.

Bibliography

Martín Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 805–817. ACM, 2016.

Amos Beimel. Secret-sharing schemes: a survey. In *International Conference on Coding and Cryptology*, pages 11–46. Springer, 2011.

Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(Jan):165–202, 2012.

Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer, 2006a.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, pages 265–284. Springer, 2006b.

Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The composition theorem for differential privacy. *IEEE Transactions on Information Theory*, 63(6):4037–4049, 2017.

Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. *IACR Cryptology ePrint Archive*, 2017:396, 2017.

Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

Shuang Song, Kamalika Chaudhuri, and Anand D Sarwate. Stochastic gradient descent with differentially private updates. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pages 245–248. IEEE, 2013.

Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

A Additional proofs

Lemma 3. *For every* z_1, z_2*:* $((z_1 \text{ smod } C) + (z_2 \text{ smod } C)) \text{ smod } C = (z_1 + z_2) \text{ smod } C$.

Proof. The proof follows since:

$$
\begin{aligned}
& ((z_1 \bmod C) + (z_2 \bmod C)) \bmod C \\
= \; & ((z_1 + C \bmod 2C) + (z_2 + C \bmod 2C) - 2C) \bmod C \\
= \; & (((z_1 + z_2 + 2C) \bmod 2C) - 2C) \bmod \mathrm{C} \\
= \; & (((z_1 + z_2) \bmod 2C) - 2C) \bmod C \\
= \; & (C + ((z_1 + z_2) \bmod 2C) - 2C) \bmod 2C - C \\
= \; & (C + z_1 + z_2) \bmod 2C - C \\
= \; & (z_1 + z_2) \bmod C \,.
\end{aligned}
$$

□

Lemma 4. *If $|z| < C$ then z smod C=z.*

Proof. Note that if $|z| < C$ then $0 \leq z + C \leq 2C$ and therefore,

$$z \bmod C = ((z + C) \bmod 2C) - C = (z + C) - C = C \,.$$

□

Theorem 3. *A function $f : X^* \mapsto \mathbb{R}^d$ is said to be Δ-sensitive in the L_∞ norm if for every $D, D' \in X^*$ that differ in exactly one element it holds that $\|f(D) - f(D')\|_\infty \leq \Delta$.*

Let $\epsilon, \delta \geq 0$ such that $\delta < e^{-1/2}$ and let f be Δ-sensitive in the L_∞ norm then the Laplace mechanism applied to f is (ϵ, δ)-DP for

$$b \geq \frac{\Delta\sqrt{d}\left(\sqrt{\log\frac{1}{\delta}} + \sqrt{\log\frac{1}{\delta} + 2\epsilon}\right)}{\epsilon\sqrt{2}} \,.$$

Note that another way to think about the setting discussed in this theorem is to think about f as a set of k non-adaptive queries. Therefore, this bound holds in this case too.

Proof. Let D and D' be such that they differ on a single element and let $\gamma = f(D) - f(D')$. From the sensitivity of f it follows that $\|\gamma\|_\infty \leq \Delta$. To simplify notation, assume w.l.o.g. that γ is such that $\forall i,\ \gamma_i = \Delta$. Let

$$A = \left\{ x \in \mathbb{R}^d \,:\, |\{i \,:\, x_i \geq \Delta\}| \geq \frac{d}{2} - \frac{\epsilon b}{2\Delta} \right\} \,.$$

Since

$$|x_i - \Delta| - |x_i| = \begin{cases} -\Delta & \text{if } x_i \geq \Delta \\ \Delta & \text{if } x_i \leq 0 \\ \Delta - 2x_i & \text{if } 0 < x_i < \Delta \end{cases}$$

then $\forall x \in A$ we have that

$$
\begin{aligned}
\frac{\exp\left(-\frac{\|x\|_1}{b}\right)}{\exp\left(-\frac{\|x-\delta\|_1}{b}\right)} &= \exp\left(\frac{1}{b}\sum_i |x_i - \Delta| - |x_i|\right) \\
&\leq \exp\left(\frac{\Delta}{b}\left(d - 2\,|\{i \,:\, x_i \geq \Delta\}|\right)\right) \\
&\leq \exp(\epsilon) \,.
\end{aligned}
$$

Let $S \subseteq \mathbb{R}^d$ and let

$$T = S - f(D) = \{x - f(D) : x \in S\} \quad .$$

$$\begin{aligned} & \Pr[f(D) + \text{Lap}(b) \in S] \\ = \ & \Pr[\text{Lap}(b) \in T] \\ = \ & \Pr[\text{Lap}(b) \in T \cap A] + \Pr[\text{Lap}(b) \in T \setminus A] \\ \leq \ & \Pr[\text{Lap}(b) \in T \cap A] + \Pr[\text{Lap}(b) \notin A] \quad . \end{aligned}$$

Since

$$\begin{aligned} & \Pr[\text{Lap}(b) \in T \cap A] \\ = \ & \int_{x \in T \cap A} \frac{1}{2b} \exp\left(-\frac{\|x\|_1}{b}\right) dx \\ \leq \ & e^{\epsilon} \int_{x \in T \cap A} \frac{1}{2b} \exp\left(-\frac{\|x - \gamma\|_1}{b}\right) dx \\ = \ & e^{\epsilon} \Pr[\gamma + \text{Lap}(b) \in T \cap A] \\ \leq \ & e^{\epsilon} \Pr[f(D') + \text{Lap}(b) \in S] \quad . \end{aligned}$$

then

$$\begin{aligned} & \Pr[f(D) + \text{Lap}(b) \in S] \\ \leq \ & e^{\epsilon} \Pr[f(D') + \text{Lap}(b) \in S] + \Pr[\text{Lap}(b) \notin A] \end{aligned}$$

and it suffices to show that $\Pr[\text{Lap}(b) \notin A] \leq \delta$.

Recall that if $x_i \sim \text{Lap}(b)$ then $\Pr[x_i > \Delta] = \frac{1}{2}\exp(-\Delta/b)$. Therefore, $\Pr[\text{Lap}(b) \notin A]$ equals the probability of tossing d coins with a probability of $\frac{1}{2}\exp(-\Delta/b)$ for heads and observing at most $\frac{d}{2} - \frac{\epsilon b}{2\Delta}$ heads. Since $\exp(-\Delta/b) \geq 1 - \Delta/b$ it holds that $\Pr[\text{Lap}(b) \notin A]$ is smaller that the probability of tossing d coins with probability $\frac{1}{2}(1 - \Delta/b)$ for heads and observing at most $\frac{d}{2} - \frac{\epsilon b}{2\Delta}$ heads. Using Hoeffding's inequality it follows that

$$\begin{aligned} & \Pr[\text{Lap}(b) \notin A] \\ \leq \ & \exp\left(-2d\left(\frac{1}{2} - \frac{\epsilon b}{2d\Delta} - \frac{1}{2}\left(1 - \frac{\Delta}{b}\right)\right)^2\right) \\ = \ & \exp\left(-\frac{d}{2}\left(\frac{\Delta}{b} - \frac{\epsilon b}{d\Delta}\right)^2\right) \end{aligned}$$

We would like the last term to be $\leq \delta$, which is equivalent to

$$\frac{d}{2}\left(\frac{\Delta}{b} - \frac{\epsilon b}{d\Delta}\right)^2 \geq \log\frac{1}{\delta},$$

which in turn is equivalent to

$$\left|\frac{\Delta}{b} - \frac{\epsilon b}{d\Delta}\right| \geq \sqrt{\frac{2}{d}\log\frac{1}{\delta}}. \tag{6}$$

Note that if b is as stated in the statement of the theorem, then

$$b \geq \Delta\sqrt{\frac{d}{\epsilon}}$$

and hence

$$\frac{\epsilon b}{d\Delta} \geq \frac{\Delta}{b}\,.$$

Using this observation, we have that (6) holds if

$$\frac{\epsilon b}{d\Delta} - \frac{\Delta}{b} \geq \sqrt{\frac{2}{d}\log\frac{1}{\delta}}\,,$$

which holds if

$$\epsilon b^2 - d\Delta^2 - \Delta b\sqrt{2d\log\frac{1}{\delta}} \geq 0\,,$$

while $b, d, \Delta > 0$. Therefore, for (6) to hold, it suffices that

$$\begin{aligned} b &\geq \frac{\Delta\sqrt{2d\log\frac{1}{\delta}} + \sqrt{2\Delta^2 d\log\frac{1}{\delta} + 4d\Delta^2\epsilon}}{2\epsilon} \\ &= \frac{\Delta\sqrt{d}\left(\sqrt{\log\frac{1}{\delta}} + \sqrt{\log\frac{1}{\delta} + 2\epsilon}\right)}{\epsilon\sqrt{2}}\,. \end{aligned}$$

□