Practical, Label Private Deep Learning Training based on Secure Multiparty Computation and Differential Privacy

Sen Yuan
Facebook Inc.
yuansen@fb.com

Milan Shen
Facebook Inc.
milanshen@fb.com

Ilya Mironov
Facebook Inc.
imironov@fb.com

Anderson C. A. Nascimento
School of Engineering & Technology
University of Washington Tacoma
andclay@uw.edu

Abstract

Secure Multiparty Computation (MPC) is an invaluable tool for training machine learning models when the training data cannot be directly accessed by the model trainer. Unfortunately, complex algorithms, such as deep learning models, have their computational complexities increased by orders of magnitude when performed using MPC protocols. In this contribution, we study how to efficiently train an important class of machine learning problems by using MPC where features are known by one of the computing parties and only the labels are private. We propose new protocols combining differential privacy (DP) and MPC in order to privately and efficiently train a deep learning model in such scenario. More specifically, we release differentially private information during the MPC computation to dramatically reduce the training time. All released information idoes not compromise the privacy of the labels at the individual level. Our protocols can have running times that are orders of magnitude better than a straightforward use of MPC at a moderate cost in model accuracy.

1 Introduction

Secure multiparty computation (MPC) [1, 2] is one of the basic building blocks of privacy preserving machine learning (PPML). MPC is particularly useful in a scenario where several parties need to compute a function of a dataset that cannot be directly accessed by the computing parties. This can happen, for example, when the private input data does not come directly from data holders but is the result of previous secure computations. In such scenarios, no party holds the private data in the clear: it is secret shared among the computing parties. The cost of training the a machine learning model using solely MPC can be several orders of magnitude more expensive than training in the clear.

In this contribution, we show that it is possible to dramatically speed-up applications of privacy preserving machine learning on top of MPC by releasing differentially private information about the players inputs throughout the computation. Because of the guarantees of differential privacy, the information release does not affect the privacy of individual entries, while runtimes can be improved by orders of magnitude. We apply our ideas to the problem of training machine learning models when one of the parties is in possession of training features, and the corresponding labels for each input are secret, potentially coming from previous computations, and are secret shared between the computing parties. Note that since the labels are not directly accessible by any of the computing

Preprint. Under review.

parties, a direct application of local differential privacy is not possible. This problem is relevant for a wealth of applications. For example, this scenario is relevant for computing models for predicting ads conversions where one party knows the training features, while the labels (conversion or no conversion) are usually not held neither of them and need to be computed using MPC with the parties' inputs (timestamps of clicks and/or impressions).

A direct application of MPC to solve this problem would secret share all the inputs to the computation (features and labels) and use them as input to an MPC protocol. In the case of a deep learning model, that would imply the computation of several activation functions, gradients of loss functions, inner products - all expensive computations when carried over MPC. For a network with thousands of neurons, that would imply millions of secure computations per dataset entry per epoch. We will show that by releasing information during the MPC execution (in a differentially private way), we can drastically reduce the computational complexity of such protocols. We present two different protocols for solving this problem: one based on local differential privacy [3] and another one based on a modification of the well-known differentially private stochastic gradient descent (DP-SGD). [4].

1.1 Motivations and Related Works

In this paper, we show that it is possible to dramatically increase the efficiency of deep learning training on top of MPC for a practical case: label privacy in deep learning. These are computations where the features are known by one of the computing parties and the labels are private – secret shared among the computing parties.

Secure Multiparty Computations Secure multiparty computations have been hailed as a potential solution for the problem of computing machine learning models when the computing parties are not allowed to observe the dataset directly. We will concentrate our analysis on secret shared based secure multiparty computation [5]. Threshold secret sharing among n parties and with a threshold t is a cryptographic protocol that takes an input x and distribute *shares* $\{x_1, \ldots, x_n\}$ to parties $\{P_1, .., P_n\}$ such that: (i) No subset of $t-1$ or less parties can learn any information on the secret x; and (ii) any subset of t or more parties can fully recover x. In a threshold secure multiparty computation protocol, n parties $\{P_1, .., P_n\}$ want to compute a function $f(x_1, x_2, ..., x_n)$ depending on secret inputs so that, at the end of the computation, no set of up to $t-1$ parties knows more than the function output, its own input and everything else that can be computed from these data. Secret sharing based MPC is carried out by assuming that the computing parties $P_1, .., P_n$ have secret shared all the inputs $x_1, \ldots, x_n$ at the beginning of the protocol. The secret sharing scheme in question is usually a linear one - where linear operations on the players' inputs can be computed locally on shares. These linear operations do not require the parties to exchange messages and are computed locally and efficiently. To compute a generic function f, the parties first agree on a description of f as a circuit consisting of additions and multiplication gates. The addition gates can be computed using the linearity of the underlying secret sharing scheme - these operations do not require any communication among the computing parties. Multiplication gates are expensive and require communication among all the computing parties. Because every function f has to be broken down as a sequence of secure additions and multiplications, and the high cost of secure multiplications, functions that can be easily computed in the clear become prohibitively expensive when computed using MPC. For example, computing x^{-1} for a given input x can require between 50 and 100 secure multiplications (depending on the required numerical accuracy) [6] - each secure multiplication requiring exchange of data among all the computing parties. *For neural networks, activation functions are the most expensive computation when performed on top MPC. They are responsible for over 90% of the total computational costs.*

When computing a complex deep learning model, we can have millions of secure multiplications performed for each dataset entry forward pass in the network. Making such MPC deep learning training practical for large datasets (potentially consisting of billions of entries) is still an open problem. Despite these difficulties, MPC has been applied to an extensive range of machine learning problems [7, 8, 9, 10, 11, 6, 12, 13]. In our problem, we leverage the fact that only labels are kept private such that we can reduce the number of multiplications required for training a deep learning models drastically.

Differential Privacy Differential privacy (DP) has initially been proposed as a way to ensure that queries obtained from a dataset did not reveal information about single entries of such dataset [14]. It has been generalized to more complex tasks such as making sure that a machine learning model does not reveal information about individual entries belonging to the training dataset [4]. There are two scenarios in DP. Global differential privacy assumes a trusted curator that access the entire dataset and prepares/releases the differentially private queries. In local differential privacy, such trusted curator is not available/assumed and individual data holders randomize their data and release the randomized results, which are, in turn, processed by a non-trusted party to produce the differentially private desired computations. In the specific case of deep learning models, differentially private stochastic gradient descent [4] is a very popular technique for training deep neural networks with differential privacy. DP-SGD aims at protecting the entire dataset (features and labels). Note that in local and global differential privacy, one needs to access the private data directly (a trusted curator in the case of Global DP and the data holders in the case of Local DP). We work in a scenario where that is not possible. The private data, labels in our case, is never directly accessible to anyone. It is secret shared among the computing parties.

Label Privacy and Privacy Preserving Machine Learning Chase et. al. [15] proposed a protocol for training a neural network with differential privacy in a scenario where a dataset is distributed horizontally across several parties, so labels and features are known to these parties. MPC is used to aggregate and add differential privacy to batches locally trained in the clear by each one of these parties. This is clearly different from our scenario where labels are not known by any party. Thus, training batches using local computations in the clear is not possible. Label privacy was initially studied by [16] and was applied to linear regression in [17]. In a recent paper [18], Ghazi et. al. attacked the problem of using deep learning when only labels need to be kept private, which is similar to our problem. However, in their paper, they assume that the labels are accessible either by a trusted curator responsible for implementing differential privacy mechanisms or by the data owners themselves. It differs from our practical applications hence MPC computation is well suited to be leveraged.

2 Technical Preliminaries

In this section we review some basic concepts and definitions that will be used in the remainder of the paper.

Definition 1 *Differential Privacy [14]- An algorithm $\mathcal{A}$ is said to be (ϵ, δ) differentially private if for any neighboring datasets D and D', any subset S of outputs of $\mathcal{A}$, we have that $Pr[\mathcal{A}(D) \in S] \leq e^{\epsilon} Pr[\mathcal{A}(D') \in S] + \delta$ for non-negative constants ϵ and δ.*

Definition 2 *Label Differential Privacy [18] - Let ϵ and δ be non-negative constants. A randomized training algorithm $\mathcal{A}$ taking as input a dataset is said to be (ϵ, δ) label differentially private (ϵ, δ)-LabelDP) if for any two training datasets D and D' that differ in the label of a single example, and for any subset S of outputs of $\mathcal{A}$, we have that $Pr[\mathcal{A}(D) \in S] \leq e^{\epsilon} Pr[\mathcal{A}(D') \in S] + \delta$.*

Definition 3 *Gaussian Mechanism [14] - The Gaussian mechanism for a d-dimensional function f adds noise $N(0, \sigma^2)$ to each one of the components of f. For $c^2 > 2ln(1.25/\delta)$, $\Delta = L_2$ sensitivity of f and $\sigma \geq c\Delta/\epsilon$, the Gaussian mechanism is (ϵ, δ) DP.*

Secure Multi Party Computation (MPC) Building Blocks Secure multiparty computation protocols for, n players, are possible with information theoretical security if all the players are connected by pairwise, private and authenticated channels and at least $2/3n + 1$ players are honest [1, 2]. In case a broadcast channel is available, MPC is possible with an honest majority [19]. Protocols tolerating a dishonest majority need further assumptions. These assumptions can be computational ones [20], the existence and availability of other cryptographic protocols (such as oblivious transfer)[21] or pre-distributed trusted correlated data [22]. Our proposed solutions work under any secure general multiparty computational protocol. We provide implementations using Crypten[1] [23]. We refer to [23] for a detailed description of the implementation of our basic MPC building blocks. For ease of comprehension, we will state our protocols for n=2 (two computing parties) and using additive secret sharing. An additive secret sharing of a value $x \in \mathbb{Z}_q = \{0, 1, \cdots, q-1\}$ is a pair of random numbers x_a, x_b chosen uniformly from $\mathbb{Z}_q$ subject to the restriction that $x = x_a + x_b \mod q$. We denote the secret sharing of x by $[\![x]\!]_q = (x_a, x_b)$. Notice that two players can easily

[1] https://crypten.ai

compute shares of the addition of two secrets by just locally adding their respective shares, i.e. $[\![x]\!]_q + [\![y]\!]_q = [\![x + y]\!]_q = (x_a + y_a \mod q, x_b + y_b \mod q)$. To compute the multiplication of two secrets we use the protocol proposed by Beaver in [24], which is based on random pre-computed triples and requires one round of communications between Alice and Bob and the exchange of two ring elements from $\mathbb{Z}_q$. Let $x_1, x_2, \cdots, x_l$ be the binary representation of x. It is convenient to transform $[\![x]\!]_q$ into $[\![x_1]\!]_2, [\![x_2]\!]_2, \cdots, [\![x_l]\!]_2$, that is to transform a secret x shared over $\mathbb{Z}_q$ into binary secret shares of the bit of the binary representation of x. Such a task is accomplished by bit decomposition protocols [25]. There are also simple procedures for doing the reverse task, computing $[\![x]\!]_q$ from $[\![x_1]\!]_2, [\![x_2]\!]_2, \cdots, [\![x_l]\!]_2$ [11]. We also use secure comparison protocols [26, 27, 28]. In a secure comparison protocol, two parties hold secret inputs x and y respectively, and would like to check if $x < y$ or not. The outcome of the protocol is a bit shared between the two players which indicates if the input x is smaller than the input y or not but without revealing any other information about the inputs. We slightly abuse our notation and also use $[\![x]\!]_q$ when x is a vector with coordinates belonging into $\mathbb{Z}_q$. We denote the secure component wise multiplication (Hadamard product) of two vectors x and y by $[\![x]\!]_q \odot [\![y]\!]_q$. Finally, we note that since all of our computations happen over $\mathbb{Z}_q$, we need to map finite precision real valued inputs into appropriate integers. Details on how this mapping happens are presented in [23].

3 Problem Description and Protocols

Problem Description: We work in a two-party scenario. We are motivated by the use case when the model label is created from two data holders combining their data together and it is considered private such that we don't want to reveal it to neither party. Thus, in our scenario, one of the parties holds training features, while the label is secret shared between them. The parties have shared identifiers for each row of data, or training example, that can be used to align the dataset during the training process. We'll refer to the party holding the training features as Alice and the other party as Bob. We are interested in training a deep learning model in such a situation. The model will be made available to Alice after training. Bob will not receive any output in our protocols. Informally, we say a protocol is secure if, upon protocol completion, Bob learns nothing, and Alice learns solely differentially private information about the labels.

We work in a scenario where there is no trusted curator to obtain global differential privacy. Moreover, since the labels, cannot be directly accessed by any party and are secretly shared between Alice and Bob, a direct use of local differential privacy is also out of question. The model will be trained using pairs $(x_i, y_i), 1 \leq i \leq n$. Alice is in possession of x_i, while y_i is a label secret shared between Alice and Bob and not known to any of them. The desired output is a vector of weights $W = (w_1, ..., w_d)$ that will be known only to Alice. Bob receives no output at the end of the protocol. Our results are stated for this specific scenario of two-party computations, but they trivially generalize to a multiparty case where one of the computing parties holds the features x_i while the labels y_i are shared among three or more parties.

Protocol I - Randomized Response: Our first protocol is based on the randomized response mechanism. For the sake of simplicity, we restrict our analysis to the case where the labels are binary. An extension to multiple classes is presented in the appendix A.1. The main idea here is for Alice and Bob randomize the label y_i and reveal it in the clear. Since Alice and Bob cannot directly access the labels in the clear, by randomizing we mean that Alice and Bob will compute (using a secure two-party computation protocol) a bit $\hat{y}_i$ so that $\hat{y}_i = 1$ with probability p, for an appropriately chosen bias p. Alice and Bob then privately XOR $y_i$and $\hat{y}_i$ and release the result. A direct approach to generate $\hat{y}_i$ is for Alice and Bob to generate λ random bits each one of them and interpret these random bits as the binary expansion of a uniform number r in $[0, 1]$. Alice and Bob then use a private comparison protocol and check if $r < p$. Alice and Bob can privately evaluate the XOR of bits a and b by computing the formula: a XOR $b = a + b - 2ab$ over MPC. Once the $\hat{y}_i$ XOR y_i are available, we can use the training strategy proposed in Ghazi et. al. [18] to train a deep learning model with the randomized labels and the corresponding features x_i. Before presenting our protocol, we briefly remark that Alice and Bob can non-interactively produce shares of a random bit $[\![r]\!]_2$ by locally picking up random bits r_a and r_b and defining $r = r_a + r_b \mod 2$.

Algorithm 1: Randomized Response based Solution

```
input  : Alice inputs $x_i$, Alice and Bob input shares $[\![y]\!]_i$, $\{1 \leq i \leq n\}$, $p$ is a public parameter
         in $[0, 1]$
output : Trained Model
1  Alice and Bob locally generate lambda shares of random bits $[\![r_j]\!]_2, 1 \leq j \leq \lambda$, and define
   $r_1, ..., r_\lambda$ as the binary extension of a real number $r$ in $[0, 1]$;
2. Alice and Bob run a secure comparison protocol and obtain $[\![r < p]\!]_q$. The output of this
   protocol is a bit that is equal to one if $r < p$ or zero if $r \geq p$. Denote the output by $o_i$ and note it
   is secret shared between Alice and Bob. None of these parties know its value;
3  Alice and Bob compute $[\![\hat{y}_i]\!]_q = [\![y_i]\!]_q$ XOR $[\![o_i]\!]_q = [\![y_i]\!]_q + [\![o_i]\!]_q - 2[\![y_i]\!]_q[\![o_i]\!]_q$. Bob announces
   his shares of this computation in the clear to Alice. Alice recovers $\hat{y}_i$ in the clear;
4  Alice and Bob repeat steps 1,2 and 3 $n$ times for $y_1, ..., y_n$;
5  Alice uses $(x_i, \hat{y}_i)$ to compute the model in the clear using the training strategy proposed in [18];
```

Theorem 1 *Protocol 1 correctly computes randomized labels and is label-differentially private for Alice and Bob.*

Proof Sketch: To prove correctness, we observe that the bit o_i is one with probability p and thus y_i will be flipped with probability p. Privacy for Bob comes from the fact he never receives any output and only deals with secret shares in the protocol. Because of the security of the privacy protocol, the only information observed by Alice during the computation are the randomized labels. By making $p = 1/(e^\epsilon + 1)$ we we end up with an ϵ-labelDP.

Second Protocol - Label DP-SGD: While our randomized response algorithm is efficient and provides good accuracy performance, the utility of randomized response data decreases substantially for the case when the label space cardinality is high. Additionally, it releases information about individual, albeit randomized, entries in the dataset. That might not be compatible with privacy requirements that demand that every information publicly released has to be aggregated. In order to cope with these cases, we propose another protocol - an adaptation of DP-SGD to a label privacy scenario. Interestingly, we show that when only label privacy is required, no gradient clippings is necessary, contrary to what happens in the original DP-SGD [4].

Assume our dataset is of the form (x_i, y_i), where x_i represents the features in possession of Alice for the i-th data set entry, and y_i represents the corresponding attributed labels that are secret shared between Alice and Bob. The goal is to train a model that depends on d parameters here represented by W.

Our start idea is to run DP-SGD [4] on top of MPC. Since Alice has all the features x_i, forward passes can be performed in the clear. MPC will be used for computing gradients, aggregating them and adding noise. The result is then released in the clear and Alice can do a back propagation, updating the weights in the clear. Note that label information is only revealed in an aggregated format and after noise is added, so no violation of individual label privacy happens. Moreover, Alice's information is never sent to the Bob. However, we now show that since in our problem features are known by Alice and only the labels are unknown, we can improve the utility of differentially private stochastic gradient descent.

In the usual DP-SGD algorithm, gradients need to be kept differentially private. Thus, they need to be clipped, aggregated and noise should be added to them. We now point out that when only labels need to be kept private, clipping is not needed.

We work with cross-entropy loss function and with a binary classification problem (for the sake of simplicity). However, our results naturally extend to other loss functions as well as to multi class problems - these extensions are presented in the appendix A.2 . In the following, $\mathcal{L}$ denotes the loss function, p_i denotes the output probability of the output neuron for input x_i, and W represents the set of parameters. The design hinges on a key observation from the chain rule: $\frac{d\mathcal{L}(y_i,p_i)}{dW} = \frac{d\mathcal{L}(y_i,p_i)}{dZ_i} \cdot \frac{dZ_i}{dW} = (p_i - y_i) \cdot \frac{dZ_i}{dW}$, where Z_i is the output layer neuron value before the sigmoid transformation. Assume that the size of the mini batch is N.

Thus, the gradient depends on a scalar component $(p_i - y_i)$ times a vector dZ_i/dW (which depends on the label y_i). The scalar quantity is bounded $[-1, 1]$ and since only the labels need to be kept differentially private, we compute the sensitivity of $d\mathcal{L}(y_i, p_i)/dW$ based on variations on y_i. The overall sensitivity of $d\mathcal{L}(y_i, p_i)/dW$ is bounded by the maximum L_2 norm among all the vector dZ_i/dW within a mini batch. Denote such value by g and note it is known by Alice. We then should compute the aggregated noisy gradients $1/N(\sum_{i=1}^{N} d\mathcal{L}(y_i, p_i)/dW + \mathcal{N}(0, (g\sigma)^2 I_d))$, where $\mathcal{N}(0, (g\sigma)^2 I_d)$ denotes a Gaussian vector of dimension d (the total number of parameters of the model) and variance $(g\sigma)^2$ for an appropriately chosen constant $\sigma > 0$. The aggregated noisy gradient is then revealed and Alice can use it to update the d parameters of her model. Note that when computing $d\mathcal{L}(y_i, p_i)/dW$ only the quantity $(p_i - y_i)$ needs to be computed jointly by Alice and Bob. dZ_i/dW can be computed in the clear by Alice.

We now show that we can reduce the amount of noise necessary to make our solution differentially private. We do so by, again, exploiting the fact we are interested in label privacy and by using the iterative nature of the back propagation algorithm. Let L denote the total number of layers in our network, the L-th layer being the output layer. Rather than adding noise to the entire d-dimensional vector $1/N(\sum_{i=1}^{N} d\mathcal{L}(y_i, p_i)/dW)$ at once, we add noise to the aggregated gradient corresponding to the $L - 1$ layer (representing the weights connected to the output layer of the neural network). Denote such gradient by $1/N(\sum_{i=1}^{N} d\mathcal{L}(y_i, p_i)/dW^{L-1})$. We compute $1/N(\sum_{i=1}^{N} d\mathcal{L}(y_i, p_i)/dW^{L-1} + \mathcal{N}(0, (g_t\sigma)^2 I_t))$, where g_t is the sensitivity of $1/N(\sum_{i=1}^{N} d\mathcal{L}(y_i, p_i)/dW^{L-1})$ wrt variations on y_i. Alice uses the noisy aggregated t-dimensional gradient $1/N(\sum_{i=1}^{N} d\mathcal{L}(y_i, p_i)/dW^{L-1} + \mathcal{N}(0, (g_t\sigma)^2 I_t))$ to update the t parameters corresponding to the weights connected to the output layer of the network. These t dimensional noisy gradients are then back propagated to the parameters in remaining layers of the network. Differential privacy is ensured by its post-processing property. We remark that the variance of the noise added is independent of the total number of parameters of the model d, depending only on the number of weights connected to the last layer t. Our solution is presented in Algorithm 2. We now prove its security.

```
Algorithm 2: Label DP-SGD
input  : Alice inputs $x_i$, Alice and Bob input shares $\llbracket y_i \rrbracket_q$ $\{1 \leq i \leq n\}$, mini-batch size $N$, the
         number of weights connected to the output layer $t$, and $\sigma > 0$.
output : Trained model for Alice. No output for Bob
for each mini batch do
    Alice forward prop and outputs $p_i$ (output probability of the output neuron) using input $x_i$;
    Alice and Bob Compute $\llbracket p_i - y_i \rrbracket_q$ ;
    Alice computes $dZ_i/dW^{L-1}$ and secret shares it with Bob;
    Alice and Bob compute per sample gradient
      $\llbracket d\mathcal{L}(y_i, p_i)/dW \rrbracket_q = \llbracket (p_i - y_i) \rrbracket_q \llbracket dZ_i/dW^{L-1} \rrbracket_q$;
    Alice and Bob compute aggregated gradient $\llbracket \sum_{1}^{N} (p_i - y_i) dZ_i/dW^{L-1} \rrbracket_q$;
    Alice computes $g_t$ (maximum $L_2$ norm of $dZ_i/dW^{L-1}$ across all the mini batch);
    Bob generates in the clear $\mathcal{N}(0, I_t)$, a $t$-dimensional Gaussian noise vector with mean zero
      and variance one;
    Bob computes the secret share of the Gaussian noise vector $\llbracket \mathcal{N}(0, I_t) \rrbracket_q$ with Alice;
    Alice generates in the clear the square root of the variance needed for DP noise, $(g_t\sigma)^2$;
    Alice computes the secret shares the square root of the variance $\llbracket (g_t\sigma)^2 \rrbracket_q$ with Bob;
    Alice and Bob multiply $\llbracket \mathcal{N}(0, I_t) \rrbracket_q$ times $\llbracket \sqrt{(g_t\sigma)^2} \rrbracket_q$. This result is added to the
      aggregated gradients for the mini batch and divided by $N$ resulting in
      $\llbracket 1/N\{\sum_{1}^{N} (p_i - y_i) dZ_i/dW^{L-1} + \mathcal{N}(0, (g_t\sigma)^2 I_t)\} \rrbracket$;
    Bob sends his shares to Alice. Alice opens the noisy average gradients and updates the
      weights connected to the output layer of her model. Alice then back propagates these noisy
      weights to the remaining layers/weights;
end
```

Theorem 2 *Bob learns nothing in Protocol 2.*

Proof Sketch: Bob only receives secret shared data during the protocol execution and Bob receives no output once the protocol is finished.

Theorem 3 *If $\sigma = \sqrt{2ln(1.25/\delta)}/\epsilon$, Protocol 2 is (ϵ, δ) label DP for Alice for a single mini batch.*

Proof Sketch: First, we need to compute the sensitivity of a function $f = \sum_{i=1}^{N}(p_i - y_i)dZ_i^t/dW^{L-1}$, where N is the size of the mini batch and dZ_i^t/dW^{L-1} denotes the gradient corresponding to the t weights connected to the output layer. The sensitivity of f is defined as the maximum across any pair of datasets D and D' that differ in a single label y_i. Noting that $(p_i - y_i) \leq 1$, the sensitivity of f is upper bounded by $g_t = max_i||dZ_i^t/dW^{L-1}||_2$, where $|| \cdot ||_2$ denotes the L_2 norm. It immediately follows that by adding noise $N(0, \sigma^2 g_t^2)$ to each coordinate of $1\sum_{i=1}^{L}(p_i - y_i)dZ_i^t/dW^{L-1}$, where $\sigma = \sqrt{2ln(1.25/\delta)}/\epsilon$ we end up with an (ϵ, δ) label private differentially private scheme. The privacy of the labels when computing the remaining coordinates of the gradient is ensured by the post-processing property of differential privacy.

By using parallel composition, the protocol will be $\max_j(\epsilon_j, \delta_j)$ for one pass across all the j mini-batches (one epoch). For T epochs, we can obtain the resulting (ϵ^*, δ^*) by using the accountants method [4].

Remark: Forward and backward passes happen in the clear. MPC is solely used for computing $2t$ private multiplications per data set entry. So, expensive activation function computations are all performed in the clear. The total number of secure multiplications depends solely on the number of neurons in the second last layer, so it is independent of the total number of layers of the network. Our result makes it possible to train deep networks on MPC for label privacy case.

4 Implementation Results

Complexity Analysis We measure the complexity of our protocols by the number of secure multiplications and the round complexity. Our protocol based on randomized response for the binary classification case performs one secure comparison and one secure multiplication per data set entry. The private comparison protocol needs 3λ multiplications. We use $\lambda = 16$ in our experiments. The round complexity of the comparison protocol is equal to λ [23]. The randomized response for c classes needs two secure comparison protocols plus four multiplications per data set entry. The cost of each one of the comparison protocols is 3λ and $\log_2 c$ multiplications, respectively. The round complexity of the protocol is $\max\{3\lambda, \log_2 c\}$. Our protocol based on Label Private DP-SGD requires $2tc$ secure multiplications per data set entry, where t is the number of neurons of the second last layer, and c is the number of classes. The round complexity of the protocol is 3 rounds per mini batch.

Experimental Setup and Runtimes We now describe runtimes for our protocols. We present runtime only for the operations that happen over MPC, computations that happen in the clear are not included. We start by presenting our results for the randomized response inspired protocols. We have chosen $q = 64$ bits, $\lambda = 16$ bits, and analyze binary and a ten classes classification problems. The results presented are averaged over 10 runs. We performed our simulations on a virtual machine - Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz with 251G of RAM.

	Time for 50k entries	Time for 100k entries
RR 2 Classes	1.77s	2.2s
RR 10 Classes	2.48s	4.2s

Table 1: Runtime for Randomized Response based Protocols

We now present results for our label private DP-SGD protocols. We have chosen $q = 64$ bits and batch size = 128. Results are averaged over 5 runs.

We can see that the runtime performance of the randomized response-based mechanisms is better than that of Label DP-SGD. This result is expected since, for the parameters used in this experiments, we have a much higher number of private multiplications in Label DP-SGD. For ten classes, Label

	Time for 50k entries and $t = 128$	Time for 100k entries and $t = 128$	Time for 50k entries and $t = 512$	Time for 100k entries and $t = 512$
Label DP-SGD 2 Classes	4.02s	7.72s	55.5s	1min 45s
Label DP-SGD 10 Classes	1min 32s	2min 40s	4min 8 s	8 min 25s

Table 2: Runtime for Label DP-SGD Protocols

DP-SGD computes about $1.3 * 10^6$ secure multiplications per mini batch (128 data set entries), while the randomized response correspondent runs about $3k$ private multiplications per 128 data set entries. Additionally, the randomized response protocols are run only once, independent of the number of epochs used in the subsequent deep neural network training. For the sake of comparison, we trained a neural network with one convolutional layer (kernel size =5), three fully connected layers (with 256, 256, and 128 neurons respectively) and one output layer (1 neuron) using the traditional approach (secret share all the inputs and train the model using MPC - no release of DP information). The time required for MPC operations in the traditional approach was 38.4s per 100 data set entries. Using Label DP-SGD for (2 classes and $t = 128$) the runtime for the required MPC operations was 8ms per 100 data set entries.

Accuracy Estimation Experimental Setup The goal of accuracy analysis is to understand the accuracy deterioration due to DP noise for Protocol 2 - the label DP-SGD. The corresponding analysis for the randomized response-based protocol is exactly the same as presented in [18]. We use CIFAR-10[2] image dataset for model training and accuracy analysis. We apply ResNet18 [29] to the model training. The ResNet18 is known to be one of the state of the art models on the CIFAR-10 dataset. The fully connected layer connecting to the output layer has 500 neurons. We train the model with 100 epochs, fixed learning rate 0.005, momentum 0.9 and batch size 128. The epsilon for Protocol 2 is calculated by the Moment Accountant [4], which is a function of noise multiplier, delta and number of epochs. Particularly, the epsilon grows as we increase the number of epochs. We set delta as 1e-5. Notice that the accuracy is also a function of the number of epochs. Instead of reporting the accuracy result for epoch=100, we choose the optimal epoch number to balance the epsilon and accuracy.

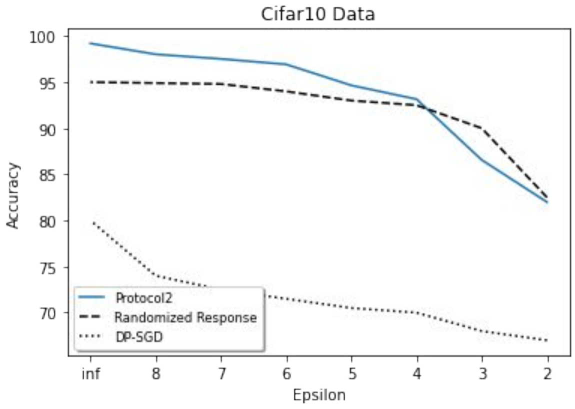

Figure 1: Accuracy results for the Randomized Response (RR) Protocol, Label DP-SGD (Protocol 2) and DP-SGD for the CIFAR10 data set (10 Classes). Results for the RR protocol are from [18]. Results for DP-SGD are from [4]

Observations for Accuracy Estimation The blue curve in Figure 1 depicts how accuracy changes as the epsilon decreases. The baseline accuracy (best accuracy for the model without any noise) is 99.2% under our experiment setup. The accuracy drop is less than 4.2% when epsilon is greater than or

[2] https://www.cs.toronto.edu/ kriz/cifar.html

equal to 4. The accuracy falls relatively sharply when epsilon slides from 4 to 2. To better understand how our label privacy mechanism performs relative to existing methods, we add randomized response and DP-SGD lines in Figure 1. The numbers are from [18]. It is worth noting that the baselines (when epsilon is infinity) are different due to model training differences and care should be taken when comparing the mechanisms. However, the learnings are still insightful. Specifically, we observe that our Protocol 2is much better suited for label privacy compared to DP-SGD. The DP-SGD shows a sharper drop when epsilon goes from infinity to 8. Moreover, the accuracy degradation from randomized response looks flatter than our Protocol 2 when epsilon is above 4. When epsilon is less than 4, the decline slopes are similar.

5 Conclusions

We have presented protocols that substantially increase the efficiency of MPC based training of machine learning models for the private label scenario. Our protocols exploit the idea of releasing differentially private information during the model training phase. We have two solutions, one based on randomized response (RR), and another one based on an adaptation of DP-SGD to a scenario where only the privacy of the labels is considered. For our RR protocol, MPC is used solely for flipping labels and the actual learning happens in the clear. For the Label DP-SGD protocol, heavy computations (activation functions, inner products, etc.) all happen outside MPC. The resulting complexity of Label DP-SGD is *independent* of the total size of the network, depending solely on the number of neurons connected to the output layer. Our results show that label private deep learning based on MPC is practical for very complex networks and large data sets. Also, to the best of our knowledge, our adaptation of DP-SGD to the label private scenario is novel and might be useful in other scenarios, such as federated learning. **Limitations:** Our solutions are dramatically more efficient than traditional MPC ones because we release differentially private information during training. From the privacy perspective, we do not see this release as negative, since it cannot be used to break the individual privacy of single entries. Moreover, a model trained with MPC without the use of any DP mechanism will leak information when used for inference after training, and this leak is not differentially private. However, w do pay a small price in accuracy for obtaining these advantages. Another limitation of our work is that we work with honest-but-curious adversaries. While there are compilers that can transform our protocol into one secure against fully malicious adversaries [30], there is a significant cost in performance. thus, we leave as an open problem to obtain efficient protocols for label private deep learning training secure against malicious players. **Societal Impacts:** By making practical the training of label private deep learning models over MPC, we bring the benefits of machine learning solutions while ensuring that the privacy of data contributors is respected. We believe that the solutions here presented can be applicable in social relevant applications. We do not see any negative societal impacts of our work.

Acknowledgements

We thank Prasad Buddhavarapu, Shripad Gade, Brian Knott, and Laurens van der Maaten for valuable comments and insights while working on this paper.

References

[1] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, 1988.

[2] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 307–328. 2019.

[3] Graham Cormode, Somesh Jha, Tejas Kulkarni, Ninghui Li, Divesh Srivastava, and Tianhao Wang. Privacy at scale: Local differential privacy in practice. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1655–1658, 2018.

[4] Martin 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, 2016.

[5] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015.

[6] Martine de Cock, Rafael Dowsley, Anderson CA Nascimento, and Stacey C Newman. Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 3–14, 2015.

[7] Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure mpc over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1102–1120. IEEE, 2019.

[8] Mark Abspoel, Daniel Escudero, and Nikolaj Volgushev. Secure training of decision trees with continuous attributes. *Proceedings on Privacy Enhancing Technologies*, 2021(1):167–187, 2021.

[9] Shreya Sharma, Chaoping Xing, and Yang Liu. Privacy-preserving deep learning with spdz. In *The AAAI Workshop on Privacy-Preserving Artificial Intelligence*, 2019.

[10] Valerie Chen, Valerio Pastro, and Mariana Raykova. Secure computation for machine learning with spdz. *arXiv preprint arXiv:1901.00329*, 2019.

[11] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson CA Nascimento, Wing-Sea Poon, and Stacey Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE Transactions on Dependable and Secure Computing*, 16(2):217–230, 2017.

[12] Devin Reich, Ariel Todoki, Rafael Dowsley, Martine De Cock, and Anderson CA Nascimento. Privacy-preserving classification of personal text messages with secure multi-party computation: An application to hate-speech detection. *arXiv preprint arXiv:1906.02325*, 2019.

[13] Martine De Cock, Rafael Dowsley, Anderson CA Nascimento, Davis Railsback, Jianwei Shen, and Ariel Todoki. High performance logistic regression for privacy-preserving genome analysis. *BMC Medical Genomics*, 14(1):1–18, 2021.

[14] 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.

[15] Melissa Chase, Ran Gilad-Bachrach, Kim Laine, Kristin E Lauter, and Peter Rindal. Private collaborative neural network learning. *IACR Cryptol. ePrint Arch.*, 2017:762, 2017.

[16] Kamalika Chaudhuri and Daniel Hsu. Sample complexity bounds for differentially private learning. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 155–186. JMLR Workshop and Conference Proceedings, 2011.

[17] Di Wang and Jinhui Xu. On sparse linear regression in the local differential privacy model. In *International Conference on Machine Learning*, pages 6628–6637. PMLR, 2019.

[18] Badih Ghazi, Noah Golowich, Ravi Kumar, Pasin Manurangsi, and Chiyuan Zhang. On deep learning with label differential privacy. *arXiv preprint arXiv:2102.06062*, 2021.

[19] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85, 1989.

[20] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[21] Donald Beaver and Shaft Goldwasser. Multiparty computation with faulty majority. In *Conference on the Theory and Application of Cryptology*, pages 589–590. Springer, 1989.

[22] Donald Beaver. Commodity-based cryptography. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 446–455, 1997.

[23] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. In *Proceedings of the NeurIPS Workshop on Privacy-Preserving Machine Learning*, 2020.

[24] Donald Beaver. One-time tables for two-party computation. In *International Computing and Combinatorics Conference*, pages 361–370. Springer, 1998.

[25] Tord Reistad and Tomas Toft. Linear, constant-rounds bit-decomposition. In *International Conference on Information Security and Cryptology*, pages 245–257. Springer, 2009.

[26] Thijs Veugen, Frank Blom, Sebastiaan JA de Hoogh, and Zekeriya Erkin. Secure comparison protocols in the semi-honest model. *IEEE Journal of Selected Topics in Signal Processing*, 9(7):1217–1228, 2015.

[27] Juan Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *International Workshop on Public Key Cryptography*, pages 330–342. Springer, 2007.

[28] Florian Kerschbaum, Debmalya Biswas, and Sebastiaan de Hoogh. Performance comparison of secure comparison protocols. In *2009 20th International Workshop on Database and Expert Systems Application*, pages 133–136. IEEE, 2009.

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

[30] Yehuda Lindell, Eli Oxman, and Benny Pinkas. The ips compiler: Optimizations, variants and concrete efficiency. In *Annual Cryptology Conference*, pages 259–276. Springer, 2011.

A Extension to Multi Class Cases

A.1 Randomized Response based Protocol

We now extend our randomized response-based protocol to the multi class case. Wlog we assume that $y_i \in \mathbb{Z}_c = \{0, 1, \cdots, c-1\}$, for some integer $c > 2$. Denote by y_i the label for the $i-$th input and by $\hat{y}_i$ its randomized version. The probability that y_i is not flipped is $Pr[y_i = \hat{y}_i] = \frac{e^\epsilon}{e^\epsilon + c - 1}$.

Otherwise, the label y_i will be flipped (with probability $\frac{1}{e^\epsilon+c-1}$) into one of the $c-1$ values $\mathbb{Z}_c \setminus y_i$ according to a uniform probability distribution.

Algorithm 3: Randomized Response based Solution - Multi Class Extension

```
input  : Alice inputs $x_i$, Alice and Bob input the secret shares shares $[\![y_i]\!]$, $\{1 \leq i \leq n\}$, $p$ is a
         public parameter in $[0, 1]$
output : Trained Model
2  Alice and Bob locally generate lambda shares of random bits $[\![r_j]\!]_2, 1 \leq j \leq \lambda$, and define
     $r_1, ..., r_\lambda$ as the binary extension of a real number $r$ in $[0, 1]$;
3  Alice and Bob privately compute $[\![r-p]\!]_q$ and run a private comparison protocol to check if
     $r-p<0$. The output of this protocol is a bit $o_i$ secret shared between Alice and Bob and it is
     equal to one if $r-p<0$ or zero if $r-p \geq 0$;
4  Alice and Bob secret share the publicly known value $[\![c-1]\!]_q$;
5  Alice picks up a random integer $a_i \in \mathbb{Z}_{c-1}$ according to a uniform distribution;
6  Alice picks up a random integer $b_i \in \mathbb{Z}_{c-1}$ according to a uniform distribution;
7  Define $f_i = a_i + b_i \mod c-1$;
8  Alice and Bob convert $[\![f_i]\!]_{c-1}$ into $[\![f_i]\!]_q$;
9  Alice and Bob run a private equality test protocol between $[\![y_i]\!]_q$ and $[\![f_i]\!]_q$. Denote the outcome
     of the comparison by $[\![h_i]\!]_q$. So, $h_i = 1$ iff $y_i = f_i$;
10 Alice and Bob compute $[\![\hat{y}_i]\!]_q = [\![o_i]\!]_q[\![y_i]\!]_q + (1 - [\![o_i]\!]_q)([\![h_i]\!]_q[\![c-1]\!] + (1 - [\![h_i]\!]_q)[\![f_i]\!]_q)$;
11 Bob announces his shares of this computation in the clear to Alice. Alice recovers $\hat{y}_i$ in the clear;
12 Alice and Bob repeat steps $1, 2, \cdots, 12$ $n$ times for $y_1, ..., y_n$;
13 Alice uses $(x_i, \hat{y}_i)$ to compute the model in the clear using the training strategy proposed in [18];
```

the arguments for proving correctness and security are the same as in the binary case.

A.2 Local DP-SGD

In order to generalize protocol 2 to the multi class case, we need to basically change the sensitivity analysis and the amount of noise that needs to be added to gradients to obtain an (ϵ, δ) label private solution. Let's start by the sensitivity analysis.

Assume the multi class label is one hot encoded and c is the position of the ground truth label. For example, if the label is $(0, 0, 1, 0)$, then c is 2 assuming our index starting from 0. Then the cross entropy loss can be defined as

$$\mathcal{L}(c, \mathbf{z}) = -\log \frac{e^{z_c}}{\sum_j e^{z_j}} = -\log p_c \tag{1}$$

where $\mathbf{z} = (z_1, ..., z_K)$ is output represented by logits and p_j is the output probability for output neuron j. Let W_j be the weight vector connecting to the jth logit in the output layer. And let $W = (W_1, ..., W_K)^T$ be the vector representing all weights connecting to output layer. By chain rule,

$$\frac{d\mathcal{L}}{dW} = \sum_{j=1}^{K} [p_j - I(j=c)] \frac{dz_j}{dW} \tag{2}$$

As we change the ground truth label c to c', the sensitivity is

$$|| \frac{d\mathcal{L}}{dW}(c) - \frac{d\mathcal{L}}{dW}(c') || = || \sum_{j=1}^{K} [I(j=c) - I(j=c')] \frac{dz_j}{dW} || \leq 2 \max_j || \frac{dz_j}{dW} || \tag{3}$$

If we consider sample index i, then equation (2) becomes

$$\frac{d\mathcal{L}}{dW} = \frac{1}{N} \sum_{ij}^{N} [p_{ji} - I(j=c_i)] \frac{dz_{ji}}{dW} \tag{4}$$

Then for any c_i and c_i' the sensitivity analysis becomes:

$$||\frac{d\mathcal{L}}{dW}(c_i) - \frac{d\mathcal{L}}{dW}(c_i')|| = ||\frac{1}{N}\sum_{ij}[I(j = c_i) - I(j = c_i')]\frac{dz_{ji}}{dW}|| \leq \frac{2}{N}\max_{ij}||\frac{dz_{ji}}{dW}|| \tag{5}$$

The nominator is 2 is because there $|I(j = c_i) - I(j = c_i')|$ is 1 only if j is c_i or c_i', otherwise 0. Therefore, the noise added to gradient (average across samples) $\frac{d\mathcal{L}}{dW}$ should be $N(0, \frac{4\sigma^2}{N^2}\max_{ij}||\frac{dz_{ji}}{dW}||^2)$.

The new protocol will be exactly like protocol 2, but with an updated noise variance. In the following $[\![y_i]\!]_q$ denotes secret shares of an output vector y_i (a one-hot encoding of the label for input i). $[\![p_i]\!]_q$ represents secret shares of a vector where each component is the output of one neuron in the output layer. Accordingly, dZ_i/dW^{L-1} is a tensor where each coordinate is the corresponding gradient for one of the output neurons.

Algorithm 4: Label Private DP-SGD - Multi Class

```
input  : Alice inputs $x_i$, Alice and Bob input shares $[\![y_i]\!]_q$ $\{1 \leq i \leq n\}$, mini-batch size $N$, the
         number of weights connected to the output layer $t$, and $\sigma > 0$.
output : Trained model for Alice. No output for Bob
for each mini batch do
    Alice forward prop and outputs $p_i$ (vector with output probability of each output neuron)
      using input $x_i$;
    Alice and Bob Compute $[\![p_i - y_i]\!]_q$ ;
    Alice computes $dZ_i/dW^{L-1}$ and secret shares it with Bob;
    Alice and Bob compute per sample gradient
      $[\![d\mathcal{L}(y_i, p_i)/dW]\!]_q = [\![(p_i - y_i)]\!]_q \odot [\![dZ_i/dW^{L-1}]\!]_q$, where $\odot$ represents the
      coordinate-wise product;
    Alice and Bob compute aggregated gradient $[\![\sum_1^N (p_i - y_i)dZ_i/dW^{L-1}]\!]_q$;
    Alice computes $g_t = 2\max_{ij}||\frac{dz_{ji}}{dW}||$;
    Bob generates in the clear $\mathcal{N}(0, I_t)$, a $t$-dimensional Gaussian noise vector with mean zero
      and variance one;
    Bob computes the secret share of the Gaussian noise vector $[\![\mathcal{N}(0, I_t)]\!]_q$ with Alice;
    Alice generates in the clear the square root of the variance needed for DP noise, $(g_t\sigma)^2$;
    Alice computes the secret shares the square root of the variance $[\![(g_t\sigma)^2]\!]_q$ with Bob;
    Alice and Bob multiply $[\![\mathcal{N}(0, I_t)]\!]_q$ times $[\![\sqrt{(g_t\sigma)^2}]\!]_q$. This result is added to the
      aggregated gradients for the mini batch and divided by $N$ resulting in
      $[\![1/N\{\sum_1^N (p_i - y_i)dZ_i/dW^{L-1} + \mathcal{N}(0, (g_t\sigma)^2 I_t)\}]\!]$;
    Bob sends his shares to Alice. Alice opens the noisy average gradients and updates the
      weights connected to the output layer of her model. Alice then back propagates these noisy
      weights to the remaining layers/weights;
end
```

Proofs os security and correctness are exactly the same as in Protocol 2.