FHE-compatible Batch Normalization for Privacy Preserving Deep Learning

Alberto Ibarrondo and Melek Önen

EURECOM
Sophia-Antipolis, France
`http//www.eurecom.fr`

Abstract. Deep Learning has recently become very popular thanks to major advances in cloud computing technology. However, pushing Deep Learning computations to the cloud poses a risk to the privacy of the data involved. Recent solutions propose to encrypt data with Fully Homomorphic Encryption (FHE) enabling the execution of operations over encrypted data. Given the serious performance constraints of this technology, recent privacy preserving deep learning solutions aim at first customizing the underlying neural network operations and further apply encryption. While the main neural network layer investigated so far is the activation layer, in this paper we study the Batch Normalization (BN) layer: a modern layer that, by addressing internal covariance shift, has been proved very effective in increasing the accuracy of Deep Neural Networks. In order to be compatible with the use of FHE, we propose to reformulate batch normalization which results in a moderate decrease on the number of operations. Furthermore, we devise a re-parametrization method that allows the absorption of batch normalization by previous layers. We show that whenever these two methods are integrated during the inference phase and executed over FHE-encrypted data, there is a significant performance gain with no loss on accuracy. We also note that this gain is valid both in the encrypted and unencrypted domains.

Keywords: Fully Homomorphic Encryption · Privacy · Deep Learning · Encryption · Cryptography · Neural Networks · Batch Normalization

1 Introduction

Deep Learning has recently become increasingly popular mainly due to the unprecedented computing capabilities promised by the cloud computing paradigm and the exponential increase on the size and amount of available datasets. Problems such as speech recognition, image classification, object detection/recognition or prediction have experienced major breakthroughs thanks to the use of highly complex Deep Neural Networks (DNN). DNN have two different phases: *training*, where a DNN model is optimized sequentially using large amounts of known and categorized data, and *inference* (often referred to as classification), where the optimized, trained DNN model is employed to process new data. While training poses an open challenge for academic and industrial actors alike, it is when performing inference that the real worth of DNN unfolds, generating substantial added value to organizations using them. It is common nowadays to reuse highly optimized DNN models by slightly adjusting them to fit particular needs (also known as transfer learning and fine tuning [16]), and then deploying them.

With the advent of cloud computing, the expensive computations required by DNN are being pushed to the cloud. Nevertheless, such an outsourcing poses a risk to the privacy and security of the data involved. When targeting problems where sensitive data is used, such as predicting illness using a database of patients or forecasting the likelihood of an individual to commit fraud by inspecting his bank movements, both the input data and the outcome require data protection.

Traditional data encryption solutions unfortunately fall short in ensuring the confidentiality of the data and taking advantage of cloud computing capabilities, ie. to apply the DNN model over encrypted data. While Fully Homomorphic Encryption (FHE) [5] allows any operation over encrypted inputs, obtaining the corresponding result in the encrypted domain, it unfortunately suffers from serious performance limitations. Some efficient versions of FHE, for instance Leveled Homomorphic Encryption (LHE) [3], have been later proposed encompassing additions and a limited number of multiplications, that is, low-degree polynomials.

The use of LHE for DNN inference purposes, immediately preserves the privacy of input and output data. However, given that LHE only supports polynomials, implementing DNN with LHE imposes the linearization of all the DNN operations or their approximations into low-degree polynomial equivalents. While some of the DNN operations such as fully connected and/or convolution layers are already linear, other functions, namely activation functions, pooling and batch normalization require some transformation in order to support LHE.

Most of recent privacy preserving neural networks mainly focus on the linearization and, in fact, on the approximation of the sigmoid activation function [6, 13, 4]. This paper studies the batch normalization layer which mainly consists of subtraction of the mean and division by the standard deviation for intermediate values in the network. As also observed by [4],this additional layer significantly improves the accuracy of the model. In this paper, we study this new layer and show how to adapt it in order to compute inference in LHE-encrypted inputs. In short, we observe that computations in the layers preceding batch normalization (namely fully connected or convolution) can be easily tweaked while remaining linear, and therefore propose to reformulate their parameters in a way that these layers mathematically absorb the BN layer. Thus, during the inference phase, instead of relying on two separate layers, the DNN only implements a single layer that inherently applies batch normalization.

This paper is organized as follows: Section 2 covers background knowledge on Fully Homomorphic Encryption, Deep Neural Networks, and Batch Normalization (BN). Section 3 reviews related work on privacy preserving neural networks and further describes the conflict between BN and LHE. Section 4 describes the details of the proposed re-parametrization method that allow convolution or fully connected layer to absorb the batch normalization and hence support the use of LHE. Section 5 analyzes the impact of the proposed method on accuracy and performance. Finally, Section 6 provides conclusive remarks, exploring the implications of this technique and foreseeing future research based on it.

2 Background

Notation In the rest of this paper we use the following notation:

- v: Scalar,
- $\mathbf{v}$: Vector,
- $\mathbf{V}$: Matrix or higher dimension tensor,
- $\langle \mathtt{d} \rangle$: Data d (scalar/vector/matrix) encrypted using FHE.

2.1 Fully Homomorphic Encryption (FHE)

Homomorphic encryption is the main cryptographic building block for outsourcing/delegating data and computation to an untrusted third party such as the cloud server. By definition, an encryption scheme E_k is defined as being "homomorphic" with respect to a function f, if given some inputs $(x_1, x_2, .., x_n)$, one can obtain $f(x_1, x_2, ..., x_n)$ by performing some operations over the individually encrypted inputs $(c_1, c_2, .., c_n)$ and decrypting the resulting value. While initial homomorphic encryption schemes named as partially homomorphic encryption schemes were supporting only additions [15] or multiplications [17], in 2009, Gentry introduced the first fully homomorphic encryption scheme (FHE) [5] which allows the execution of any arbitrary function over encrypted inputs. Unfortunately, this initial scheme and some of its subsequent improvements suffer from poor computation efficiency and prohibitive growth of ciphertext size. Therefore, researchers investigate leveled homomorphic encryption (LHE) solutions [3] that can handle polynomials over encrypted inputs. The encryption operation includes some noise in the encrypted output, which grows when performing some computations. Performance-wise, LHE slows down computations in a factor of 1000 or more. More concretely, while addition is rather fast and does not increase the noise meaningfully, multiplication is slow and it increases the noise considerably [7]. There is a limit on how many multiplications can be performed over the encrypted data due to high noise. Above this limit decryption of the ciphertext becomes impossible. A bootstrapping procedure can be used to control the noise growth and hence support higher degree polynomials. In that case, the encryption scheme becomes fully homomorphic. The bootstrapping procedure unfortunately remains very costly in terms of computation. LHE schemes which do not involve any bootstrapping operation but handle polynomials only, remain much more efficient. To sum up, the two main requirements for LHE in privacy preserving Deep Neural Networks are **transforming all operations into polynomials** and **avoiding** as many **multiplications** as possible while keeping high accuracy. The two popular libraries that implement leveled homomorphic encryption are "SEAL"[1] and "HELib"[2], which support additions and

[1] `https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library`

[2] `https://github.com/shaih/HElib`

multiplications over encrypted operands ($\langle a \odot b\rangle = \langle a\rangle \odot \langle b\rangle$) and unencrypted operands ($\langle a \odot b\rangle = \langle a\rangle \odot b$).

2.2 Deep Neural Networks (DNN)

DNN are a particular type of machine learning techniques, where sequential transformations called layers are applied to the input. Neural Networks fall into the category of *supervised learning*, where the data used to train the model is labeled: if the neural network is being trained to recognize handwritten digits (e.g.: MNIST dataset [3]), then it requires the dataset with images containing the handwriting samples and the corresponding real value (from 0 to 9). Modern DNN are composed of several kinds of layers:

– **Fully Connected (FC)** is the classical layer present in legacy Neural Networks [2]. Also known as Dense layer, it consists of a vector to vector transformation, where the input $\mathbf{x}$ is multiplied by a matrix $\mathbf{W}$ of weights and subtracted a vector of biases $\mathbf{b}$. Conventionally, each value in input and output vectors is denominated as neuron. The FC layer is expressed as:

$$\mathbf{y_{FC}} \equiv FC(\mathbf{x}) = \mathbf{x} * \mathbf{W} - \mathbf{b} \tag{1}$$

– **Convolutional Layer (Conv)** applies spatial convolution to a matrix $\mathbf{X}$ (figure 1), multiplying the values of a filter $\mathbf{W}$ to contiguous sub-regions in $\mathbf{X}$ and then adding a bias $\mathbf{B}$ to the result. By convention, all values in $\mathbf{B}$ are the same. The spacing between sub-regions is named *strides (s)*, and the border appended to $\mathbf{X}$ in order to maintain the same size between $\mathbf{X}$ and $\mathbf{Y}$ is defined as *padding (p)*. The Conv layer is formulated as:

$$\mathbf{Y_{Conv}} \equiv Conv_{s,p}(\mathbf{X}) = \mathbf{X} \oplus \mathbf{W} + \mathbf{B} \tag{2}$$

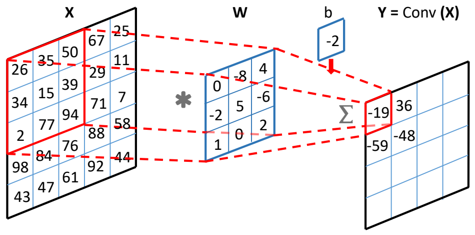

Fig. 1: Spatial Convolution in Conv layer

Although the process of applying the filter to sub-regions is iterative and slow, by appropriately vectorizing both the input $\mathbf{X}$ and the filter $\mathbf{W}$, as

[3] `http://yann.lecun.com/exdb/mnist/`

well as replicating the latter one, spatial convolution can be expressed as a matrix multiplication (similar to FC layer). Alternatively, applying FFT to the whole layer turns convolution into a multiplication.

– **Activation Function** is a mathematical function applied to individual values of a tensor, therefore it is easily parallelizable. It is generally non-linear, constituting the main non-linearity of Deep Neural Networks: this allows DNN to solve non-linear problems. Activation functions are located after FC and Convolutional layers. The most common variants are sigmoid σ, hyperbolic tangent $tanh$ and Rectifier Linear Unit $ReLU$ (see figure 2).

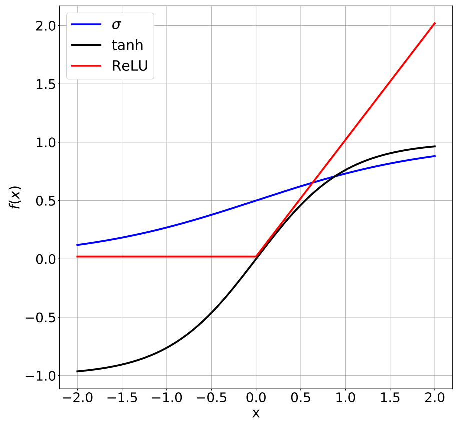

Fig. 2: Common activation functions around their non-linearity at $x = 0$

– **Pooling layer (Pool)** computes a reduction function over sub-regions of the input image $\mathbf{X}$, thus reducing its size while maintaining the number of dimensions. Most typical reduction functions are max and $average$.

– **Batch Normalization (BN)**. Due to its relative importance for this paper, we will dive deep in its understanding in the next subsection.

2.3 Batch Normalization (BN)

Batch Normalization [11] is a layer that is trained over batches of input data. The BN layer reduces internal covariance shift by 'normalizing' each data point with respect to the batch $\mathbf{B}$: subtracting mean of the batch and dividing by standard deviation of the batch. It is generally applied before activation functions, since it packs the values in a small interval around $x = 0$, obtaining a distribution that makes smarter use of their non-linearities. This prevents the model training from getting stuck in saturated modes, also helping to handle gradient explosion. During the training phase, a BN is computed using the following operations over each batch $\mathbf{B}$ of size m and each training step k:

$$\begin{aligned}
\textbf{Input} \rightarrow & \mathbf{B} = \{x_0, x_1, ..., x_m\} \\
& \mu_{\mathbf{B}} \leftarrow \frac{1}{m} \sum_{m}^{i=1} x_i \\
& \sigma_{\mathbf{B}}^2 \leftarrow \frac{1}{m} \sum_{m}^{i=1} (x_i - \mu_{\mathbf{B}})^2 \\
& \hat{x_i} \leftarrow \frac{x_i - \mu_{\mathbf{B}}}{\sqrt{\sigma_{\mathbf{B}}^2 + \epsilon}} \\
& y_i \leftarrow \gamma \hat{x_i} + \beta \equiv BN_{\gamma,\beta}(x_i) \\
\textbf{Output} \rightarrow & \{y_0, y_1, ..., y_m\}
\end{aligned} \tag{3}$$

In a Batch Normalization layer there are two trainable parameters γ and β, optimized using backpropagation. β is a shifting parameter, while γ is a scaling parameter. Mean μ_B and variance σ_B^2 of each batch are calculated and stored. To avoid zero division, BN also includes a very small constant $\epsilon \approx 10^{-9}$.

Once the network has been trained, "the values of the mean and variance are fixed during inference" (see [11] page 4). This is accomplished computing the unbiased estimators for μ_B and σ_B^2 across all N batches B of size m:

$$\mathbf{E}(\mathbf{x}) \equiv \mu_{\mathbf{T}} = \frac{1}{N} \sum_{k=1}^{N} \mu_{\mathbf{B_k}} \qquad \mathbf{Var}(\mathbf{x}) \equiv \sigma_{\mathbf{T}}^{\mathbf{2}} = \frac{m}{m-1} \frac{1}{N} \sum_{k=1}^{N} \sigma_{\mathbf{B_k}}^{\mathbf{2}} \tag{4}$$

This allows **inference** to be performed with static parameters: no mean or variance is calculated on this phase. The BN layer during inference uses μ_T and σ_T^2 to perform the scalar transformation:

$$y_{BN} \equiv BN_{\gamma,\beta,\mu_T,\sigma_T^2}(x) = \gamma * \frac{x - \mu_T}{\sqrt{\sigma_T^2 + \epsilon}} + \beta \tag{5}$$

In practice, BN has become part of de-facto standard architectures such as ResNet [9], and its contribution to accuracy improvement is more than established in the Deep Learning community.

3 Problem Statement

3.1 Encrypting Deep Neural Networks

As mentioned in section 1, our goal is to protect data when performing inference using DNN. Indeed, the cloud who performs the inference operation, is considered as an honest-but-curious adversary. Hence data needs to be encrypted Applying FHE to encrypt data during inference phase leads to executing all operations inside each layer over encrypted data. Regarding their compatibility with LHE, DNN layers can be regrouped into two categories:

– *Linear layers*: their internal operations are intrinsically linear/polynomial. FC and Conv layers fall within this category. Encrypting these layers using LHE is completely straightforward:

$$\begin{aligned} \langle \mathbf{y_{FC}} \rangle \equiv \langle FC(\mathbf{x}) \rangle = \langle \mathbf{x} * \mathbf{W} - \mathbf{b} \rangle = \langle \mathbf{x} \rangle * \langle \mathbf{W} \rangle - \langle \mathbf{b} \rangle \\ \langle \mathbf{Y_{Conv}} \rangle \equiv \langle Conv(\mathbf{X}) \rangle = \langle \mathbf{X} \oplus \mathbf{W} - \mathbf{B} \rangle = \langle \mathbf{X} \rangle \oplus \langle \mathbf{W} \rangle - \langle \mathbf{B} \rangle \end{aligned} \tag{6}$$

– *Non-linear layers*: Activation functions, pooling and Batch Normalization are non-linear: They include non-polynomial functions/operations. In order to be compatible with LHE, activation functions have been subject to many previous research papers (see section 3.2) which use techniques to approximate them such as Taylor or Chebyshev polynomials. Furthermore, in [6], pooling layers were proven to have decent approximations when substituting *max* and *mean* functions by *scaled mean.* On the other hand, batch normalization (non-linear due to division and square root) has not yet been studied to become compatible with LHE. Authors in [4] mention this layer without giving further information on how to adapt it to the encrypted domain.

3.2 Related work

Existing solutions that ensure data privacy for neural networks usually are based on either homomoprhic encryption [6, 4] or secure two-party computation [14, 13]. Early solutions such as the one in[1] leave the server with the execution of the linear operations over encrypted data only and require the data owner to locally perform the non-linear operations over intermediary decrypted data. More recent solutions apply approximation of the non-linear functions (the activation functions, like sigmoid or hyperbolic tangent) with low-degree polynomials over which homomorphic encryption can be performed more efficiently. Initially, Gilad-Bachrach et al. [6] proposed CryptoNets, a convolutional NN for data encrypted with LHE and suggest to approximate the activation function with x^2 and the max pooling function with mean pooling. This inherently causes some loss in terms of inference accuracy. Later on, solutions such as [4] mainly focused on the improvement of this accuracy. On the other hand, [14] and [13] define a two-server model whereby the data owners distribute the shares of their data among two non-colluding servers that further perform classification on the joint (but private) data using secure two-party computation.

Among LHE solutions approximating activation functions, Chabanne et al. [4] attempt to improve the accuracy of the inference model by introducing a batch normalization layer before activation functions, similarly to our work. Unfortunately, the paper does not give any detail on the integration of this function when combined with LHE. Our proposed solution not only makes use of BN to improve the accuracy of the model but also integrates the underlying BN operations within FC or Conv in order to have linear operations only and easily support FHE. Hence, BN is not considered as a separate layer, and we show that the number of operations is noticeably reduced. Furthermore, this new solution does not have any impact on the accuracy of the model and increases the performance of the inference phase (see section 5, for more details).

3.3 BN vs. LHE: how to address conflicting requirements?

The targeted problem is how to preserve privacy of data (input and output) when performing inference in Deep Neural Networks. In a context where Deep Learning is being pushed to the cloud, we consider a scenario where a data owner initially gets his hands on a trained DNN model, either by training it himself (be it from scratch, be it from applying transfer learning[16]) or by obtaining an already trained model. Benefiting from cost savings, ubiquity and high availability, this DNN model is then outsourced to the cloud to run the inference phase. In order to ensure data privacy against the cloud while enabling DNN inference calculation, data encryption becomes mandatory.

In our solution, as in previous solutions, this is achieved by applying LHE thanks to its homomorphic nature. Notwithstanding that most layers (FC, Conv, pooling and activation functions) have already been implemented in the LHE encrypted domain, batch normalization remains outside the scope of LHE-encrypted DNNs. The operations included in batch normalization include a division and a square root, which cannot be directly implemented with LHE. In order to linearize it, given that Taylor or other polynomial approximations are clearly inefficient in the sense that they require several multiplications while yielding poor approximations, we suggest a different approach: considering BN during inference as a linear transformation, we first reformulate their parameters, and then combine it with preceding linear layers (Fully Connected or Convolutional), expressing the concatenated layers as a single layer.

We hereafter propose a re-parametrization trick for BN which allows batch normalization layers to be included in privacy preserving DNN while remaining linear and thus compatible with the use of LHE.

4 Solution

4.1 A first approach: reformulating Batch Normalization

Firstly, provided that σ_T^2 is the only parameter in BN layers that is operating with functions other than sums and multiplications, we propose a small reformulation inside BN layers for all the operations to be compliant with the two requirements identified in section 2, for the compatibility with LHE. We start with encrypting the formula in equation 5 which corresponds to the BN layer and we obtain the following equation:

$$\begin{aligned}\langle y_{BN}\rangle \equiv \left\langle BN_{\gamma,\beta,\mu_T,\sigma_T^2}(x)\right\rangle &= \left\langle \gamma * \frac{x-\mu_T}{\sqrt{\sigma_T^2+\epsilon}} + \beta \right\rangle \\ &= \langle\gamma\rangle * (\langle x\rangle - \langle\mu_T\rangle) * \left\langle \frac{1}{\sqrt{\sigma_T^2+\epsilon}} \right\rangle + \langle\beta\rangle\end{aligned} \tag{7}$$

We cannot easily compute the square root and the division in the LHE encrypted domain. However, we realize that we may not need to compute them and instead, we take σ_T^2 from the freshly trained DNN and compute the inverse of its square root over plaintext values. This newly transformed parameter denoted by ϕ is stored to be used in encrypted inference. We can take advantage of :

$$\phi \equiv \frac{1}{\sqrt{\sigma_T^2+\epsilon}} \rightarrow \langle y_{BN'} \rangle \equiv \langle BN_{\gamma,\beta,\mu_T,\phi}(x) \rangle = \langle \gamma \rangle * (\langle x \rangle - \langle \mu_T \rangle) * \langle \phi \rangle + \langle \beta \rangle \quad (8)$$

We can push this reformulation even further, and minimize the number of operations performed inside an encrypted BN layer by grouping parameters:

$$\left.\begin{array}{l} \upsilon \equiv \frac{\gamma}{\sqrt{\sigma_T^2+\epsilon}} \\ \tau \equiv \beta - \frac{\mu_T * \gamma}{\sqrt{\sigma_T^2+\epsilon}} \end{array}\right\} \rightarrow \langle y_{BN''} \rangle \equiv \langle BN_{\upsilon,\tau}(x) \rangle = \langle \upsilon \rangle * \langle x \rangle + \langle \tau \rangle \quad (9)$$

This way, the batch normalization layer can be computed in the LHE-encrypted domain by performing only one addition and one multiplication.

4.2 The re-parametrization trick: absorbing BN layer

Despite the reformulation of BN detailed in the previous section, BN still involves some computations. We will now show how to make these computations completely disappear while keeping the effect of BN. We start with a trained DNN with BN layers. Our only requirement for the DNN is to have a linear layer (FC or Conv) right before the BN layers, which is indeed the standard case for existing DNN architectures [8] [10]. The idea is to absorb the BN operations using the parameters from FC and Conv ($\mathbf{W}$ and $\mathbf{b}$). With this setup, we can merge FC/Conv equations (1 and 2) with Batch Normalization (equation 5). For the FC layer we would obtain:

$$\begin{aligned} \mathbf{y_{BN\&FC}} &\equiv BN(FC(\mathbf{x})) = BN(\mathbf{W} * \mathbf{x} - \mathbf{b}) \\ &= \gamma * \left(\frac{(\mathbf{W} * \mathbf{x} - \mathbf{b}) - \mu_T}{\sqrt{\sigma_T^2+\epsilon}} \right) + \beta \end{aligned} \quad (10)$$

By rearranging all the parameters we obtain:

$$\begin{aligned} \mathbf{y_{BN\&FC}} &= \gamma * \left(\frac{(\mathbf{W} * \mathbf{x}) - (\mathbf{b} + \mu_T)}{\sqrt{\sigma_T^2+\epsilon}} \right) + \beta \\ &= \gamma * \left(\frac{\mathbf{W} * \mathbf{x}}{\sqrt{\sigma_T^2+\epsilon}} - \frac{\mathbf{b} + \mu_T}{\sqrt{\sigma_T^2+\epsilon}} \right) + \beta \\ &= \frac{\gamma * \mathbf{W} * \mathbf{x}}{\sqrt{\sigma_T^2+\epsilon}} - \frac{\gamma * (\mathbf{b} + \mu_T)}{\sqrt{\sigma_T^2+\epsilon}} + \beta \\ &= \left(\frac{\mathbf{W} * \gamma}{\sqrt{\sigma_T^2+\epsilon}} \right) * \mathbf{x} - \left(\frac{\gamma * (\mathbf{b} + \mu_T)}{\sqrt{\sigma_T^2+\epsilon}} - \beta \right) \end{aligned} \quad (11)$$

We now define the reparametrized weights and biases for the FC layer as follows:

$$\mathbf{W_{new}} = \mathbf{W} * \frac{\gamma}{\sqrt{\sigma_T^2 + \epsilon}} \qquad \mathbf{b_{new}} = (\mathbf{b} + \mu_T) * \frac{\gamma}{\sqrt{\sigma_T^2 + \epsilon}} - \beta \tag{12}$$

With these new weights and biases we have absorbed the BN layer into the preceding FC layer, while still performing mathematically equivalent operations:

$$\mathbf{y_{BN\&FC}} \equiv BN(FC(\mathbf{x})) \equiv FC_{new} = \mathbf{W_{new}} * \mathbf{x} - \mathbf{b_{new}} \tag{13}$$

Since spatial convolution is also linear, the same reparametrization trick can be applied to Conv layers followed by a BN layer:

$$\mathbf{Y_{BN\&Conv}} \equiv BN(Conv(\mathbf{X})) \equiv Conv_{new} = \mathbf{W_{new}} \oplus \mathbf{X} - \mathbf{B_{new}} \tag{14}$$

The re-parametrization trick allows us to completely absorb BN layers, whereas reformulation of BN layers only reduced the number of operations. Applying LHE to equations 13 and 14 is just as straightforward as it was in equation 6.

4.3 Integration with neural networks

In this section, we finally study how the re-parametrization trick is integrated within the actual neural network. As an overview, the training phase of the DNN remains unmodified, allowing all previous DNN to be trained as they were before, as well as obtaining already trained networks. Before using the trained DNN to perform inference over encrypted data, we apply both the reparametrization and the reformulation transformations. First of all, the blocks of $[FC \rightarrow BN]$ and $[Conv \rightarrow BN]$ present in the DNN are **reparametrized** into FC_{new} and $Conv_{new}$ respectively using equation 12. Secondly, for the remaining BN layers that are not preceded by a FC/Conv layer, we **reformulate** them employing equation 9: $BN_{\gamma,\beta,\mu_T,\phi}(x) \rightarrow BN_{\upsilon,\tau}(x)$.

Whilst the NN is transformed, the overall mathematical computations remain equivalent, thus having theoretically zero impact on the accuracy of the DNN while conserving the properties of BN layers. At this point we could perform inference over LHE-encrypted data at the untrusted (honest-but-curious) cloud, preserving privacy of data. The process is depicted in figure 3. Note that, when **deploying the model** to the cloud, thanks to the available operations in the LHE domain, we can choose to **either encrypt** the model (degrading performance), **or leave it as plaintext** (unsafe but substantially faster for computations). It should be noted that existing privacy preserving neural network solutions consider that the cloud knows the model (the model is not encrypted) but cannot discover the inputs and outputs of the inference phase (the data is encrypted).

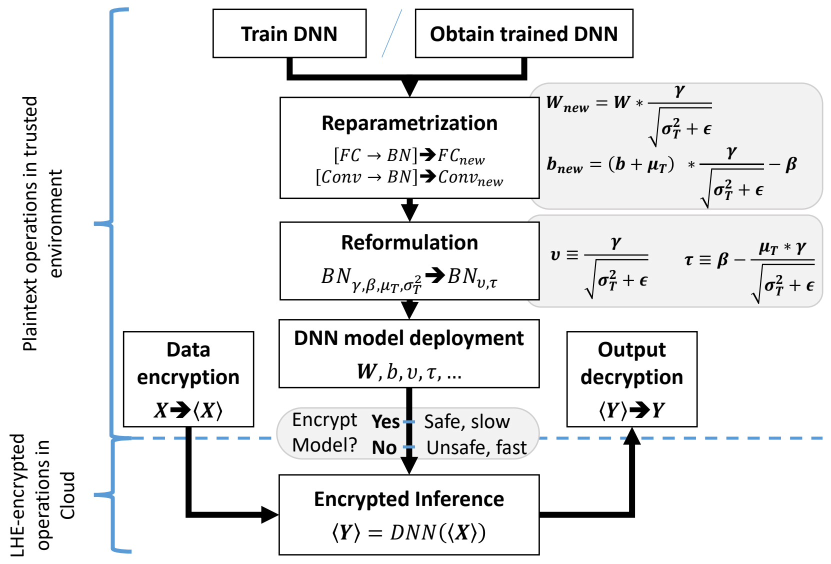
Plaintext operations in trusted environment
LHE-encrypted operations in Cloud
Train DNN
Obtain trained DNN
Reparametrization
$[FC \rightarrow BN] \rightarrow FC_{new}$
$[Conv \rightarrow BN] \rightarrow Conv_{new}$
$W_{new} = W * \frac{\gamma}{\sqrt{\sigma_T^2 + \epsilon}}$
$b_{new} = (b + \mu_T) * \frac{\gamma}{\sqrt{\sigma_T^2 + \epsilon}} - \beta$
Reformulation
$BN_{\gamma,\beta,\mu_T,\sigma_T^2} \rightarrow BN_{\upsilon,\tau}$
$\upsilon \equiv \frac{\gamma}{\sqrt{\sigma_T^2 + \epsilon}}$
$\tau \equiv \beta - \frac{\mu_T * \gamma}{\sqrt{\sigma_T^2 + \epsilon}}$
Data encryption
$X \rightarrow \langle X \rangle$
DNN model deployment
$W, b, \upsilon, \tau, \ldots$
Output decryption
$\langle Y \rangle \rightarrow Y$
Encrypt Model?
Yes
Safe, slow
No
Unsafe, fast
Encrypted Inference
$\langle Y \rangle = DNN(\langle X \rangle)$

Fig. 3: Encrypted DNN inference with BN reformulation & reparametrization

5 Evaluation

5.1 Impact on Accuracy

In this section we test the zero impact of re-parametrization and reformulation on the overall accuracy of the DNN **during inference**. In order to test the veracity of this statement, we have implemented two unencrypted DNNs in Tensorflow [4], one with more than 15M parameters (figure 4, top) and one with less than 200k parameters (figure 4, bottom). Both DNN possess one BN layer that could fuse with a Conv layer and one BN that could fuse with a FC layer. In both cases we use ReLU as activation function. Details can be found in Appendix I.

Using the MNIST dataset [12], we briefly trained each network (Adam optimizer, 5 epochs with learning rate 0.01 and 5 epochs with learning rate 0.001), then performing inference to obtain 99.02% and 98.80% of test accuracy respectively. Afterwards, we applied the re-parametrization trick on $[Conv + BN]$ and $[FC + BN]$ blocks, and re-evaluated the test accuracy. We observed that we indeed obtained exactly the same results for both networks. Finally, we applied the proposed reformulation technique to both BN layers and performed the test again, obtaining the exact same accuracy scores. This validates our approach.

5.2 Performance analysis

This section analyzes the performance of the new solution, revealing noticeable computational savings with respect to a NN with standard BN.

[4] Code available in `https://github.com/ibarrond/reparametrization-BN.git`

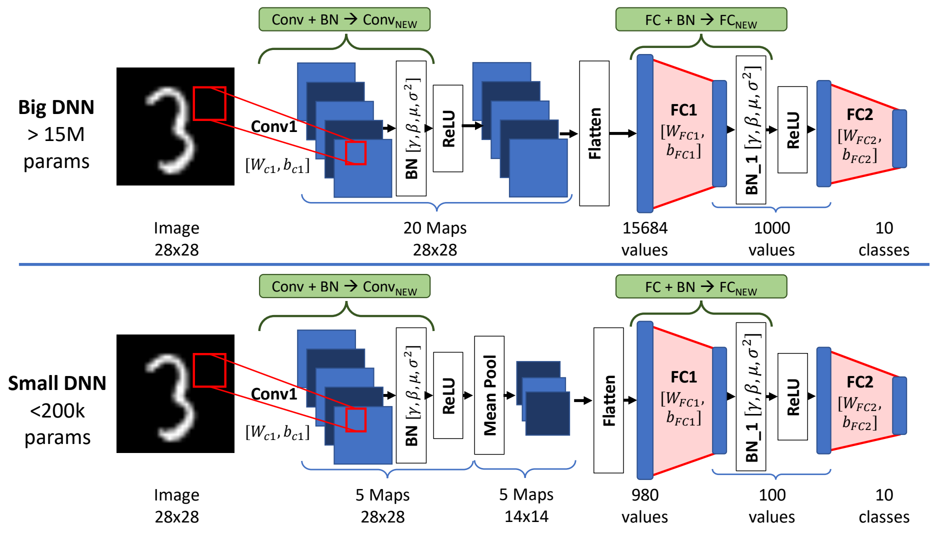

Fig. 4: DNNs with Batch Normalization after Conv and FC layers used for testing.

Case with no privacy Empirically, we propose to measure the time taken to perform inference over 10.000 images for the two networks described in section 5.1. We then compare it with the time that the re-parametrized and reformulated versions take. Table 1 shows the total inference time (in ms) for 10,000 images averaged over 30 executions and in two different settings (with a CPU and a GPU).

Table 1: Performance of DNN with reparametrization and reformulation without any privacy protection

Platform	CPU Intel i7 6700HQ		GPU Tesla V100	
Network	15M par.	200K par.	15M par.	200K par.
Original	48.1 ms	34.6 ms	11 ms	10.2 ms
Reformulated	43.2 ms	31.8 ms	10.1 ms	9.4 ms
Reparametrized	38.0 ms	27.2 ms	8.6 ms	7.93 ms

As shown in this table, we can conclude that the proposed solution significantly improves the performance of the inference within the plaintext domain. Indeed, in average, reformulation of BN layers yields a 9% performance boost, while re-parametrization with FC/Conv layers shows a 21% performance boost.

Case where data is encrypted with LHE In order to evaluate the cost of integrating privacy, we have followed an incremental approach. We have first considered that the model remains in plaintext and the data is encrypted only. Further on, we have encrypted the model as well and applied it over the encrypted data. We have used a small DNN and have implemented it with naive algorithms each of the DNN layers using the LHE open source library *HElib*[7], an implementation of the BGV encryption scheme developed purely over CPU. Similarly to existing solutions, we have used Taylor polynomials of degree 2 to approximate ReLU activation functions around $x = 0$. We have performed inference over one single image employing the trained 200k model. The results, shown in table 2, testify the large overhead present when dealing with LHE operations. Nonetheless, the performance gain observed in plaintext is still perceivable, although it has decreased in magnitude: using re-parametrization, we observe a 14% of increased performance with the unencrypted/plaintext model and a 12% with the encrypted DNN model. This is due to the fact that we're already avoiding calculation of square root and division by applying the ϕ reformulation. Additionally, we also notice the x7 drop in performance when encrypting the DNN model.

Table 2: Performance of DNN (LHE-encrypted or not) for inference over a single encrypted image

DNN model	200K unencrypted model	200K LHE-encrypted model
Original (ϕ)	6.48 min	47.4 min
Reformulated (τ, υ)	6.16 min	45.2 min
Re-parametrized	5.54 min	41.7 min

6 Conclusion

This paper has studied the problem of privacy preserving Deep Neural Networks when the inference phase is outsourced to the cloud and is executed over data encrypted with LHE. While existing work mostly focus on the compatibility of activation functions with FHE/LHE, we investigated the batch normalization layer and propose a new solution that is suitable to the use of LHE. We hence propose to reformulate the BN layer, linearize the operations and further integrate within the convolution or fully connected layers. The proposed techniques show a performance gain of 21% over plaintext data, and 12% - 14% over encrypted data using an encrypted - unencrypted model respectively; all of this without any drop in the model accuracy.

Thanks to the proposed solution, complex modern DNN models that make heavy use of Batch Normalization are now compatible with FHE. This allows the execution of inference models over encrypted data by an untrusted powerful server such as a cloud service provider. Furthermore, even in the unencrypted domain, the proposed re-parametrization shows significant performance results and can be useful if data cannot be outsourced and therefore remain in plaintext. Thus, the novelty and the performance gains of the proposed solution holds both on the encrypted and on the plaintext domain.

As future work, we plan to implement reformulation and reparametrization tricks using well known Deep Learning frameworks such as Tensorflow or PyTorch, automatizing its computation in order to apply it more efficiently.

7 Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable feedback and comments. This work was partly supported by the PAPAYA project funded by the European Union's Horizon 2020 Research and Innovation Programme, under Grant Agreement no. 786767.

References

1. Barni, M., Orlandi, C., Piva, A.: A privacy-preserving protocol for neural-network-based computation. In: 8th ACM Workshop on Multimedia and Security (2006)
2. Bishop, C.M.: Neural networks for pattern recognition. Oxford university press (1995)
3. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory **6**(3), 13:1–13:36 (Jul 2014). https://doi.org/10.1145/2633600, `http://doi.acm.org/10.1145/2633600`
4. Chabanne, H., de Wargny, A., Milgram, J., Morel, C., Prouff, E.: Privacy-Preserving Classification on Deep Neural Network. In: ePrint Archive (2017)
5. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing. STOC '09 (2009)
6. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: International Conference on Machine Learning (2016)
7. Halevi, S., Shoup, V.: Algorithms in helib. In: International cryptology conference. pp. 554–571. Springer (2014)
8. HasanPour, S.H., Rouhani, M., Fayyaz, M., Sabokrou, M.: Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. CoRR **abs/1608.06037** (2016), `http://arxiv.org/abs/1608.06037`
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015), `http://arxiv.org/abs/1512.03385`
10. Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K.Q.: Deep networks with stochastic depth. In: European Conference on Computer Vision. pp. 646–661. Springer (2016)
11. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456 (2015)
12. LeCun, Y., Cortes, C., Burges, C.J.: The MNIST database of handwritten digits. Tech. rep. (1998), `ttp://yann.lecun.com/exdb/mnist/.`
13. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via minionn transformations. In: ACM CCS 2017. pp. 619–631. ACM (2017)
14. Mohassel, P., Zhang, Y.: SecureML: A System for Scalable Privacy-Preserving Machine Learning. In: IEEE Symposium on Security and Privacy (SP) (2017)

15. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: IN ADVANCES IN CRYPTOLOGY — EUROCRYPT 1999. pp. 223–238. Springer-Verlag (1999)
16. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on knowledge and data engineering **22**(10), 1345–1359 (2010)
17. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (Feb 1978). https://doi.org/10.1145/359340.359342, `http://doi.acm.org/10.1145/359340.359342`

Appendix I: DNN architectures used for section 5

- **Input**: 28x28 greyscale images.
- **Output**: [0-9] Single digit with the class the image belongs to.
- **Layers in order**:

Table 3: DNN architectures used for performance study

DNN architecture	15 M	200k
Conv1	20 filters 5x5, stride 1	5 filters 5x5, stride 1
BN	20 $(\beta, \gamma, \mu, \sigma^2)$	5 $(\beta, \gamma, \mu, \sigma^2)$
ReLU	No parameters	No parameters
Mean Pool	-	stride 2x2
FC1	15684*1000 neurons	980*100 neurons
BN1	1000 $(\beta, \gamma, \mu, \sigma^2)$	100 $(\beta, \gamma, \mu, \sigma^2)$
ReLU	No parameters	No parameters
FC2	1000*10 neurons	100*10 neurons