Practical fully homomorphic encryption for fully masked neural networks

Malika Izabachène, Renaud Sirdey, Martin Zuber*

CEA, LIST,
91191 Gif-sur-Yvette Cedex, France
email: name.surname@cea.fr

Abstract. Machine learning applications are spreading in many fields and more often than not manipulate private data in order to derive classifications impacting the lives of many individuals. In this context, it becomes important to work on privacy preserving mechanisms associated to different privacy scenarios: protecting the training data, the classification data, the weights of a neural network.. In this paper, we study the possibility of using FHE techniques to address the above issues. In particular, we are able to evaluate a neural network where both its topology and its weights as well as the user data it operates on remain sealed in the encrypted domain. We do so by relying on Hopfield neural networks which are much more "FHE friendly" than their feed-forward counterparts. In doing so, we thus also argue the case of considering different (yet existing) Neural Network models better adapted to FHE, in order to more efficiently address real-world applications. The paper is concluded by experimental results on a face recognition application demonstrating the ability of the approach to provide reasonable recognition timings ($\approx 0.6s$) on a single standard processor core.

Fully Homomorphic Encryption, LWE, GSW, Hopfield Neural Networks, Face Recognition, FHE Performances

1 Introduction

A Fully Homomorphic Encryption (FHE) scheme is, ideally, an encryption scheme permeable to any kind of operation on its encrypted data. With any input x and function f, with E the encryption function, we can obtain $E(f(x))$ non interactively from $E(x)$. This property is particularly valuable when privacy-preserving computations on remote servers are required. Unfortunately, all FHE known today still imply high computation overheads and tackling real applications euphemistically remain a challenge. Nevertheless, since Gentry's theoretical breakthrough around 2010, there has been steady progress towards more and more efficient (or less and less inefficient) FHE cryptosystems. In recent years, machine learning applications have been wider spread than ever in various domains. In this context, a growing concern is put on data privacy and confidentiality for both sensitive personal information and valuable intellectual property (e.g. models). In this context, Fully Homomorphic Encryption has the potential to provide an interesting counter-measure but its state of the art does not

* Corresponding author.

(yet?) allow to directly apply existing FHE schemes to the highly complex real-world machine learning-based systems. While looking for faster and more efficient FHE schemes is of paramount importance, our paper fall into a line of work which addresses the needs of finding new FHE-friendly Machine Learning methods possibly by revisiting old or less conventional tools from that field and to tune existing FHE specifically to running these methods as efficiently as possible in the encrypted domain.

Prior Work. Most of the existing works on the subject have done so with artificial feed-forward neural networks. These kind of neural networks are the most popular in the Machine Learning field and successfully applying FHE on them in an efficient manner remains a challenging task. There have been different ways this has been tried and not always through the use of HE. For instance, in [2], [17] and [13] the authors restrict themselves to a set-up in which the training data is shared between two parties and never fully revealed to any one of them, yet allowing the network to be trained with the whole data-set. Feed-forward networks require to have non-linear activation functions, such as a sigmoid, which are challenging to implement efficiently (and exactly) over FHE. Previous works have handled this by either using the sign function instead (as with [3]) or a polynomial function (as with [16]). Another issue with neural networks that previous work has grappled with is how to deal with its (multiplicative) depth. With that respect, [16] provides a leveled scheme that has its parameters grow with the depth of the network. It was improved most recently in [4]. Alternatively, using an efficient bootstrapping algorithm from [5] which was then refined in [6], [3] proposes a scheme where the size of the parameters does not depend on the depth of the network hence demonstrating the practical relevance of this new breed of FHE schemes in the context of Machine Learning applications. To the best of our knowledge, what all the previous works on the subject have in common is the fact the network itself is always in the clear domain and that only the data that are aimed to be classified reside in the encrypted domain. What this paper aims for is to encrypt both the weights of a neural network as well as the data to be classified. Yet, doing this for feed forward networks appears very difficult as building on previous works - for instance the most recent work by [3] - to fit these constraints would lead to unreasonable timings. What we choose to do in the face of this difficulty is to solve the same problem by switching to a different kind of neural network: Hopfield networks.

Our Contribution. In this paper, we present a method to classify encrypted objects by means of a fully encrypted neural network with practically relevant timings and accuracy. We specifically present different approaches to homomorphic face recognition using neural networks. It is worth emphasizing that, because of the nature of the network used (it is discrete and uses a sign function as an activation function), no approximations are made in any of the homomorphic computations. Our main contributions are twofold. First, we provide results in the case where encrypted data are classified by a clear network. This method is both quite fast and lightweight as it takes less than ≈ 0.2s seconds to perform a face recognition operation with e.g. 110-bit security. Then, we provide the specifications for a second scheme in the case where encrypted data are classified by an encrypted network. We introduce a method that uses FHE to allow for an arbitrary number of rounds with parameters that are depth-invariant. It classifies in less than 0.6s with 80-bit security. This paper is

organized as follows. We start by reviewing the challenges that combining neural networks and fully homomorphic encryption presents. Some of those challenges have already been tackled in part in the recent literature and some have yet to be. For self-containedness, we then go into the basics of discrete Hopfield Networks at least to the extent that we use them in this paper. This is then followed by a brief presentation of the LWE encryption scheme as well as of TFHE, the LWE-based FHE scheme and library that we use. Then, we present the building blocks that make the bulk of our contribution: several encoding, encryption and computational methods designed for different types of computations involved in various flavors of Hopfield network homomorphic evaluation. Experimental results are finally provided with an emphasis on the very reasonable timings that we obtain for classification on a masked neural network and other variants, on a face-recognition application.

2 Homomorphic Neural Networks

2.1 Models for evaluation of encrypted Neural Networks

There are not many works yet on the subject of FHE applied to neural networks. To the best of our knowledge, all of the previous works on the question have focused on the classification of an encrypted input sent by a client but through the use of a public or clear-domain neural network. However, this is only one of the ways one could wish to apply FHE to neural networks. One could want a number of things to be masked:

Encrypted training data Good-quality (labeled) training data is costly to obtain and some companies make a business out of providing such data. Therefore, one might very understandably want to protect the data one spent time or money acquiring when sending it to a remote server to be used for training.

Encrypted weights Training an efficient NN is generally harder than it seems and requires both computation time and know-how. Hence, the owner of a NN might want to avoid for its weights to be leaked or for some entity (e.g. the server homomorphically evaluating the network) to be able to use it without restriction.

Hidden topology of the NN Training a NN is as much about the training data and the training algorithm as it is about the topology of the NN (number of hidden layers and neurons per layer for instance) on which will depend a lot of the properties of the NN (too large and it has a tendency to generalize badly; too small and it might not be precise enough in its classification). So, in some cases, there may be an interest in masking the network topology and not only its weights, again w. r. t. the server homomorphically evaluating the network.

Encrypted classification data This is arguably the first sought-after property as it pertains to user privacy and allows the network to run on encrypted data therefore providing a service without access to the actual user information.
On top of finding new, efficient ways to classify over encrypted data, this paper also aims to hide the topology and the weights of the network as well. The way we achieve this is by switching from a feed-forward network - one that was used by all previous works - to a different kind of NN: a Hopfield network (HN).

2.2 Hopfield Networks Basics

Hopfield networks are recursive neural networks with only one layer of neurons, all connected. They were first introduced by J. J. Hopfield in 1982 in [10, 14]. See also [12] and [8] for further uses in the literature. Let us thus consider a network made up of M neurons with each having a value a_i and weights $w_{i,j}$ between neurons i and j.
During the evaluation of a network, the single layer updates the values it stores at every iteration according to a correlation property. The weight $w_{i,j}$ between two neurons i and j represents the correlation that exists between them. An update of neuron i translates to a weighted sum of all other neuron values with an activation function θ: $a_i = \theta\left(\sum_{j=1}^{n} a_j w_{i,j}\right)$ Therefore every neuron is "drawn" towards the values of the neurons most correlated to it. This correlation is symmetric ($w_{i,j} = w_{j,i}$) and a neuron is not correlated with itself ($w_{i,i} = 0$). In practice what the Hopfield network does naturally is storing a number of patterns by highly correlating the bits of the patterns. The network does not need to globally converge to be useful. If we define certain bits as "classification bits" (for example, in the face recognition experiment of Sect. 4.1 only 3 bits out of 256 are used to classify between up to 8 patterns), then only those bits need to be updated to obtain a first result. With a sign function as an activation function these networks are computationally lighter than their feed-forward counterparts.
The training of the network (determining the weights) requires us to have (as usual for NNs) three sets of data: the training set, the validation set and the testing set. We trained the network used in this work ourselves.

3 LWE encryption scheme

In this work, we use the TFHE encryption scheme by Chillotti et al. [5, 6] for an homomorphic evaluation of Hopfield network with both encrypted weights and encrypted activation values. We refer to those papers for an in-depth presentation of TFHE and only refer here to what is necessary for the understandng of our work.

Notation. We use $\mathbb{B}$ to denote the set $\{0,1\}$. We denote the reals $\mathbb{R}$ and use $\mathbb{T}$ to denote the real torus mod 1. The ring $\mathbb{Z}[X]/(X^N+1)$ is denoted $\mathfrak{R}$ and $\mathbb{B}_N[X]$ is its subset composed of the polynomials with binary coefficients. We write $\mathbb{T}_N[X]$ the quotient $\mathbb{R}[X]/(X^N+1)$ mod1 where N is a fixed integer. Vectors are denoted with an arrow as such: $\vec{*}$. Ciphertexts are denoted with boldface letters. In the following, λ is the security parameter, M is the size of the network as presented in section 2.2 and B is the maximum value taken by the weights of the network.

3.1 TFHE basics

`TRLWE` is the ring version of LWE introduced in [11] over the torus. It encrypts messages in $\mathbb{T}_N[X]$. `TLWE` is its "scalar" version and encrypts messages in $\mathbb{T}$. `TRGSW` is the ring version of the GSW scheme introduced in [7]. A `TRGSW` ciphertext encrypts messages in $\mathfrak{R}$. We note here that in the rest of the paper, we use bold lower-case letters for `TLWE` ciphertexts and bold upper-case letters for `TRLWE` and `TRGSW` ciphertexts. A `TLWE` encryption $\boldsymbol{c}$ of μ with standard deviation σ and with key $\boldsymbol{s}$ will be noted as $\boldsymbol{c} \in \texttt{TLWE}_{\boldsymbol{s},\sigma}(\mu)$ or $\boldsymbol{c} \in \texttt{TLWE}(\mu, \boldsymbol{s}, \sigma)$. The same notations are valid for `TRLWE` and `TRGSW` encryptions.

Parameters. We have the following parameters involved for TLWE, TRLWE and TRGSW encryption schemes respectively.

TLWE parameters: Given a minimal noise overhead α and a security parameter λ, we derive a minimal key size n.
TRLWE parameters: we call n the key size and we have $n=k\times N$. The fact we use n for both TLWE and TRLWE parameters is not a problem. We see in the next paragraph that through the TRLWE to TLWE extraction process one obtains a TLWE ciphertext with $n=k\times N$. In the rest of the paper we will have $k=1$.
TRGSW parameters: We have decomposition parameters ℓ, B_g. To define homomorphic multiplication, we decompose a TRGSW ciphertext as a small linear combination of rows of a gadget matrix defined with respect to a basis B_g as a ℓ repeated super-decreasing sequence $(1/B_g,...,1/B_g^\ell)$. Since we use an approximated decomposition, we have an additional precision parameter ε. We will take $\beta=B_g/2$ and $\varepsilon=\frac{1}{2B_g^l}$.

Known operations over TFHE ciphertexts. We present linear algebra computations for homomorphic neural network evaluation. These are all operations presented in the TFHE papers [5, 6].

Linear combination over TLWE : $\texttt{TLWE}^p\rightarrow\texttt{TLWE}$
For p TLWE ciphertexts $\boldsymbol{c_i}$ of $\mu_i\in\mathbb{T}$, and for p integers δ_i, we can obtain an encryption of $\sum_{i=1}^p\delta_i\mu_i$ through the operation $\sum_{i=1}^p\delta_i\cdot\boldsymbol{c_i}$. We write the operation $\mathsf{TLWEScalarProduct}(\boldsymbol{c},\delta)$. For $p=2$ and $\delta_1=\delta_2=1$, we write $c_1+c_2=\mathsf{AddTLWE}(c_1,c_2)$. The TRLWE equivalent is $\mathsf{AddTRLWE}$.

Extraction : $\texttt{TRLWE}\rightarrow\texttt{TLWE}$
From a TRLWE ciphertext $\boldsymbol{C}$ of $\mu=\sum_{i=0}^{N-1}\mu_i\in\mathbb{T}_N[X]$, it is possible to extract a TLWE ciphertext $\boldsymbol{c}$ of a single coefficient of μ_p at a position $p\in[0,N-1]$. We write this operation $\boldsymbol{c}=\mathsf{SampleExtract}(\boldsymbol{C},p)$.

External Product : $\texttt{TRLWE}\times\texttt{TRGSW}\rightarrow\texttt{TRLWE}$
The external product between a TRGSW ciphertext $\boldsymbol{C_1}$ of $\delta\in\mathbb{Z}_N[X]$ and a TRLWE ciphertext $\boldsymbol{C_2}$ of $\mu\in\mathbb{T}_N[X]$ produces a TRLWE ciphertext $\boldsymbol{C}$ of the product $\delta\cdot\mu\in\mathbb{T}_N[X]$. We write $\boldsymbol{C}=\mathsf{ExternalProduct}(\boldsymbol{C_2},\boldsymbol{C_1})$.

Public Rotation : $\texttt{TRLWE}\rightarrow\texttt{TRLWE}$
Given a TRLWE ciphertext $\boldsymbol{C}$ of μ and an integer p we can obtain a TRLWE ciphertext $\boldsymbol{C'}$ of $\mu\times X^p$ at no cost to the variance of the ciphertext. We write $\boldsymbol{C'}=\mathsf{RotateTRLWE}(\boldsymbol{C},p)$.

Key-switch : $\texttt{TLWE}\rightarrow\texttt{TLWE}$
We give a TLWE ciphertext $\boldsymbol{c}\in\texttt{TLWE}(\mu,\boldsymbol{s})$. For a given TLWE key $\boldsymbol{s}$ and two integers base and t. Given $\mathsf{KS}_{i,j}\in\texttt{TLWE}(s_i/\mathsf{base}^j,\boldsymbol{s'},\gamma)$ for $j\in[1,t]$ and $i\in[1,n]$ and with γ a given standard deviation. The key-switching procedure outputs $\boldsymbol{c}\in\texttt{TLWE}_{\boldsymbol{s'}}(\mu)$. We write $\mathsf{KS}_{\boldsymbol{s}\rightarrow\boldsymbol{s'}}=(\mathsf{KS}_{i,j})_{(i,j)\in[1,n]\times[1,t]}$ and we call it the key-switching key.

Public Functional key-switch : $\texttt{TLWE}^p\rightarrow\texttt{TRLWE}$
We give p TLWE ciphertext $\boldsymbol{c}_i\in\texttt{TLWE}(\mu_i,\boldsymbol{s})$, and a public R-lipschitzian morphism $f:\mathbb{T}^p\mapsto\mathbb{T}_N[X]$ of $\mathbb{Z}$-modules. For a given TRLWE key $\boldsymbol{s'}$ and two integers base and t, we

take $\mathsf{KS}_{i,j} \in \mathtt{TRLWE}(s_i/\mathsf{base}^j, \boldsymbol{s'}, \gamma)$ for $j \in [1,t]$ and $i \in [1,n]$ and with γ a given standard deviation. The functional keyswitching outputs $\boldsymbol{C} \in \mathtt{TRLWE}_{\boldsymbol{\mathfrak{K}'}}(f(\mu_1,...,\mu_p))$. We write $\mathsf{KS}_{\boldsymbol{s}\to\boldsymbol{s'}} = (\mathsf{KS}_{i,j})_{(i,j)\in[1,n]\times[1,t]}$ and we call it the key-switching key. We exclusively use a very specific function in this paper. We will call it the identity function and refer to the key-switch operation as $\mathsf{Keyswitch}_{\mathsf{Id}}$ It is defined for any $t \in \mathbb{T}$: $t \mapsto t \cdot X^0$.

Bootstrapping : $\mathtt{TLWE} \to \mathtt{TLWE}$
Given an integer S, a $\mathtt{TLWE}$ ciphertext $\boldsymbol{c} \in \mathtt{TLWE}(\mu, \boldsymbol{s})$, n $\mathtt{TRGSW}$ ciphertexts $\mathsf{BK}_i \in \mathtt{TRGSW}(s_i, \boldsymbol{s'}, \alpha_b)$ where the s_i are coefficients of the equivalent $\mathtt{TLWE}$ key, we can obtain a ciphertext $\boldsymbol{c_o} \in \mathtt{TLWE}(\mu_o, \boldsymbol{s'}, \alpha_{\mathrm{boot}})$ where $\mu_o = 1/S$ if $\mu \in [0, \frac{1}{2}]$ and $\mu_o = -1/S$ if $\mu \in [\frac{1}{2}, 1]$. Most importantly, the output standard deviation α_{boot} is fixed by the parameters of the bootstrapping key BK.

Homomorphic operations and variance overhead. Table 1 summarizes the elementary operations we use in this paper, and the noise propagation they induce. We set several constants which depend on the parameters of the underlying encryption schemes. We set δ to be a plaintext integer encrypted as a $\mathtt{TRGSW}$ ciphertext in the external product operation.

Operation	Variance
$\mathsf{TLWEScalarProduct}(\boldsymbol{c}, \delta)$	$\|\|\delta\|\|_2^2 \cdot \vartheta_{\boldsymbol{c}}$
$\mathsf{AddTLWE}(\boldsymbol{c_1}, \boldsymbol{c_2})$	$\vartheta_1 + \vartheta_2$
$\mathsf{AddTRLWE}(\boldsymbol{C_1}, \boldsymbol{C_2})$	$\vartheta_1 + \vartheta_2$
$\mathsf{SampleExtract}(\boldsymbol{C}, j)$	$\vartheta_{\boldsymbol{C}}$
$\mathsf{Keyswitch}(\boldsymbol{c})$	$\vartheta_{\boldsymbol{c}} + nt\vartheta_{\mathsf{KS}} + n\mathsf{base}^{-2(t+1)}$
$\mathsf{Keyswitch}_{\mathrm{Id}}(\boldsymbol{c})$, $p=1$	$\vartheta_{\boldsymbol{c}} + nt\vartheta_{\mathsf{KS}} + n\mathsf{base}^{-2(t+1)}$
$\mathsf{ExternalProduct}(\boldsymbol{C_1}, \boldsymbol{C_2})$	$\boldsymbol{B}\vartheta_2 + \boldsymbol{C} + \boldsymbol{D}\vartheta_1$
$\mathsf{Bootstrapping}$	$\vartheta_{\mathrm{boot}}$

Table 1. Elementary operations and their variance overhead The ϑ_cs, the $\vartheta_{\boldsymbol{c}}$s and the $\vartheta_{\boldsymbol{C}}$s are the noise variances of their respective ciphertexts. ϑ_{KS} is the variance for the encryption of the key-switching key KS and ϑ_{BK} for the bootstrapping key BK. In the case of the external product, we write δ the message encrypted by the input $\mathtt{TRGSW}$ ciphertext $\boldsymbol{C_2}$. Furthermore, $\boldsymbol{B} = 2\ell N\beta^2$, $\boldsymbol{C} = \epsilon^2(1+N) \cdot ||\delta||_2^2$, $\boldsymbol{D} = ||\delta||_2^2$. As for the bootstrapping, the output variance overhead is given by: $\vartheta_{\mathrm{boot}} = 4Nn\ell\beta^2 \times \vartheta_{\mathsf{BK}} + n(1+N)\epsilon^2$.

Data encoding. There are two types of data we want to encrypt : the activation values a_i which are equal to 1 or -1 and the weight values $w_{i,j} \in [-B, B]$ where $B \in \mathbb{Z}$. For $\mathtt{TRLWE}$ and $\mathtt{TLWE}$ encryption, we encode integers into torus values as follows: we divide the torus into S slices and encode any $x \in \mathbb{Z}/S\mathbb{Z}$ as its corresponding slice in the torus representation. In other words, we have $\frac{x}{S} \mod 1 \in \mathbb{T}$. In this paper, we only encode activation values in the torus and weights stay integers. The bigger the slice S is, the more error propagation can be allowed before the error makes the decryption overflow into the adjacent slice. Choosing to reduce the number of slices in order to relax the

constraints on the parameters while still ensuring a correct output is a choice made also in [3]. Heuristically, we found that $S \geq M$ is appropriate for our case. And in effect we will choose $S = M$. Finally, we will call ϑ_{max} be the maximum ciphertext variance that still allows the message to be deciphered correctly. It depends only on S.

3.2 Encrypted inputs through a clear network

In this paper, we present several methods for classifying encrypted inputs via a Hopfield Network. In this section, we present the case where the network is not encrypted, but the input activation values are. We encrypt the activation values a_i as TLWE ciphertexts $\boldsymbol{c} \in \mathtt{TLWE}(\frac{1+2a_i}{2S})$. Now, in order to update the activation value of one neuron, we need to compute a scalar product and take the sign of the result and then insert it back into the network for further computations. For a given neuron p, this corresponds to the following computation : $a_p = \mathsf{sign}\left(\sum_{i=1}^{M} a_i w_{p,i}\right)$ The TLWE ciphertexts $\boldsymbol{c_i} \in \mathtt{TLWE}(a_i)$ are grouped in a vector $\vec{\boldsymbol{c}}$. and the weights $w_{i,p}$ are in clear. Assuming we want to update the p^{th} neuron, we have an algorithm depicted in figure 1 below. Then our scheme consists of applying this algorithm on a given number of activation values to update them. Once the number has been reached the resulting encrypted activation values can be sent back to the owner of their key.

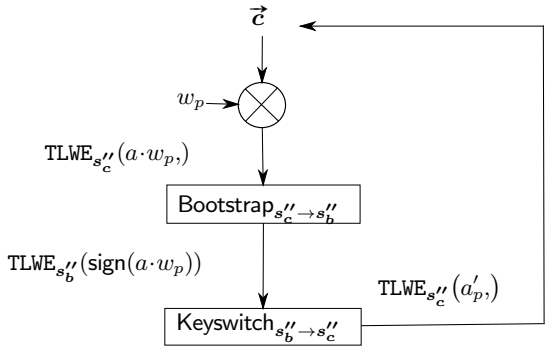

Fig. 1. This figure illustrates the "clear-case" update algorithm. The output of the Keyswitch operation is reincorporated into the vector of ciphertexts as its p^{th} component.

If we choose the parameters of the scheme and ϑ_c such that

$$MB^2\left(\vartheta_{\mathsf{boot}} + nt\vartheta_{\mathsf{KS}} + n\mathsf{base}^{-2(t+1)}\right) \leq \vartheta_{\max} \quad \text{and} \quad \vartheta_c = \frac{\vartheta_{\max}}{MB^2}$$

then our computation is fully homomorphic.

Scheme. We now present a scheme based on this algorithm. For this we define three actors:

– "the network owner" (or "owner" for short). She trained a Hopfield network on a given type of formatted data and owns the resulting network.

– "the cloud". It has the computational ability to perform tasks with the owner's network at a low cost, making it interesting for other actors to use its services.

– "the client". She owns formatted data compatible with the owner's network and which she would like to run through the Hopfield network.

We present here the case where **the owner** does not encrypt the network. She therefore shares it with the cloud which has access to the weights in clear. She also shares instructions for the updates to perform for a classification. Furthermore, she has to determine the parameters to use for encryption both for activation values and the weights. She shares these parameters with the client.
The client does want to protect her data. She chooses a TLWE key, a TRLWE key and creates a bootstrapping key BK and a key-switching key KS. Finally she encrypts her data and the ciphertexts are sent, along with BK and KS, to the cloud.
The cloud then performs the necessary updates, outputs an updated ciphertext and sends it back to the client.
The client decrypts and has the result to the classification problem.

3.3 Encrypted inputs through an encrypted network

We now consider the case where both the weights and the activation values are encrypted. We have $\vec{a}=(a_0,...,a_{M-1})$ and $\overrightarrow{w_p}=(w_{0,p},...,w_{M-1,p})$ for every p.

Encryption Methodology. We use the TRLWE and TRGSW encryption schemes to encrypt the activation values and the weights respectively. We are going to use the same key for both: $\boldsymbol{s_c}$.

We constrain $N \geq M$. We first encode the data as two $(N-1)$-degree polynomials (with 0 coefficients to pad if need be):

$$\forall p \in [0,M-1], \quad W_p = \sum_{i=0}^{N-1} w_{i,p} \cdot X^{N-1-i}, \quad A = \sum_{i=0}^{N-1} a_i \cdot X^i \in \mathbb{T}_N[X]$$

We then encrypt both the weight polynomials and the activation polynomial respectively as TRGSW and TRLWE ciphertexts:

$$\boldsymbol{C_w^{(p)}} \in \texttt{TRGSW}(W_p, \boldsymbol{s_c}, \alpha_c) \quad \text{and} \quad \boldsymbol{C_a} \in \texttt{TRLWE}(A, \boldsymbol{s_c}, \alpha_c)$$

These are both encrypted using the same standard deviation α_c. We also have M TRLWE ciphertext for each individual activation value:

$$\forall p \in [0,M-1], \quad \boldsymbol{C_p} \in \texttt{TRLWE}(a_p X^p, \boldsymbol{s_c}, \alpha'_c)$$

We group them in a vector of ciphertexts $\vec{\boldsymbol{C}} = (\boldsymbol{C_0},...,\boldsymbol{C_{M-1}})$.

Update Algorithm. Figure 2 presents an update of the p^{th} activation value for a given p. Note that the Keyswitch operation hides a rotation operation that transforms $\texttt{TRLWE}(a'_p, \boldsymbol{s_c})$ into $\texttt{TRLWE}(a'_p X^p, \boldsymbol{s_c})$ at no cost to the error propagation and virtually no time cost. Keeping TRLWE ciphertexts of individual activation values (the $\boldsymbol{C_i}$ ciphertexts) allows us to rebuild a new and updated $\boldsymbol{C_a}$ ciphertext by summing them all up at the end.

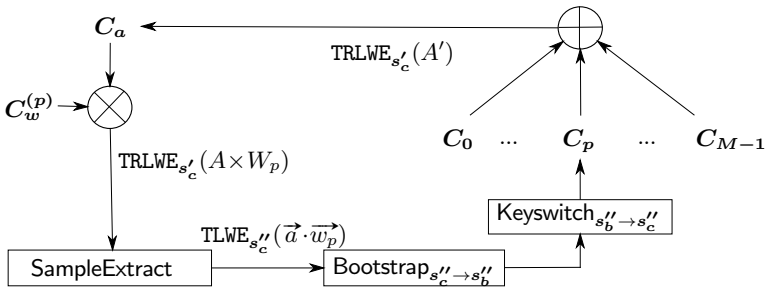

Fig. 2. This figure illustrates the update of the p^{th} neuron in the "masked case".

Correctness. There are two sets of parameters defined both for the initial ciphertexts and for the bootstrapping key BK. We will therefore use the notation $*_c$ for the initial ciphertext parameters and $*_b$ for the bootstrapping key parameters. As for the key-switching key KS, we will use the notation $*_s$. We have $N_s = N_c$. This can work because all of the input ciphertexts have the same parameters with only one exception: the variance of the $\boldsymbol{C_p}$ ciphertext which is not equal to ϑ_c; we will refer to it by ϑ'_c. For this next theorem, we set some constants in order to simplify notations. We set $\mathbf{A} = 4N_c N_b l_b \beta_b^2$; $\mathbf{B} = N_c(1+N_b)\epsilon_b^2$; $\mathbf{C} = N_c t$; $\mathbf{D} = N_c \mathsf{base}^{-2(t+1)}$; $\mathbf{E} = MB^2$; $\mathbf{F} = \epsilon_c^2(1+N_c)$ and $\mathbf{G} = 2\ell_c N_c \beta_i^2$.
Then, if we choose the parameters of the scheme and ϑ_c, ϑ'_c such that

$$M(\mathbf{G}+\mathbf{E}) \times (\mathbf{A}\vartheta_{\mathsf{BK}} + \mathbf{B} + \mathbf{C}\vartheta_{\mathsf{KS}} + \mathbf{D}) + \mathbf{F} \times \mathbf{E} \leq \vartheta_{\max} \tag{1}$$

$$\vartheta_c \leq \frac{\vartheta_{\max} - \mathbf{F} \times \mathbf{E}}{\mathbf{G}+\mathbf{E}} \quad \text{and} \quad \vartheta'_c = \mathbf{A}\vartheta_{\mathsf{BK}} + \mathbf{B} + \mathbf{C}\vartheta_{\mathsf{KS}} + \mathbf{D} \tag{2}$$

then our computation is fully homomorphic.

Scheme. This scheme is identical to the one presented in section 3.2 with the only difference that the owner and the client share the same key $\boldsymbol{s_c}$. This means collusion between the client and the cloud gives them access to the owner's network. The same goes for collusion between the owner and the cloud. There are ways around this but none is perfect. One way is to introduce a Trusted Third Party (TTP).

4 Experimental results

4.1 Face recognition via Hopfield networks

Although their applicability is less universal than CNN, Hopfield networks are known to perform well for certain tasks including image analysis and recognition. In this section, we provide experimental results in the case of a face recognition function. We used a free database available for research purposes. It consists of pictures of

153 individuals - male and female alike - with 20 pictures per individual with different facial expressions on each person. The database can be downloaded from here: https://cswww.essex.ac.uk/mv/allfaces/faces94.html. The feature extraction for the images was done using the LTP (Local Ternary Patterns) introduced in [15] and from the Java library Openimaj (see [9]). From a face image, the feature extraction gives us 128 80×80 matrices of raw binary data that we use for the classification. For this test we selected 8 random faces. By training the network on those patterns we wish to be able to classify a picture of a person as one of those 8 faces. For this we apply one iteration of classification on 3 neurons of a 256-neuron network. Empirically, the network stabilizes after the first iteration, hence we do not find better classification results with more iterations. We find that reducing the number of neurons to 256 maximizes the time/performance ratio, giving us an 86.2% classification performance (however we are confident that better results could be obtained with deeper machine learning know-how).

4.2 Parameter choices

Choosing the parameters, we have to take into account the security constraints from the underlying LWE problem on top of the correctness constraints for the two schemes we present. The overall security of the scheme relies entirely on the security of the underlying TFHE scheme and therefore the LWE problem on which it is built. For a given size of the key and a given security parameter, we can obtain a minimum variance using the `lwe-estimator`[1] script. The estimator is based on the work presented in [1]. The structural parameters of the network are $M=256$ and $B=8$. We have $S=M=256$.

Parameters for the clear Hopfield network. For both an 80-bit security and a 110-bit security, the parameters in table 2 are appropriate. However we need to choose different values for n. We set $n=470$ for an 80-bit security; And $n=650$ for a 110-bit security.

N	α_b	α_c	Bg_b	ℓ_b	t	base
1024	4.26e-13	2.41e-06	64	4	3	32

Table 2. Parameter values for both security settings in the case of a clear network. We have $\alpha_s=\alpha_c$ and k is set to 1.

Parameters for the masked Hopfield network. Table 3 presents the parameters used for the initial ciphertext encryption, the bootstrapping key and the key-switching key. These parameters are valid for an 80-bit security. We were not able to go beyond this security level: the parameter constraints did not hold anymore.

[1] https://bitbucket.org/malb/lwe-estimator/raw/HEAD/estimator.py

λ	α_c	α'_c	N_c	Bg_c	ℓ_c
80	4.26e-13	4.26e-13	1024	64	4

λ	α_b	N_b	Bg_b	ℓ_b
80	8.88e-16	2048	256	5

λ	α_s	N_s	t	base
80	4.26e-13	1024	4	128

Table 3. Parameter values for initial ciphertext encryption (top left), for the bootstrapping key BK (top right) and for the key-switching key KS (bottom) for an 80-bit security

We provide the sizes of the ciphertexts associated with these parameters. The size of a single initial TRLWE ciphertext is 16.5 KBytes and the size of the initial TRGSW ciphertext 132 KBytes. The size of the Bootstrapping key is 337 MBytes and the size of the Key-switching key 135 MBytes. The total size of the encrypted network is 33.7 MBytes and the total size of the encrypted activation values 4.22 MBytes.

4.3 Network Performance

In this subsection, we present the timing results for our classification. All timings are measured on an Intel Core i7-6600U CPU. The performances are given in table 4. Overall we can see that a 1-iteration classification does not take more than 0.21 seconds in the clear case and under 0.6 seconds for a fully masked, fully homomorphic, 1-iteration classification.

TLWEScalarProduct	Bootstrap	Keyswitch
7.5×10^{-7}	0.06	0.009
Total time for 1 update		≤ 0.07

ExternalProduct	Bootstrap	Keyswitch	AddTRLWE
3×10^{-4}	1.6×10^{-1}	1.4×10^{-2}	5×10^{-4}
Total time for 1 update		≤ 0.2	

Table 4. Algorithm timings (in seconds) for the clear-case classification implementation (top) and for the masked case (bottom). As a reminder, a classification requires 3 updates.

5 Conclusion

In this paper, we investigated a method to classify encrypted objects by means of a fully encrypted neural network with practically relevant timings and accuracy on a face recognition application, thus achieving both network and data privacy for the first time to the best of our knowledge. We have done so by considering Hopfield networks, a kind of neural network well-known in the Machine Learning community yet more amenable to practical FHE performances than their feed-forward counterparts. As such, this paper also provides insights as to how to intricate an FHE scheme with an algorithm in order to achieve decent FHE execution performances. As a matter of perspective, our next step is to achieve the training of a Hopfield network on an encrypted data set.

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Cryptology ePrint Archive, Report 2015/046 (2015)
2. Bansal, A., Chen, T., Zhong, S.: Privacy preserving back-propagation neural network learning over arbitrarily partitioned data. Neural Computing and Applications **20** (2011)
3. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Proceedings of CRYPTO 2018. Springer (2018)
4. Chabanne, H., de Wargny, A., Milgram, J., Morel, C., Prouff, E.: Privacy-preserving classification on deep neural network. IACR Cryptology ePrint Archive **2017** (2017)
5. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Advances in Cryptology - ASIACRYPT 2016 Proceedings, Part I. Springer (2016)
6. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Improving tfhe: faster packed homomorphic operations and efficient circuit bootstrapping. In: Proc. of Asicacrypt. Springer-Verlag (2017)
7. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Crypto (2013)
8. Gurney, K.: An introduction to neural networks (1997)
9. Hare, J.S., Samangooei, S., Dupplaw, D.P.: Openimaj and imageterrier: Java libraries and tools for scalable multimedia analysis and indexing of images. In: Proceedings of the 19th ACM international conference on Multimedia. ACM (2011)
10. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences **79** (1982)
11. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: EUROCRYPT. Springer (2010)
12. MacKay, D.: Information theory, inference, and learning algorithms (2003)
13. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton) (2015)
14. Sulehria, H.K., Zhang, Y.: Hopfield neural networks: A survey. In: Proceedings of the 6th WSEAS Conference on Artificial Intelligence, Knowledge Engineering and Data Bases. WSEAS (2007)
15. Tan, Xiaoyang, Triggs, Bill: Enhanced local texture feature sets for face recognition under difficult lighting conditions. Trans. Img. Proc. **19** (2010)
16. Xie, P., Bilenko, M., Finley, T., Gilad-Bachrach, R., Lauter, K.E., Naehrig, M.: Crypto-nets: Neural networks over encrypted data. CoRR (2014)
17. Yuan, J., Yu, S.: Privacy preserving back-propagation neural network learning made practical with cloud computing. IEEE Transactions on Parallel and Distributed Systems **25** (Jan 2014)