Simulating Homomorphic Evaluation of Deep Learning Predictions

Christina Boura[1,4], Nicolas Gama[1,2], Mariya Georgieva[2,3(✉)], and Dimitar Jetchev[2,3]

[1] Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université Paris-Saclay, Versailles, France
christina.boura@uvsq.fr
[2] Inpher, Lausanne, Switzerland
{nicolas,mariya,dimitar}@inpher.io
[3] EPFL, Lausanne, Switzerland
[4] Inria, Paris, France

Abstract. Convolutional neural networks (CNNs) is a category of deep neural networks that are primarily used for classifying image data. Yet, their continuous gain in popularity poses important privacy concerns for the potentially sensitive data that they process. A solution to this problem is to combine CNNs with Fully Homomorphic Encryption (FHE) techniques. In this work, we study this approach by focusing on two popular FHE schemes, TFHE and HEAAN, that can work in the approximated computational model. We start by providing an analysis of the noise after each principal homomorphic operation, i.e. multiplication, linear combination, rotation and bootstrapping. Then, we provide a theoretical study on how the most important non-linear operations of a CNN (i.e. max, `Abs`, `ReLU`), can be evaluated in each scheme. Finally, we measure via practical experiments on the plaintext the robustness of different neural networks against perturbations of their internal weights that could potentially result from the propagation of large homomorphic noise. This allows us to simulate homomorphic evaluations with large amounts of noise and to predict the effect on the classification accuracy without a real evaluation of heavy and time-consuming homomorphic operations. In addition, this approach enables us to correctly choose smaller and more efficient parameter sets for both schemes.

Keywords: Neural networks · Homomorphic encryption · TFHE · HEAAN

1 Introduction

Neural networks (NN) are extremely powerful machine learning algorithms for classification or recognition of complex data such as images, handwriting or speech. These algorithms are used in many domains and so, they often treat highly sensitive data like medical records or confidential financial information.

S. Dolev et al. (Eds.): CSCML 2019, LNCS 11527, pp. 212–230, 2019.
https://doi.org/10.1007/978-3-030-20951-3_20

One of the most popular choices for achieving privacy guarantee in these settings is Fully Homomorphic Encryption (FHE) [7,25]. FHE schemes allow for computing on encrypted data in the sense that decrypting the encrypted result yields the result that would have been produced if the computation had been performed on the plaintext. Compared to other privacy-preserving solutions (e.g. Multiparty Computation (MPC)), FHE operations are non-interactive and thus, they save on communication costs. Second, MPC schemes require non-collusion assumptions on the computing parties in order to achieve privacy and such assumptions can often be challenging. In FHE applications, no such assumptions are needed.

The three FHE schemes B/FV [8,12,20], TFHE [16,17], based on [18,21] and HEAAN [14,15], all based on the Ring-LWE problem, are currently among the most efficient constructions. For all of them, a homomorphic noise is added on the top of the plaintext. The scheme B/FV was initially designed to perform exact SIMD operations modulo a prime. Thus, for B/FV the added noise does not affect the outcome of the decryption in the sense that the decrypted value is exactly the plaintext. In this context the noise quantifies how many operations can be homomorphically executed with the decrypted value remaining correct. The notion of bootstrapping in B/FV has therefore the meaning of "noise reduction".

On the other hand, in HEAAN, homomorphic operations are floating-point operations where the least-significant bits of the mantissa are randomly rounded at each arithmetic operation to a value that is close to the exact result. The entropy in the errors induced by these least-significant roundings arise from many factors such as the randomness of the ciphertexts as well as the randomness of the large evaluation keys. These errors, unpredictable to the users, are corrected by neither decryption nor bootstrapping (the decryption of the encryption is not the original message) and accumulate throughout the whole computation.

There is still a notion of the maximum number of homomorphic operations, or *multiplicative level*, that designs the maximum number of operations that can be applied on a given ciphertext: if we do less operations than this level, the decryption produces an approximate result (as opposed to the exact result in B/FV); if we exceed this level, the decryption fails completely in an undefined behaviour manner. The bootstrapping of HEAAN can still extend this level to allow further computations, but it does not reduce noise. As shown in [6], TFHE (the last of the above-mentioned schemes) can be interfaced with both B/FV and HEAAN, and thus, supports both exact or approximated arithmetic: in this paper, we only consider the approximate mode of operation.

For evaluating CNNs with FHE, one can either select one scheme over the other, or propose a hybrid solution combining HEAAN and TFHE. Indeed, some of the computations performed during a CNN evaluation are easier with TFHE while some others are more natural with HEAAN. For instance, in the case when many approximated computations have to be performed and a decision must be taken on the result, it is optimal to use the HEAAN scheme for the first part and switch to TFHE for evaluating the decision function. This hybrid approach based on the Chimera framework has already been used by one of the solutions

proposed to the Idash'18 Track 2 [2,10] competition on designing homomorphic solutions for semi-parallel Genome Wide Association Studies (GWAS) based on logistic regression using homomorphically encrypted data. Here, the logistic regression requiring many iterations was performed with TFHE in order to use a fast bootstrapping to reduce the noise, whereas the linear algebra computations on matrix of large dimensions was performed with HEAAN using massively vectorized SIMD computations offered by the scheme.

Our Contributions. The goal of this paper is to efficienty simulate homomorphic evaluation of neural network predictions (in particular, CNN predictions) in order to analyze the stability of the performance (evaluation accuracy) of neural networks in the presence of noise due to FHE decryptions (what we refer to as the approximated computational model). Performing such an analysis on encrypted data can be extremely time consuming, the reason why we choose to do our analysis on the plaintext and simulate the noise resulting from the homomorphic computations and the function approximations.

In order to perform our experiments on approximate operations, we have chosen exclusively the HEAAN and TFHE schemes implemented in the context of the Chimera [6] framework. After analyzing the noise in the homomorphic operations in Sect. 2 and explaining how/why this noise can be modeled with Gaussian distributions in the context of both TFHE and HEAAN, we show in Sect. 3 how one can homomorphically evaluate various commonly used functions for deep learning (e.g., `Abs`, `Sign`, max and `ReLU`) with TFHE and HEAAN and discuss the potential difficulties of such an approach. We remark that the choice of a scheme also depends on the desired level of precision in the output. Through the simulation approach, one is able to efficiently determine the best CNN structure and the smallest FHE parameters required during a preliminary study phase.

We performed experiments with perturbations on three distinct convolutional neural networks of small (LeNet-5), medium (cat-and-dog-9) and large (ResNet-34) size and observed that all these networks support large relative errors of at least 10% without almost any impact on the global accuracy (see Sect. 2). This means, as we show, that only 4 bits of precision (instead of 20 to 40 bits usually) are needed on all fixed-point operations throughout the network, which yields very small parameter sets and fast homomorphic operations. Finally, these experiments allowed to make useful deductions about the stability of some common CNN operations (e.g., different pooling functions). As we show, all operations are not equally stable and thus, some of them should be preferred when used in a FHE context.

Outline. In Sect. 2 we recall the homomorphic schemes TFHE and HEAAN and we analyse the noise propagation for basic arithmetic (linear combination, multiplication, rotation/permutation and bootstrapping). In Sect. 3, we show how to tweak the bootstrapping to evaluate the activation functions of the CNN. Finally, in Sect. 4, we simulate the noise propagation during the homomorphic evaluation and measure its effect on the CNN prediction accuracy.

2 Homomorphic RLWE Encryption Schemes and Noise Propagation

The goal of this section is to introduce the key concepts concerning both the TFHE and HEAAN schemes that are necessary to understand the core of this article. As explained earlier, both schemes will be introduced via the Chimera framework [6] that provides a unified representation of these schemes as well as a uniform noise analysis. We start by introducing some necessary notation.

Notation. Let $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ be the real torus, that is the set of real numbers modulo 1. We denote further by $\mathbb{Z}_N[X] = \mathbb{Z}[X]/(X^N + 1)$ the ring of polynomials with integer coefficients modulo $X^N + 1$. Respectively, $\mathbb{R}_N[X] = \mathbb{R}[X]/(X^N + 1)$ is the ring of real polynomials modulo $X^N + 1$. Informally, the elements of $\mathbb{Z}_N[X]$ are seen as integer polynomials with N coefficients whereas the elements of $\mathbb{R}_N[X]$ are seen as real polynomials with N coefficients.

In order to introduce the notion of slots (real or complex), we use the following two isomorphisms of $\mathbb{R}$-vector spaces:

$$\mathbb{R}[X]/(X^N + 1) \simeq \mathbb{R}^N, \qquad f = a_0 + \cdots + a_{N-1}X^{N-1} \mapsto (a_0, \ldots, a_N), \tag{1}$$

and

$$\mathbb{R}[X]/(X^N + 1) \simeq \mathbb{C}^{N/2}, \qquad f \mapsto (f(\zeta), f(\zeta^3), \ldots, f(\zeta^{N-1})). \tag{2}$$

Here $\zeta = e^{\pi i/N}$ is a primitive root of $X^N + 1$. Representation (1) corresponds to what is called the *coefficient packing* and representation (2) corresponds to what is called the *slot packing*.

2.1 HEAAN and TFHE Through the Chimera Framework

The Chimera framework introduced in [6] allows to apply elementary operations either from the HEAAN or the TFHE library to RLWE ciphertexts [29] within the same FHE computation. Both use the same ciphertext space.

In this work, we describe these libraries mostly from the user point of view without going into the details of their internal representation. In particular, we view an RLWE ciphertext as an encryption of a plaintext in $\mathbb{C}^{N/2}$ (i.e., $N/2$ complex plaintext slots under the isomorphism (2)) on which one can perform approximated arithmetic. The coefficients to slots representation and slots to coefficients representation can be used at any moment to switch between a slot-based representation in $\mathbb{C}^{N/2}$ and a coefficient-based representation in $\mathbb{R}^N$.

The slots in a given ciphertext vector have a fixed public precision $\rho > 0$ in the following sense: the complex coordinates of the vector are all of the form $(x + iy) \cdot 2^\tau$ for some public exponent $\tau > 0$ (uniform across all the coordinates and precomputed in advance) and some secret $x, y \in [-1, 1]$. In addition, both x and y are assumed to have ρ fractional bits of precision (i.e., the size of the mantissa is exactly ρ bits, where ρ is usually a fixed constant across the entire computation).

During the FHE computation, only the ρ-bits of the mantissa are secret and are the only ones that are homomorphically evaluated.

In a pure floating-point model, the result of some operations cannot always be exactly represented on the target precision: these results are usually rounded to the nearest mantissa. In FHE, these roundings are more random and difficult to predict and we modelize this via a noise propagation model, whose mean and standard deviation depend on the elementary operation.

We will only refer to the internal cryptographic representation of the ciphertexts in the section where we define our noise propagation model. Namely, both TFHE and HEAAN schemes use RLWE ciphertexts in $\mathbb{R}_N[X]^2 \mod 1$ (or $\mathbb{R}_N[X]^2 \mod q$), the same key space $\mathbb{Z}_N[X]$ with small coefficients and the same phase function $\varphi_s(a,b) = b - s \cdot a$ introduced in [16]. In this framework, the approximated decryption, common to HEAAN and TFHE, considers that the phase is always close to the actual message and is a good enough approximation thereof. Then, accumulated errors are not corrected by the cryptosystem but rather by the numerical stability of the homomorphically evaluated algorithm.

Finally, the notion of level common to TFHE and HEAAN is defined as the maximal multiplicative depth supported by the ciphertext. Each homomorphic product reduces the level of the resulting ciphertext; when the level 0 is reached, the ciphertext must be bootstrapped to continue operating on it.

Consider a security parameter λ, a maximal level L and a target precision ρ, then these parameters implicitly define a minimal key size N. For more details see the FHE standardization workshop security document [3].

Below, we describe the algorithms for encryption and decryption that are used in TFHE and HEAAN, both enabling error-tolerant decryption functions, and hence approximated arithmetic.

KeyGen: A uniformly random binary key $s \in \mathbb{Z}_N[X]$ (with small coefficients). In order to support non-linear operations, **KeyGen** also needs to generate various encryptions of s, such as evaluation, key-switching or bootstrapping keys, which are not essential to this paper (see [9,15,20] for more details).

EncryptAtLevel$_{\tau,L}(x,s)$**:** The plaintext x is in $\mathbb{C}^{N/2}$ (complex slots bounded by $|x| \leq 2^\tau$). Divide x by $2^{\tau+L}$ and apply the isomorphism (1) to obtain a small real polynomial μ bounded by 2^{-L}. Then, pick a uniformly random $a \in \mathbb{R}_N[X] \mod 1$, and a small Gaussian error e with standard deviation $2^{-L-\rho}$, and return $(a, s \cdot a + \mu + e)$.

DecryptApproxAtLevel$_{\tau,L}(c,s)$**:** Compute the phase $\varphi_s(c) = b - s \cdot a \mod 1$, lift all its coefficients to the real field in the interval $[-\frac{1}{2}, \frac{1}{2})$ which recovers an approximation of μ, then apply the isomorphism (1) and multiply by $2^{\tau+L}$ to recover the slots x (up to an error $2^{\tau-\rho}$).

Remark 1. Here we describe only a symmetric key version. Note however that the public key version is obtained by evaluation of constant functions using the secret key.

2.2 Noise Models for Homomorphic Operations

We now analyze the resulting output noise of the main homomorphic operations for TFHE and HEAAN. The most common operations are linear combinations, multiplications, slot permutations as well as functional bootstrapping.

Linear Combination. Let $\sum_{i=1}^{k} \alpha_i c_i$ be a linear combination, where the c_i are RLWE ciphertexts that encrypt the plaintexts x_i and $\alpha_i \in \mathbb{Z}$ are small integers. Given independent normally distributed Gaussian noises $e_i \in \mathbb{C}^{N/2}$ (slot representation) with multivariate normal distribution $N(x_i, \sigma_i^2)$ ($x_i \in \mathbb{C}^{N/2}$ is the mean and $\sigma_i \in \mathbb{C}^{N/2 \times N/2}$ is the covariance matrix), the noise of the decryption of $\sum_{i=1}^{k} \alpha_i c_i$ is $\sum_{i=1}^{k} \alpha_i e_i$ which is normally distributed with multivariate distribution $N\left(\sum_{i=1}^{k} \alpha_i \mu_i, \sum_{i=1}^{k} \alpha_i^2 \sigma_i^2\right)$.

We can thus simulate this noise by computing the exact result $\sum_{i=1}^{k} \alpha_i x_i$, applying a random multivariate (discrete) Gaussian noise of amplitude $\sum \alpha_i 2^{\tau_i - \rho}$ and expressing the outcome as an exact multiple of $2^{\tau-\rho}$.

Multiplication. The homomorphic evaluation of a multiplication corresponds to the internal product of ciphertexts of HEAAN [15] or to the external product of Chimera/TFHE [6,17] if one of the operands is a fresh ciphertext.

Assuming that c_1 and c_2 are ciphertexts corresponding to the two plaintexts x_1, x_2 and assuming that the noise parameters $e_1, e_2 \in \mathbb{C}^{N/2}$ in the decryptions of c_1 and c_2, respectively, are independent and normally distributed according to $N(\mu_i, \sigma_i)$, then the distribution of the noise parameter for the decryption of the product $c_1 c_2$ can be approximated with a normal distribution. Indeed, note that,

$$(x_1 + e_1)(x_2 + e_2) = x_1 x_2 + x_1 e_2 + x_2 e_1 + e_1 e_2.$$

Now, for fixed x_1, the terms $x_1 e_2$ and $x_2 e_1$ are clearly normally distributed and $e_1 e_2$ is negligible, so the distribution of the noise in the decryption of the product can be approximated with the normal distribution for $x_1 e_2 + x_2 e_1$. This has already been studied (see e.g. [15,16]). Thus, when multiplying homomorphically two ciphertexts c_1, c_2 representing plaintexts x_1, x_2 with public exponents τ_1, τ_2 and precision ρ, we obtain a ciphertext c with exponent $\tau = \tau_1 + \tau_2$ and precision ρ, which can be modeled as follows: compute the exact product $x_1 x_2$, add a random (discrete) multivariate Gaussian noise of amplitude $2^{\tau_i - \rho}$ and express the outcome as an exact multiple of $2^{\tau-\rho}$.

Rotations/Permutations. One of the possibilities for permuting or rotating the elements in the slot representation is to switch to the coefficient packing. This last operation is easy. Knowing that the transformation between coefficients to slots representation and inversely corresponds to applying an orthogonal (or hermitian) matrix, the effect on the noise is numerically stable and it preserves the Gaussian noise amount. However, this consumes (at least) one homomorphic multiplicative level, because the transformation involves a homomorphic evaluation of a Discrete Fourier Transformation.

Bootstrapping. Traditionally, a bootstrapping applies homomorphically the identity function to the plaintext and resets the multiplicative level to a high value. Here, we omit the noise-reduction part which does not occur in the floating-point mode [14]. Complex non-linear functions are traditionally evaluated by interleaving bootstrappings, SIMD additions and multiplications and slot rotations. However, it is not optimal to proceed this way for three reasons: (1) After a costly bootstrapping, one still needs to evaluate the non-linear function which is time consuming, thus sacrificing efficiency. (2) One can approximate the non-linear function by polynomials: if the approximation can be made arbitrarily precise within a fixed range, the degree and the size of the coefficients rapidly diverge for large ranges and the expression gets numerically unstable outside the specified range (Runge's phenomenon). Therefore, any plaintext outlier can destroy the correctness of the result, which leads to a precision sacrifice. (3) Finally, the bootstrapping needs to raise the multiplicative level very high to leave room for the homomorphic function evaluation, thus requiring excessively large parameters (again, sacrificing efficiency). In the cases of both TFHE and HEAAN, we thus focus on a more numerically stable strategy where the bootstrapping includes the evaluation of the non-linear function.

Functional Bootstrapping in TFHE. Recall that the TFHE scheme evaluates functions via evaluating lookup tables on discretized input [17, §4.3, Alg. 4]. As such, the bootstrapping of TFHE approximates a given function by a step function (in exactly the same way as one performs Riemann integration) and then evaluates the approximation by a homomorphic lookup table evaluation. For example, the `ReLU` function $f(x) = \max(0, x)$ for $-1 \leq x \leq 1$ can be approximated by the step function defined as follows:

$$f_\delta(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ k\delta & \text{if } x \in [(k-1)\delta, k\delta), \end{cases}$$

where $k \in \mathbb{Z}$ and $k \leq 1/\delta$ (see Fig. 1 (left)). Thus, given a plaintext x, instead of computing $f(x)$, one obtains the value $f_\delta(x + e_1) + e_2$ where e_1 and e_2 are two error terms, e_2 being Gaussian noise and e_1 corresponding to an internal rounding error (see the rounding in Step 2 of [16, Alg. 3]).

Functional Bootstrapping in HEAAN. In contrast to TFHE, the original version of HEAAN evaluates the `sine` function by Taylor approximation [14, §3.2]. Moreover, the extension of HEAAN proposed in Chimera generalizes this method to evaluation of Fourier series and thus, evaluation of the given target function via a low-degree Fourier series. Graphically, the target function f is replaced by a smooth function S_f and then a Gaussian noise is added on the top of that (see Fig. 1 (right)). Finally, when the function has a point of singularity (such as the `ReLU` at the point $x = 0$), the HEAAN approximation is biased at that point (strictly above $x = 0$). It is thus desirable to validate the effect of this biased approximation to the quality of prediction of the trained convolutional neural network.

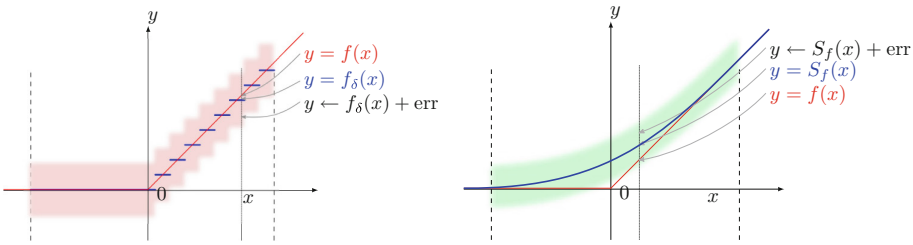

Fig. 1. Functional bootstrap in TFHE for the `ReLU` function (in left) and Functional bootstrap in HEAAN for the `ReLU` function (in right). (Color figure online)

In conclusion of this section, since every elementary FHE operation has a Gaussian noise in output, we can omit the input noise from the bootstrapping and merge it with the output noise of the previous operation in our simulation.

3 Evaluation of Nonlinear Functions in Neural Networks

Non-linear functions are central building blocks in deep learning and as such it is important to analyse how to homomorphically evaluate them. Examples of such operations are comparisons, max functions, piecewise functions (e.g. the REctified Linear Unit (`ReLU`) := $\max(0, x)$ activation function), rounding, a decryption function (equivalent to the sign function) or continuous functions such as the sigmoid $\mathsf{sigmoid}(x) = 1/1 + \exp(-x)$.

Note that the `ReLU` and max are easily expressed with the absolute value: for x, y in $(-1/4, 1/4)$, $2\max(x, y) = (x + y) + |x - y|$, and for $2\texttt{ReLU}(x) = x + |x|$.

3.1 Non-linear Functions in TFHE

In TFHE, given a non-linear function $f : \mathbb{T} \to \mathbb{T}$, one can compute $f(\varphi_s(c))$ (see Sect. 2.1) for a LWE ciphertext c via functional boostrapping under the following constraints: the domain of the function is restricted to multiples of $1/2N$ where N is the bootstrapping key size (in particular, it is a medium-sized power of 2), and the function must be $(1/2)$-antiperiodic, i.e. $f(x + 1/2) = -f(x)$. On the half-period, the function can be defined pointwise, so its graph can be arbitrary. Some particular functions such as $\texttt{Abs}(x) - 1/4$ and $\texttt{Sign}(x)$ already coincide with a $(1/2)$-antiperiodic function over $[-1/2, 1/2]$ (see Fig. 2). More general functions such as $\mathsf{sigmoid}(\gamma x) - 1/2$ can be defined over $[-1/2, 1/2]$ and extended to $\mathbb{R}$ by anti-periodicity.

Once the $(1/2)$-antiperiodic function f to evaluate is chosen, its graph is mapped to the element $\nu = \sum_{i=0}^{N-1} \nu_i X^i \in \mathbb{R}_N[X] \mod 1$ where $\nu_i = f(i/2N)$ and used as a test vector in the bootstrapping of TFHE to evaluate f (see [17, §6.1]). In the output of the bootstrapping, the decrypted value is within a small Gaussian error around $f(x)$ as discussed in Sect. 2.

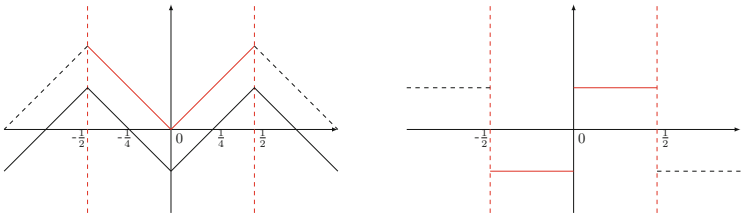

Fig. 2. Absolute (on the left) and Sign (on the right) values TFHE

3.2 Non-linear Functions in HEAAN

In HEAAN, non-linear functions can be evaluated via approximations by either complex-valued polynomials (via traditional products) or trigonometric polynomials (Fourier approach within the bootstrapping).

As explained in [5], Fourier series of smooth and regular functions converge rapidly: for instance, the Fourier series of a C^∞-function converges superalgebraically and if one smooths any periodic function by convolution with a small Gaussian, its Fourier series converges exponentially fast. However, the convergence is slower if the function has discontinuities (pointwise convergence in $\Omega(1/k)$), or discontinuities in its derivative (uniform convergence in $\Omega(1/k^2)$) where k is the number of harmonics used in the series.

For example, the absolute value is a triangular signal on $[-1/2, 1/2)$ which extends naturally to a 1-periodic continuous function (piecewise C^1). Given $N/2$ LWE ciphertexts, we can efficiently pack the complex exponential of their phases $\exp(2i\pi\mu)$ in the slots of a single HEAAN ciphertext. Subsequently, we can evaluate any trigonometric polynomial of small degree and extract the results back to LWE samples. For instance, the triangular signal (corresponding to the absolute value) has the following Fourier series with only cosine terms of odd degrees that converge in $O(k^2)$ and the square signal (corresponding to the sign or decryption function) has only sine terms of odd degrees.

$$\texttt{Abs}(x) = K_1 \sum_{k=0}^{\infty} \frac{\cos 2\pi(2k+1)x}{(2k+1)^2} + K_2, \texttt{Sign}(x) = K_1 \sum_{k=0}^{\infty} \frac{\sin 2\pi(2k+1)x}{(2k+1)} + K_2$$

Figure 3 shows that the first three (resp. six) terms of the Fourier series of the absolute value and the sign function already provide a good approximation on the interval $[-1/2, 1/2)$.

Compared to classical approximations of functions by polynomials in [11,22] (i.e. Taylor series or Weierstrass approximation theorem), Fourier series have three main advantages: they do not diverge to ∞ outside of the interval (better numerical stability), the Fourier coefficients are small (square integrable), and the series converge uniformly to the function on any interval that does not contain any discontinuity in the derivative. However, in the particular case of `Abs` and `Sign`, the presence of a singularity or discontinuity at $x = 0$ in both graphs

implies that the series converge poorly around 0. Unfortunately, native plaintexts in HEAAN ciphertext at level L have by definition tiny phases in the interval $\left[-1/2^L, 1/2^L\right)$. We address this problem using the bootstrapping capability of HEAAN: First, we decrease the level $L = 0$ or $L = 1$ (using the algorithm of re-scaling defined in [15]), so that input phases range over a large torus interval $(-1/2, 1/2)$ or $(-1/4, 1/4)$, and then, divide K_1 by 2^L so that the output has level L.

With this bootstrapping trick, HEAAN can at the same time evaluate a non-linear function and bootstrap its output to a level L even higher than its input. Taking this fact into account, instead of writing $\texttt{ReLU}(x) = \max(0, x)$ as $\frac{1}{2}(|x|+x)$ like in TFHE, where the term $+x/2$ is not bootstrapped, it is actually better to extend the graph of ReLU from a half period $(-1/4, 1/4)$ directly to a 1-periodic continuous function and to decompose the whole graph as a Fourier series. In the latter case, the output level L can be freely set to an arbitrary large value. Figure 3 shows a degree-7 approximation of the odd-even periodic extension of the graph of $\texttt{ReLU}(x)$. If the ReLU is evaluated via this technique, the output message is the Fourier approximation, and the phase still carries an additional Gaussian noise on top of it, as shown in Sect. 2. In the next section, we also study the robustness of neural networks with this approximation and perturbation model.

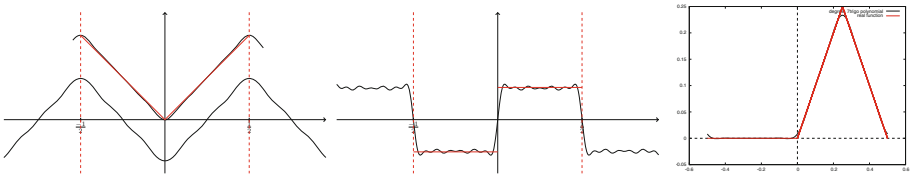

Fig. 3. Abs (on the left), Sign (in the middle) and ReLU (on the right) for HEAAN

4 Predictions for Deep Learning

Neural networks (NN) are computing systems trained to solve among others classification problems. Networks with multiple layers are known as *deep*. *Convolutional neural networks* (CNN) are a special type of deep neural networks that have been proven very successful in image recognition and classification. Preserving the privacy of sensitive data (e.g., medical or financial) while applying machine learning algorithms and still ensuring good performance and high output accuracy is currently a problem of interest to both the cryptographic and the machine learning communities [4,7,10,11,23,25,32]. We briefly describe now the main layers composing a CNN from a FHE point of view.

Convolution: It is an operation that extracts features from the input image (such as lines or borders) and is achieved by computing convolutions (via

element-wise products) of the input matrix and a filter. Convolution is viewed as a secret affine function that can be efficiently evaluated using the external product of [6,17].

Non-linearity: To introduce non-linearity, an activation function is then applied to the output of the convolution. Nowadays, this is almost always achieved by the `ReLU` function. In almost all previous works, the standard approach was to replace the `ReLU` by a function with a lower multiplicative depth. In [23], `ReLU` is notably approximated by the square function $f(x) = x^2$, in [7] it is replaced by the sign function, while in [11] the `ReLU` is approximated by low-degree polynomials.

Pooling: This layer reduces the dimensions of the input by retaining the most important information. This is typically done by a procedure called *max pooling* or more rarely by *average pooling* that compute the maximum (resp. average) value for every disjoint region of the input. Today, no efficient algorithm is known to compute the maximum of a large number of values. On the contrary, average pooling is linear with public coefficients and therefore FHE-friendly. In [23] the authors replace max pooling by sum pooling, while in [11] max pooling is replaced by average pooling.

Fully Connected (FC) Layer: All the neurons of this layer are all connected to all neurons of the previous layer. Their activation is computed by a matrix multiplication plus a bias offset. This is again a secret precomputed affine step that can be achieved via the external product.

Loss Layer: This is normally the last layer of a CNN. During the evaluation, the loss layer becomes an argmax operation. This last step is in general ignored in other homomorphic implementations of neural networks. For example, in [7, 23], the authors simply output the score vectors and the recipient computes the best one after decryption. To do this final step homomorphically, the boolean approach of TFHE seems to be the most suited to this non-SIMD step.

4.1 Robustness Against the FHE Error Models

In this section, we simulate the homomorphic execution of the neural network by replacing the value output of each non-linear layer by a random sample which has the same distribution as the phase of RLWE samples after a homomorphic evaluation of the layer. This approach allows us to simulate a homomorphic evaluation, and to obtain accurate predictions on the outcome without having to run the expensive homomorphic computation. This allows to estimate the largest noise standard deviation α that can be tolerated by the network, and therefore, the smallest FHE parameters required to evaluate it. In our experiments we add Gaussian noise with varying standard deviation and look for the maximal standard deviation of the noise that can be tolerated by the network.

As explained above, in the context of FHE, the training of networks is usually done on the plaintexts without any perturbations occurring, and only then, the network is encrypted to the cloud to protect the privacy of the model during

predictions. In this direction, we carried out many experiments on three different convolutional neural networks structures, using the TFHE and HEAAN noise models of Sect. 2, in order to measure their robustness against such perturbations. This approach is not new. For example, in [13] the authors studied the stability of CNNs by applying among others a Gaussian perturbation to the internal weights inside the convolutional layers. The applied Gaussian was centered at zero and had a standard deviation relative to the standard deviation of that layer's original weight distribution. This type of perturbation modifies the average value of the inputs to the convolutional layer. Even, if the motivation of this paper is not linked to homomorphic computations, their conclusions and ours intersect at some points. Indeed, the authors of [13] noticed that the last convolutional layers are surprisingly stable, while the first convolutional layers are much more fragile and so the accuracy depends on the level the perturbation applies. The most surprising result that we obtain in our experiments is that all the neural networks we tested support quite large relative errors of at least 10% of 2^{τ}, without any impact on the global accuracy. In a TFHE context, raising the error amplitude from a usually required 2^{-40} negligible amount to 2^{-4} means that the depth of leveled circuits (number of transitions in automata in leveled circuits in [17]) can be increased by a factor $(2^{36})^2$ without changing the parameter sets. This also means that only 4 bits of precision (instead of 20 to 40 bits usually) are needed on all fixed point operations throughout the network, which results notably in very small parameter sets for HEAAN.

4.2 Experiments

We conducted experiments with three different convolutional neural networks and for all of them we used the `dlib C++` library [26]. The first network is LeNet-5 [27], that can be trained to recognize handwritten digits, the second-one is a 9-layer CNN trained to distinguish cat from dog pictures, and the last one is the ResNet-34 network [24], a deep network of 34 layers able to classify an input image into one of 1000 objects. We briefly describe each of the networks and the experiments done on it.

LeNet-5: Recognition of Handwritten Digits. LeNet-5 is a well-known convolutional 7-layer neural network designed by LeCun et al. in 1998 to recognize handwritten digits [27]. In the original version of the network, the sigmoid was used as the activation function. In the version that we manipulated (`dlib` library [26]), the `ReLU` activation function is used instead.

We trained this network on the MNIST dataset [28], composed of 60000 training and 10000 testing images, with two different versions of the pooling algorithm. We first trained the network by using max pool for both pooling layers and at a second stage we re-trained it from scratch by replacing now max pool by average pool. Our goal was to see how each version reacts to perturbations. In particular, we added to each output value of the activation function a value drawn from a Gaussian distribution with mean value zero and some standard deviation σ. This was done for the activation function of all levels. For

our experiments we further used two different activation functions: the original ReLU activation function and then an approximation of the ReLU function by a trigonometric function, depicted in Fig. 3 (right), or in green in Fig. 1 (right) which can be used in HEAAN as a replacement of $\max(0, x)$. Finally, we perturbed the output of the activation function in two different ways. First by a Gaussian distribution of fixed standard deviation σ and in a second experiment by a standard deviation proportional to the input's standard deviation (which can be publicly estimated during training).

The results of these experiments are summarized in Table 1 and Fig. 4. In this example, we pushed standard deviation from 0.0 to 1.0 for both trained CNNs, the one trained with max pool and the other one trained with average pool. In Table 1 we give both the accuracy on the testing set but also on the training set. In order to correctly interpret the right part of Fig. 4 it has to be noted that the mean value of the ReLU entries was measured between 0.4 and 1.91 and the standard deviation between 0.97 and 2.63.

Table 1. Experiments on the LeNet-5 network trained first with max pool and then with average pool. ReLU means that during the evaluation the original ReLU function was used, while $\widetilde{\text{ReLU}}$ signifies that an approximation was used instead.

Pool type	σ	Non-proportional perturbation				Proportional perturbation			
		ReLU		$\widetilde{\text{ReLU}}$		ReLU		$\widetilde{\text{ReLU}}$	
		Train acc.	Test acc.	Train acc.	Test acc.	Train acc.	Test acc.	Train acc.	Test acc.
Max	0.0	0.9999	0.9924	0.9999	0.9924	0.9999	0.9924	0.9999	0.9924
Average		0.9994	0.9903	0.9994	0.9903	0.9994	0.9903	0.9975	0.9903
Max	0.1	0.9998	0.9918	0.9996	0.9916	0.9984	0.9908	0.9980	0.9905
Average		0.9994	0.9903	0.9993	0.9904	0.9977	0.9891	0.9976	0.9892
Max	0.2	0.9990	0.9910	0.9976	0.9899	0.9883	0.9835	0.9842	0.9787
Average		0.9991	0.9901	0.9985	0.9894	0.9897	0.9843	0.9878	0.9826
Max	0.3	0.9966	0.9894	0.9901	0.9833	0.9540	0.9501	0.9199	0.9192
Average		0.9981	0.9898	0.9960	0.9872	0.9699	0.9655	0.9595	0.9581
Max	0.4	0.9919	0.9843	0.9654	0.9610	0.8686	0.8723	0.7695	0.7815
Average		0.9968	0.9884	0.9908	0.9845	0.9308	0.9296	0.9014	0.9039
Max	0.5	0.9823	0.9766	0.8942	0.8966	0.7475	0.7587	0.5901	0.5959
Average		0.9947	0.9869	0.9792	0.9737	0.8728	0.8745	0.8156	0.8214
Max	0.6	0.9626	0.9610	0.7644	0.7737	0.6199	0.6248	0.4325	0.4317
Average		0.9919	0.9842	0.9552	0.9517	0.8007	0.8054	0.7179	0.7245
Max	0.7	0.9284	0.9280	0.6166	0.6288	0.5013	0.5024	0.3233	0.3274
Average		0.9883	0.9816	0.9171	0.917	0.7219	0.7288	0.6212	0.6332
Max	0.8	0.8756	0.8808	0.4809	0.4953	0.4040	0.4056	0.2526	0.2576
Average		0.9843	0.9779	0.8633	0.8698	0.6433	0.6506	0.5295	0.5383
Max	0.9	0.8103	0.8191	0.3826	0.3884	0.3316	0.3322	0.2036	0.2094
Average		0.9779	0.9724	0.8044	0.8135	0.5691	0.5727	0.4498	0.4538
Max	1.0	0.7399	0.7462	0.3179	0.326	0.2757	0.2803	0.1719	0.1732
Average		0.9696	0.9636	0.7434	0.7548	0.4989	0.5062	0.3822	0.3862

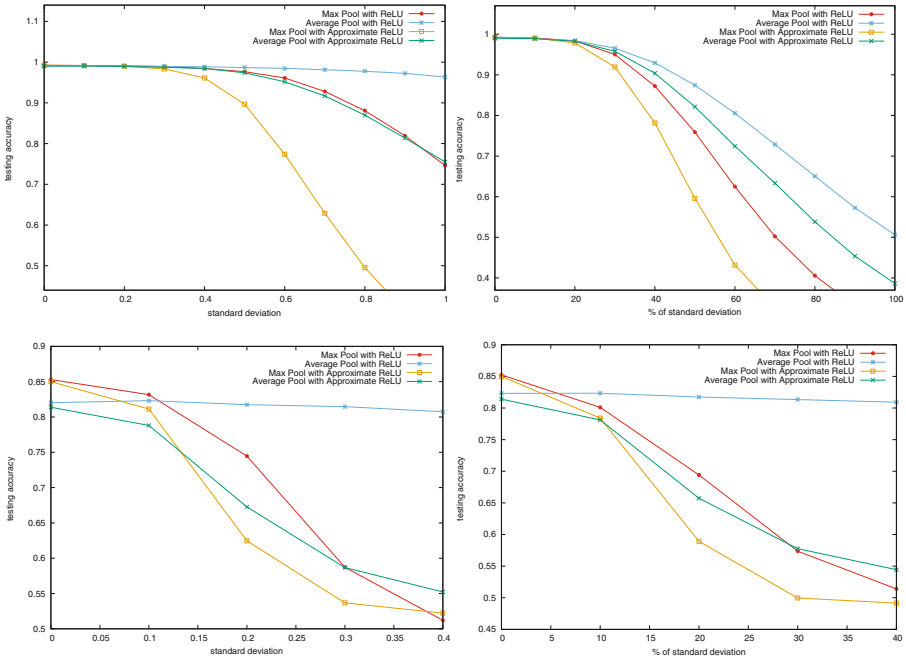

Fig. 4. Experiments with LeNet-5 (up) and the cat versus dog classifier (down). The results with proportional perturbations are on the right, while with non-proportional perturbations on the left.

The first remark that can be done by looking into these experiments is that average pool is much more stable to perturbations than max pool and provides a high accuracy even for large values of the standard deviation. The second remark concerns the accuracy when an approximation of the `ReLU` function is used instead of the original one. As it can be seen from the left part of Fig. 4, the accuracy for the average pool version is clearly lower when a `ReLU` approximation is used, but still has a very good score (over 95%) for standard deviations as high as 0.6. Finally, special care has to be taken when interpreting the results corresponding to the application of a proportional perturbation of the input data standard deviation. In the right part of Fig. 4 the x-axis corresponds to a perturbation equal to the percentage of the inputs' standard deviation. Depending on the original deviation of the input distribution, the perturbation can be extremely important and this is why the accuracy shows to drop. Therefore, one has to keep in mind that the perturbation of the right-side figures is in general more important and probably also more meaningful than the one of the left-side figures.

Cats versus Dogs Classifier. In this section we present our results and remarks on a simple 9-layer neural network that was trained to classify pictures as cats or dogs. For this, we used again the `dlib` library [26] and coded with it

the 9-layer NN presented in [1]. The structure of this NN is depicted in Fig. 5. This network is composed of 3 convolution layers followed by the `ReLU` activation function, two fully connected (FC) layers and two pooling layers. In the original net, the max pool operation is used at this step. The 7-th layer is a dropout layer, that is a standard technique for reducing overfitting and consists in ignoring a different randomly chosen part of neurons during the different stages of the training phase [31]. We trained this network on the Asirra dataset [19] used by Microsoft Research in the context of a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) challenge. Most of the good CNNs trained to distinguish dogs from cats achieve more than 80% accuracy on the testing set while the accuracy on the training set is usually around 100%. The difference in the two performances is usually due to some overfitting occuring.

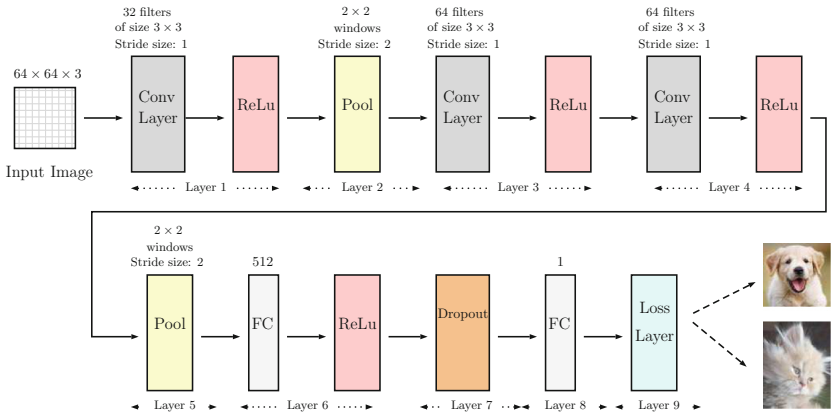

Fig. 5. 9-layer neural network [1] trained to classify pictures as cats or dogs.

We did exactly the same type of experiments for this network and the results can be found in Table 2 or visualized in the lower part of Fig. 4. This network is a little-bit more complex than LeNet-5 and seems to be less stable. For this reason, the higher standard deviation considered here is 0.4. However, globally, the same remarks as for LeNet-5 network result. Again, for correctly interpreting the right part of the table, it has to be noted that the mean value of the inputs of the activation function ranges between 0.0004 and 0.628 and the standard deviation ranges between 0.0067 and 3.51.

ResNet-34. ResNet (Residual Network) is a recent family of very deep convolutional neural networks showed to perform extremely well [24]. The global layer structure is very similar to a classical CNN, however better performances are achieved by the introduction of a shortcut connection, that consists in skipping one or more layers. The version that we used is composed of 34 layers, and is

Table 2. Experiments on a 9-layer CNN trained to distinguish cats from dogs.

Pool type	σ	Non-proportional perturbation			Proportional perturbation	
		ReLU		$\widetilde{\text{ReLU}}$	ReLU	$\widetilde{\text{ReLU}}$
		Training acc.	Test acc.	Test acc.	Test acc.	Test acc.
Max	0.0	0.9999	0.8530	0.8500	0.8524	0.85
Average		0.99995	0.8202	0.8138	0.8232	0.814
Max	0.1	0.9944	0.8316	0.8112	0.801	0.784
Average		0.99995	0.8232	0.7880	0.8234	0.7812
Max	0.2	0.8782	0.7446	0.6246	0.6942	0.5892
Average		0.9999	0.8174	0.6726	0.8174	0.6574
Max	0.3	0.6234	0.5872	0.5368	0.5736	0.4996
Average		0.99965	0.8146	0.5868	0.8134	0.5776
Max	0.4	0.5228	0.512	0.5222	0.514	0.4916
Average		0.998	0.8074	0.5522	0.8092	0.5444

abbreviated as ResNet-34. This network, once trained, is able to classify photos of objects into 1000 distinct object categories.

The training of such residual networks is extremely time consuming (two weeks on a 16-GB Titan GPU, and about 20 times more on 16-CPU cores) and because of time constraints we were not able to finish the training on a network where max pooling is replaced by average pooling. Thus, we were only able to perform our experiments on the pre-trained network on the imagenet ILSVRC2015 dataset [30] and the results are reported in Table 3. Top 1 and Top 5 labels report respectively the percentage of the pictures in the validation set that were correctly classified (Top 1) and whose correct label appeared in the five top suggestions provided by the network (Top 5).

Table 3. Experiments on ResNet-34 with max pooling and with perturbations of standard deviation ranging from 0.0 to 0.5. The right columns correspond to perturbations proportional to the input's standard deviation.

Pool type	σ	Non-proportional perturbation				Proportional perturbation			
		ReLU		$\widetilde{\text{ReLU}}$		ReLU		$\widetilde{\text{ReLU}}$	
		Top 1	Top 5	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5
Max	0.0	0.7416	0.9158	0.7428	0.9187	0.7439	0.9202	0.7398	0.9166
Max	0.1	0.7357	0.9132	0.7056	0.9165	0.7132	0.8948	0.7586	0.9252
Max	0.2	0.6991	0.8860	0.7056	0.8967	0.1562	0.3294	0.3658	0.6027
Max	0.3	0.5068	0.7267	0.4829	0.7171	0.0019	0.0089	0.0012	0.0079
Max	0.4	0.1500	0.0498	0.1065	0.0817	0.0018	0.0085	0.000	0.0009
Max	0.5	0.0233	0.0608	0.0017	0.0066	0.0001	0.0044	0.000	0.0010

4.3 Conclusion/Discussion

Finally, we summarize the experiments with the three different CNNs, provide links to Sect. 3 and give recommendations on which operation should be performed with which FHE scheme (depending on the given use case).

Max pool versus Average pool: We conducted experiments on LeNet-5 and the 9-layer CNN classifying cats and dogs, by replacing during the training and the evaluation the classical max pooling operation by the average pool. This modification, applied also in [11] and to some extend in [23], offers a significant advantage for all FHE schemes, as this operation is affine with public coefficients, compared to max pool that is non-linear. Our experiments showed that this approach offers a further advantage in FHE, as it is way more stable than max pool to perturbations. This behaviour has a natural mathematical explanation, since the standard deviation of an average of independent samples is smaller than the input standard deviations.

Proportional versus non-proportional perturbations: We applied two types of perturbations to all three networks. The first type of perturbations was the addition at the output of the activation function of a value drawn from a Gaussian distribution with zero mean and a fixed standard deviation. In the second type of perturbations, the value added had a standard deviation proportional to the standard deviation of the input distribution. The second scenario corresponds to the fixed-point arithmetic model, where the public plaintext exponent τ is set to match the amplitude during the training phase, and therefore, the noise α is by definition relative to 2^{τ}. Surprisingly, without impacting the result, neural networks are able to absorb very large relative errors between 10% and 20% after each `ReLU` (there are respectively thousands, millions, and billions of them in the three tested networks). This means homomorphic parameters need only to ensure $\rho = 4$ bits of precision on the plaintext, instead of the usually recommended $\rho = 30$.

Approximating the `ReLU` activation function: The main source of non-linearity of a convolutional neural network is coming from the `ReLU` function. In TFHE these functions are evaluated exactly either as circuits, or as point-wise-defined arbitrary functions. Approximating the `ReLU` by something easier is thus a natural approach [7,11,23]. In HEAAN such continuous functions can be approximated accurately by low degree trigonometric polynomials. In our experiments with ResNet-34 (see Table 3) the output accuracy is surprisingly even better with an approximated `ReLU` of this type than with the classical one, in the presence of small noise, which proves that this approach is realistic.

Number of layers: In the plaintext model, the accuracy can in general be improved by adding more layers, if no overfitting occurs. However, in the homomorphic model, what happens with accuracy is still an open question, because with the number of layers, the complexity of computation grows and the activation function can only be approximated. This generates additional noise that can affect the accuracy.

References

1. Cats and dogs and convolutional neural networks, September 2016. http://www.subsubroutine.com/sub-subroutine/2016/9/30/cats-and-dogs-and-convolutional-neural-networks
2. Track 2: Secure parallel genome wide association studies using homomorphic encryption (2018). www.humangenomeprivacy.org/2018/competition-tasks.html
3. Albrecht, M., et al.: Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018
4. Badawi, A.A., et al.: The AlexNet moment for homomorphic encryption: HCNN, the first homomorphic CNN on encrypted data with GPUs. Cryptology ePrint Archive, Report 2018/1056 (2018). https://eprint.iacr.org/2018/1056
5. Boura, C., Chillotti, I., Gama, N., Jetchev, D., Peceny, S., Petric, A.: High-precision privacy-preserving real-valued function evaluation. IACR Cryptology ePrint Archive 2017, 1234 (2017)
6. Boura, C., Gama, N., Georgieva, M.: Chimera: a unified framework for B/FV, TFHE and HEAAN fully homomorphic encryption and predictions for deep learning. Cryptology ePrint Archive, Report 2018/758 (2018)
7. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10993, pp. 483–512. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_17
8. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_50
9. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: ITCS 2012, pp. 309–325. ACM (2012)
10. Carpov, S., Gama, N., Georgieva, M., Troncoso-Pastoriza, J.R.: Privacy-preserving semi-parallel logistic regression training with fully homomorphic encryption. Cryptology ePrint Archive, Report 2019/101 (2019). https://eprint.iacr.org/2019/101
11. Chabanne, H., de Wargny, A., Milgram, J., Morel, C., Prouff, E.: Privacy-preserving classification on deep neural network. Cryptology ePrint Archive, Report 2017/035 (2017). https://eprint.iacr.org/2017/035
12. Chen, H., Laine, K., Player, R.: Simple encrypted arithmetic library - SEAL v2.1. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 3–18. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_1
13. Cheney, N., Schrimpf, M., Kreiman, G.: On the robustness of convolutional neural networks to internal architecture and weight perturbations. CoRR, abs/1703.08245 (2017)
14. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 360–384. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_14
15. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15

16. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 3–33. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_1
17. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. Cryptology ePrint Archive, Report 2018/421 (2018). https://eprint.iacr.org/2018/421
18. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 617–640. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_24
19. Elson, J., Douceur, J.R., Howell, J., Saul. J.: Asirra: a CAPTCHA that exploits interest-aligned manual image categorization. In: Proceedings of the 2007 ACM Security, CCS 2007, pp. 366–374. ACM (2007)
20. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive 2012, 144 (2012)
21. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_5
22. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. In: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, 19–24 June 2016, pp. 201–210 (2016)
23. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. In: ICML 2016. JMLR Workshop and Conference Proceedings, vol. 48, pp. 201–210. JMLR.org (2016)
24. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR 2016, pp. 770–778. IEEE Computer Society (2016)
25. Jiang, X., Kim, M., Lauter, K.E., Song, Y.: Secure outsourced matrix computation and application to neural networks. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, 15–19 October 2018, pp. 1209–1222. ACM (2018)
26. King, D.E.: Dlib-ml: a machine learning toolkit. J. Mach. Learn. Res. **10**, 1755–1758 (2009)
27. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE, pp. 2278–2324 (1998)
28. Lecun, Y., Cortes, C., Burges, C.J.: The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/
29. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_1
30. Russakovsky, O., et al.: ImageNet large scale visual recognition challenge. IJCV **115**(3), 211–252 (2015)
31. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
32. Wagh, S., Gupta, D., Chandran, N.: SecureNN: efficient and private neural network training. Cryptology ePrint Archive, Report 2018/442 (2018). https://eprint.iacr.org/2018/442