New Challenges for Fully Homomorphic Encryption

Ilaria Chillotti
Zama, France

Marc Joye
Zama, France

Pascal Paillier
Zama, France

Abstract

Machine learning and privacy are sometimes perceived to be at odds. Privacy concerns are especially relevant when the involved data are sensitive. This work deals with the privacy-preserving inference of deep neural networks.

We report on first experiments that were done with a new library developed at Zama. The library implements a variant of the TFHE fully homomorphic encryption scheme that features an efficient bootstrapping. Our preliminary results indicate that evaluating deep neural networks is now within the reach of fully homomorphic encryption. They also call for new challenges for fully homomorphic encryption when applied to the inference of deep neural networks.

1 Privacy-Preserving Machine Learning

Machine learning algorithms are extremely useful in many areas but the type of data that they deal with is often sensitive. Typical examples include algorithms for the detection of certain genetic diseases from DNA samples or the ones used for face recognition, email classification, to name a few. The processed data contain private information about users and could be used in many ways, from target advertising to blackmail or even threat in some cases. This is why it is essential to protect the data being used in machine learning applications. Privacy requirements are also pushed by recent regulations companies dealing with user's data must comply with, like the GDPR (General Data Protection Regulation) [11] in Europe or the CCPA (California Consumer Privacy Act) [6] in the US are two examples.

Cryptographic techniques are methods of choice when it comes to the protection of data. But traditional encryption algorithms merely protect data while it is in transit or at rest. Indeed, one limitation and structural property of traditional encryption schemes is that data first needs to be decrypted prior to being processed. A fundamentally different approach is to rely on *fully homomorphic encryption* (FHE). In contrast with traditional encryption schemes where the privacy control lies in the hands of the recipient of the encrypted data, fully homomorphic encryption schemes allow the recipient to directly operate on encrypted data.

Furthermore, as the interactivity is minimal in an FHE-based approach, it can be a better fit for the use-case of *Machine Learning as a Service* (MLaaS), compared to other privacy preserving approaches such as multi-party computation. A typical application scenario is shown in Fig. 1. It involves a user sending data to a service provider (server) running a trained model for a given task (e.g., medical diagnosis). In this work, the focus is on FHE-based techniques for the privacy-preserving inference of neural networks.

2 The Road Towards Practical FHE-based Implementations

First posed as a challenge by Rivest *et al.* in 1978 [19], fully homomorphic encryption—the ability to evaluate any function over encrypted data—was only solved in 2009 in a breakthrough result by Gentry [12]. After a decade of intense research, practical solutions have emerged and are being pushed for standardization [14].

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

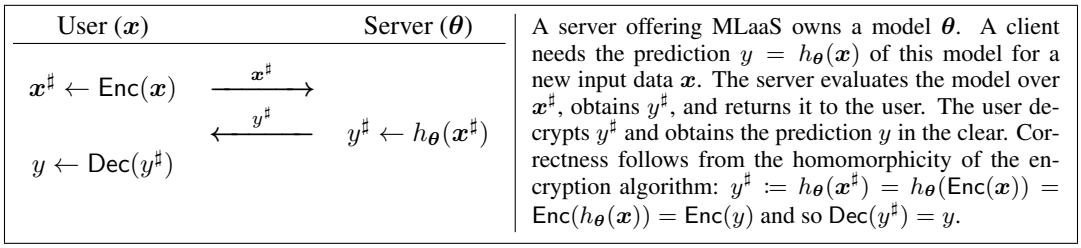

Figure 1: MLaaS over encrypted data.

All known instantiations of fully homomorphic encryption schemes produce noisy ciphertexts. Running homomorphic operations on these ciphertexts in turn increases the noise level in the resulting ciphertext. At the some point, the noise present in a ciphertext may become too large and the ciphertext is no longer decryptable. A homomorphic encryption scheme supporting a predetermined noise threshold is termed *leveled* (or somewhat homomorphic).

At the core of Gentry's result resides the technique of *bootstrapping*. Bootstrapping is a generic technique that allows refreshing ciphertexts. It therefore enables to turn leveled homomorphic encryption schemes into *fully* homomorphic encryption schemes, and so to make them evaluate any possible function over ciphertexts. The key idea behind bootstrapping is to homomorphically evaluate the decryption circuit. An encryption of the decryption key (matching the encryption key used to produce the ciphertext) has to be provided in order to perform the bootstrapping procedure.

The works that followed Gentry's publication were aimed at proposing new schemes or at improving the bootstrapping in order to make FHE more efficient in practice. The most famous constructions are DGHV [8], BGV [5], GSW [13], and their variants. While the constructions that were successively proposed made the bootstrapping more practical, it still constituted the bottleneck (each bootstrapping taking a few minutes). A much faster bootstrapping, based on a GSW-type scheme, was later devised by Ducas and Micciancio [10], reducing the bootstrapping time to a split second. Their technique was further improved and refined, which led to the development of the TFHE scheme [7].

3 Homomorphic Evaluation of Deep Neural Networks

Our starting point is the state-of-the-art TFHE scheme. TFHE can operate in two modes: leveled and bootstrapped. The *leveled mode* supports linear combinations and a predetermined number of products. The operations evaluated in this mode make the noise always grow. The leveled mode can be used to evaluate small-depth circuits. As for the *bootstrapped mode*, it enables a fine control of the noise by reducing it to a given level whenever it exceeds a certain threshold. For problems involving circuits of large depth, only the bootstrapped mode is applicable. Deep neural networks belong to that case.

3.1 Programmable Bootstrapping

In [4], Bourse *et al.* considered a special type of discretized networks where signals are restricted to the set $\{-1, 1\}$ and where the activation function is the sign function. They adapted the TFHE scheme so as to enable the evaluation of the [non-linear] sign function during a bootstrapping step.

Actually, it turns out that the bootstrapped mode of TFHE can be extended to support the evaluation of any function during the bootstrapping step. More specifically, any function (including non-linear functions) of an input ciphertext can be obtained as the output of the bootstrapping. Interestingly, the resulting ciphertext features a controlled level of noise. The process can therefore be iterated over and over. So, in the case of machine learning applications, the depth of neural networks can be arbitrarily large.

A discretized variant of the TFHE scheme including the programmable bootstrapping was developed. So, using this framework, the evaluation of complex functions is achieved thanks to a combination of programmable bootstrapping techniques and leveled operations. The techniques are efficient and directly operate on words of a chosen size.

3.2 Homomorphic Inference

We review below a number of representative layers that are commonly used to build neural networks. The list is non-exhaustive. Our techniques are generic and support all known types of layers.

Dense/linear layer and convolution layer A (fully connected) dense layer computes the dot product between the inputs and a matrix of weights. A bias vector can be added. An activation function is then applied component-wise to produce the outputs. When there is no activation function, a dense layer is also called linear layer. Similarly, a convolution layer convolves the input layer with a convolution kernel (a.k.a. filter) that is composed of a tensor of weights so as to produce a tensor of outputs. Biases can be added to the outputs. Moreover, an activation function can be applied to the outputs.

When evaluated homomorphically, the weights and the biases are provided in the clear. Hence, the evaluation of these two layers (the activation excepted) consists of a series of multiplications by constants and additions, which are all leveled operations. The activation functions are treated below.

Activation layer An activation layer is used to inject non-linearity in the neural networks. It is crucial in the learning. There are many activation functions that can be used in an activation layer. One of the most popular activation functions is the Rectified Linear Unit (ReLU) function. Other commonly used activations include the sigmoid function or the hyperbolic tangent function.

As aforementioned, the homomorphic evaluation of an activation function (as any function) can be performed via a programmable bootstrapping (PBS), with the outputs of the function encoded inside the test polynomial.

Global average pooling layer A global average pooling layer computes the average of the components of its inputs. Specifically, if n denotes the number of components and a_i denotes the value of component i, the global average pooling function computes $\left(\sum_{i=1}^{n} a_i\right)/n$.

In a homomorphic evaluation, the global average pooling can be reduced to the computation of the sum $\sum_{i=1}^{n} a_i$. The division by n is then performed in the next programmable bootstrapping—e.g., in a dense layer or a convolution layer—by dividing the weights by the same quantity. Hence, the sole homomorphic operation required to evaluate a global average pooling layer is the addition of ciphertexts.

Max-pooling layer A max-pooling layer extracts a fixed-size subset of components from the inputs and computes their maximum.

At first sight, as the max function is multivariate (i.e., its takes multiple arguments on input), it is unclear how it can be evaluated homomorphically. With two arguments, the max function can however be expressed using the [univariate] ReLU function, $\max(x, y) = y + \mathrm{ReLU}(x - y)$. In order to evaluate the max-pooling on more arguments, the basic relation $\max(x_1, \ldots, x_{k-1}, x_k) = \max(y_k, x_k)$ with $y_k = \max(x_1, \ldots, x_{k-1})$ can be used.

3.3 Experimental Results

We conducted a series of numerical experiments to assess the performance against the MNIST dataset [18]. We report below preliminary results for depth-20, 50, 100 neural networks, respectively noted NN-20, NN-50 and NN-100. These networks all include dense and convolution layers with activation functions; every hidden layer possesses at least 92 active neurons.

Parameters Using the notation of [7], selected cryptographic parameters are $(k, N, \sigma) = (1, 4096, 2^{-62})$ for GLWE encryption and $(n, \sigma) = (938, 2^{-23})$ for LWE encryption. The word-size is $\Omega = 64$ bits. These two parameter sets meet a 128-bit security level and were validated using the `lwe-estimator` (https://bitbucket.org/malb/lwe-estimator/) [1].

Performance analysis Experiments were performed on two different types on machine: a personal computer with 2.6 GHz 6-Core Intel® Core™ i7 processor, and a 3.00 GHz Intel® Xeon® Platinum 8275CL processor with 96 vCPUs hosted on AWS. The two machines are referred to as PC and AWS. The running times are given in the table below. For reference, we also included the

times for an unencrypted inference. It is important to note that the given times correspond to the evaluation of a single inference run independently; in particular, the times are not amortized over a batch of inferences. The AWS implementation takes advantage of the 96 vCPUs; in particular, the neurons in the hidden layers are processed in parallel.

	In the clear		Encrypted		
	PC	Accuracy	PC	AWS	Accuracy
NN-20	0.17 ms	97.5 %	115.52 s	17.96 s	97.5 %
NN-50	0.20 ms	95.4 %	233.55 s	37.69 s	95.4 %
NN-100	0.33 ms	95.2 %	481.61 s	69.32 s	90.5 %

Regarding the output accuracy, we observe a drop of nearly 5% for NN-100 compared to the accuracy when run over clear data. We note that our results are preliminary. The gap should be narrowed by spending more time in the parameter tuning. Our next work is to fully automate the parameter selection. Meanwhile, some manual adjustments and trial-and-error need to be made.

4 Discussion and Perspectives

Earlier works attempted to evaluate neural networks using fully homomorphic encryption. Cryptonets [9] was the first initiative towards this goal. They were able to perform a homomorphic inference over 5 layers against the MNIST dataset [18]. In order to limit the noise growth, the standard activation function was replaced with the square function. A number of subsequent works have adopted a similar approach and improved it in various directions. Among them, it is worth mentioning the results of the iDASH competition [15], whose goal is to find privacy-preserving solutions in the context of genome analysis. The winning solutions of the homomorphic encryption track of the last two editions (namely, [17, 2] for 2018 and [16] for 2019) have all in common to rely on *leveled* homomorphic encryption.

In a recent work, Boura *et al.* [3] investigated the applicability of fully homomorphic encryption for *classical* deep neural networks. They simulated the effect of noise propagation by adding a noise value drawn from a normal distribution to intermediate values, while evaluating the model in the clear. These experiments were carried out with models making use of the standard ReLU activation function but also with models making use of FHE-friendly variants thereof. Similar experiments were run by replacing max-pooling layers with FHE-friendly average-pooling layers. As a conclusion of their study, the authors of [3] recommend to favor FHE-friendly operations as they appear to be usually more resilient to noise perturbations.

While applicable to certain use-cases, leveled solutions are however inherently limited in the type of tasks they can perform. In particular, in the case of neural networks, they can only accommodate networks with a moderate number of layers. The techniques and results presented in the previous section indicate that depth is no longer necessarily an issue and that deep neural networks can actually be evaluated homomorphically.

In a way similar to iDASH for leveled solutions, we believe it would be useful to have challenges for bootstrapped solutions. The goal would not be to design new operations nor to modify the topology in order to make a given neural network more amenable to an FHE-based implementation. On the contrary, the goal would be to stick to the original neural network model. Doing so presents the tremendous advantage of not requiring to re-train a new model. As the operations and topology are unchanged, the already-trained model can be used as is. The efforts to train a neural network should not be overlooked. This is a costly and time-consuming operation. Furthermore, in many cases, producing a new model is not even possible as this implies having access to the training dataset, which may demand the prior approval of the data owners or of some regulatory authorities.

The iDASH competition shined a spotlight on privacy-preserving technologies and their application to genome analysis. In the same way, we believe that a similar competition with bootstrapped FHE-based challenges would create a positive emulation among the research teams to seek for always better and innovative solutions. Our hope is to make in the long run the homomorphic evaluation of deep neural networks a commodity.

Availability

The library implementing our extended version of TFHE has been developed in Rust. It is available as an open-source project on GitHub at URL https://github.com/zama-ai/concrete.

References

[1] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. doi:10.1515/jmc-2015-0016.

[2] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, and Shafi Goldwasser. Secure large-scale genome-wide association studies using homomorphic encryption. Cryptology ePrint Archive, Report 2020/563, 2020. https://ia.cr/2020/563.

[3] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Simulating homomorphic evaluation of deep learning predictions. In *Cyber Security Cryptography and Machine Learning (CSCML 2019)*, volume 11527 of *Lecture Notes in Computer Science*, pages 212–230. Springer, 2019. doi:10.1007/978-3-030-20951-3_20.

[4] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 483–512. Springer, 2018. doi:10.1007/978-3-319-96878-0_17.

[5] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions Computation Theory*, 6(3):13:1–13:36, 2014. Earlier version in ITCS 2012. doi:10.1145/2633600.

[6] California Consumer Privacy Act (CCPA). https://www.oag.ca.gov/privacy/ccpa.

[7] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020. Earlier versions in ASIACRYPT 2016 and 2017. doi:10.1007/s00145-019-09319-x.

[8] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010. doi:10.1007/978-3-642-13190-5_2.

[9] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *33rd International Conference on Machine Learning (ICML 2016)*, volume 48 of *Proceedings of Machine Learning Research*, pages 201–210. PMLR, 2016. URL: http://proceedings.mlr.press/v48/gilad-bachrach16.html.

[10] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015. doi:10.1007/978-3-662-46800-5_24.

[11] The EU General Data Protection Regulation (GDPR). https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN.

[12] Craig Gentry. Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53(3):97–105, 2010. Earlier version in STOC 2009. doi:10.1145/1666420.1666444.

[13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2013. doi:10.1007/978-3-642-40041-4_5.

[14] Homomorphic encryption standardization. https://homomorphicencryption.org/.

[15] iDASH secure genome analysis competition. http://www.humangenomeprivacy.org.

[16] Miran Kim, Arif Harmanci, Jean-Philippe Bossuat, Sergiu Carpov, Jung Hee Cheon, Ilaria Chillotti, Wonhee Cho, David Froelicher, Nicolas Gama, Mariya Georgieva, Seungwan Hong, Jean-Pierre Hubaux, Duhyeong Kim, Kristin Lauter, Yiping Ma, Lucila Ohno-Machado, Heidi

Sofia, Yongha Son, Yongsoo Song, Juan Troncoso-Pastoriza, and Xiaoqian Jiang. Ultra-fast homomorphic encryption models enable secure outsourcing of genotype imputation. bioXxiv, 2020. doi:10.1101/2020.07.02.183459.

[17] Miran Kim, Yongsoo Song, Baiyu Li, and Daniele Micciancio. Semi-parallel logistic regression for GWAS on encrypted data. Cryptology ePrint Archive, Report 2019/294, 2019. https://ia.cr/2019/294.

[18] Yann LeCun, Corinna Cortez, and Christopher C. J. Burges. The MNIST database of handwritten digits, 1998. Available at http://yann.lecun.com/exdb/mnist/.

[19] Ronald L. Rivest, Len Adleman, and Michael L. Detouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 165–179. Academic Press, 1978. Available at https://people.csail.mit.edu/rivest/pubs.html#RAD78.