Homomorphic encryption and Machine learning

Jainth Chaudhary

Bachelor of Information Technology, Concordia University of Edmonton

Abstract

Basically, I have work on implying Homomorphic encryption on the machine learning. How can machine can learn it and do it on practically. So, on this report is bascially,What is Homomorphic encryption?. how it works? How to use it on python. Homomorphic encryption on the aims at allowing computations on encrypted data without decryption other than that of the final result. This could provide an elegant solution to the issue of privacy preservation in data-based applications, such as those using machine learning.

Keywords: Homomorphic encryption, Machine learning

1 Introduction

Information security is concern for all the world as nowadays breaches becoming more and more common. For one example, in this year a lot of medical data has been leaked and misused. This may concern important issue as everyone wants their privacy and security. For that homomorphic encryption helps a lot. As not everyone sees the data of anyone but just the required thing they want to see. So, I am working on homomorphic encryption method and exploring the new possible ways to enhance the security. Homomorphic Encryption is a special type of encryption though. It allows someone to modify the encrypted information in specific ways without being able to read the information. For example, homomorphic encryption can be performed on numbers such that multiplication and addition can be performed on encrypted values without decrypting them.

Homomorphic Encryption has manifested around Data Privacy. As it turns out, when you homomorphically encrypt data, you can't read it but you still maintain most of the interesting statistical structure. This has allowed people to train models on encrypted data. Furthermore a startup hedge fund called Numer.ai encrypts expensive, proprietary data and allows anyone to attempt to train machine learning models to predict the stock market. Normally they wouldn't be able to do this because it would constitute giving away incredibly expensive information.

Fully homomorphic encryption (FHE) allows evaluation of arbitrary functions on encrypted data, and as such has a myriad of potential applications such as private cloud computing. Gentry [7, 8] was the first to show that FHE is theoretically possible. His construction consisted of three parts: first, construct an encryption scheme that is somewhat homomorphic, i.e. that can evaluate functions of limited complexity (think low degree), secondly, simplify the decryption function of this scheme as much as possible (so called squashing), thirdly, evaluate this simplified decryption function homomorphically to obtain ciphertexts with a fixed inherent noise size (so called bootstrapping).

The first variants [21, 18, 20, 4, 5] of Gentry's scheme all followed the same struc- ture and as such had to make additional security assumptions to enable the squashing step. More recent schemes [9, 4, 3, 2] avoid squashing all together and can bootstrap by evaluating the real decryption circuit. Another advantage of the more recent schemes is that their security is based on the Learning with Errors (LWE) problem [17] or its ring variant RLWE [15], the hardness of which can be related to classical problems on (ideal) lattices.

2 Homomorphic Encryption

In this section, its the introduction of the homomorphic encryption that tells the details of key generation, encryption, decryption, and homomorphic function evaluation.

2.1 Types

Homomorphic Encryption is a relatively new field, with the major landmark being the discovery of the first Fully Homomorphic algorithm by Craig Gentry in 2009. This landmark event created a foothold for many to follow. Most of the excitement around Homomorphic Encryption has been around developing Turing Complete, homomorphically encrypted computers. Thus, the quest for a fully homomorphic scheme seeks to find an algorithm that can efficiently and securely compute the various logic gates required to run arbitrary computation. The general hope is that people would be able to securely offload work to the cloud with no risk that the data being sent could be read by anyone other than the sender. It's a very cool idea, and a lot of progress has been made.

However, there are some drawbacks. In general, most Fully Homomorphic Encryption schemes are incredibly slow relative to normal computers (not yet practical). This has sparked an interesting thread of research to limit the number of operations to be Somewhat homomorphic so that at least some computations could be performed. Less flexible but faster, a common tradeoff in computation.

This is where we want to start looking. In theory, we want a homomorphic encryption scheme that operates on floats (but we'll settle for integers, as we'll see) instead of binary values. Binary values would work, but not only would it require the flexibility of Fully Homomorphic Encryption (costing performance), but we'd have to manage the logic between binary representations and the math operations we want to compute. A less powerful, tailored HE algorithm for floating point operations would be a better fit.

2.2 Homomorphic Encryption in Python

Let's start by covering a bit of the Homomorphic Encryption:
Plaintext: this is your un-encrypted data. It's also called the "message". In our case, this will be a bunch of numbers representing our neural network.
Cyphertext: this is your encrypted data. We'll do math operations on the cyphertext which will change the underlying Plaintext.
Public Key: this is a pseudo-random sequence of numbers that allows anyone to encrypt data. It's ok to share this with people because (in theory) they can only use it for encryption.
Private/Secret Key: this is a pseudo-random sequence of numbers that allows you to decrypt data that was encrypted by the Public Key. You do NOT want to share this with people. Otherwise, they could decrypt your messages.
So, those are the major moving parts. They also correspond to particular variables with names that are pretty standard across different homomorphic encryption techniques. In this paper, they are the following:
S: this is a matrix that represents your Secret/Private Key. You need it to decrypt stuff.
M: This is your public key. You'll use it to encrypt stuff and perform math operations. Some algorithms don't require the public key for all math operations but this one uses it quite extensively.
c: This vector is your encrypted data, your "cyphertext".
x: This corresponds to your message, or your "plaintext". Some papers use the variable "m" instead.
w: This is a single "weighting" scalar variable which we use to re-weight our input message x (make it consistently bigger or smaller). We use this variable to help tune the signal/noise ratio. Making the signal "bigger" makes it less susceptible to noise at any given operation. However, making it too big increases our likelihood of corrupting our data entirely. It's a balance.
E or e: generally refers to random noise. In some cases, this refers to noise added to the data before encrypting it with the public key. This noise is generally what makes the decryption difficult. It's what allows two encryptions of the same message to be different, which is important to make the message hard to crack. Note, this can be a vector or a matrix depending on the algorithm and implementation. In other cases, this can refer to the noise that accumulates over operations. More on that later.
As is convention with many math papers, capital letters correspond to matrices, lowercase letters correspond to vectors, and italic lowercase letters correspond to scalars. Homomorphic Encryption has four kinds of operations that we care about: public/private keypair generation, one-way encryption, decryption, and

the math operations. Let's start with decryption.

The formula on the left describes the general relationship between our secret key S and our message x. The formula on the right shows how we can use our secret key to decrypt our message. Notice that "e" is gone? Basically, the general philosophy of Homomorphic Encryption techniques is to introduce just enough noise that the original message is hard to get back without the secret key, but a small enough amount of noise that it amounts to a rounding error when you DO have the secret key. The brackets on the top and bottom represent "round to the nearest integer". Other Homomorphic Encryption algorithms round to various amounts. Modulus operators are nearly ubiquitous. Encryption, then, is about generating a c so that this relationship holds true. If S is a random matrix, then c will be hard to decrypt. The simpler, non-symmetric way of generating an encryption key is to just find the inverse of the secret key.

Several open-source HE libraries have emerged in recent years, each one with different properties based on the employed encryption scheme [35]. Microsoft's Simple Encrypted Arithmetic Library (SEAL) [36] scheme, with support for the Brakerski/Fan-Vercauteren (BFV) [37] and the Cheon-Kim-Kim-Song (CKKS) scheme [38], and IBM's HELib [39] based on the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [40] are two of the most widely used HE libraries. The first noticeable shortcoming of HELib is the lack of support for floating-point numbers. To allow for computations to be performed on rational numbers, SEAL takes advantage of a particular property of the CKKS scheme: rescaling can be performed without changing the encrypted value. Since a plaintext is represented as a polynomial with integer coefficients, floating-point parameters of the message are scaled by a parameter that affects the precision of the computations. Homomorphic operations performed with both HELib and SEAL introduce noise, thus limiting the number of operations that can be performed with ciphertexts. Noise-management techniques have been integrated to maintain the noise level below a certain threshold, such that the ciphertext does not become corrupted. While HELib uses the expensive procedure of bootstrapping to enable unlimited computations, SEAL uses a scale-invariant error reduction technique which requires an estimation of the number of operations that will be performed as an a priori information. Moreover, there are some limitations on the types of operations that can be performed on the ciphertext. The schemes used in HELib and SEAL are fully homomorphic with respect to addition and multiplication, and only polynomial functions can be easily performed. As a consequence, there is no implicit support for division, and nonlinear functions have to be approximated by low-degree polynomials.

While these schemes are known for their efficiency in terms of proven security, the above-mentioned restrictions, alongside their computational overhead, introduce noticeable constraints in the neural network topology, which in turn affect the performance of privacy-preserving neural networks [41].

Other proposed methodologies rely on employing partially homomorphic encryption (PHE) instead of FHE. Since FHE is currently practically impossible to be used in a real-world system, a viable approach is a system based on PHE that is specialized only for certain operations. Such an approach introduces a clear advantage in terms of running time and may be used in a practical application with reasonable overhead [42]. Various encryption schemes have homomorphic properties, out of which we mention the Paillier scheme [43], an additive homomorphic scheme where addition in the ciphertext space corresponds to multiplication in the plaintext space, and the ElGamal scheme [44], a multiplicative homomorphic scheme, which, through some modifications, can become additive. Other PHE with the potential to be used in a practical application are Goldwasser-Micaly [45] that allows computing the XOR operation on encrypted data, searchable encryption [46] with support for keyword search, order-preserving encryption [47] for sorting encrypted values, and deterministic encryption [43], that allows equality checks on encrypted values.

Another promising method is the algebra homomorphic encryption scheme (AHEE) [48], an encryption scheme that is homomorphic with respect to algebraic addition and multiplication, i.e., both addition and multiplications can be performed on encrypted data. A key advantage of this scheme is that it has a relatively small computational complexity, same as Paillier and ElGamal, while being homomorphic with both addition and multiplication. The main limitation of this scheme as well as for Paillier and ElGamal is that it only allows the encryption of relatively small integer numbers. More specifically, during the encryption process, an exponentiation operation needs to be evaluated, where the exponent is the message to be encrypted. Hence, even with a multiprecision arithmetic library, the operation can still cause an overflow. Using 1024 bit integers, we found that only numbers with up to about can be encrypted. This limitation becomes even more important when performing computations on encrypted data, i.e., one cannot determine if an encrypted number is too large for performing a certain operation.

In order to facilitate the privacy-preserving deep learning-based analysis, the cryptosystem must allow the computations within the model to be performed on rational numbers. To address this requirement, an encryption mechanism is typically used to encode a given rational number as a sequence of integers [49]. As some of the basic operations are difficult, if not impossible, to apply on encoded data, such an approach has limited functionality when applied on real data. Furthermore, the encoding strategy not only explicitly limits the data utility but also directly affects the results of the computations.

Over the past few years, many HE schemes have been proven to meet the security requirements. Although sufficiently secure, most of these approaches offer poor performance as they suffer greatly from runtime bloat, i.e., several orders of magnitude slower than the plaintext computations. This clearly restrains their usability in real-world applications. Consequently, simplified encryption schemes based on linear transformations emerged in the field as more practical alternatives. Despite the criticism for weaker security [50, 51], this type of cryptosystems seems to be the currently only feasible method with the potential to enable privacy-preserving computations in real-world applications.

As a consequence, the herein employed methodology relies on a variant of the matrix-based homomorphic encryption scheme proposed in 2012 by Kipnis and Hibshoosh [4]. In contrast with the currently adopted schemes in privacy-preserving neural network-based solutions [25, 26, 29], the MORE (Matrix Operation for Randomization or Encryption) encryption scheme is noise free and nondeterministic (multiple encryptions of the same plaintext data, with the same key, result in different ciphertexts). An unlimited number of operations can therefore be performed on ciphertext data. Moreover, the MORE scheme enables all four basic arithmetic operations over encrypted data. Herein, MORE was redesigned to directly support floating-point arithmetic in order to address the floating-point precision constraint of privacy-preserving deep learning-based analysis on real-world data.

3 Conclusion

With advances in machine learning and cloud computing the enormous value of data for commerce, society, and people's personal lives is becoming more and more evident. In order to realize this value it will be crucial to make data available for analysis while at the same time protect it from unwanted access.This report is basically the about the Homomorphic Encryption. Homomorphic encryption is a rapidly advancing field and so expect that more complex ML algorithms applied to larger data sets requiring fewer computational resources may soon be possible. For example, it should soon be possible to use kernel methods to derive low-degree polynomial machine learning algorithms implementing non-linear mappings. Other open problems include the question, which protocols will be useful in practical data analysis scenarios, and how the computational burden can be optimally distributed between cloud and client taking into account the cost of communication.