Homomorphic Encryption of Neural Networks

Piyush Vishal Sharma

1700154C202, 1700178C202

CSC 2017 Batch

# Abstract

The modern world is moving towards an intelligent future where AI and ML will be playing an important role in improving the technologies across virtually all domains. The most basic requirement for training an ML model is data. Without the availability of data there is no use of AI or ML. But there are some hindrances to obtaining the data. Sometimes the data is freely and openly available, but there are some case where the privacy of the data is much more important that creating an ML model out of it.

Such scenarios can prove to be a hurdle in developing new technologies. So we suggest a way which can be used to train ML models on such private data while maintaining privacy of both the data as well as the ML model, using homomorphic encryption. We used a somewhat homomorphic encryption system proposed by Zhou and Wornell [1].

# Introduction

In today’s world, cryptography plays an important role in our lives. Virtually every communication of ours over the internet is encrypted. This all started with the development of symmetric encryptions where the two communicating parties, say Alice and Bob used a secret key to encrypt and decrypt their communications. The key was meant to be kept as a secret by Alice and Bob so that an adversary, say Eve, is not able to eavesdrop into their conversation. The main problem arose, when it was the time to exchange these secret keys over the internet. There was no method to securely transfer it over internet. This is when public key encryption came into the picture. It allowed two people to exchange secrets over the internet without using a key which need to be shared by both of them. The key which Alice and Bob were using for encryption could now be shared using the asymmetric encryption techniques. But then the researchers started to think of a technique which not only would encrypt the data to protect it from eavesdroppers but would also allow the user to keep it hidden from the person/resource which is providing the user some services using his/her data. So, researchers started finding encryption techniques which would allow doing computation on the encrypted data as if the computations were being performed on plain data. This would solve the issue of sharing private, personal and sensitive data with third parties for the services provided by them. This is called homomorphic encryption technique.

Homomorphic encryption basically enables us to perform computations over cipher texts. And when we decrypt it, we get the plain text which would reflect all those operations performed on the encrypted text. There are different kinds of homomorphic encryptions techniques like, partially homomorphic, which basically either allows only one kind of operation being performed or limits the number of times an operation can be performed; the second one is fully homomorphic encryption, which allows a number of operations any number of times.

We used a somewhat homomorphic encryption system proposed by Zhou and Wornell [1]. We selected this encryption system because it is very efficient compared to other techniques. And efficiency is a major bottleneck for using homomorphic encryption on Neural networks.

# Homomorphism

Homomorphism may be defined according to two operations, i.e. addition and multiplication. Below are the equations representing the properties of homomorphic encryption:

*“Homomorphic encryption is additive if :*

*Enc(x + y) = Enc(x) + Enc(y)*

*Homomorphic encryption is multiplicative if :*

*Enc(x\*y) = Enc(x) \* Enc(y) “*

A homomorphic encryption system is made up of some building blocks which are presented below:

1. **Key generation:** This block outputs a public key and a secret key to be used by the user.
2. **Encryption algorithm:** This block takes the message and the public key as the input and outputs the cipher text.
3. **Decryption algorithm:** This block takes the secret key and the cipher text as the input and give the message as the output.
4. **Evaluation function:** This block takes input as evaluation key, circuit (realized using logic gates), and a group of cipher texts and outputs a single cipher text which would represent the result of the calculations performed on the cipher texts. [2]

## Partially Homomorphic Systems

Partially homomorphic systems have some limitations to the number and type of operations which can be performed on the cipher texts. Some techniques support only addition, some support only multiplication, while some support both but one of the operations can be performed for a limited number of times. Based on these properties, these systems can be divided into the following categories:

### Additive Homomorphic Systems

These systems allow any number of additions on the cipher texts. Some of the techniques include Goldwasser-Micali system [3], Pallier system [4]. It allows any number of addition operations over the cipher texts. The cipher in this scheme is calculated according to the following equations:

*where ,*

*C = cipher text  
 g = number calculated from cryptographic techniques  
 r = random number  
 m = message / plain text*

As we can see in the above scheme that if the cipher texts are multiplied and the resultant cipher text is decrypted, we will get the addition of the plain texts that were encrypted.

### Multiplicative Homomorphic Systems

These systems calculate the cipher texts in such a way that allows the performing operations on cipher text in such a way that it reflects multiplication on plain text. Some techniques which allow such operations include RSA algorithm [5], ElGamal encryption scheme [6]. Equations which support the above claims are listed below:

*where,   
 C = cipher text  
 m = message*

*e = number generated by cryptographic methods*

As we can see from the above equations, we can perform certain operations on the cipher text so as to produce a cipher text which when decrypted would replicate the multiplication of the plain texts.

### Additive and Multiplicative Homomorphic Encryption Systems

All the previously discussed techniques allowed only one operation, addition or multiplication, to be reflected on plain text. But there are some other techniques as well which allow one of those operations any number of times and other operation a limited number of times. One of such techniques is Boneh- Goh-Nissim cryptosystem [7]. This technique allows any number of additions but allows “at most one multiplication”.

## Somewhat Homomorphic Encryption

The way from partially homomorphic systems to fully homomorphic systems goes through somewhat homomorphic systems. These systems like all other have encrypt and decrypt modules. But they also have an additional module called *evaluate* which basically is used to perform operations on the cipher texts. While encrypting the plain text, a noise is added to it which is removed while decrypting. Addition usually doubles this noise and multiplication squares the noise. If the noise rises above a certain level, the cipher text cannot be decrypted. So we cannot perform operations beyond that. Because of this these systems are called *somewhat homomorphic encryption systems.* But his can be overcome by using *bootstrapping.* In this process, evaluate function, which normally works on encrypted texts, is made to run decrypt module because of which we get other cipher text but with much lesser amount of noise. So virtually because of bootstrapping, we can perform unlimited number of operations. This leads the way to fully homomorphic encryption. The equations supporting the concept of this technique are listed below:

From the above equations we can clearly see the effect of addition and multiplication on the rise in amount of noise. Also they show how somewhat homomorphic encryption can be used for computations involving both additions and multiplications.

## Fully Homomorphic Encryption Systems

Scheme from Brakerski and Vaikuntanathan is known as “Fully Homomorphic Encryption without Bootstrapping" where their approach is based on “Learning with Errors (LWE) and Ring Learning with Errors (RLWE) problems “ [8]. This technique is proposed over a ring which makes it to support both addition and multiplication as operators over the cipher text. The algorithm doesn’t need any bootstrapping, because it is optimized in such a way that any operation performed on the cipher text affects the noise only linearly and that too to a very small amount. So the noise remains below the threshold and the final cipher text can be decrypted without any problems.

# Encryption Scheme

This encryption scheme was proposed by Zhou and Wornell [1]. Before introducing to the scheme, we will explain all the variables and notations which are used in the algorithm.

Variables which are used:

* K: This is a matrix which represents Private Key. This will be used decryption.
* M: This is a matrix which represents Public key. This will only be used for encryption.
* c: This is a vector representing encrypted data.
* p: This is vector representing message or plaintext.
* w: This scaler vector represents weight.
* e: This represents added Noise.
* row and col are size of matrix (Number of rows and Number of columns respectively).

For a Private Key matrix of row x col K, a plain text or message p, error term e, weight w and Ciphertext c satisfies:

For Decryption:

Here, represents nearest integer or rounded integer value.

## Key Switching

Using key switching technique one can change private key of the message without changing the original plaintext.

We compute a new private key K’ such that:

It is a four-step process.

1. First we convert K’ to an intermediate private key K\* and represent c in its binary form and such that new ciphertext |c\*|= 1. We also ensure that:
2. Now the intermediate private key and ciphertext are converted to desired key K’. For this we construct a switch key matrix M such that:

Here, e is random noise matrix.

1. Now we represent and calculate M:
2. Now we solve c’ such that:

## Linear Transformation

If we want to calculate transformation of encrypted text c by R then

Now, we can use key switching technique discussed above to compute switch matrix M and thus c’.

# Proposed Algorithm

We propose a homomorphic encrypted Neural Network. A neural network has many different types of operations and for some operations we need to need to apply some tricks to work with homographically encrypted data.

* Since neural network involves gradient descent and weights are updated by a very small value which is represent in decimals and the homographic encryption technique which we are using rounds the original message to integer, so we scale decimals into integers. For example, we can scale up every number by 10000 for precision up to 4 decimal digits. But scaling can sometimes result in overflow of the variable storing the number, so we need to choose scaling factor appropriately and uniquely for each number.
* We can perform Vector multiplication by using key switch technique which we discuss above.
* Dot product operations can be computed using Linear Transformation technique, which is proposed by Yu, Lai and Paylor [9].
* For a neural network we need a non-linear activation function. We will be using Sigmoid function:

Since this homographic encryption technique does not supports exponential function so we propose to use Taylor series to do polynomial operations and thus we can calculate sigmoid function approximately.

Above, it is Taylor series expansion of .

* Other basic operations such as multiplication and addition are well supported by this homographic encryption technique.

# Software used

We used Python to implement the algorithm and used NumPy for matrix operations.

# Applications

As homomorphic encryption systems become more efficient, its applications will cover very diverse industries. With homomorphic encryption people won’t need to trust 3rd parties with their data. Everything is secure and private fully end-to-end. The User uploads the encrypted data on the cloud server and then 3rd party performs computations over the cipher text and the result, which is also encrypted, can only be decrypted by the user with his key.

## Medical Records

With advancements in machine learning and AI, we are able to predict and diagnose diseases with very fast speed and even better than human experts. CheXNet is convolution neural network developed by researchers at Stanford which has achieved better results at predicting pneumonia from X-Ray than expert radiologists [10]. This solution can be easily deployed on cloud servers for usages of doctors around the world. But people are very conscious of sharing their medical data with 3rd parties. If medical records of a person are leaked to his employers, then he may be subject to discrimination and maybe even fired from his job. So, there's a need to protect data as well as results.

## Genome Analysis

Today with advancements in DNA analysis, we are able to recognize disease risk and create personalized medicines for people. Soon people will be able to share their DNA for analysis and get results about their health profile and disease risks [11]. But to achieve this requires people to share DNA sequences of people. DNA sequences are identity markers which can't be changed. If DNA of a person is leaked to financial organizations then it may be possible that they reject loan applications from the person just because his DNA sequence suggests that he is at higher risk. So, protecting privacy of DNA sequences is very important for success of DNA Analysis industry. With the use of homomorphic encryption, the privacy of the person and his results can be protected.

## Finance and Advertising

Nowadays, banks keep records of their customers to create a risk profile using Machine Learning. But theirs is threat of data being leaked when it is in decrypted form while being prepared for computations. There are web applications which calculate can our credit score, but people don’t trust the scoring company for such private information. Now, it’s possible that a web application is able to file our tax returns without decrypting our financial data and ensuring privacy.

Recently, there was Facebook-Cambridge Analytica Scandal where Facebook said that [12] they trusted Cambridge Analytica for using data of users only for Advertising, but Cambridge Analytica broke their trust and used data to manipulate voters in US presidential elections. If Facebook had used Homographic Encryption on its data, then the privacy of the users could have been saved.

# Additions Made

The following are the additions made by us to the already proposed algorithm to increase the functionality of the algorithm and to increase the domain of its applicability.

## Neural Network support

The current paper mentions the implementation of the HE technique in some ML algorithms like Support Vector Machines, Naïve Bayes, Linear Regression, but it does not discuss the use in Neural Networks.

## Approximation of Nonlinear function

As this technique of Homomorphic Encryption deals with only integers, implementing a nonlinear function directly as the activation function in a neural network was not possible. To solve this problem, we made use of Taylor series for the activation function and used it to get the approximation of the nonlinear activation function which was very close to the real value and made the prediction more accurate.

## Support for Floating point numbers

The nonlinear activation function would give a result in floating points. The current technique only uses integers. Converting the floating points to numbers would result in drastically reduced accuracy. So, we needed to add the support for floating points while keeping the calculations still in integers. We did that by scaling up all the calculations to a pre-decided power of 10. The results at the end would be scaled down again. This helps us to have those decimals also contributing to the calculations in between.

## Encrypting Neural Network weights

There might be some sources which would not be willing to share even the homomorphically encrypted data with us for training of the model. For this we came up with a solution in which we encrypt our NN and send it to the source for training. The network gets trained over the data and is decrypted once it comes back to us. This way, privacy of the data of the source as well as the intelligence of our NN is preserved.

# Future Works

## Bootstrapping

The calculation of the cipher text in this technique includes some random noise. The operations like multiplication, addition increase this amount of noise. If the value of the noise exceeds the value of the message, the cipher will not be decryptable anymore. So as the amount of noise increases, we need to perform bootstrapping which reduces the amount of noise while keeping the message intact. But this process will take a lot of time and the neural network will reduce drastically. So an efficient method of bootstrapping must be found to implement in the present model.

# Applicable Scenarios

## Scenario 1

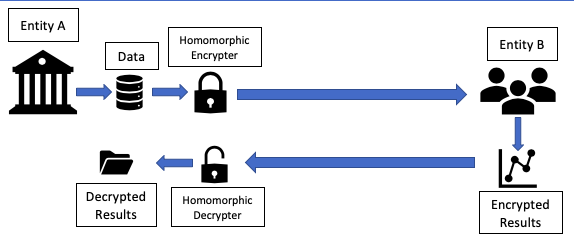
In this scenario, an entity A is in possession of a highly sensitive data. This data might be medical records of patients or data pertaining to some other fields. Another entity wants to automate some process which would require training of a model on the data which entity A possesses. But since the data is highly sensitive, entity A would not be willing to share the data. But using homomorphic encrypter, Entity A can send the data to entity B which can train its model on the given data since homomorphic encryption allows arithmetic operations over encrypted data.

A picture containing screenshot, drawing

Description automatically generated

## Scenario 2

This is the scenario where an entity wants to use the service of another entity by fetching the results/predictions for its data using a pre-trained model in possession of entity B. But the data is private to entity A and is not willing to share the data with anyone else. In this case too, entity A can homomorphically encrypt the data and sent it to the entity in possession of the trained model. The entity B will not be able to see the data but will be able to perform arithmetic operations over it since it is homomorphically encrypted. The results obtained too will be encrypted, which will be sent back to entity A. Entity A can decrypt the predictions and use them. This way both the data and the results were hidden from entity B.



## Scenario 3

This is the scenario where an entity in possession of an ML model wants to train its model on some data in possession of entity B who is unwilling to share even the unencrypted data with anyone else. In this case, entity A can encrypt the weights of the network and send the model to entity B which trains the encrypted model over the data he is in possession of. The weights are hidden from entity B. Training over the data will improve the weights of the model. The encrypted model will be sent back to entity A which can decrypt the network and use the model somewhere else. This way both the privacy of the data and the intelligence of the network are maintained.

A screenshot of a cell phone

Description automatically generated

# Result and Analysis

To compare the proposed neural network including encryption with an unencrypted neural network, we plotted their loss vs time graph. The graph of unencrypted neural network converges very early (1000x) when compared with the encrypted neural network because of the obvious delay in calculation in encrypted numbers. The graphs are given below:

A screenshot of a cell phone

Description automatically generated

*Fig 1. Loss vs Time graph for normal Neural network*

A screenshot of a cell phone

Description automatically generated

*Fig2. Loss vs Time graph for Encrypted Neural Network*

Also it was noted that the loss for the encrypted neural network shot up unexpectedly during training when it reached a value below a point. This is because since the loss calculated originally was encrypted which had to be decrypted for the purpose of comparison. But since this technique does not use bootstrapping, when the value of the loss goes below a certain point, the error value overshadows the original value of the loss and decryption becomes impossible and hence we see the unexpected rise beyond a certain point.

A screenshot of a cell phone

Description automatically generated

*Fig 3. Loss vs Time where loss shoots up unexpectedly*

We also made an network which would train itself on unencrypted weights and inputs. It will be able to make predictions on a given encrypted data as input. This type of model will be used in a scenario where an entity wants to get predictions on its data without revealing the data to any second entity.

A screenshot of a cell phone

Description automatically generated

*Fig 4. Training network on unencrypted inputs using linear activation function*

As it is clearly visible from the graphs above that training a model over encrypted inputs and encrypted weights takes a lot of time because we are dealing with encrypted data all along. Also because of lack of bootstrapping, the model’s loss was not at its lowest point which means the accuracy of the model would not be great. But the second scenario is practically applicable since the model can be trained over unencrypted data, and then can be used to predict results over encrypted data.

Also use of non-linear functions is necessary since the real world data is multi-dimensional and complex and is often not linearly separable.

# Conclusion

Homomorphic Encryption was first theoretically introduced in 1978 [13]. But till 2009 there was no practical implementation. When Gentry proposed first Fully Homomorphic Encryption, he solved decades long problem. After this breakthrough, research in Homomorphic encryptions has exploded as it has capability to revolutionize security of cloud services and also protecting data privacy of cloud users. In this paper we tried to implement a neural network on homomorphic encryption. We used somewhat homomorphic encryption technique and this technique is not scalable. So, further work and more research work is required in Homomorphic Encryption systems so that they can be applied in Machine Learning systems.

# Bibliography

|  |  |
| --- | --- |
| [1] | H. Zhou and G. Wornell, “Efficient Homomorphic Encryption on IntegerVectors and Its Applications,” IEEE, 2014. |
| [2] | D. H. a. S. Picek, “Homomorphic Encryption in the Cloud,” ieee, Zagreb, 2014. |
| [3] | C. Gentry, “A fully homomorphic encryption scheme,” Stanford, 2009. |
| [4] | P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” Springer, Berlin, 1999. |
| [5] | L. A. a. M. L. D. R. L. Rivest, “On data banks and privacy homomorphisms,” Academia Press, 1978. |
| [6] | T. E. Gamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” Springer, New York, 1985. |
| [7] | E.-J. G. a. K. N. D. Boneh, “Evaluating 2-dnf formulas on ciphertexts,” Springer, Berlin, 2005. |
| [8] | Z. B. a. V. Vaikuntanathan, “Efficient fully homomorphic encryption from (standard) lwe,” FOCS, 2011. |
| [9] | A. Yu, L. L. Wai and J. Payor, “Efficient Integer VectorHomomorphic Encryption,” MIT, 2015. |
| [10] | P. Rajpurkar, J. Irvin, K. Zhu, B. Yang and H. Mehta, “CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning,” arXiv preprint arXiv:1711.05225, 2017. |
| [11] | S. Farah, M. S. Sushma, T. Asha, B. Cauvery and K. S. Shivanand, “DNA Based Disease Prediction Using Pathway Analysis,” 2017 IEEE 7th International Advance Computing Conference (IACC), 2017. |
| [12] | “The Guardian,” 2018. [Online]. Available: https://www.theguardian.com/technology/2018/mar/21/mark-zuckerberg-response-facebook-cambridge-analytica. |
| [13] | R. L. Rivest, A. Shamir and L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” Association for Computing Machinery, 1978. |