Forecasting the future by (not) looking at the past: time-series forecasting with Homomorphic Encryption

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Andrea Murino, 10713305

Advisor:
Prof. Manuel Roveri

Co-advisors:
Alessandro Falcetta

Academic year:
2020-2021

Abstract: The use of Artificial Intelligence and Machine Learning algorithms becomes more and more present every day, with applications ranging from *smart home* devices to *digital health care*. Nonetheless, the increasing use of such algorithms - which need more and more data in order to work - raises an alarm regarding the issues of data privacy and user protection.
The aim of this thesis is to present an architecture for private time-series forecasting, i.e. able to provide forecasts on time-series sent by users meanwhile guaranteeing the privacy of users' data. In order to achieve this goal, Homomorphic Encryption will be used, which will allow to operate directly on encrypted data without having access to plaintext data.

Key-words: Deep Learning, Privacy preserving, Time-Series-Forecasting, Homomorphic Encryption

1. Introduction

1.1. Problem and motivation

Machine learning (ML) [13] is a branch in evolution of computational algorithms that are designed to simulate the human intelligence and to learn from the surrounding environment. Techniques based on ML have been applied successfully in many fields such as pattern recognition, computer vision[34], finance[11] and computational biology to biomedical and medical application [25, 28]. These algorithms showed how was it possible to learn from the current context. Considering patients with cancer that are involved in radiotherapy which requires a large set of processes, often the degrees of the complexity of this processes may involve several stages of difficult human-machine interactions[32], which would invite the use of machine learning algorithms to optimize and automate these processes. We can think about contouring and treatment planning[28], image-guided radiotherapy[27] and so on.
These algorithms work regardless the nature of the domain, generally the more the data fed to these algorithms, the higher the effectiveness.
Such techniques arouse the interest in many industries and have quickly become a powerful tool to solve issues and making problems solvable where these were not until that moment.
In the past decade the paradigm of *cloud computing* [31] has emerged overwhelmingly. With cloud computing, instead of setting up their own physical infrastructure, users could rely on a large-scale network of machines managed by a provider. The users have to pay only for the services they had used and the workload can be

shifted on the cloud benefiting in terms of availability, maintainability and scalability.
There are three main services provided by cloud computing that are Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) but recently a new one has been introduced, namely *machine learning as a service* (MLaaS). The main difference with respect to the three kind of services stated above is substantially that what is offered this time are ready-to-use remotely machine learning services.
The main reasons that contributed to the success of this modality are in first place that usually ML solutions are complex and require an high computational power, often not available for users on their personal machines, indeed they could be severally time and space consuming; secondly, it is likely that users are looking for something already known and available at the state of art (e.g. face recognition, speech analysis, conversion text-to-speech)

1.1.1. Privacy preserving

Cloud computing modalities, included MLaaS paradigm, can expose users to risks related to the nature of data involved.
To be effective, these algorithms need to be fed with data that may even be sensitive for whose sharing them, imagine in fact data such as personal photos, hearthbeats or medical information, companies information about sales and revenues.
From the many definitions of privacy, Lane et al. [19] suggest that privacy is respected only when the information is treated and confined accordingly with some defined boundaries. The consequences in violating these terms depend on the magnitude of the violation.
Considering then the case of MLaaS, such sensitive data could get into the wrong hands and used for social engineering attacks. Although the usefulness of machine learning is undeniable [35], the same training data that has made it so successful also presents at the same time serious privacy problems. Centralised collection of photos, speeches and videos from millions of individuals are fraught with privacy risks.
Firstly, the companies that collect this data keep it forever; users from whom data was collected can neither delete it, nor control how it will be used, nor influence what will be learned. Secondly, images and voice recordings often contain sensitive data acquired accidentally by smart objects: faces, number plates, computer screens, the sound of other people talking and environmental noise, etc. Furthermore, the big Internet companies monopoly on "big data" collected from millions of users leads to their monopoly on the AI models learned from this data. Users clearly benefit from new services, such as powerful image search, voice-activated personal assistants, and machine translation of web pages in foreign languages, but the models constructed from their collective data remain proprietary of the companies that created them.

1.2. Goal and Results

This work is aimed to present a solution that face the problem of privacy in the time-series forecasting topic. Accordingly to the definition, *time series* [38] is a series of data points indexed in time order. Typically this sequence is split at equally spaced points in time. Some examples of time series are the number of daily Covid-19 cases in the last month, the average daily temperature in Toronto during the last year or the monthly revenue of a company in the last decade. The word forecasting means to predict new data based on the past and the present and most commonly by analyzing trends. It might refer to formal statistic methods employing time-series, as we mentioned above, cross-section or longitudinal data or less judgemental methods.
The usage can really differ between areas of application, for example in supply chain management it can be used to ensure that the right product is at the right place at the right time, but even for earthquake prediction, energy forecasting, sales forecasting, weather forecasting and many others.
Among all the possible forecasting models, there are also *machine learning* ones. Using part of data coming from the past we can train a Machine Learning model and then use it to make new predictions.
Unfortunately, in real case scenarios, if we want to keep the concept of privacy preserving for real, it's possible that one won't have any data from the past. Usually a company would like to protect its own data without sharing such information.
The solution is still possible if it is considered to train the model with publicly accessible data, not protected by any kind of privacy agreement. Another solution could be to use data provided by the user that are either not covered by confidentiality or do not present serious privacy problems. While from the literature is possible to find several algorithms to make forecasting, this work will mainly focus on a Machine learning model. In particular the proposed solution for this thesis consists into applying a a special type of Encryption, i.e. *Homomorphic Encryption* (HE) to CNN's, a famous model type of ML, for the time series forecasting.
Time series forecasting [40] is an important area of forecasting in which past observations of a variable are collected to develop a model describing the underlying relationship. The model is finally used to extrapolate

the time series into the future. This modeling approach is particularly useful when there's not so much knowledge about the underlying data generating process or when there is no satisfactory explanatory model that relates the prediction variable to other explanatory variables.
Thanks to the HE, the user can encrypt the data on his own machine (e.g. a personal computer or a mobile device) through a public key, then the encrypted data could be sent to a Cloud-based special MLaaS, amended to work on encrypted data.
When the computation is finished, the encrypted result (a forecast in this case) is returned back to the user who can decrypt and make it available.
This process involves that plain data never leaves the user device keeping the privacy and confidentiality; the MLaaS provider will not be able to access the plain data and it will work only on the encrypted version. This is possible because of the service offered by the HE, it allows to operate directly on ciphertext, i.e. encrypted data, without the need to have the data in plain text or its private key to decrypt it. All the advantages of the cloud computing are kept.
Unfortunately there are no free lunches, working on encrypted data comes at the cost of some issues:

- **Higher computational workload** and this justify the needing of a Cloud based solution;
- The majority of the HE schemes, included the one treated in this work, allows **only a limited set of operations**, i.e. additions and multiplications, both in terms of type of operations but also in terms of the number of operations actually feasible. Hence we need to approximate models in order to not introduce non-linearity;
- Homomorphic Encryption requires an **analysis in terms of chosen parameters** because they will determine the correctness, the computational load and the precision of the HE operations.

However, speaking of CNN, additions and multiplications are enough to get good results. Clearly the model used need to take into consideration these aspects and has to be properly reshaped, redesigned and retrained.
This work includes an implementation of a proposed architecture. The implementation, developed in Python code, has been used to realize an experimental campaign that consists of test suites on CNNs models on plain data to verify the forecasting performance compared to others well known algorithms, and from another side, such models are applied on encrypted data to verify the correct functioning of the solution.

1.3. Thesis Structure

Section 2 introduces the main technical tools used in this thesis, i.e., Homomorphic Encryption, Machine Learning, and Cloud Computing. In particular, this Section focuses on Convolutional Neural Networks and its time-series tailored version.
Section 3 is about the related literature, in this section some existing privacy-preserving solutions are presented.
Section 4 is the core of the work, here the proposed architecture is shown in detail.
Section 5 explores TS-PyCrCNN, the library written in Python that implements the proposed architecture.
Section 6 includes all the experiments carried out and a comparison with others Deep Learning algorithms is discussed.
Finally in Section 7 conclusions are drawn.

2. Background

2.1. Cloud Computing

One of the accepted definition of cloud computing is the following [26]:

Definition 2.1. ***Cloud computing*** *Cloud Computing is a model for enabling convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly released with minimal management effort or service provider interaction.*

Usually cloud is composed of several machines with an important total amount of network bandwidth, computational power and storage. The peculiarity of the Cloud paradigm is in the way these resources can be accessed, indeed clients can see the cloud as a potentially infinite resources pool, and use only the amount they need, usually with a *"pay-as-you-go"* payment model.

Cloud service paradigms

The National Institute of Standards and Technology (NIST) indicates three fundamentals services paradigms offered by a cloud:

- **Software as a Service (SaaS):** software solutions are made available to the customers; This applications are installed on the Cloud provider's machines and it's responsibility of the provider to install the software, keep it updated and guarantee its correct functioning.
  Examples of SaaS are online office suites, accessed through browsers by costumers, or online games.
- **Platform as a Service (PaaS):** In this case the running application is not managed by the cloud provider, the customer has the freedom to choose which application deploy on the cloud infrastructure using services like execution stacks, compilers, database or, in general, middleware.
  Clearly the platform is set to give specific boundaries on permitted operations, customers can not access the operating system or the hardware configuration.
  Provider responsibility is simply to guarantee the correct functioning of the underlying stack and the allocation of the adequate amount of resources.
- **Infrastructure as a Service (IaaS):** In this scenario, costumers have access to every function of their systems, which includes the possibility to change OS, access network settings and plain storage space.
  The Cloud provider has to run the minimal software needed to make the computing resources available to consumers, including abstractions like virtual machines and virtual network interfaces. The physical access to hardware remains responsibility of the Cloud provider
- **Machine Learning as a Service (MLaaS):** This paradigm does not have an official definition, but it has been recognised by dominant cloud companies in the world, such as Google, Amazon and Microsoft. It's a particular case of PaaS but in this case cloud providers make available platforms already optimized for the deployment of machine learning algorithms because of the huge amount of resources needed.

Pros and Cons of Cloud solutions

One of the most advantageous aspect for which people invest in cloud solutions is that they significantly reduce the entry level barriers in terms of cost and infrastructure especially for small firms and start-up companies.
Resources are highly scalable and flexible depending on customers requirements, this lead to lower costs and results in a higher efficiency.
Last but not the least, cloud solutions are characterized by an high level of *dependability* namely they includes a variety of characteristics like resistance to failures, guarantees on the correctness of provided services and so on. On the other side, two main drawbacks can be spotted, firstly the vendor lock-in issue, in which a customer is tied to a specific cloud provider with a poor support in terms of portability, either for legal constraints or simply because of technical incompatibilities.
The second one is the **privacy**, discussed in the first section of the paper.

2.2. Machine Learning

Accordingly to [13], the term Machine Learning (ML) indicates a subset of artificial intelligence. In particular, ML is the study of computer algorithms which are able to improve automatically in solving a specific task: they reflect the behaviour of human beings who are able to improve their performance with experience.
Another definition is the following:

Definition 2.2. *A program learns from a certain experience E, with respect to a class of tasks T, obtaining a performance P, if its performance in solving tasks of type T, measured by performance P, improves with the experience E.*

Techniques based on ML have been applied in many fields such as pattern recognition, computer vision, spacecraft engineering, finance, entertainment and computational biology to biomedical and medical application. This algorithms showed how is possible to learn from the current context. We can mainly distinguish three different paradigms used to build models described in the next part of this section.

Supervised Learning

Supervised learning [9] represents much of the research activity in machine learning, and many supervised learning techniques have found application in multimedia and content processing. The distinguishing feature of supervised learning is when annotated training data is available. The name recalls the idea of a "supervisor" who instructs the learning system on the labels to associate with training examples. Typically, these labels are class labels in classification problems. Supervised learning algorithms infer patterns from these training data and these patterns can be used to classify other unlabelled data.

Unsupervised Learning

Differently from the supervised case, here inputs are not labeled beforehand. Typical problem of unsupervised learning is clustering: trying to look for common patterns in inputs and group them accordingly to these common traits.

Reinforcement Learning

RL is situated in between supervised learning and unsupervised learning, this paradigm is about learning in sequential decision making problems in which there is limited feedback. It is substantially based on a reward system with which it makes the program optimal in doing a complex task.
Basically each move gives a feedback, either positive or negative in terms of a reward, and the goal of the algorithm is to maximize the total reward.

2.2.1. Deep Learning

As [20] states, deep learning allows, in an automatic fashion, learning multiple levels of representations of the underlying distribution of the data to be modeled. Deep Learning (DL) is one of the most important types of ML algorithms. DL models are composed of various **layers**, placed one after the other. Input data flows through every layer, being modified by each of them. This lets the model identify patterns in input data in a progressive way, making it possible to obtain much more precise mappings.
Among all the possible DL models, this work will focus in particular on Convolutional Neural Networks.

2.3. Convolutional Neural Network

CNNs are a type of Artificial Neural Networks (ANN) [1] with multi layers. During the last years, it has been considered to be one of the most powerful tools, and has become very popular as it is able to handle a huge amount of data. Convolutional Neural Network (CNN) take this name from mathematical linear operation between matrices called convolution.
They are basically composed of a series of connected nodes called artificial neurons. When a neuron receives an input, it process it accordingly to a specific function, and then forward the result to the connected nodes. Generally, CNNs are organized in layers and every layer propagate the signal only to the next one, this process continue until the last layer (output layer) is not reached.

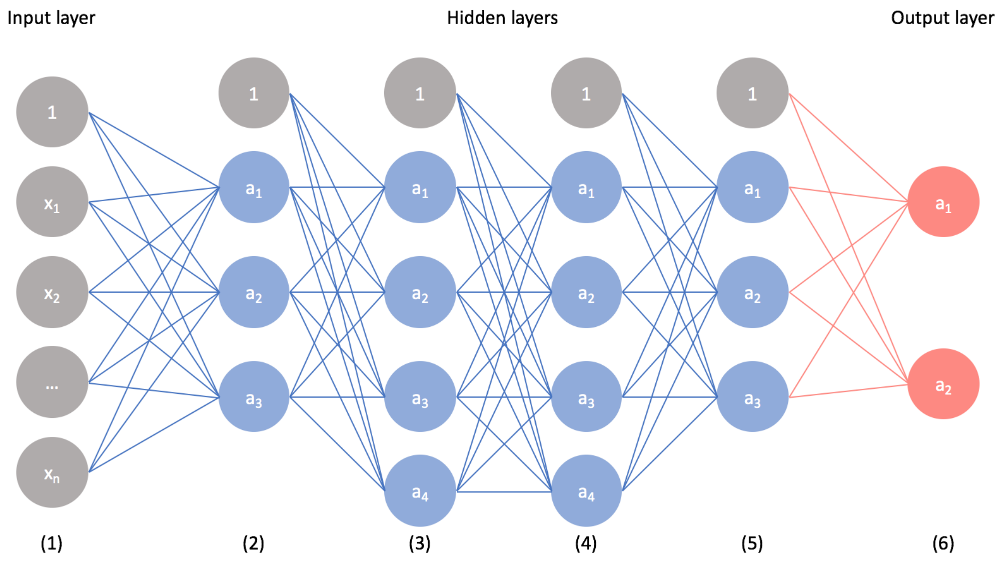


Figure 1: Structure of a Convolutional Neural Network

The structure of the CNN let automatically learn patterns in a progressive way, every layer some features are extracted for each training as long as the size of the training set is large enough and set of parameters required are finely tuned.
The typical layers of a CNN are reported below for reference.

Convolutional layer

This is the core layer of the net. The convolutional layer is composed of a set of filters, also called kernels, with which the nodes are mathematically convoluted. The result of this operation is an activation map of the specific filter involved, indeed the number of filters will determine the number of output that will be forwarded to the next layer of the network.

Activation function

An activation function can be applied within the layers of the net in order to transform the input in some specific output.
Generally in the context of CNN, only non-linear function are useful, once the function is defined, it will transform the input accordingly to this function.
Among all the possible activation function, the most used are the *Rectifiers* (employed in Rectified Linear Units - ReLU).
A ReLU computes the function:

$$f(x) = max(0, x)$$

Pooling layer

Generally input data can be simplified (lowering the computation load of the CNN) using the pooling layers. Usually they are placed between consecutive convolutional layers, they offer different functions to implement such simplification: the most common ones are average poolings and max poolings.
In the process, portions of the input images are analyzed. Let's consider for example a small window of 2×2 pixels, the 4 values contained in such window are averaged if it is the case of an average pool layer, or the maximum one is taken (Max pooling case). Then, the window is moved at a certain distance, called stride and the process is repeated until the whole input is consumed. The result will have of lower dimensions; however, the important features are still correctly placed with respect to each other.
Often, this is sufficient to get a correct final prediction.

Fully connected layer

Together with convolutional layers, fully connected layers (also known as *dense layers*) are the most common ones.
Generally they are placed at the bottom of the network and their input is a flattened array of values received from the previous layer. Mathematically, they compute a dot-product with a matrix of weights and eventually the result is summed to a vector of bias.
They play a key role in the network because are in charge to convert the partial output obtained so far in a 1D-vector (if we are dealing with a classification task) that is the key to obtain predictions.

Temporal Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a class of DL models usually applied in computer vision for image recognition, hence they are a good fit for image classification task.
If we wants to deal with a different nature of data, i.e. time series, Temporal Convolutional Neural Networks (TCNN) come into play. TCNNs are a kind of Convolutional Neural Networks for sequence modelling tasks, which work by combining aspects of RNN (Recurrent Neural Networks) and CNN architectures. TCCNs [36] possess very long effective history sizes (i.e., the ability for the networks to look very far into the past to make a prediction) using a combination of very deep networks.

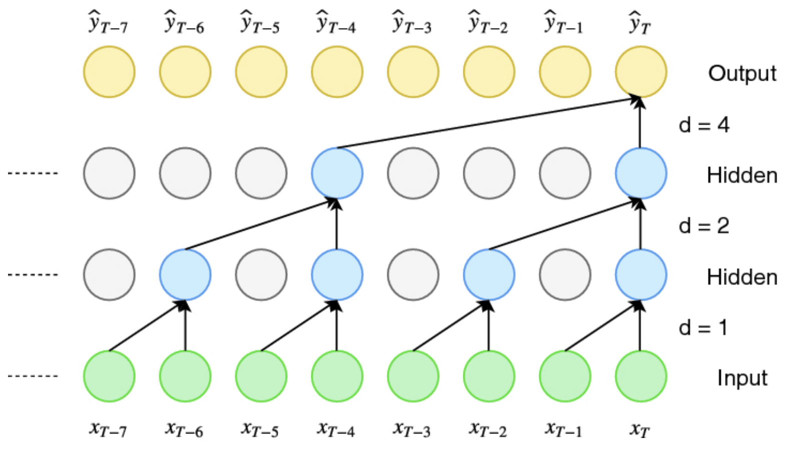


Figure 2: How a TCNN looks like

The main difference with respect to CNN is definitely on the type of input data fed into the network. Except for this the process remains exactly the same, the data goes through the several layers, forwarding the intermediate results that could come from the convolutional layers, average pool, etc. to the next layer. In general also the kind of output we're expecting changes. Indeed, the use of TCNN is mostly referred to regression problems in which the output of the network consists in a forecast dependent on the input data fed to the network. Let's consider to feed the network with the daily average temperature of a city, then after having trained the model, we want to use it to forecast the average temperature for the next 7 days (it is possible to choose a specific time-horizon: 1 day, 3 days, 7 days etc.).
The proposed solution is based on TCNN and it will be deeply analyzed in the following sections.

2.4. Homomorphic Encryption (HE)

Among the different types of encryption, HE allows a (limited) set of operations directly on the ciphertexts (i.e. the result of encryption performed on plaintexts using an algorithm).
As stated in [4], an encryption function E and its decryption function D are homomorphic with respect to a class of functions $\mathcal{F}$ if, for any function $f \in \mathcal{F}$, we can construct a function g such that $f(x) = D(g(E(x)))$ for a set of input x. This is possible because after the encryption the algebraic structure of the plaintexts is preserved.
Common encryption scheme, in this sense, are not homomorphic, indeed no algebraic structure is kept after the encryption phase. In this case the ciphertexts obtained are meaningless in terms of operation executed on them. The results of algebraic operations, such as additions and multiplications, bring to incorrect values with respect the ones that would come from the plaintext space.

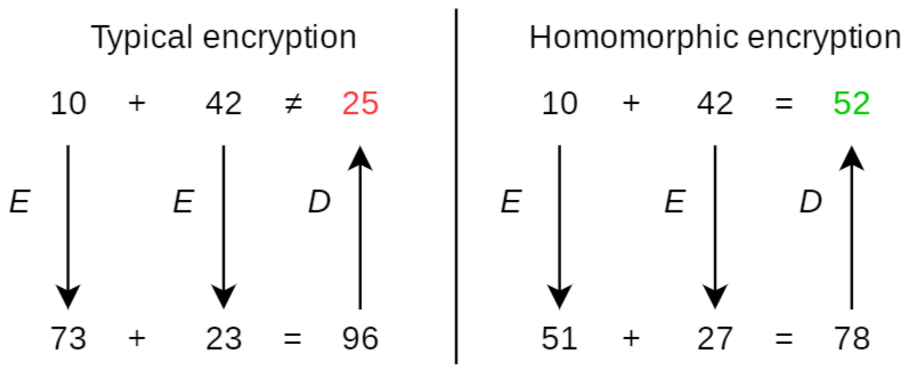


Figure 3: Comparison between common Encryption scheme and HE scheme [14]

It is possible to distinguish three main categories of HE schemes:

1. **Somewhat homomorphic schemes**, in which operations such as additions and multiplications are supported while all the others operations need to be approximated with the first two.
   Additionaly it is not possible to operate on the same ciphertext for an unbounded number of consecutive times, indeed at every operation a certain quantity of noise is added to the ciphertext and if a certain threshold of noise is reached, then it will not be anymore possible to make the decryption without corrupting the data;
2. **Partially homomorphic schemes**, this kind of schemes support an unbounded number of operations on ciphertexts but just for a single type of operation;
3. **Fully homomorphic schemes**, in this case both additions and multiplications are supported for an unbounded number of times on a ciphertext.

2.4.1. Focus on the Brakersi/Fan-Vercauteren scheme (BFV)

The Brakerski/Fan-Vercauteren (BFV) [16] scheme is the one considered in this paper and it is based on the Ring-Learning With Errors (RLWE) [6] problem.
The idea behind this scheme was to modify the scheme proposed by Brakerski [7], going from the setting of a Learning With Error (LWE) problem to a RLWE.

Polynomial rings

Polynomial ring is the main object used in the BFV scheme, some preliminary notions are required in order to explore and understand the scheme.
Definition 2.3. *A ring is a set R with the operations of addition and multiplication. Addition is defined as $+ : R \times R \to R, (a, b) \longmapsto a + b$, while multiplication is defined as $* : R \times R \to R, (a, b) \longmapsto a * b$ [22].*
A ring satisfies the following conditions:
1. $(R, +)$ is a commutative group, i.e. $a + b = b + a$. The element $0 \in R$ is the neutral element in this group. The inverse element of $a \in R$ is $-a$.
2. The multiplication is associative, i.e. $(a * b) * c = a * (b * c)$ for every $a, b, c \in R$.

3. The distributive laws hold, i.e. for all $a, b, c \in R$:

$$a * (b + c) = a * b + a * c,$$

$$(a + b) * c = a * c + b * c.$$

The element $1 \in R$ is called unit and $1 * a = a * 1 = a$ for all $a \in R$. If R is a ring, the polynomial ring $R[x]$ is the ring of every polynomials with coefficients in R:

$$R[x] = \{a_0 + a_1 x + \cdots + a_n x^n : a_i \in R \, \forall i\}$$

The addition and multiplications operations are defined as:

$$\sum_{i=0}^{n} a_i x^i + \sum_{i=0}^{n} b_i x^i = \sum_{i=0}^{n} (a_i + b_i) x^i$$

$$\sum_{i=0}^{n} a_i x^i * \sum_{j=0}^{m} b_j x^j = \sum_{i=0}^{n} \sum_{j=0}^{m} a_i b_j x^{i+j}$$

The ring used in the BFV scheme is:

$$R = \mathbb{Z}[x]/(f(x))$$

$f(x) \in \mathbb{Z}[x]$ is a monic irreducible polynomial of degree m. In particular, a cyclotomic polynomial $\Phi_m(x)$ of degree m is used, with m being a positive power of 2.

Ring Learning With Errors problem

Ring Learning With Errors is a computational problem used in many fields of cryptography, including the developing of encryption schemes resistant to attacks.
BFV uses the "decision" version of the problem.

Definition 2.4. ***RLWE-Decision*** *Given:*

- $\mathbf{a_i}$ *a set of random but known polynomials from R_q, which is the ring R with coefficients in $\mathbb{Z}_q$;*
- $\mathbf{e_i}$ *is a set of small random and unknown polynomials relative to a bound b in the ring $\mathbb{Z}_q$;*
- $\mathbf{s}$ *be a small unknown polynomial relative to a bound b in the ring $\mathbb{Z}_q$.*
- $\mathbf{b}_i = (\mathbf{a}_i \cdot \mathbf{s}) + \mathbf{e}_i$

And a list of polynomial pairs $(\mathbf{a}_i, \mathbf{b}_i)$, the RLWE-Decision problem consists in determining if the $\mathbf{b}_i$ polynomials were constructed as $\mathbf{b}_i = (\mathbf{a}_i \cdot \mathbf{s}) + \mathbf{e}_i$ or were generated randomly from R_q.

Encryption Scheme

[24] describes the encryption scheme which has been used for the base of the BFV scheme [16]. R_t is the plaintext space, for some $t > 1$. Consider $\Delta = \lfloor q/t \rfloor$, and $r_t(q) = q \bmod t$. Thus, it holds:

$$q = \Delta \cdot t + r_t(q)$$

As reported by the authors, q and t have not to be prime nor coprime. The basic operations of the scheme are:

- SecretKeyGen (1^λ): the sample $\mathbf{s} \leftarrow \chi$ and the output sk$=\mathbf{s}$.
- PublicKeyGen (sk): set $\mathbf{s}$=sk, sample $\mathbf{a} \leftarrow R_q$, $\mathbf{e} \leftarrow \chi$ and output

$$\text{pk} = ([-(\mathbf{a} \cdot \mathbf{s} + \mathbf{e})]_q, \mathbf{a})$$

- EvaluateKeyGen (sk, p): sample $\mathbf{a} \leftarrow R_{p \cdot q}, \mathbf{e} \leftarrow \chi'$ and return

$$\text{rlk} = ([-(\mathbf{a} \cdot \mathbf{s} + \mathbf{e}) + p \cdot \mathbf{s}^2]_{p \cdot q}, \mathbf{a})$$

  Evaluation keys are needed for the operation of *relinearization.*
- Encrypt (pk, $\mathbf{m}$): to encrypt a message $\mathbf{m} \in R_t$, let:

$$\mathbf{p}_0 = \text{pk}[0]$$

$$\mathbf{p}_1 = \text{pk}[1]$$

  Then, sample $\mathbf{u}$, $\mathbf{e}_1$, $\mathbf{e_2} \leftarrow \chi$ and return:

$$\text{ct} = ([\mathbf{p}_0 \cdot \mathbf{u} + \mathbf{e}_1 + \Delta \cdot \mathbf{m}]_q, [\mathbf{p}_1 \cdot \mathbf{u} + \mathbf{e}_2]_q)$$

- Decrypt (sk, ct): set $\mathbf{s} = \mathrm{sk}, \mathbf{c}_0 = \mathrm{ct}[0], \mathbf{c}_1 = \mathrm{ct}[1]$. The decrypted value is:

$$\left[\left\lfloor \frac{t \cdot [\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q}{q} \right\rceil\right]_t$$

- Add $(\mathrm{ct}_0, \mathrm{ct}_1)$: Output $(\mathrm{ct}_0[0] + \mathrm{ct}_1[0], \mathrm{ct}_0[1] + \mathrm{ct}_1[1])$
- Multiply $(\mathrm{ct}_0, \mathrm{ct}_1, \mathrm{rlk})$: compute

$$\mathbf{c}_0 = \left[\left\lfloor \frac{t \cdot (\mathrm{ct}_1[0] \cdot \mathrm{ct}_2[0])}{q} \right\rceil\right]_q$$
$$\mathbf{c}_1 = \left[\left\lfloor \frac{t \cdot (\mathrm{ct}_1[0] \cdot \mathrm{ct}_2[1] + \mathrm{ct}_1[1] \cdot \mathrm{ct}_2[0])}{q} \right\rceil\right]_q$$
$$\mathbf{c}_2 = \left[\left\lfloor \frac{t \cdot (\mathrm{ct}_1[1] \cdot \mathrm{ct}_2[1])}{q} \right\rceil\right]_q$$

These are the basic operations employed in the BFV scheme and it can be shown to be semantically secure [24]. The operation of relinearization is related to noise and noise growth in the ciphertexts.
Indeed this operation is used to limit the noise growth during multiplications. Relinearization consists in writing $\mathbf{c}_2$ in base T and set:

$$\mathbf{c}_0^{'} = \left[\mathbf{c}_0 + \sum_{i=0}^{l} \mathrm{rlk}[i][0] \cdot \mathbf{c}_2^{(i)}\right]_q$$
$$\mathbf{c}_1^{'} = \left[\mathbf{c}_1 + \sum_{i=0}^{l} \mathrm{rlk}[i][1] \cdot \mathbf{c}_2^{(i)}\right]_q$$

Finally, return $(\mathbf{c}_0^{'}, \mathbf{c}_1^{'})$.
These concepts are fundamental from a practical point of view for the use of the BFV scheme.

The Noise concept

Another key element for the HE-BFV scheme is the concept of noise.

Definition 2.5. ***Invariant noise*** *Let $ct = (c_0, c_1, \dots, c_k)$ be a ciphertext encrypting the message $m \in R_t$. Its invariant noise v is the polynomial with the smallest infinity norm such that:*

$$\frac{t}{q} ct(s) = \frac{t}{q}\left(c_0 + c_1 s + \cdots + c_k s^k\right) = m + v + at$$

for some polynomial a with integer coefficients [18].

During the encryption, noise is added to the ciphertexts to guarantee that, being $p_1 = p_2$ two plain values to be encrypted with the same public key, the corresponding ciphertexts c_1 and c_2 are different (i.e., $c_1 \neq c_2$). However, every time operations are performed on ciphertexts, their noise increases. When the noise reaches a certain threshold, it becomes impossible to decrypt the data without corrupting it.

Definition 2.6. ***Noise budget*** *Let v be the invariant noise of a ciphertext ct encrypting the message $m \in R_t$. Then the noise budget (NB) of ct is $-\log_2(2||v||)$.*

The NB is an integer positive number. It if is greater than 0, it still possible to decrypt a ciphertext and obtain the correct plaintext value hence without corrupting the data. As stated in [18]:

Lemma 2.1. *The function Decrypt correctly decrypts a ciphertext ct encrypting a message m, as long as the NB of ct is positive.*

It is useful to understand how noise grows when an operation is performed on a ciphertext. Generally, the important aspect is that additions and subtractions impact in a light fashion on the noise growth, while multiplications are the operations that consume more noise budget.

Plain operations

The plain operation term refers to operations that involve ciphertext and a plaintext in the ciphertext space. This type of operations is really frequent, many times a ciphertext has to be used in a function with other values which are not encrypted. Plain operations are less computational consuming with respect to operations between ciphertexts; moreover, they consume much less noise. As stated in [18], the noise of the result of a plain addition is:

Lemma 2.2. *Let $ct = (x_0, \cdots, x_j)$ be a ciphertext encrypting m_1 with noise v and let m_2 be a plaintext polynomial. Let ct_{padd} denote the ciphertext obtained by plain addition of ct with $m2$. Then the noise of ct_{padd} is $v_{padd} = v - \frac{r_t(q)}{q} m_2$, with the bound:*

$$||v_{padd}|| \leq ||v|| + \frac{r_t(q)}{q}||m_2||$$

While for plain multiplication:

Lemma 2.3. *Let $ct = (x_0, \cdots, x_j)$ be a ciphertext encrypting m_1 with noise v and let m_2 be a plaintext polynomial. Let N_{m_2} be an upper bound on the number of non-zero terms in the polynomial m_2. Let ct_{pmult} denote the ciphertext obtained by plain multiplication of ct with $m2$. Then the noise of ct_{pmult} is $v_{pmult} = m_2 v$, with the bound:*

$$||v_{pmult}|| \leq N_{m_2}||m_2||||v||$$

Nevertheless, plain values have to be encoded before they can be used in operations with other ciphertexts.

Encoding Process

Generally data processing involves numbers, integers or reals. Hence there is need to encode numbers into polynomial to exploit the full potentiality of HE. BFV provides operations to execute this encoding. Every number must be first encoded into a plaintext polynomial in R_t, and can only be encrypted after that [18]. It is clear that the rings $\mathbb{Z}$ and R_t are different: for example the set of integers is infinite, while R_t is not. This is the main reason for which HE applications must be carefully designed.
Two main encoder are available, the integer and the fractional one.

Integer encoder

Many integer encoders exist, one for each *base*. The base is denoted by $B \geq 2$. Consider the case with $B = 2$. A possible way [18] to encode an integer in the range $-(2^n - 1) \leq a \leq 2^n - 1$ is to form the n-bit binary expansion of $|a|$, say $a_{n_1}, \cdot, a_1 a_0$. The binary encoding of a, then, is:

$$\mathsf{IntegerEncode}(a, B = 2) = \text{sign}(a) \cdot (a_{n-1}x^{n-1} + \cdot + a_1 x + a_0)$$

If $B > 2$, instead of a binary expansion, a base-B expansion is used. The coefficients are chosen from the symmetric set $[-\frac{(B-1)}{2}, \cdot, \frac{(B-1)}{2}]$, given that there is a unique representation with at most n coefficients for each integer in $[-\frac{(B^n-1)}{2}, \frac{(B^n-1)}{2}]$. Usually, it is used $B = 2$ or $B = 3$. Decoding is simple and consists in evaluating the plaintext polynomial at $x = B$.

Fractional encoder

Fractional encoder are meant to encode real numbers. As for the integer encoder, fractional encoders are parametrized by an integer base $B \geq 2$ [12], with the same role it had in the previous case. First the integer part of the value is encoded, using the usual integer encoder. n is added to each exponent of the fractional part of the binary expansion of the value. Then, the base B is changed into the variable x; lastly, the signs of each term are flipped. [18] presents a practical example. Consider $B = 2$ and the rational number 5.8125. It has a finite binary expansion:

$$5.8125 = 2^2 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4}$$

The integer part is encoded as usual, obtaining the polynomial $\mathsf{IntegerEncode}(5, B = 2) = x^2 + 1$. Then, n is added to each exponent of the fractional part $(2^{-1} + 2^{-2} + 2^{-4})$ and the base 2 is changed into x: the result is $x^{n-1} + x^{n-2} + x^{n-4}$. Lastly, each sign of the terms is switched:

$$-x^{n-1} - x^{n-2} - x^{n-4}$$

Formally, for any rational number r with finite binary expansion it holds:

$$\begin{aligned}\mathsf{FracEncode}(r, B = 2) = \operatorname{sign}(r) \cdot [&\mathsf{IntegerEncode}(\lfloor |r| \rfloor, B = 2) \\ &+ \mathsf{FracEncode}(\{|r|\}, B = 2)]\end{aligned}$$

The decoding phase consists in the application of the steps described above but simply in reverse order. Consider that not every rational number has a finite binary expansion. If it is the case, the expansion of the fractional part has to be truncated to some precision (which can be expressed as n_f bits).

2.5. Homomorphic Encryption for the practical use

The BFV scheme previously introduced relies on the choice of a set of parameters, namely:

- m: Polynomial modulus degree,
- p: Plaintext modulus, and
- q: Ciphertext coefficient modulus.

The choice of such parameters determines the performance obtained from the HE setting.
Indeed, this is reflected in the number of operations that can be performed on a ciphertext without corrupting it, the computational load requested to perform the operations, the memory requested and finally the level of security against ciphertexts attacks.
As we stated before, the Noise Budget (NB) is a practical indicator for the number of operations still allowed on a specific ciphertext, if the NB reaches zero it means that no more operations are allowed on such ciphertext. Hence, there is the needing to carefully tune the encryption parameters to carry out all the operations without corrupting the data.

Parameters choice

In this part the choice of the set of parameters is discussed, in particular every choice for a singular parameter affects the environment in a specific way.
Let's dive into it:

- $\boldsymbol{m}$: increasing this parameter reflects into an higher noise budget for a fresh encrypted value, on the other hand it significantly increase the computational load and memory occupation and consequently the time requested to perform an operation on a ciphertext.
- $\boldsymbol{p}$: this parameter is strictly related to the accuracy of the operations performed on the ciphertext, the higher p, the higher the accuracy. The drawback is that with a greater value, the NB is more fastly reduce, again, if it reaches 0, then it will corrupt the value when the ciphertext is decrypted.
- $\boldsymbol{q}$: affects the security of the scheme, and represents a very difficult parameter to set. For this work the choice of q was delegated to the SEAL library [33] which provides a way to automatically set q given m and the desired AES-equivalent security level [18].

In the following sections, the solution proposed take into consideration all these aspects and the best parameters set has been chosen after an appropriate tuning phase.

3. Related Literature

From the literature many interesting works related to Homomorphic Encryption are available.
A first example is the paper of " Privacy-Preserving collective learning with HE " [30] published on 22 September 2021.
The work is intended to present a protocol to share classified time series data within entities to train parameters. Differently from the proposed solution, here the authors of [30] aim to use encrypted data to train a deep learning model while respecting the concept of privacy preserving.
The CKKS (Cheon-Kim-Kim-Song) encryption scheme is used because of its ability to work with real number using approximate arithmetic.
The experiment has been carried out with a LSTM architecture, a special case of RNN (Recurrent Neural Network), but due to the limits of HE especially in terms of number of operations allowed, a particular approach has been used used.
Only the last layer of the network was fine-tuned since HE schemes are not powerful enough to train deep-learning methods end-to-end.
Fully Homomorphic Encryption (FHE) instead has been treated in the work presented in [23] published on 19 December 2016 by Wen-jie Lu and Shohei Kawasaki and Jun Sakuma.
The paper illustrate how statistical analysis can be performed on encrypted data. The authors of [23] proposed how to conduct privacy-preserving statistical analysis using fully homomorphic encryption. In particular they showed how to conduct descriptive and predictive statistics on FHE ciphertexts efficiently. They also experimentally demonstrated the utility of their procedure, for example it took about 20 minutes to perform the model building of a linear regression on about 30k data of 6 features as input.
BFV encryption scheme has been used in the paper " Privacy-preserving time series medical images analysis using a hybrid deep learning framework " [39] published on October 2019.
The work is intended to propose a computer-aided diagnosis (CAD) algorithm to analyze time-series medical images automatically. LSTM is the reference DL model for this work and several improvements on the original network layers are made to ensure the HE-Convolutional-LSTM can support calculation on encrypted images.
The proposed paper " Privacy-friendly Forecasting for the Smart Grid using Homomorphic Encryption and the Group Method of Data Handling " [5] focus on predicting energy consumption while preserving individual consumer usage information.
Also here, as for the proposed work [23], fully homomorphic encryption (FHE) is investigated. Popular class of deep learning methods can not be applied because of the computation of the sigmoid activation function which introduce non linearity. The paper [5] shows how the usage of *Ivakhnenko's group method of data handling (GMDH)*, a family of inductive algorithms for computer-based mathematical modeling, can be implemented homomorphically; moreover, after a comparison analysis between different forecasting methods it showed that GMDH produced in a significant way more accurate results compared to the other methods considered.

4. Architecture of the proposed solution

The goal of this work is to propose a solution that provides a service of forecasting while the privacy of the data sent by the user is maintained. This solution is thought to be used in a Cloud computing scenario, in particular as a privacy-preserving DLaaS. This is similar to what proposed by Disabato et al. [10]. In that work, authors proposed a privacy-preserving DLaaS able to make image classification on encrypted user images. In the same way, the solution proposed in this thesis employs a DL model able to work on input data previously encrypted by a user. Differently, the considered task is time-series forecasting, even though an important part of the infrastructure can be maintained. Note that this desirable for a Cloud provider: having the possibility to offer many services with minimal changes in the underlying infrastructure allows to minimize the costs.
The general flow of the data consists in the encryption of user's time-series on her/his device, its transmission to the Cloud, the processing of the DL model on the encrypted data, the transmission of the still encrypted results to the user. Then, the user can simply decrypt the result and obtain the final forecasting for their time-series.

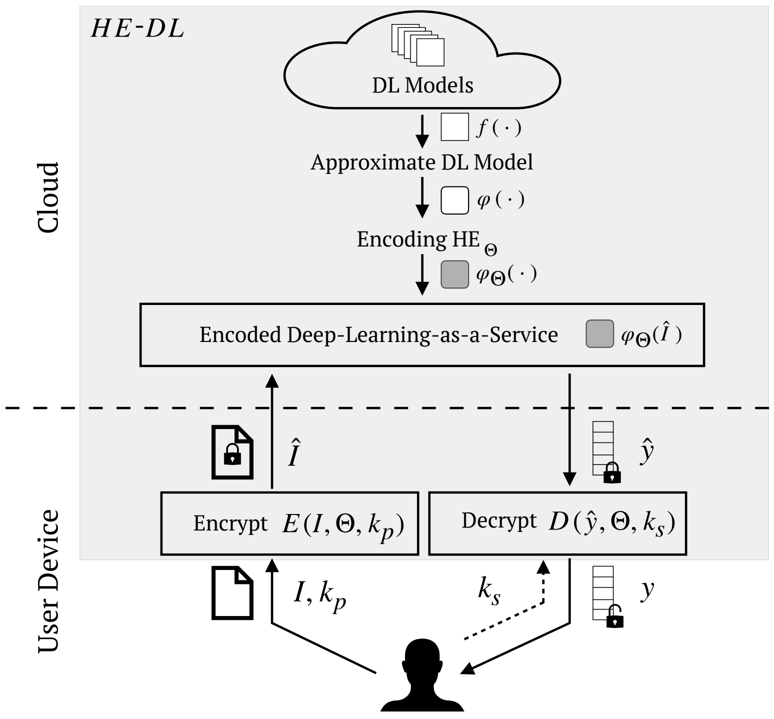


Figure 4: The privacy-preserving architecture for deep-learning-as-a-service presented in [10].

4.1. TCNN: a CNN in time-series forecasting perspective

Time series forecasting is not a simple task.
Differently from the simpler problems of classification and regression, time series problems add the complexity of order or temporal dependence between observations. This temporal structure can also aid in modelling, providing additional structure like trends and seasonality that can be used to improve model skill.
Generally, time series problems have been addressed by linear models like *ARIMA* because well understood and effective on a large scale of problems.
ARIMA model was introduced by Box and Jenkins in 1970. It refers to *Auto-Regressive Integrated Moving Average* because it accounts both for an auto-regressive and a moving average part. In such model, the future value of a variable is a linear combination of past values and past errors [3].
Classical models such this suffer from some limitations, machine learning methods, in this sense, can be effective on more complex time series forecasting problems with multiple input variables, complex non linear relationships, and missing data [8]. Convolutional neural networks can be applied to time series forecasting. Many types of CNN models are available from the literature and can be used for each specific type of time series forecasting problem.
Although CNNs are traditionally developed for two-dimensional image data, they can also be used to address time-series forecasting problem; in this case, they assume the name of *Temporal Convolutional Neural Network* (TCNN). Note that TCNNs may be used both on *multivariate* time-series or in *univariate* ones. In this work, we focused on the univariate time-series scenario.

4.2. CNN as a stateless deep learning model

As mentioned in the previous sections, BFV encryption scheme allows only a certain quantity of operations on a ciphertext.
The noise budget is an overall indicator of how many operations can be performed on a ciphertext without corrupting the data, making impossible the decryption.
Mathematical operations such as multiplications and summations impact differently on the NB, indeed the formers are way more NB-consuming with respect to the latters. Also the type of data involved in the operations impacts in a different way, for example, performing an operation between two ciphertexts consumes more noise budget with respect to operate with a ciphertext and a plaintext.
It is clear that a model that elaborates the minimum (but sufficient to produce good results) number of operations on data should be preferred.
This is exactly why the work focus on CNN's models rather than more advanced and computationally complex ones such as RNN (Recurrent Neural Network).
The key concept behind is that RNNs are *stateful*, they are networks with loops in them, allowing the information to persist.

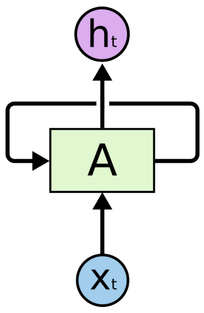


Figure 5: RNNs have loops.

The image above is useful to illustrate the concept of persistence inside a RNN, considering A as a chunk of a neural network, it's clear how the partial output is kept inside the network to be re-used and possibly increase the accuracy of the predictions.
Considering the homomorphic encryption perspective, the re-use of such data is something that should be avoided because of the issue of noise budget consuming. Every new re-usage of the ciphertext would unavoidably lead to a NB consumption increasing the risk to corrupt the data.
CNNs do not operate in such fashion, indeed they are considered ***stateless*** because no persistence is implemented.
Even though this seems to be a shortcoming of the model (because RNN could perform better than CNN on the same task), the results carried out with this work are satisfactory and any risk linked to the exhaustion of the noise budget has been prevented.

4.3. Encryption and Decryption

Privacy preserving of user data is totally based on a specific part of the architecture, namely the encryption and decryption phase.
Weak encryption parameters or incorrect processes would lead user data to be exposed or to be corrupted and hence useless. That is why this phase has to be carried out carefully.
The choice of the set of encryption parameters $\Theta = \{m, p, q\}$ directly impacts the security of data, the computational load and correctness of results.
The encryption scheme is asymmetric, such parameters are used to generate a **public key** (that will be provided also to the cloud service provider) and **private key** that never leaves the user device and is used to decrypt the final result.
The ***encryption*** process is defined as follows:

$$\hat{I} = E\left(I, \Theta, k_p\right)$$

being:

- I: the user data, i.e., the time-series to be forecasted;
- Θ: the encryption parameters;
- k_p: the public key.

The result $\hat{I}$ represents the encrypted user data, i.e. the encrypted time-series to be forecasted. We emphasize that the notation is the one used in [10], but adapted for the time-series forecasting task.
The encrypted time-series is ready to be processed by a cloud service provider, which will use an *encoded* TCNN for the forecasting. This process returns a result denoted by $\hat{y}$, while the meaning of *encoded* is explained in the next Section.
The result, which is the forecast for the time-series, is still encrypted and only the user can access to it after having performed the ***decryption*** process:

$$y = D(\hat{y}, \Theta, k_s).$$

Similarly to the encryption, the parameters of $D(\cdot)$ are:

- $\hat{y}$: the output of $\varphi_\Theta(\hat{I})$, i.e., the encrypted time-series forecast;
- Θ: the encryption parameters;
- k_s: the secret key.

Obviously, the secret key k_s, generated during the encryption process, never leaves the user device. Only the user has the rights to access and use it in order to decrypt the results coming from the cloud. y represents the plain output of the user requested service. The meaning of $\varphi_\Theta(\hat{I})$ is explained in the next Section but at this point it is just necessary to know that it represents the process that the service provider performs on the encrypted data.

4.4. Approximated and encoded Deep Learning processing

As stated in previous sections, one of the main limits of HE-BFV is that only additions and multiplications can be carried out on encrypted data. If *DL models* $f(\cdot)$s are denoted as a set of deep-learning models, it has to be clear that $f(\cdot)$ can not be used as they are, they have to be modified in order to not use operations unavailable in HE-BFV.
This process is called *Approximation*. After being approximated the set of approximated *DL models* $\varphi(\cdot)$s is obtained. Moreover, these approximated models need to be encoded in order to work with ciphertexts, in particular all the weights of every model are encoded; to carry out the encoding operation, the set of encryption parameters θ is required. Finally the set of *encoded deep-learning-as-a-service* $\varphi_\Theta(\cdot)$ is obtained.
The cloud after the approximation and encoding of the DL models is ready to process the encrypted data $\hat{I}$ and provide results (still encrypted):

$$\hat{y} = \varphi_\Theta(\hat{I})$$

Let $f(\mathcal{I})$ be a CNN composed of L layers $\eta^{(l)}_{\theta_l}$ with parameters θ_l and $l = 1, \ldots, L$.
Since layers such as convolutional and dense layers are composed only by additions and multiplications, there is no need to approximate them.
The approximation is required on the activation function and the pooling layers. Generally the most used activation function in CNN is ReLU which consist in a max operation:

$$f(x) = \max(0, x)$$

Unfortunately, the max operation is not polynomial and this makes it incompatible with the HE-BFV scheme. In this case the Square activation function will be used, whose function is:

$$f(x) = x^2$$

The max-pool layer (typically used in CNN) cannot be executed directly in a HE-BFV scenario. Generally, the pooling operation transforms a sub-matrix coming from a previous layer into a smaller one whose values depend on the type of pooling applied. The max-pooling operator, similarly to the ReLU activation function requires a max operation that is not polynomial, hence in this case it should be substituted with an average-pool operator that, as the name suggests, simply takes the average of such sub-matrix, an operation that can be achieved with simple additions and multiplications.
Note that, after performing the approximation process, the model has to be trained again. This is important because changing the architecture of the model changes also the flow of the data, hence, the weights used before the approximation process are not suited anymore.
Obviously, if the original model $f(\cdot)$ already contains only HE-compatible processing layers, this procedure is not necessary.

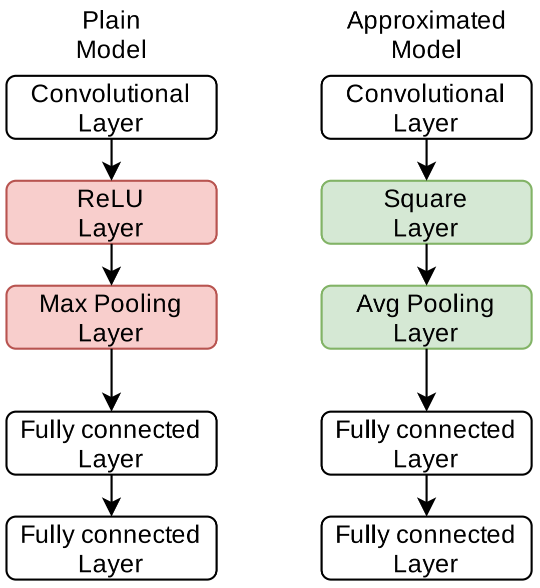


Figure 6: Example of a plain model with the corresponding approximated model [14]

Approximation is not enough to make a DL model able to work on encrypt data; infact its weights need to be encoded.
Encoding has nothing to do with encryption, it is just a matter of changing the representation of data (in this case a polynomial one) but everything is still “plain“.
To encode the model, the values of Θ are necessary; the used (who encrypted the data) has to provide them, and they have to be the same used to generate the encryption keys and to encrypt the values. The approximated and encoded CNN is finally denoted by φ_Θ. Summing up, the HE-based encrypted processing can be formalized as follows:

$$y = D\left(\varphi_\Theta\left(E\left(I, \Theta, k_p\right)\right), \Theta, k_s\right), \tag{1}$$

The equation clarifies every part of the process, first the client encrypt its own data with an encryption function E, this data is used by the approximated and encoded CNN φ_Θ on the cloud and an encrypted result $\hat{y}$ is returned; finally the user decrypt on his/her device the encrypted result $\hat{y}$ with a decryption function D that takes in input also the set of encryption parameters θ and the private key k_s originally generated.

4.5. Encryption parameters

As mentioned in the previous sections, the choice of the encryption parameters, the Polynomial modulus degree m, the Plaintext modulus p and the Ciphertext coefficient modulus q plays a key role in the encrypting process. In general increasing m and p result in an higher computational memory load but in a higher noise budget for fresh encrypted values. p directly affects the precision of the results but consequently the noise budget consumed at each operation is higher.
The Ciphertext coefficient modulus q can severely affects the security of encrypted data: this is why choosing it is not an easy task.
Thankfully, for this purpose, q will established through the SEAL library [33] which provides a specific function that, given m and the desired AES-equivalent security level *sec*, returns a suggested value for q [18].
Hence in the architecture, the user can choose the two parameters m and p. It is important to notice that they are visible to the cloud and they takes into account the correctness of the results and the computational load. The user can let the cloud service provider propose some values for these parameters, indeed it is just necessary a tuning phase in which looking how would increasing and decreasing such parameters impact on the results in terms of time and correctness.
In this work a security level of 128 *bits* is always used. This is an arbitrary choice motivated by the fact that the higher is the desired level of security, the higher is q and consequently the computational overhead increase.

5. Implementation of the proposed solution

This section is meant to introduce all the details concerning the implementation of the solution.
Specifically, after a gentle introduction to the external libraries used, the actual structure of work will be presented as well as the compatible models used to implement the solution and finally a look on how the testing phase was carried out.

5.1. External libraries used

To develop the proposed solution, several external libraries were used:

- Pyfhel [2]: Homomorphic encryption library;
- numpy [37]: Scientific computation;
- jsonpickle[1]: Serialization and deserialization of complex Python objects to and from JSON;
- PyCrCNN [14]: Library for privacy-preserving image-recognition as-a-service with Homomorphic Encryption.

5.2. Structure

The entire project is a library written in Python language.
It relies on the Pyfhel library v2.3.1 [2], Laurent (SAP) and Onen (EUROCOM) licensed under the GNU GPL v3 license for the tasks of HE; Pyfhel is a wrapper of the Microsoft SEAL library [33].
The library can be decomposed in different sections:

1. **convolutional**: it accounts for the code relative to the convolutional layer
2. **crypto**: it comes from the PyCrCNN [14] library and it accounts for the code of cryptographic operations, like encrypting matrices, encoding matrices, etc.
3. **functional**: it comes from the PyCrCNN [14] and accounts for the functional operations such as Square Layers.
4. **linear**: it accounts for the code relative to the linear (dense) layer.
5. **net_builder**: it comes from the PyCrCNN [14] and accounts for the code aimed to build an encoded model starting from a pytorch model.

The implemented library was developed to handle various kind of layers of a CNN, namely:

- Convolutional Layer
- Square Layer
- Average Pool Layer
- Flatten Layer
- Linear Layer

Every stated layer that was built has a similar shape to the same layer of from the PyTorch [29] (an open source machine learning library) library version.
There are two mainly methods inside each layer class:

- `__init__`: this method is the constructor of the layer, it takes in input the same parameters that the same pytorch layer would have been expected;
- `__call__`: this method takes in input as a parameter a tensor and process that tensor in the called layer returning an output.

Below an example of a network layer rebuilt from scratch.

Listing 1: Convolutional layer

```
class ConvolutionalLayer:
    """
    A class used to represent a convolutional layer
    ...


    """

    def __init__(self, HE, weights, stride=(1, 1), padding=(0, 0), bias=None):
        self.HE = HE
        self.weights = c.encode_matrix(HE, weights)
```

[1]https://jsonpickle.github.io/index.html

```
        self.stride = stride
        self.padding = padding
        self.bias = bias
        if bias is not None:
            self.bias = c.encode_matrix(HE, bias)



    def __call__(self, t):
        out = []
        for ts in t:
            for b,filter in enumerate(self.weights):
                i = 0
                while(i <= ts.shape[1] - filter.shape[1]):
                    out.append(np.sum(np.multiply(np.expand_dims(subMatrix(ts,i,
                        filter.shape[1]),0),filter)) + self.bias[b])
                    i += 1

        return (np.array(out).reshape(t.shape[0],self.weights.shape[0],t.shape
            [2]-self.weights.shape[2]+1))

def subMatrix(ts,idx,kernel_length):
    return ts[0][idx:kernel_length+idx]
```

5.3. Compatible models

The net_builder sub-package that comes from the PyCrCNN library [14] is responsible to accept a PyTorch CNN model, extract its features and encode them to let them be suitable for the HE-BFV scheme.
To be used, a PyTorch model has to fulfill some requirements:

- It must be a *Sequential* model [2];
- It must have extension .pt / .pth;
- It must have been saved in PyTorch with the `save()` function.

[2]https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html

6. Experimental results

This section provides an overview on the execution of the implemented solution with a focus on performance and accuracy.
The TCNN's performance is compared with other popular Deep Learning algorithms considering plain data (i.e. not encrypted), then the approximated and encoded CNN φ_θ is used and tested and the difference against the plain case are spotted.

6.1. Description of the CNN

The Convolutional Neural Network used for this work has been kept really simple in order to not heavily impact on the computational memory load.

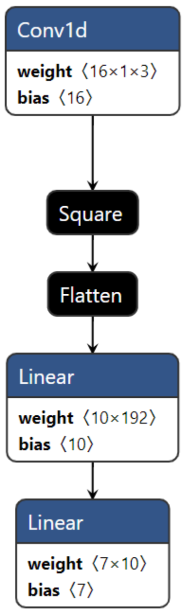


Figure 7: CNN model

The presented model $f(\cdot)$ already contains only HE-compatible processing layers, indeed the activation function is a Square one, and there is no pooling layer at all, hence the approximation procedure in this specific case is not necessary.
Let's analyze all the layers one by one:

- *Convolutional layer*: The time-series involved in the experiments are univariated and hence 1-dimensional. They have been chunked in smaller samples of arbitrary length that will be called *seq_length*.
  The layer has 16 3 × 1 filters with stride 1 × 1. Thus, the output is 1 × 16 × *seq_length* - 3 + 1.
- *Square activation layer*: This layer applies the Square function to each input node. The dimension of the batch is unchanged.
- *Flatten layer*: This layer is used to flatten a contiguous range of dimensions into a tensor. It's required before to forward data to a linear layer.
  The output has dimension 1 × (16 × (seq_length - 3 + 1))
- *Fully connected layer*: This layer connects the incoming 1 × (16 × (seq_length - 3 + 1)) nodes to 10 exit nodes.
- *Fully connected layer*: Accordingly to the desired time-horizon forecasting window (*forecast horizon*) this layer connects the incoming 10 nodes to the final *forecast horizon* nodes.

The forecast horizon is simply the number of future points which will be forecasted by the model. While it is possible to always fix this value to 1 and, eventually, use the TCNN more times if more forecast points are needed, this involves that the encrypted output of the model is re-used as input to forecast new points. This leads to the same situation shown in Section 4.2; hence, it is better to have model with a fixed and pre-determined forecast horizon.

6.1.1. Training the model

As discussed in the Introduction, in order to be able to make predictions, the model has to be trained with a training set of data.
There are two case scenario that could be considered:

1. The CNN can be trained with **publicly accessible data, not protected by any kind of privacy agreement**.
   Hence this data are unrelated to the ones we would make predictions on, the privacy is totally preserved because data are not accessed even when training the model.
   Then, once the model is trained, it can be used to make predictions feeding it with encrypted data.
2. The second scenario, that is also the one chosen for this work, is that **the user provides data coming from the past that are either not covered by confidentiality or do not present serious privacy problems**.
   Also in this case, after the model is trained, it is ready to be used on new unseen and encrypted data to make predictions.

Considering the real case scenario, namely the case in which we deal with the privacy preserving of data, it is unlikely that we will be able to re-train our model.
The user most probably will provide an unique set of data for training the model and nothing more. Hence the real case scenario is that after a single training (hopefully with an important amount of data) we will immediately deploy a ready-to-use model to make predictions.
Nevertheless, we carried out two different experimental campaign: in the first the model is trained just once and immediately used to make predictions, and in the second one the model is retrained at every new prediction and observe how the accuracy of the model behaves.
This second experiment was carried out mainly for comparative purposes, indeed in the next subsections a comparison, considering a continous retraining, with several deep learning algorithms is presented.

6.2. Datasets

In order to understand the performance of the implemented solution, four different datasets have been used.

- **Covid-19 Italy [17]:** this dataset accounts several information about the spread of the Covid-19 pandemic in Italy such as Daily cases, Total positives, Total positives variation, Total intensive care, Daily deaths etc. recorded from 25-02-2020 to 25-05-2021 and with a data range of 40260 for the daily cases and 992 for the daily deaths.
  For this dataset, two different information were extracted, namely *Daily Deaths* and *Daily Cases*.
  The model has been trained with data in the interval from 25-02-2020 to 18-08-2020 in both cases of daily deaths and cases, then the performance of the TCNN has been evaluated predicting the new daily cases or deaths from the end of training set to the end of the time series.
- **Monthly Milk Production [21]:** This dataset accounts the monthly milk production from 01-01-1962 to 01-12-1975 and it has a data range of 196.
  Here the model has been trained with the values in the interval from 01-01-1962 to 01-01-1974 and it has been evaluated predicting from the 01-02-1974 until the end of the dataset.
- **Airline passenger [21]:** The last dataset accounts the monthly passenger of a airline company, the interval of observations starts on January 1949 and ends on December 1960 with a data range of 518.
  The model has been trained with the values from the beginning of the time series to January 1958 and has been evaluated predicting values from February 1958 until the end of the series.

The model previous described has been trained with the best hyper-parameters, after a grid search as a tuning phase, for the particular considered dataset.

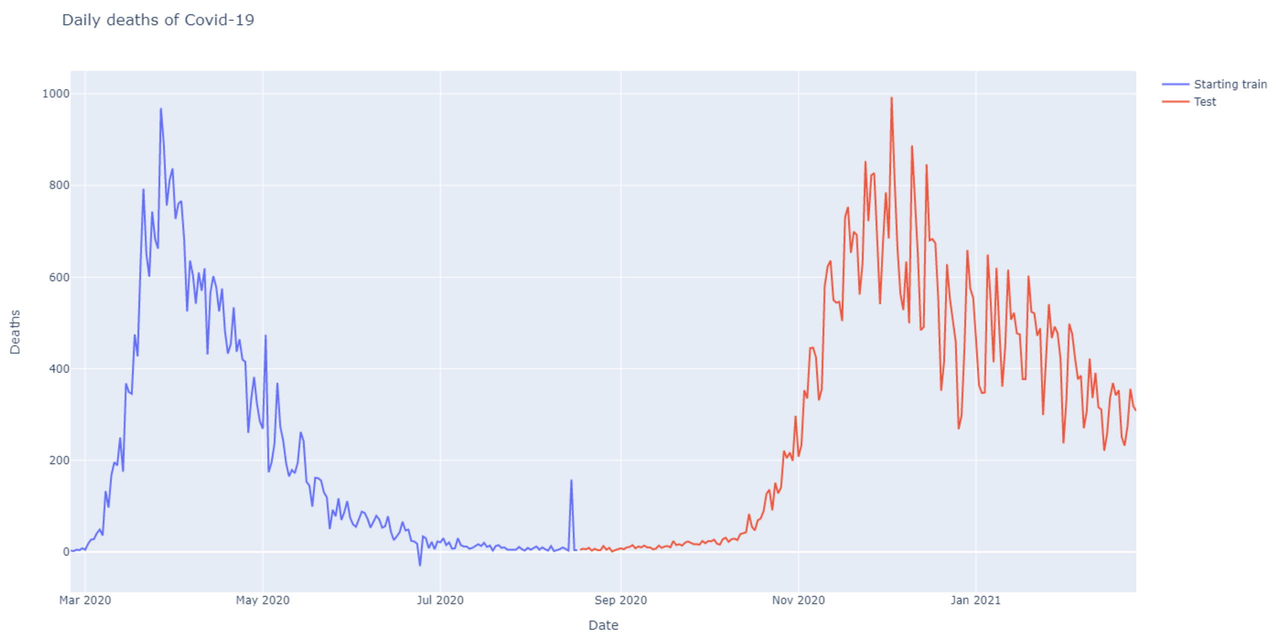


Figure 8: A representation of the daily deaths dataset

6.3. Performance on plain data

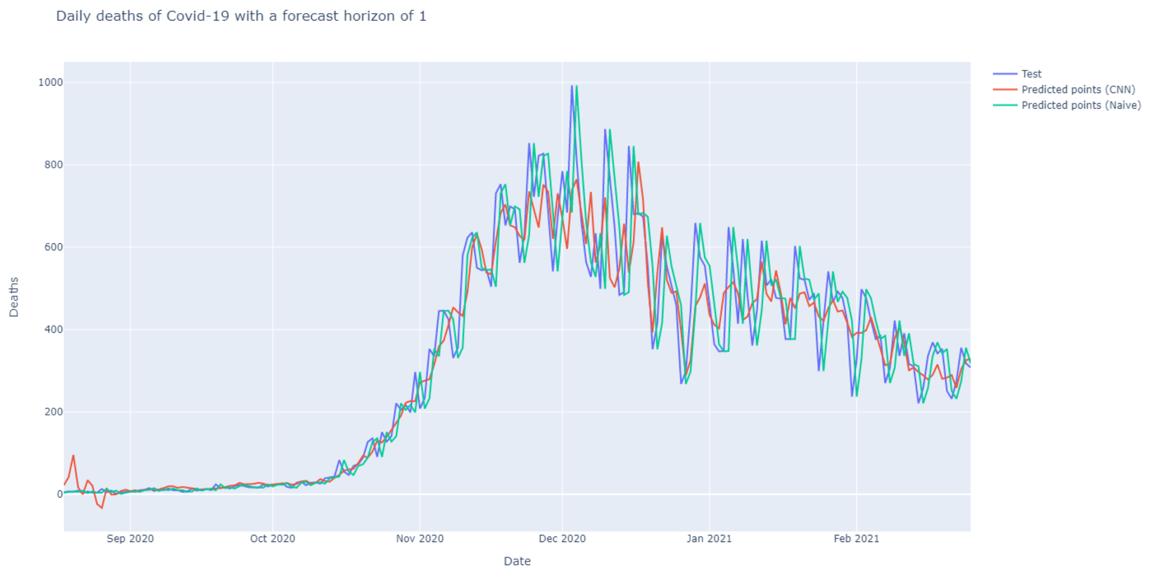
In order to evaluate performance, two error indices have been used:

- Mean Absolute Error (MAE) $= \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$ where y_i is the prediction coming from the model, while x_i is the true value and n is total number of predicted elements.
- Root Mean Squared Error (RMSE) $= \sqrt{\frac{\sum_{i=1}^{n} (y_i - x_i)^2}{n}}$ where, again, y_i is the prediction coming from the model, while x_i is the true value and n is total number of predicted elements.

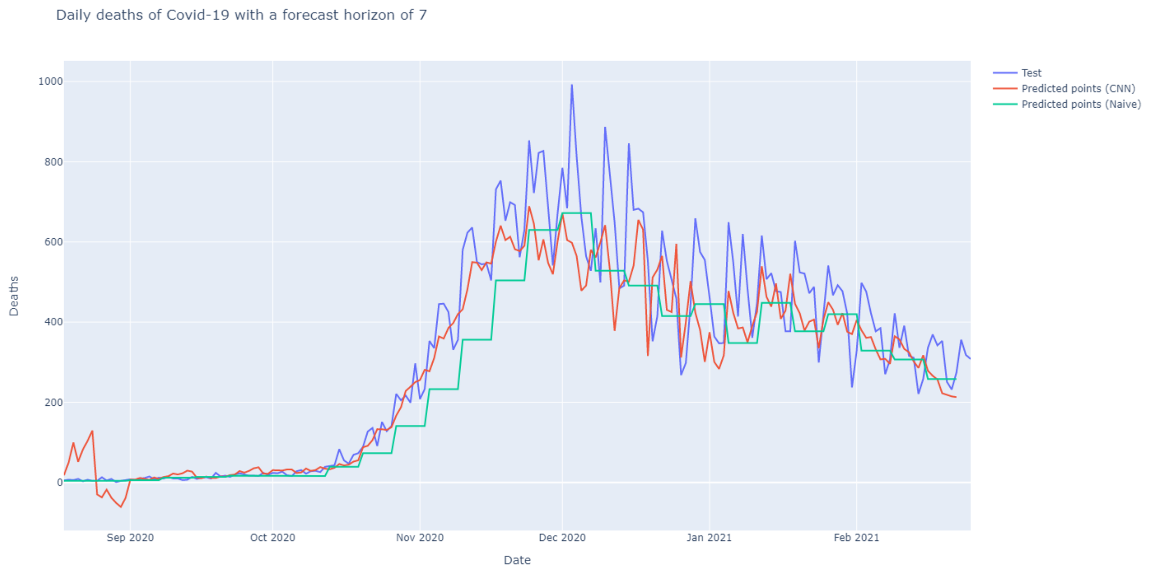
Accordingly to the specific dataset, different experiments have been carried out; for example in the *daily deaths* and *daily cases* dataset, predictions have been made with first a time horizon of 1 day, and then with a time horizon of 7 days.
For the *airline passenger* and *monthly milk productions* dataset, 3 different time-horizon have been chosen: 1 month, 3 months and 7 months.
At this moment, it is important to understand which are the performances of the model in a real case scenario, namely, as we mentioned in previous sections, with a one-time training on a training set and with predictions being made on unseen data.
Considering this fact we ended up with the following results, shown in Table 1.

Table 1: Performance results without retraining on different datasets.

		Daily cases		Daily deaths		Monthly Milk Production			Airline Monthly Passengers		
Forecast horizon		1-D	7-D	1-D	7-D	1-M	3-M	6-M	1-M	3-M	6-M
Range		40260		992		196			518		
TS-PyCrCNN without re-training	**MAE**	2471.3	2495.5	47.5	62.9	9.06	11.23	14.60	18.86	22.36	22.32
	RMSE	4061.6	3516.9	71.9	92.7	11.16	13.40	17.85	23.21	25.14	26.62
Naive forecast	**MAE**	2060.30	4131.86	59.98	78.64	42.13	50.66	85.27	43.42	56.27	87.56
	RMSE	2855.04	5616.64	94.73	113.43	49.82	57.00	90.40	51.00	76.51	107.77



(a) 1 day time horizon



(b) 7 days time horizon

Figure 9: Model predictions on Covid-19 daily deaths dataset

The two figures above are useful to give an idea of how the model performance outperforms the one from the naive forecasting; naive forecasting is simply an estimation technique for which the last actual value is used to predict the next value. Namely in the 1 day time horizon case, the actual value from a day is taken and used as a prediction for the next day; similarly, for the 7 days time horizon case, the actual value of a day is taken and used as prediction for the next 7 days.

		Covid 19 Daily Deaths	
Forecast horizon		1 day	7 days
Range		992	
Naive prediction	**MAE**	59.98	78.64
	RMSE	94.73	113.43
TS-PyCrCNN	**MAE**	**47.5**	**62.9**
	RMSE	**71.9**	**92.7**

6.3.1. Performance comparison with different Deep Learning algorithms

Table 2: Comparison of the forecasting performance with other DL solutions for time-series forecasting.

			Daily cases		Daily deaths		Monthly Milk Production			Airline Monthly Passengers		
Forecast horizon			1-D	7-D	1-D	7-D	1-M	3-M	6-M	1-M	3-M	6-M
Range			40260		992		196			518		
TS-PyCrCNN with re-training		**MAE**	1513.01	2331.23	**45.30**	58.16	8.79	10.56	14.14	**18.37**	**21.80**	**21.54**
		RMSE	2341.37	3499.27	72.52	86.93	10.69	13.13	16.98	23.06	25.38	25.95
TS-PyCrCNN without re-training		**MAE**	2471.3	2495.5	47.5	62.9	9.06	11.23	14.60	18.86	22.36	22.32
		RMSE	4061.6	3516.9	71.9	92.7	11.16	13.40	17.85	23.21	25.14	26.62
TIMEX-Uni	Prophet	MAE	1780.4	2518.3	53.2	59.1	12.2	15.4	20.0			
		RMSE	3149.6	4528.0	83.9	99.7	14.4	17.5	22.1			
	LSTM	MAE	2632.3	2917.7	68.4	67.9	37.6	58.0	44.6			
		RMSE	3956.5	4028.3	108.2	99.5	42.3	70.3	53.1			
	Arima	MAE	1601.6	**2169.0**	53.2	55.4	30.1	41.2	61.8			
		RMSE	2627.7	3240.3	83.9	94.0	39.4	49.9	75.8			
	Exp. Smooth.	MAE	**1380.5**	2319.2	45.3	**42.1**	**5.0**	**8.0**	**11.9**			
		RMSE	2173.0	3517.7	68.4	64.1	5.9	10.0	14.6			
TIMEX-Multi	Prophet	MAE	1789.0	2752.8	49.3	74.6	/	/	/			
		RMSE	3125.1	4958.0	77.5	170.9	/	/	/			

Table 2 shows the performance comparisons between the implemented solution and a collection of popular deep learning algorithms using the TIMEX library [15], a library for time-series forecasting as-a-service.
Each model has been trained on the same training set and the errors obtained come from exactly the same prediction's interval.
For each dataset, the best result is highlighted in green. It should be considered that all the other algorithms consider the case of a continous retraining when a new prediction is made. Hence, as it can be seen, for the *daily deaths* dataset the TS-PyCrCNN performance with 1 day time horizon outperforms all the other models. Nevertheless, as it could be expected, the error in the 1-time training case is higher but is still acceptable to be considered a good result. We stress that the *TS-PyCrCNN without re-training* row accounts for the results obtainable on encrypted data.

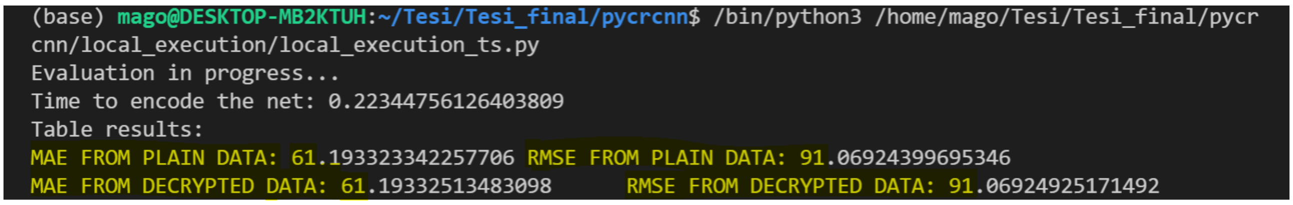
6.4. The encrypted forecasting process

At this point, since performance on plain data is convincing, it is time to test the encoded network φ_θ.
The network will be fed with a matrix of encrypted data and only the operations allowed by the HE-BFV scheme will be executed.
Finally, a comparison between *plain-case* and *encrypted-case* performances will be carried out considering the following encryption parameters:

- ***m***: 4096
- ***p***: 1511201451
- ***sec***: 128
- ***base***: 2

.

6.4.1. Results

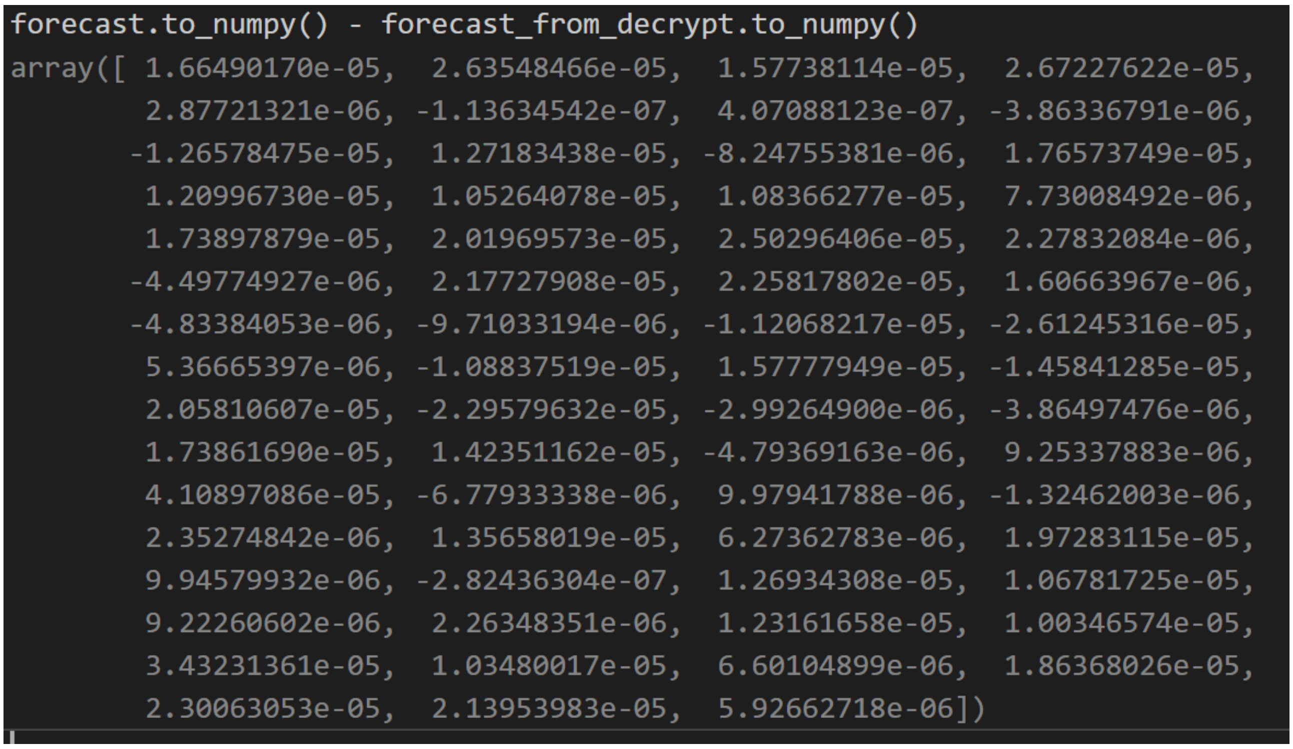


```
(base) mago@DESKTOP-MB2KTUH:~/Tesi/Tesi_final/pycrcnn$ /bin/python3 /home/mago/Tesi/Tesi_final/pycr
cnn/local_execution/local_execution_ts.py
Evaluation in progress...
Time to encode the net: 0.22344756126403809
Table results:
MAE FROM PLAIN DATA: 61.193323342257706 RMSE FROM PLAIN DATA: 91.06924399695346
MAE FROM DECRYPTED DATA: 61.19332513483098       RMSE FROM DECRYPTED DATA: 91.06924925171492
```

Figure 10: Comparison between the errors obtained running the model on plain and encrypted data.

Figure 10 shows how the CNN model performed predicting on the *Covid-19 daily deaths* with a time horizon of 7 days.
During the execution with plain data (i.e. not encrypted) the model used was the one from the PyTorch library

implementation, while the approximated and encoded one was used for the case with encrypted values.
The results are convincing because the MAE and RMSE obtained running the TCNN on plain or encrypted data ended up in having no differences between the two cases: this means that the approximated and encoded model executed exactly the same operations of the one coming from the PyTorch library.



```
forecast.to_numpy() - forecast_from_decrypt.to_numpy()
array([ 1.66490170e-05,  2.63548466e-05,  1.57738114e-05,  2.67227622e-05,
        2.87721321e-06, -1.13634542e-07,  4.07088123e-07, -3.86336791e-06,
       -1.26578475e-05,  1.27183438e-05, -8.24755381e-06,  1.76573749e-05,
        1.20996730e-05,  1.05264078e-05,  1.08366277e-05,  7.73008492e-06,
        1.73897879e-05,  2.01969573e-05,  2.50296406e-05,  2.27832084e-06,
       -4.49774927e-06,  2.17727908e-05,  2.25817802e-05,  1.60663967e-06,
       -4.83384053e-06, -9.71033194e-06, -1.12068217e-05, -2.61245316e-05,
        5.36665397e-06, -1.08837519e-05,  1.57777949e-05, -1.45841285e-05,
        2.05810607e-05, -2.29579632e-05, -2.99264900e-06, -3.86497476e-06,
        1.73861690e-05,  1.42351162e-05, -4.79369163e-06,  9.25337883e-06,
        4.10897086e-05, -6.77933338e-06,  9.97941788e-06, -1.32462003e-06,
        2.35274842e-06,  1.35658019e-05,  6.27362783e-06,  1.97283115e-05,
        9.94579932e-06, -2.82436304e-07,  1.26934308e-05,  1.06781725e-05,
        9.22260602e-06,  2.26348351e-06,  1.23161658e-05,  1.00346574e-05,
        3.43231361e-05,  1.03480017e-05,  6.60104899e-06,  1.86368026e-05,
        2.30063053e-05,  2.13953983e-05,  5.92662718e-06])
```

Figure 11: Differences between results from the two different networks

More in details, as shown in Figure 11, the intermediate results coming from the layers, both in the approximated and encoded or plain network, were almost identical (a difference in the order of 10^{-5}); this ended up in having exactly the same MAE and RMSE.
The main big difference is the execution time, considering the complexity of the network and all the encoding and encryption processes required for data, the time spent to make the same amount of predictions of the *plain* case was $97.50 secs.$ against $0.235 secs$ using the *plain* architecture.
We stress that the correctness of the encrypted processing was checked for all the related experiments, i.e, for all the MAEs and RMSEs shown in the row *TS-PyCrCNN without re-training* in Table 2.

7. Conclusions

The goal of this work was to propose a solution for time-series forecasting able to work with a deep learning algorithm focusing in the meanwhile on the privacy preserving concept. Convolutional Neural Networks applied to time series, namely TCNN, are the reference ML models used in this work.
The use of Homomorphic Encryption made possible to make strong guarantees on the confidentiality of processed data.
The process of adapting network for the encrypted case may involve the modification of several layers, the use of different activation function and can significantly impact on the computational performance and a possible loss in accuracy.
Several experiments were carried out to give to the reader an overview of some real world scenarios and some metrics about the computational overhead of a privacy-preserving solution.
Overall the aim of this thesis is to highlight how privacy problems related to technology are an important issues in the nowadays society. While Machine Learning algorithms and techniques are catching on, there's the needing to put even more effort into data security and the risks related to a misuse of that.

7.1. Future works

The proposed solution is thought to follow the paradigm of a DLaaS (Deep Learning as a Service).
What was presented is a possible solution that run "locally", hence a first extension of this work could be to actually deploy the same solution with a client-server architecture on a real Cloud infrastructure.
A second possible extension of the work is related to the choice of encryption parameters. This choice can severally impact on the computational time, for this an algorithm that automatically find the best encryption parameters would definitely mitigate this problem. It is possible to imagine an algorithm that takes in input some information about the task, a requested security level and a maximum amount of time while the constraint set of security level is respected. The result of such algorithm would be the best set of encrypted parameters to carry out the task.
Lastly, a third possible extension could be the implementation of the solution on top of a different deep learning algorithm, for example RNN (Recurrent Neural Network). As previously mentioned, such algorithms can possibly reach an higher accuracy but the non-linearity introduced by the operations involved and the number of operations required on data makes difficult the realization with HE, hence especially in this case the model has to be approximated.

8. Bibliography and citations

References

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[2] Melek Onen Alberto Ibarrondo, Laurent Gomez. Pyfhel: Python for homomorphic encryption libraries. `https://github.com/ibarrond/Pyfhel`, 2018.

[3] Adebiyi A Ariyo, Adewumi O Adewumi, and Charles K Ayo. Stock price prediction using the arima model. In *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, pages 106–112. IEEE, 2014.

[4] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. ngraph-he: a graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 3–13. ACM, 2019.

[5] Joppe W Bos, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Privacy-friendly forecasting for the smart grid using homomorphic encryption and the group method of data handling. In *International Conference on Cryptology in Africa*, pages 184–201. Springer, 2017.

[6] Joppe W Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy*, pages 553–570. IEEE, 2015.

[7] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

[8] Jason Brownlee. *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery, 2018.

[9] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008.

[10] Simone Disabato, Alessandro Falcetta, Alessio Mongelluzzo, and Manuel Roveri. A privacy-preserving distributed architecture for deep-learning-as-a-service. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[11] Matthew F Dixon, Igor Halperin, and Paul Bilokon. *Machine Learning in Finance*. Springer, 2020.

[12] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Manual for using homomorphic encryption for bioinformatics. *Proceedings of the IEEE*, 105(3):552–567, 2017.

[13] Issam El Naqa and Martin J Murphy. What is machine learning? In *machine learning in radiation oncology*, pages 3–11. Springer, 2015.

[14] Alessandro Falcetta. Privacy-preserving convolutional neural networks using homomorphic encryption. `https://github.com/AlexMV12/PyCrCNN`, 2021.

[15] Alessandro Falcetta. Timex, library for time-series-forecasting-as-a-service. `https://https://github.com/AlexMV12/TIMEX`, 2021.

[16] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[17] Protezione Civile italiana. Covid-19 italy dataset. `https://github.com/pcm-dpc/COVID-19`.

[18] Kim Laine. Simple encrypted arithmetic library 2.3.1. Technical report, Microsoft Research, WA, USA, 2017.

[19] Julia Lane, Victoria Stodden, Stefan Bender, and Helen Nissenbaum. *Privacy, big data, and the public good: Frameworks for engagement*. Cambridge University Press, 2014.

[20] Francis Quintal Lauzon. An introduction to deep learning. In *2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*, pages 1438–1439, 2012.

[21] Jason Brown Lee. Datasets. `https://github.com/jbrownlee/Datasets`.

[22] Jörg Liesen and Volker Mehrmann. *Linear algebra*. Springer, 2015.

[23] Wen-jie Lu, Shohei Kawasaki, and Jun Sakuma. Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. *Cryptology ePrint Archive*, 2016.

[24] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[25] George D Magoulas and Andriana Prentza. Machine learning in medical applications. In *Advanced course on artificial intelligence*, pages 300–307. Springer, 1999.

[26] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[27] Shinichiro Mori. Deep architecture neural network-based real-time image processing for image-guided radiotherapy. *Physica Medica*, 40:79–87, 2017.

[28] Alexandru Nicolae, Gerard Morton, Hans Chung, Andrew Loblaw, Suneil Jain, Darren Mitchell, Lin Lu, Joelle Helou, Motasem Al-Hanaqta, Emily Heath, et al. Evaluation of a machine-learning algorithm for treatment planning in prostate low-dose-rate brachytherapy. *International Journal of Radiation Oncology* Biology* Physics*, 97(4):822–829, 2017.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[30] Jestine Paul, Meenatchi Sundaram Muthu Selva Annamalai, William Ming, Ahmad Al Badawi, Bharadwaj Veeravalli, and Khin Mi Mi Aung. Privacy-preserving collective learning with homomorphic encryption. *IEEE Access*, 9:132084–132096, 2021.

[31] Rizwan Khan Priyanshu Srivastava. A review paper on cloud computing. *International Journals of Advanced Research in Computer Science and Software Engineering*, pages 17–20, 2018.

[32] Dörthe Schaue and William H McBride. Opportunities and challenges of radiotherapy for treating cancer. *Nature reviews Clinical oncology*, 12(9):527–540, 2015.

[33] Microsoft SEAL. `https://github.com/Microsoft/SEAL`. Microsoft Research, Redmond, WA.

[34] Nicu Sebe, Ira Cohen, Ashutosh Garg, and Thomas S Huang. *Machine learning in computer vision*, volume 29. Springer Science & Business Media, 2005.

[35] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015.

[36] farewell rnns welcome-tcns. `https://towardsdatascience.com/farewell-rnns-welcome-tcns-dd76674707c8`.

[37] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[38] Wikipedia. Time series description, 2021.

[39] Zijie Yue, Shuai Ding, Lei Zhao, Youtao Zhang, Zehong Cao, Muhammad Tanveer, Alireza Jolfaei, and Xi Zheng. Privacy-preserving time-series medical images analysis using a hybrid deep learning framework. *ACM Transactions on Internet Technology (TOIT)*, 21(3):1–21, 2021.

[40] G Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.

Abstract in lingua italiana

L'utilizzo di algoritmi di Intelligenza Artificiale e Machine Learning diventa ogni giorno più presente, con applicazioni che spaziano dai *smart home devices* fino al *digital health care*. Tuttavia, l'incremento nell'utilizzo di questi algoritimi - che richiedono sempre più dati per poter funzionare - han fatto scattare un campanello d'allarme per quel che riguarda le problematiche di privacy dei dati e protezione degli utenti.
Lo scopo di questa tesi è di presentare un architettura per il private time-series forecasting, ossia in grado di fornire previsioni su serie temporali fornite dagli utenti mentre viene garantita la privacy di questi ultimi. Al fine di raggiungere tale obiettivo, verrà usata la crittografia omomorfica, la quale consente di operare direttamente sui dati cifrati e impedendo l'accesso ai dati in chiaro.

Parole chiave: Deep Learning, Privacy preserving, Time-Series-Forecasting, Crittografia Omomorfica