Neural Networks for Encrypted Data using Homomorphic Encryption

MSc Advanced Computer Science

Project Report

Author:
Shreya Garge (1839643)

Supervisor:
Dr. David Galindo

September 11, 2018

University of Birmingham

School of Computer Science

Abstract

This report presents the background research, design and development, implementation, testing and results thereof of a Neural Network, implemented to work on encrypted data to provide inference using Homomorphic Encryption and evaluations. The project aims to develop a framework which enables preserving user privacy while using sensitive or confidential data for predictive analysis. This could be useful for subverting privacy concerns while providing Machine Learning as a cloud-based service. This enables the user to employ a predictive model held by a third party by submitting the data in an encrypted form, so private information is not compromised.

The background research includes a study of Machine Learning models (specifically neural networks), study of homomorphic encryption, homomorphic encryption schemes and the cryptographic foundations behind them. It also includes a study of the threats posed by the increasing pervasiveness of machine learning and predictive analysis in different fields. Finally a survey of existing work towards achieving privacy and security in Machine Learning is included.

The practical work includes the design and implementation of a neural network to classify handwritten digits. In order to achieve said privacy, the following protocol is followed : The images are encrypted by the client and sent for prediction to the server which has a trained model. The prediction model, a convolutional neural network which works on these ciphertexts, is implemented using homomorphic evaluation functions. The masked prediction sent back is then decrypted by the client. This way the server sees neither the data nor the prediction. The training network is implemented using TensorFlow and the inference network is implemented using SEAL library using Fan-Vercauteren Encryption scheme.

Keywords: Privacy in Machine Learning, Neural Networks, Homomorphic Encryption, Data-Privacy, encrypted statistical analysis, Privacy-preserving Inference models.

Acknowledgements

Firstly, I would like to thank Dr. David Galindo for his supervision on this project, and for all his timely advices and suggestions that helped me manage and organise the project, and for the many references and advices that he had provided me with, all of which were crucial in the development of this project. I would like to to thank Dr. Christophe Petit for his feedback and insights as part of inspection and presentation. I am also extremely indebted to Saif Sidhik for helping me with TensorFlow and Neural Networks. I'd also like to thank Milan Maria for her feedback and suggestions for improving the report. Finally, I would like to thank the School of computer science, University of Birmingham for providing me with the knowledge and opportunity to work on this project.

Declaration

The material contained within this thesis has not previously been submitted for a degree at the University of Birmingham or any other university. The research reported within this thesis has been conducted by the author unless indicated otherwise.

Signed ....

Contents

List of Figures

Chapter 1

Introduction

Advances in Machine Learning (ML) in the past few years have opened up a reeling array of applications in a wide variety of domains - Advertising, Finance, Health-care, Security, Autonomous systems, Robotics etc. Breakthroughs in the fields of ML and advances in computational and data storage capabilities have altered the landscape of technology. ML has become so pervasive that it is no longer unknown to most smart phone users that their data is constantly being collected for use by predictive models. With this comes the growing concern regarding user data privacy. This has also led to the understanding that ML creates and exposes new vulnerabilities in softwares that use them. For example, The data submitted to the models for prediction could be stolen or misused. Training data used by them could be tampered with, and modified to skew the model in favor of particular outcomes with malicious intent. Attackers could pose as clients to employ the services of the model, and try to steal the model for its intellectual value. Not all data used in these models is sensitive, but in fields like Finance and Health-care, both accuracy and maintaining data privacy become very important.

This project is a research and implementation of solutions to one of these problems - the prevention of misuse or theft of data submit by users for predictive analysis. To this end, the service provider allows the user to send private data in an encrypted form, and the server applies a predictive model and sends encrypted predictions back to the owner. He/she can then decrypt it to obtain the prediction. The main ingredients of building such a system are the predictive model and the encryption scheme. The choice of predictive model is what determines the accuracy. The encryption scheme needs to not only enable processing on encrypted data, but also be secure. Such encryption schemes are called Homomorphic Encryption (HE) schemes and are the most essential component of the system.

In algebra, Homomorphism is defined as "a map (function) preserving all the algebraic structures between the domain and range of the set". Similarly, Homomorphic encryption in cryptography strives to preserve the operations between the domain (plain texts) and the range (cipher texts). This kind of encryption scheme allows a service (e.g., on the cloud/ held by a third party), to perform computations on the encrypted data, preserving the features of both the function and the encryption. Although the theoretical foundation for homomorphic encryption dates well back to 1978, when Rivest, Adleman and Dertouzous who proved in [1] that it is possible to do computations on encrypted data without decrypting, the resulting encryption scheme was highly inefficient and impractical.

There was not much progress or a use case for such an encryption scheme until immense advances in Machine Learning started sparking privacy concerns among users. Craig Gentry first laid the foundation to building practical and efficient Homomorphic Encryption by introducing a technique called "bootstrapping". Since then, research in Homomorphic encryption has seen a significant rise. In the recent past, many somewhat practical HE schemes have been proposed, including the scheme used in this project proposed by Fan and Vercauteren.

1.1 Problem Statement

The setup considered consists of a client and a server. The server is willing to provide a computationally expensive prediction service for client's data. The client wants to employ this service *without* revealing the contents of this data. With the intention of solving this problem, image recognition (classification of handwritten digits) is chosen as the service provided by the server. It uses a Convolutional Neural Network (CNN) model to accomplish this task. The server has trained the network and learned parameters w of the model, and the client has an image x, previously unseen by the server. The client submits encryption of x, $\text{Enc}_{pk}(x)$ to the server, which runs a classifier C using model w on $\text{Enc}(x)$ and returns $C(w, \text{Enc}_{pk}(x))$.The client receives it and obtains the classification by decrypting : $p = \text{Dec}_{sk}(C(w, \text{Enc}_{pk}(x)))$. The server does not learn anything about the image x or the prediction p. The client does not learn anything about the server's model parameters w.

The model and the encryption scheme need to be **efficient**(Running time of the prediction network needs to be small enough to be practical for real world applications).The model also needs to be **accurate**(The performance needs to be comparable to that of the regular models). The encryption scheme needs to be **secure**.

The remaining part of the report is organized as follows - Chapter 2 covers the basic theoretical aspects of the CNN model that will be useful for justifying the choices made in the implementation and to explain the results. Chapter 3 introduces the basic concepts of Homomorphic encryption, the hard problem underlying the scheme used in this implementation. The FV scheme is introduced on a high level, and the parameters that need to be set are explained.

Chapter 4 is a literature survey and some background research on Homomorphic encryption and its applications in the context of privacy in ML done as part of the research building up to the implementation in this project. A review of existing work in this area is also included.

Chapter 5 describes the design and implementation details, models used, supporting mathematics, and a summary of the flow of the implemented system. Chapter 6 presents an analysis of the experiments with the network, and their results. The report is concluded in Chapter 7, with a discussion on the areas of improvement and possible future work.

Chapter 2

Preliminaries : Neural Networks

Machine Learning is a method of automating analysis on large data sets and building analytical models. Machine Learning tasks are usually one of the three different types - *Reinforcement Learning*, where the algorithm interacts with an environment through a sequence of actions, for which it receives rewards or penalties [2]. The model learns by trying to maximize the cumulative reward. *Unsupervised Learning*, where the algorithm tries to look at the given data and find patterns in it, without having any prior knowledge about the data[3]. *Supervised Learning*, where the model tries to find a mapping between the input and output, using a model it had learned by looking at examples in the form of input data labeled with corresponding outputs. The outputs can be categorical *(classification problem)* or real-valued *(Regression problem)* [4].

On a high level, a Neural Network is a computational model based on a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections [5] .The important components of neural networks are :

- A set of processing units, called neurons or nodes.
- Connections between the units. They represent functional dependencies (i.e, an edge from x to y indicates y is a function of x). Generally each connection is associated with a "weight".
- External inputs "bias/offset" for each unit.
- A propagation rule, which determines the effective output of a node from its internal and external inputs.

Neural network architecture is nothing but a systematic arrangement of layers. A layer is a set of processing units (nodes). Each node performs a simple job - receive inputs from other nodes (and external biases) to compute an output which is propagated to further nodes. There are three types of layers : the input layer - nodes in this layer just receive the input into the network, output layers - the nodes in this layer send results out of the network. Other than the input and output layers, for a neural network to be practical, there needs to be at least one layer in between. These are called *hidden layers* - they take input from, and give output to layers within the network.

In most cases, the propagation rule used in the network is the standard weighted summation - each node provides an additive contribution weighted by the connection to the nodes it is connected to. In the literature of standard neural networks, the output of every node is referred to as "activation". Every node applies some non-linear function on this weighted sum of inputs from other nodes, and the result is called the activation of that node.

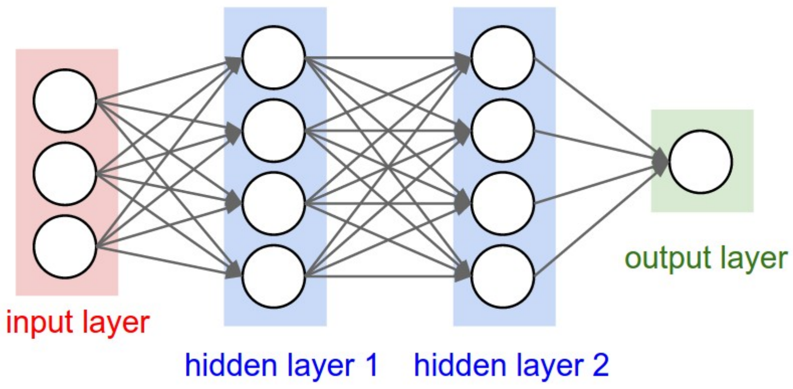


Figure 2.1: Illustration of a three layer Network.

Source: Neural Networks
Retrieved from: `http://cs231n.github.io/neural-networks-1/`

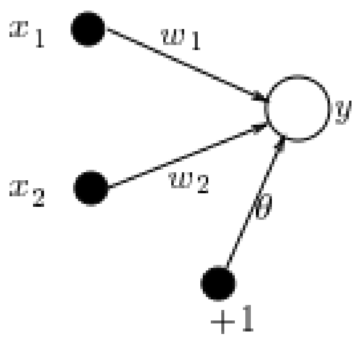


Figure 2.2: Illustration of a single node

Source: [9]

For the node in figure2.2, the output y would be

$$y = F\left(\sum_{i=1}^{2} w_i x_i + \theta\right)$$

However in this project, activation is treated as a separate layer (For simplicity in the encrypted realm), and the operation at a single node is just the weighted summation.

2.1 Convolutional Neural Networks

The problem used in this project, is the supervised learning problem of classification. Handwritten digits are classified into classes 0-9. While the design of input and output layers of the network is quite straightforward. (For example, as the input image is made up of 28×28 pixel values, the input layer would contain $28 * 28$ nodes. Since the task is to classify what the digit is, the output layer would consists 10 nodes, each representing one digit from $(0 - 9)$. The values in the output node determine which digit is the winner). But the design of hidden layers is often challenging and requires a lot of research. The handwritten image recognition task itself has been solved with very high accuracy using convolution layers in the network. Such neural networks, with a convolution layers after the input layer are called Convolutional Neural Networks (CNN)s. Figure 2.3 shows the most popular sample architecture of CNN used in many image recognition tasks with high accuracy. These networks are inspired by the biological working of the animal visual cortex

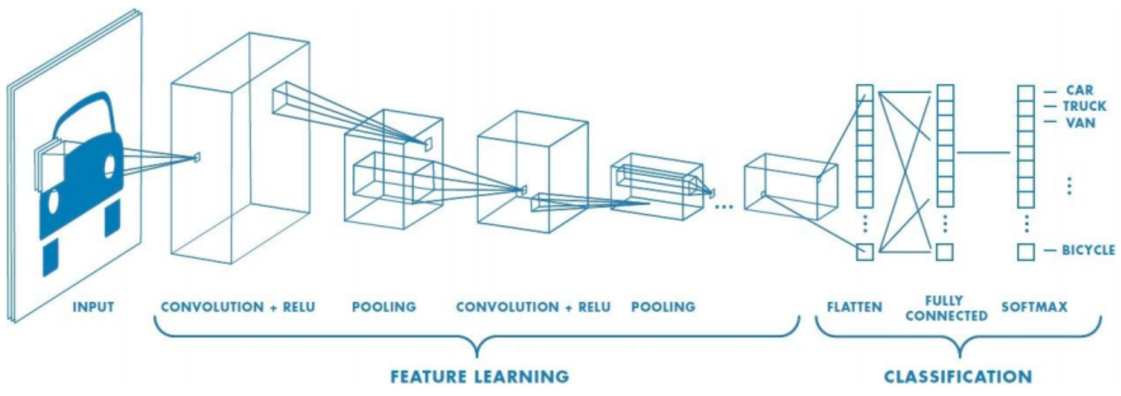


Figure 2.3: A sample CNN architecture

Source: Introduction to Convolutional Neural Networks
Retrieved from: `https://web.stanford.edu/class/cs231a/lectures/intro_cnn.pdf`

[6]. They have been proven to be effective in image classification because spatial topology is captured well by CNNs unlike standard networks. It means - if a standard network is trained on a dataset in which all the input pixels are a fixed permutation of the original dataset, the results will be identical [7].
The following sections present some theoretical aspects of CNNs on a high level, which will be used to justify the choices made in the design of the network used in the project.

2.2 Layers

2.2.1 Convolution Layer

The main purpose of this layer is to *extract features* from the inputs. A convolution is performed on the inputs with a filter (or kernel) to produce a *feature map* (output of convolution). Convolution is executed by sliding the filter over the input. A filter is just a matrix of weights and a stride. Stride is the size of the step the convolution moves every time. (i.e., if the stride is 1, the filter slides pixel by pixel). At every stop of the filter, a matrix multiplication is performed. The results are then summed onto the feature map. Numerous convolutions can be performed on a single input using numerous filters. It helps in extracting more than one features from the input. Zero-value (or same value as the edge) pixels can be added around the input image, so that the feature maps are not smaller in size than the input. This is called *padding*. This makes sure the kernel and stride on convolution with the input give output of the same size. This layer involves computation of a high number of matrix multiplications (depending also on the input size and filter size and stride length), thereby making it the most computationally expensive layer. While using a CNN, the important *Hyperparameters* to choose are :

- Kernel size.
- Filter count.
- Stride.
- Padding.

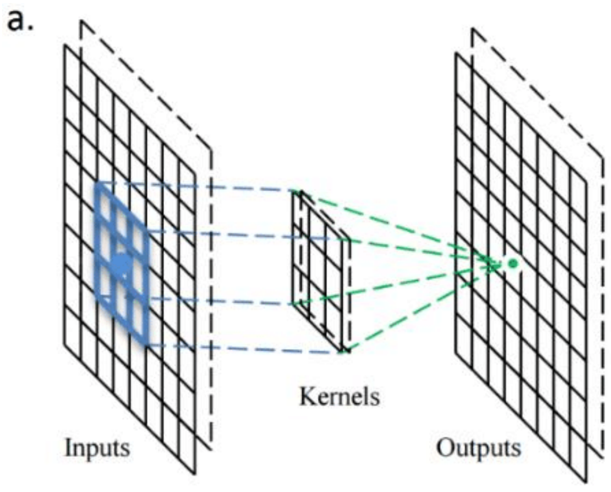


Figure 2.4: Illustration of convolution.

Source: Convolutional Neural Networks
Retrieved from: `https://drive.google.com/drive/folders/1lKGigxFGnhvlKE7MSRW5UQZaVqixcKJ_`

2.2.2 Activation Layer

The convolutional layers and fully connected layers are basically multiplication with weights followed by addition with biases. So in these layers, all the operations are linear. (of the form $w * x + b$). To introduce non-linearity into the network, activation layers are used. These layers do not have any variables like weights or biases, they only have a function. The neurons in this layer apply a single function to the input and pass the result to the next layer.

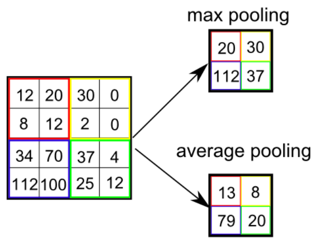
Activation functions are usually functions like sigmoid, hyperbolic tangent (tanh), Rectified linear operation(RelU) etc. Without a non-linear activation function, even with many layers in the network, summing up these layers would just give a linear function, and hence cannot be used to create effective models for non-linearly separated data. However, when working with encrypted values ReLU or Tanh or the other usual activation functions cannot be used due to limitations introduced by homomorphic evaluations (covered in the next section). Either polynomial approximations to these functions or other non-linear polynomials that work for the use-case can be used instead.

2.2.3 Pooling Layer

Pooling layers are commonly added between convolutions to serve the purpose of down-sampling.(Reducing the dimensionality). This serves the purpose of decreasing the number of parameters (weights and biases), and shortening training time. They also help in controlling over-fitting to an extent.

The most popular types of pooling are max-pooling and average-pooling, illustrated in figure2.5. In Max-Pooling, only the maximum value from the window is taken as the output, and the other values are discarded. In average(mean)-pooling, the mean of all the values in the window is taken as the output. Max-Pooling is effective in the extracting sharp and most important features, where as average pooling does the extraction smoothly [6].

Figure 2.5: Mean and max pooling.



2.2.4 Fully Connected Layer

After going through convolution, pooling and activations, for the final high level reasoning in the network, fully connected layers are used in the end. This layer can be viewed as the nodes from a regular neural network, which perform a weighted sum of the inputs and add bias.

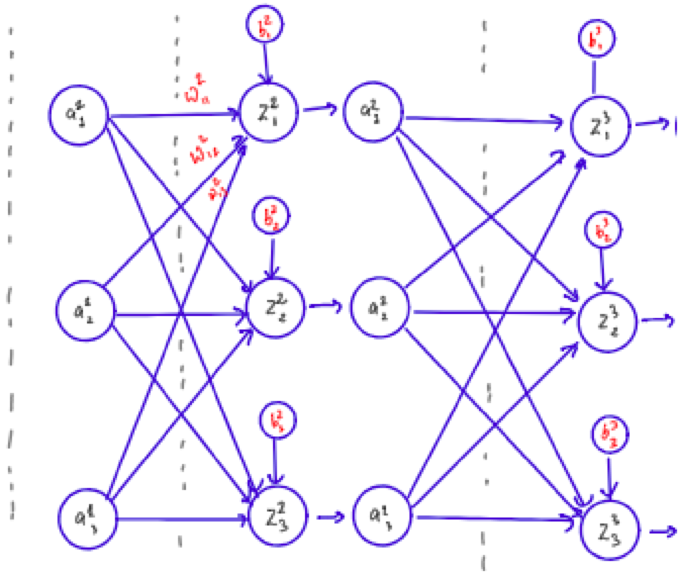


Figure 2.6: Fully connected layer.

Nodes in this layer are connected to every single node in the previous layer2.6. Theoretically, it is the final learning phase which maps the visual features extracted by convolution to desired outputs. [5].

2.2.5 Softmax

Softmax layer is used as the final layer in classification problems with discrete class labels. This is because it is more appropriate to have one output p_j per class j. p_j is interpreted as the probability of class j for the input. Softmax assigns decimal probabilities that add up to 1, to each class (figure 2.7). The output p_j for node j is given by

$$p_j = \frac{e^{z_j^L}}{Q} \qquad Q = \sum_{k=1}^{m} e^{z_k^L}$$

where

m Is the number of classes

z_k^L is the output of node k of the last layer (L)

The node with the highest probability value is decided as the most likely class for the given input.

2.3 Loss Function

After the input goes through all the layers in the network followed by the softmax layer, some values are obtained at the output layer. The purpose of loss function is to tell how

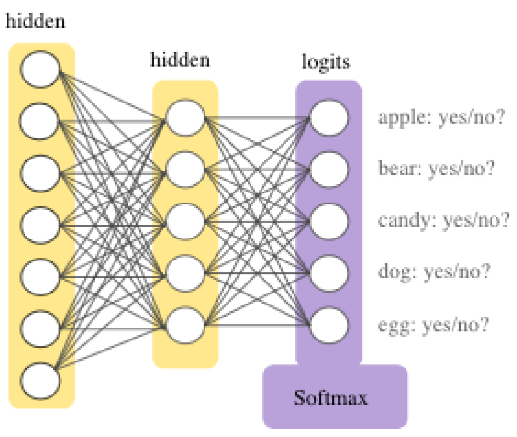


Figure 2.7: Softmax Layer.

Source: Multi-Class Neural Networks: Softmax
Retrieved from:
`https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax`

good these values are, for the classification task at hand. If the values are good, it means the weights and biases in the previous layers have been good. Specifically, the loss function compares the output with the required result (it knows what the result should be - from labels in the training samples). It measures the amount of inconsistency between the value predicted by the network ($\hat{y}$) and the actual label (y). Different types of loss functions can be used such as Mean Squared error, L_2 loss [8] ,KL Divergence, cross entropy etc. High loss usually means the weights and biases are way off and need a lot of change. This information is passed on back wards into the network using backpropagation.

2.4 Back-Propagation

Back-propagation is the algorithm used to reset the weights in the network based on the loss function results. The gradient (direction of steepest ascent) of the loss function with respect to the network's weights is calculated. Since it is required to move in the direction that minimizes the loss function, the weights and biases are reset by shifting them in the opposite direction of the gradient. This process is called (Stochastic) gradient descent (SGD). The gradient at the final (output) layer is found by finding the partial derivatives of the Loss function with respect to the weights in the previous layer. In the inner layers, local gradients are calculated with respect to outputs of the next layers - these are called *local gradients*. Starting from the final layer, the loss is calculated. Based on the gradient of this, the previous layer weights are modified and local gradients are propagated backwards to allow the flow of loss information to the previous layers. This allows efficient computation of gradient at each layer, as opposed to the naive way of calculating them at each layer separately [9].

2.5 Regularization

Over-fitting is a serious problem when it comes to deep networks with many layers. It happens when the model fits to the training data too closely. The problem with this is that the model tries to fit even the outliers in training data, and in the process, the decision boundary becomes too complex and far from the real boundary (figure 2.8). This leads to having a very high training accuracy but when new samples are classified according to this boundary, they will get classified wrongly. Thus the testing accuracy becomes low.

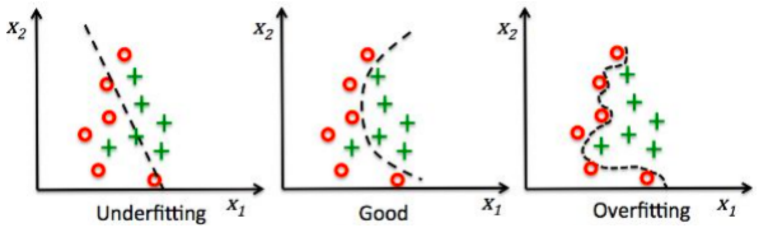


Figure 2.8: Illustration of Over-fitting in classification

Source: Introduction to Convolutional Neural Networks
Retrieved from: `https://web.stanford.edu/class/cs231a/lectures/intro_cnn.pdf`

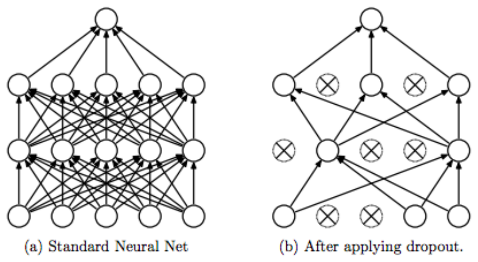


Figure 2.9: Illustration of using dropout in neural networks.

Source: [10]

Dropout is one of the most effective regularization techniques used to deal with the problem [10]. In this method, every node in the layer except the ones in the output layer, behave like a draw from a Bernoulli distribution with probability p. Each node is present in the network with a probability $1 - p$. This results in us working with a different sub network in each batch, and the weights and biases obtained finally are an average of the weights and biases obtained from each sub network. In each batch during training, some randomly selected activation units are deactivated. i.e, their weights and biases become zero. This is done only during the training phase. During testing, the average of weights obtained by training all subnetworks is used. to implement dropout, the function at every node is replaced with a dropout version [10].

$$a_j^l = \frac{1}{1-p} d_j^l \; \phi(z_j^l)$$

where

$d_j^l \sim Bernoulli(1-p)$

a_j^l is the output of node j in layer l

z_j^l is the input to said node.

ϕ is the original function the node is meant to perform.

2.6 Optimization

One of the problems with Stochastic gradient descent (SGD) is that the optimization time is strongly impacted by learning rate (figure 2.10). If the learning rate is too high,

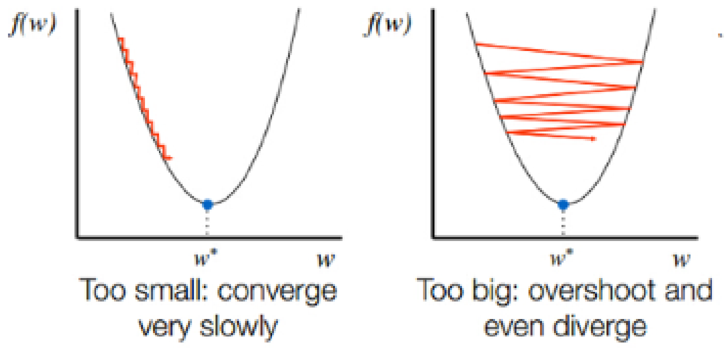


Figure 2.10: Influence of learning rate on gradient descent.

Source: Github tutorial
Retrieved from: `http://srdas.github.io/DLBook/GradientDescentTechniques.html`

the weights get shifted by a large amount, which leads to the weights overshooting the minimum, oscillating around it and never reaching it. If the learning rate is too low, the weights make very small shifts and take too long to reach the optimum [11].

This is shown in the figure 2.10. Hence, it is necessary not only to adjust the rate according to the problem, but also adaptively change how much the weights shift as the training progresses. i.e, The learning rate needs to decrease as the model parameters (weights and biases) get closer to the optimum.

There are several algorithms in literature that do this - SGD with momentum [12], adagrad, adam etc. Adam optimizer [11] is used in this implementation. The difference becomes clear from the algorithms of SGD and adam, mentioned here.

Algorithm 1: Stochastic gradient descent

```
Input: Loss Function $J$, Learning rate $\epsilon$
Result: Optimum weights $w$
1 $w \leftarrow random$;
2 while Termination condition do
3     $w \leftarrow w - \epsilon.\nabla J(w)$;
4 end
```

Algorithm 2: Adam optimizer [11]

```
   Input: Loss Function $J$, Learning rate $\epsilon$
   Result: Optimum weights $w$
 1 $w \leftarrow random$;
 2 $r, s, t \leftarrow 0$;
 3 $\rho_1 \leftarrow 0.9$;
 4 $\rho_2 \leftarrow 0.999$;
 5 $\delta \leftarrow 10^{-8}$;
 6 $\epsilon \leftarrow 0.01$;
 7 while Termination condition do
 8     $t = t + 1$;
 9     $g = \nabla_w J(w)$;
10     $s = \rho_1 s + (1 - \rho_1) g$;
11     $r = \rho_2 r + (1 - \rho_2) g \odot g$ ;
12     $\tilde{s} = \frac{s}{1-\rho_1^t}$ ,   $\tilde{r} = \frac{r}{1-\rho_2^t}$;
13     $v = -\epsilon . \frac{\tilde{s}}{\sqrt{\tilde{r}}+\delta}$;
14     $w \leftarrow w + v$
15 end
```

As can be seen from the pseudo-codes, SGD maintains a single learning rate through out the training. But in Adam, a learning rate is maintained for each learning parameter and separately adapted as the learning progress, making it more effective in finding the optima.

Chapter 3

Preliminaries: Homomorphic Encryption

A homomorphism is nothing but a "structure preserving transformation". As an example, consider the function $\varphi : \mathbb{Z} \to \mathbb{Z}_q$ such that $\varphi(z) = z \mod q$. It gives :

$$\varphi(z_1 + z_2) = \varphi(z_1) \oplus \varphi(z_2) \quad \text{and} \quad \varphi(z_1 \times z_2) = \varphi(z_1) \otimes \varphi(z_2) \quad \forall z_1, z_2 \in \mathbb{Z}$$

where $\oplus$ and $\otimes$ are addition and multiplication respectively in $\mathbb{Z}_q$ (addition and multiplication modulo q). The map φ is called a *ring homomorphism* between the rings $\mathbb{Z}$ and $\mathbb{Z}_q$. Similarly, homomorphic encryption preserves the operational structures between the rings of the plain texts and cipher texts in applying the encryption and decryption functions. Homomorphic encryption (HE), on high level, is an encryption scheme that allows operations on encrypted data preserving its structure as follows:

$$Dec_{sk}(f(Enc_{pk}(a), Enc_{pk}(b))) = f(a, b)$$

$$Dec_{sk}(f(Enc_{pk}(a), b)) = f(a, b)$$

It is also important that this encryption is also as secure as the normal encryptions, even though it is possible to do homomorphic evaluations on the encrypted data.

As an example, consider the plain version of RSA. It can be seen that deterministic version of RSA has the property of being multiplicatively homomorphic. i.e., for two messages m_1, m_2 and Public key $pk = (N, e)$, their encryptions $c_1 = m_1^e \mod N$ and $c_2 = m_2^e \mod N$,

$$\text{Enc}(m_1) \times \text{Enc}(m_2) = (m_1 \times m_2)^e \mod N = \text{Enc}(m_1 \times m_2)$$

Thus, it is intuitive to see and build homomorphisms using deterministic encryptions, but the obvious reason why it is not feasible is because the scheme would not be IND-CPA secure. To protect the cryptosystem from CPA attacks, a random component is needed in the encryptions, so that when the same message is encrypted twice, it does not yield the same cipher texts with a high probability. This creates a new problem as the random component, called "noise" grows as more operations are performed on the cipher texts. When this noise grows beyond a certain limit, the message gets corrupted and is rendered undecipherable upon decryption. In this section, the theory on how secure Fully homomorphic encryption schemes are built and effected by noise is discussed. These concepts will later be used to justify the choices in parameters and encryption schemes.

3.1 Homomorphic Encryption - Definition

Any encryption scheme by design, has three operations it needs to support - Key generation, Encryption and Decryption. Homomorphic encryption needs a fourth operation, Evaluate.

DEFINITION. *An encryption scheme is homomorphic for some functions $\circ \in \mathbb{F}_M$ in the message space if it is possible to have functions $\diamond \in \mathbb{F}_C$ in the cipher text space satisfying:*

$$\mathrm{Dec}_{sk}(\mathrm{Enc}_{pk}(m_1) \diamond \mathrm{Enc}_{pk}(m_2)) = m_1 \circ m_2$$

$$\forall m_1, m_2 \in M$$

Where M is the set of all possible messages

The algorithm has a set $\mathbb{F}_M$ of all such functions, that can be evaluated homomorphically. However, the functions in $\mathbb{F}_M$ and $\mathbb{F}_C$ do not necessarily correspond. For example, Paillier scheme described in [13] is additively homomorphic, but $\mathbb{F}_M = \{+\}$ and $\mathbb{F}_C = \{\times\}$. In order to create a scheme that allows homomorphic evaluation of arbitrary functions, it is sufficient to have a scheme that is capable of performing homomorphic addition and multiplication arbitrary number of times. Because then polynomials can be computed, and any suitably smooth function can be closely approximated (in principle). On a bit wise level, multiplication corresponds to AND and additions to XOR. This means if they can be computed on encrypted bits, it should be possible to compute any function on encrypted data (*only* in theory, in practice it would be highly impractical). This is what FHE tries to achieve. For an encryption to be fully homomorphic, it needs to be able to handle an arbitrary number of such evaluations. Meaning, circuits of AND and XOR gates of *any* depth can be evaluated, making it possible to evaluate *any* function.

DEFINITION. *An encryption scheme can be called Fully homomorphic encryption (FHE) if it can "encrypt 0 and 1, and ADD and MULTIPLY encrypted data.* This however, was feasible only in theory. In practice, the problem is that the small "noise" term added to make the schemes non-deterministic, grows under Homomorphic evaluations - especially multiplications - constituting a major obstacle to designing practical fully homomorphic encryption schemes. After certain number of operations the noise will have spilled over into the message. So only operations up to a certain degree can be performed. These are called Somewhat Homomorphic encryption schemes(SWHE). They can evaluate only a certain number of operations, for example, low degree polynomials. If a scheme is somewhat homomorphic, and then the noise can be somehow reduced when it has grown too large, Fully homomorphic encryption schemes can be constructed.

The first practical FHE was proposed in 2009 by Craig Gentry using lattice-based cryptography, and it described the plausible construction for a fully homomorphic encryption scheme by being able to lower the noise in the cipher texts [14]. This construction consists of three parts:

- constructing an encryption scheme that is *somewhat homomorphic.*
- simplifying the decryption function of this scheme as much as possible.
- Third, evaluating this simplified decryption function homomorphically to obtain cipher texts with reduced noise (called bootstrapping).

Finding a bootstrappable algorithm that was capable of adding and multiplying homomorphically made HE practical and opened doors to a cascade of Fully homomorphic encryption schemes. The first variants of Gentry's scheme such as [15], [16], [17] and others mostly followed a similar structure. Some recent schemes avoid squashing the decryption circuit (the second step) all together and can bootstrap by evaluating the decryption circuit as it is. The security of these schemes is based on the Learning with Errors (LWE) problem or its ring variant. This serves as the hard problem underlying the construction of cryptosystems with HE. In the following subsection, the notation used through out the report is detailed and the LWE problem is introduced in the next section, with little detail.

3.2 Notation

- $\mathbb{Z}_q$ denotes the set of integers $\{n : n \in \mathbb{Z}, -q/2 < n \leqslant q/2\}$, $[a]_q \in \mathbb{Z}_q$ denotes the integer $(a \mod q)$
- $\mathbb{Z}[x]$ and $\mathbb{Z}_q[x]$ are polynomials in x with their co-efficients belonging to $\mathbb{Z}$ and $\mathbb{Z}_q$ respectively.
- R denotes the polynomial ring $\mathbb{Z}[x]/\Phi_{2^d}(x)$ and R_q denotes $\mathbb{Z}_q[x]/\Phi_{2^d}(x)$
- $\Phi_{2^d}(x) = x^{2^{d-1}} + 1$, the $2^d - th$ cyclotomic (irreducible) polynomial.
- The discrete Gaussian probability distribution $D_{\mathbb{Z},\sigma}$ over $\mathbb{Z}$ from $-B$ to B ($B \simeq 10\sigma$) assigns a probability proportional to $exp(\frac{-\pi x^2}{\sigma^2})$ to each integer.
- The noise terms used in the scheme come from distribution χ on R defined from $D_{\mathbb{Z},\sigma}$. χ is defined by simply sampling co-efficients according to $D_{\mathbb{Z},\sigma}$ for the polynomial $f(x)$, but only if $f(x) = x^{2^d} + 1$. For other cyclotomic polynomials, sampling from χ is more involved [21]. This is why polynomials of the form Φ_{2^d} are used, although any monic irreducible polynomial could be used in theory.

3.3 The Underlying Hard Problem

RLWE (Ring Learning with Errors) problem was introduced by Lyubashevsky, Peikert and Regev [18]. It is simply a ring based analogue of a similar problem, Learning with Errors(LWE). It was introduced by Oded Regev in [19]. It is one of the hardest problems to solve even in a post quantum world.

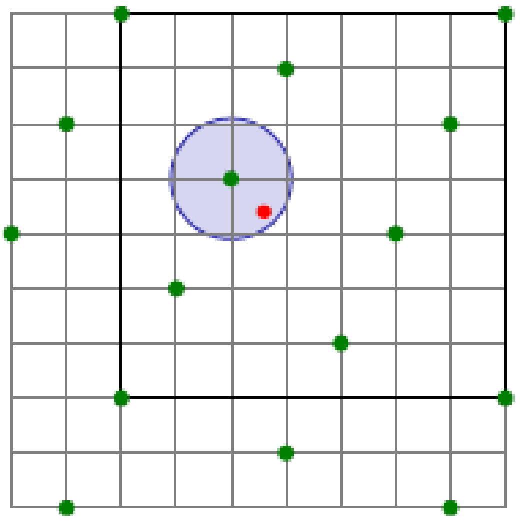
Search LWE: Find $s \in \mathbb{Z}_q^n$ (called the secret), given 'random noisy inner products'

$$\begin{aligned} a_1 \in \mathbb{Z}_q^n, &\quad b_1 = (s \cdot a_1) + e_1 \mod q \\ a_2 \in \mathbb{Z}_q^n, &\quad b_2 = (s \cdot a_2) + e_2 \mod q \\ &\quad \vdots \end{aligned}$$

Errors $e_i \sim \chi$ where χ is an error distribution over $\mathbb{Z}$

Decision LWE: Distinguish $(a_i, b_i = s \cdot a_i + e_i)$ pairs from uniform (a_i, b_i) pairs.

NOTE: Without the error terms from χ, the LWE problems are easy to solve, as the secret s can be simply recovered from the LWE samples by simple Gaussian Elimination. If m independent such samples are given, Search LWE can be seen as a bounded distance decoding problem on m-dimensional integer lattice [20]. To explain, the vector b is a dot-product with a perturbed by a small noise. So the vector b is intuitively 'close' to exactly one vector in the LWE lattice, (figure 3.1) and the problem is to find this lattice vector. Following the success of LWE, a ring based version of the problem, ring-LWE (RLWE) problem was introduced by



The red point is the perturbed point b, and the nearest green point to it is a

Figure 3.1: LWE as a lattice bounded distance problem

Lyubashevsky. The RLWE problem is an algebraic variant of LWE, which is more efficient and is reducible to worst-case problems on ideal lattices. It is also hard for polynomial-time algorithms even on quantum computers [18].

Search RLWE : With $\mathbf{s} \in R_q$ (called the secret), given m independent samples $(\mathbf{a}_i, \mathbf{b}_i = \mathbf{s} \cdot \mathbf{a}_i + \mathbf{e}_i \mod q) \in R_q \times R_q)$, find s.

Decision RLWE: Given m independent samples $(\mathbf{a}_i, \mathbf{b}_i) \in R_q \times R_q$, distinguish if they are tuples of the form $(\mathbf{a}_i, \mathbf{b}_i = \mathbf{s} \cdot \mathbf{a}_i + \mathbf{e}_i \mod q)$ or uniform $(\mathbf{a}_i, \mathbf{b}_i)$, where $\mathbf{a}_i \in R_q$, $\mathbf{s} \in R_q$ and $\mathbf{e}_i \sim \chi$, an error distribution over R

3.4 The Fan-Vercauteren Homomorphic Encryption Scheme

This encryption scheme (FV) is directly based on the decision version of the RLWE problem. The plain text space M of this scheme is the ring R_t. The Cipher text space C is $R_q \times R_q$, where $q >> t$. With $\Delta = \lfloor \frac{q}{t} \rfloor$, then : $r_t(q) = q \mod t \implies q = \Delta \cdot t + r_t(q)$.
NOTE : in this scheme, neither q nor t need to be prime [21].

The encryption scheme is defined as follows :

Key Generation:

The secret key k_s is generated by a uniform random draw from R_2 (by sampling a 2^{d-1} binary vector for polynomial coefficients.)

The public key k_p is a vector of two polynomials:

$$k_p = (k_{p1}, k_{p2}) \coloneqq ([-(a \cdot k_s + e)]_q, a) \in R_q \times R_q$$

where $a \sim R_q$ and $e \sim \chi$

The secret cannot be recovered from k_p due to the hardness of decision RLWE problem.

Encryption :

Encryption of message $m \in R_t$ with public key k_p renders a cipher text which is a vector of two polynomials

$$c = (c1, c2) := ([k_{p1} \cdot u + e_1 + \Delta \cdot m]_q, [k_{p2} \cdot u + e_2]_q) \in R_q \times R_q$$

where

$$u, e_1, e_2 \sim \chi \text{ and } \Delta = \lfloor \frac{q}{t} \rfloor$$

Decryption :

m can be recovered as :

$$m = \left[\left\lfloor \frac{t[c_1 + c_2 \cdot k_s]_q}{q} \right\rceil\right]_t \in R_t$$

Addition :

Addition of messages can be evaluated in cipher texts as the addition of the respective polynomials, followed by modulo reduction.

$$\vec{c_1} + \vec{c_2} = ([c_{11} + c_{12}]_q, [c_{12} + c_{22}]_q)$$

Multiplication :

However, multiplication is not as straight forward.

$$\vec{c_1} \times \vec{c_2} = \left(\left[\left\lfloor \frac{t(c_{11} \cdot c_{21})}{q} \right\rceil\right]_q, \left[\left\lfloor \frac{t(c_{11} \cdot c_{22} + c_{12} \cdot c_{21})}{q} \right\rceil\right]_q, \left[\left\lfloor \frac{t(c_{12} \cdot c_{22})}{q} \right\rceil\right]_q \right)$$

3.4.1 Relinearization :

The multiplication operation increases the cipher text vector length. Although it is possible to decrypt with the cipher text size more than 2, just by modifying the decryption function a little [31], for efficiency it is better to perform a procedure called relinearization, which resets the cipher text back to size 2. The mathematics behind this function is left out of the report as it is involved mathematically and not the primary focus of the project. It is described in the paper [21]. SEAL library's implementation of relinearization has been used in this project.

In order for this scheme to be secure, the decision RLWE problem stated above needs to be hard. Another point to be noted is that in the cipher text, the message is embedded into the $\log_2(t)$ most significant bits of the first polynomial [21] . The noise starts from the least significant bits. Under repeated operations, if it grows beyond this and enters into the $\log_2(t)$ bits, the cipher texts become corrupted.

Chapter 4

Practical HE Schemes and Privacy in ML - A Literature Review

The "idea" of Homomorphic encryption and its first theoretical foundation was put forward by Rivest, Adleman and Dertouzos in their paper dated as far back as 1978. They introduced and discussed a way in which third parties could work on user data in an encrypted form. This is similar to the use-case of privacy friendly cloud computation today, but they obviously used different terminology [1]. Following this, there have been several attempts to design such schemes, earliest of which date back to 1982, when Yao's garbled circuit [22] was proposed as a way to solve the Millionaire's problem (Two millionaires want to decide who has more money without revealing the amounts to each other).

There are three different types of HE schemes in the literature : Partially homomorphic encryption (PHE) schemes, which support either addition and multiplication but not both. Somewhat Homomorphic encryption (SWHE) schemes which support limited number of operations and fully homomorphic encryption (FHE) schemes support arbitrary functions. Some well-known schemes of these types have been summarized in the following sections but not in detail as they were a part of the research building up to the project but not its main focus.

4.1 Partial Homomorphic Encrytpion Schemes

There have been Partially homomorphic encryption schemes in the literature for a long time. These were the earliest of the HE schemes. For example, the Goldwasser and Micali scheme [23], El-Gamal [24], Cramer's variation of El Gamal [25] , Paillier [13] etc. However all of these schemes had limited applications as they are capable of performing either multiplications on additions homomorphically, but not both. Additively homomorphic

	Operation Supported	
Scheme	**Add**	**Mult**
RSA	no	yes
Goldwasser and Micali	yes	no
ElGamal	no	yes
Modified ElGamal (Cramer)	yes	no
Paillier	yes	no

Table 4.1: Homomorphic operations supported by some PHE schemes

schemes like Paillier are still in use as they have found a place in applications like e-voting, where addition is the main requirement. However, they cannot be used for broader applications because they can perform only either of addition or multiplication. That means very few functions can be evaluated homomorphically. To compute polynomials etc, the scheme needs to be able to perform additions and multiplications simultaneously. This was an open problem for a long time. Boneh et al's scheme in [26] came closest to being able to do both. It allows unlimited additions but only a single multiplication.

4.2 Somewhat Homomorphic Encryption Schemes

In the literature of Homomorphic Encryption, the earliest of schemes capable of performing both additions and multiplications simultaneously is the Polly Cracker scheme [27]. However, it was not practical because the size of cipher texts grows exponentially under homomorphic operations. Several SWHE schemes were proposed following this. SWHE schemes are the ones capable of evaluating circuits of *limited* depth. It means polynomials can be evaluated, but only the ones of lower degrees. However none of the schemes at that time were remotely practical, neither was there any real use-case for this sort of encryption in the industry. This changed with the advent and advancements in the fields of Big Data, Machine Learning and Cloud Computing, and the privacy concerns raised by them [28]. Around the same time, in 2009, came Gentry's seminal work [14], which laid the first foundations for a practical non deterministic scheme capable of simultaneous additions and multiplications.

4.3 Fully Homomorphic Encryption

Gentry's work [14] not only proposes an FHE scheme but also puts forth a framework to build them. Now also having found an application in the real world, this led to a whole new era in the research of HE. Since then, many schemes have been designed by several researchers following this framework. Gentry's proposed FHE scheme is based on ideal lattices, and the security of the newer FHE schemes in mostly based on the hard problems on lattices. There are mainly three families of FHE schemes in the literature - schemes over integers(Van Dijk [15]), (Ring) Learning With Errors based schemes (Brareski-Vaikuntanathan [29], Fan-Vercauteren [21]), NTRU-like schemes (Lopez-Alt [30]). The works similar to Gentry's first scheme (over integers) constituted the 'first-generation' of somewhat practical modern HE schemes.

The work by Brareski et al in [29] started a new generation of schemes based on the LWE problem. These schemes are based on better understood hardness assumptions of lattice problems [31]. They include the schemes in **brakerski2014fully** , [21] etc.

4.4 Implementations of FHE

Once practical HE was rendered possible, several implementations of these schemes have surfaced in the recent past, the first of which was an implementation of a variant of Gentry's original scheme by Smart and Vercauteren in 2010 [32]. However, the key generation took extremely long times and was practically very inefficient. The first real world implementation came in 2012, when a variant of the BGV scheme [29] was implemented in [33]. Then came a few publicly available implementations, the first of which is LibScarab [34] implementing the scheme described in [32]. However, It has many limitations as it does neither support many modern optimizations, nor techniques like relinearization. Another one is HELib, which implements the BGV scheme in [17]. It supports bootstrapping, multi-threading and lots of other features. But HELib is not easy to use for someone who is not an expert in cryptography and homomorphic encryption. Recently, another library called Simple Encrypted Arithmetic Library (SEAL) [35] was released by Microsoft, which was non-expert user friendly, including easy parameter selection and noise estimation tools. This is the library chosen in this project for the implementation of the network.

4.5 Privacy and Security in Machine Learning

In the recent past, some of the issues raised by lack of proper security, privacy, accountability etc in software system using ML algorithms have come to light. These issues raised by vulnerabilities in the flow of ML algorithms have not gone unheeded. A number of research activities, conferences such as (`http://www.fatml.org/`) etc have been gaining momentum lately.

Specifically towards preserving privacy in ML, there have been two major lines of work - one that focuses on the concept not revealing additional information about the owner of the data involved in training. This is an issue many of the big corporations are facing today, as they are collecting data continuously using phones/desktops etc to train predictive models for advertising. But it implicates the persons who are represented by the data and compromises their privacy. As a solution to this problem, this line focuses on a concept called *Differential Privacy.* On a high level, differential privacy ensures that the removal or addition of a single database record (training sample), does not affect the outcome of the analysis substantially [36]. The authors in [37] showed that deep neural networks can be trained with multi-party computations to achieve differential privacy guarantees.

However, this is relevant only in the training phase. The concept of differential privacy is not useful in the inference phase as only one record is being examined here. This brings us to the other line of work, concentrating on the privacy concerns in the inference phase. This is useful for cloud applications using ML. They may require the user to submit their data, which may be stored on the cloud for a while before applying inference models on it. So the problem is to protect the data in this process. The research in this area will be detailed in the next section, as it is closely related to the work in this project. There have also been a few recent works where training complex ML models on encrypted data has been looked at. One such algorithm is proposed in [38] where the expensive operations are offloaded to the cloud and BGV homomorphic encryption is used during the learning process.

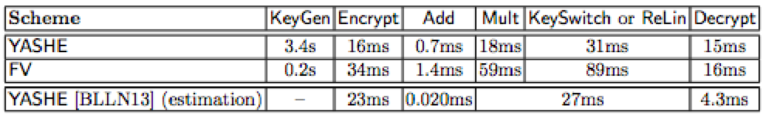
4.6 Related Work - Use of HE for Privacy Friendly ML

Before the advent of HE, the approach to securing confidential data involved *Secure Multi party Computation* (SMC) techniques. In this, the parties involved establish a communication protocol such as the ones described in the paper [39]. Combinations of this technique with regular encryption were also researched (these are called oblivious neural networks). The authors in [40] proposed one such scheme where the data owner encrypts sensitive data using normal encryption and sends to the prediction service on the cloud. The cloud then computes the outputs for the first layer and sends them back to the owner. The owner receives the results, decrypts them, applies non-linear activation functions, encrypts the results again and sends to the server. This continues until all the layers are computed. However, as they also note in a following paper [41], this procedure leaks a lot of information about the model held by the predictive service to the clients, making model theft a very plausible problem with such implementations.

The advent of non expert friendly implementations of FHE such as the one described in [31], acted as a bridge between the world of Cryptography and the world of Machine

Learning, sparking research in the application of FHE for enabling Privacy friendly machine Learning. The same authors proposed methods for implementing statistical inference algorithms in ML over encrypted data in the paper [42] They show that inference models can be implemented over encrypted data practically. For the first time, Graepel et al in [43] suggested the use of homomorphic encryption along with machine learning algorithms. In this work, several linear discriminative models such that linear means classifier, Fischer Linear Discriminant (FLD) were implemented using homomorphic encryption, and training on encrypted data was also possible because the simple linear models could have a training algorithm expressed as a low degree polynomial. Similar work has been done with nearest neighbor algorithms in [44]. Bost et al, in the paper [45] present a combination of HE schemes using Quadratic Residuosity, Paillier scheme, BGV scheme and garbled circuits for private classification for Machine Learning algorithms like Hyperplane Decision, Naive Bayes, decision trees. This method also is based partially on SMC, and is only efficient for algorithms with limited complexity and small data sets.

However, these models do not achieve the level of accuracy achieved by neural networks, especially in the task of Image recognition. CNNs are proven to have achieved the best accuracy, leaps ahead of any other machine learning algorithm. Microsoft researchers in the paper [46] first successfully applied neural networks to encrypted data using homomorphic encryption, and this project closely follows the ideas in this paper. However, the major difference is that cryptonets uses the encryption scheme described in [47] called Yet Another Somewhat Homomorphic encryption (YASHE), and this project uses the FV scheme. While FV is based on the RLWE problem, YASHE is based on a modified version of NTRU by Stehl and Steinfeld [16] and the FHE in [30]. A theoretical and practical comparison these two encryption schemes is presented in [48], where they conclude that the noise growth is smaller in FV than in YASHE (figure 4.2), but YASHE is faster than FV (figure 4.1).



Scheme	KeyGen	Encrypt	Add	Mult	KeySwitch or ReLin	Decrypt
YASHE	3.4s	16ms	0.7ms	18ms	31ms	15ms
FV	0.2s	34ms	1.4ms	59ms	89ms	16ms
YASHE [BLLN13] (estimation)	–	23ms	0.020ms	27ms		4.3ms

Figure 4.1: FV vs YASHE time

Source: [48]

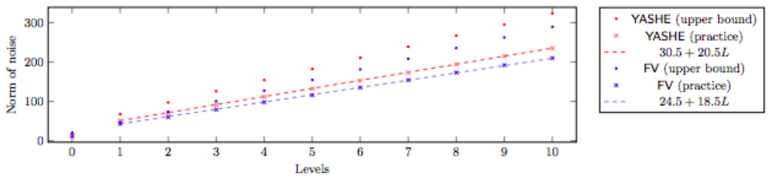


Figure 4.2: FV vs YASHE noise growth

Source: [48]

Being a beginner in the area of homomorphic encryption, it was felt that controlling and handling noise would be more of a challenge in this project, hence FV was the scheme of choice.

Chapter 5

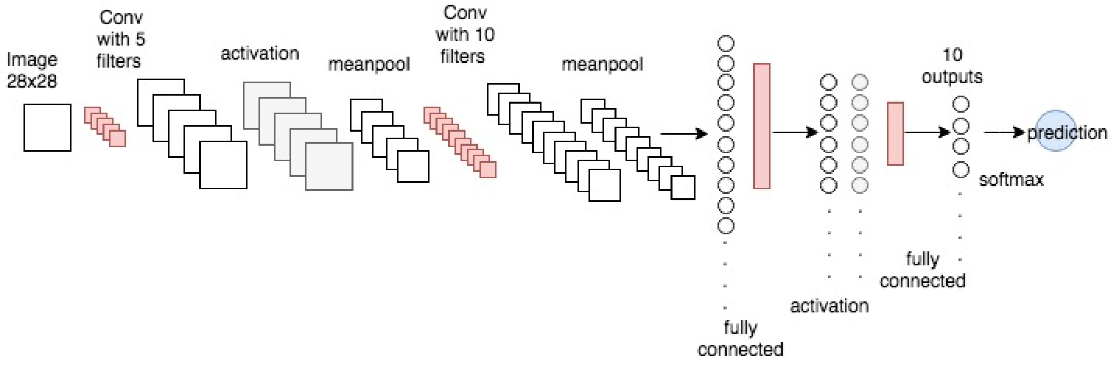
Implementation

A solution to the problem introduced in the first section is implemented as part of this project. A protocol for enabling the client to employ a predictive model held by a server without revealing his data is designed and implemented. The cloud uses a CNN model which it applies on encrypted data. But first, the model parameters (weights and biases) are learned by training on the unencrypted MNIST [49] training set. This is implemented using TensorFlow. In the inference phase, the model is applied on the encrypted data. This is written in Python 3.0, using functions from SEAL library [35] to handle encryption. Both the training and inference networks are described here.

5.1 Training Phase

5.1.1 Architecture

The inputs to training network are in the form of tuples : *the handwritten image* - a 28×28 matrix of pixels, where each pixel is represented by its gray level in the range 0-255, and *the label* - an array of length 10 in one-hot format. For example, if the handwritten digit is 2, the label would be $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$.



1. *Convolution Layer:* Input is the image, a 28×28 matrix. Convolution is done with 5 kernels of size 5×5 and a stride of $(1, 1)$. The number of input channels is 1, and 5 filters are applied on it, so 5 output channels are obtained. The output of this layer therefore, has dimensions $28 \times 28 \times 5$.

2. *Activation Layer:* Input is the $28 \times 28 \times 5$ matrix from the previous layer. It simply squares each value in the input matrix, therefore the output has the same dimensions.

3. *Mean Pool Layer:* Mean pooling is performed with a window of size $2 \times 2 \times 1$ and stride $(1, 1)$. Hence the output has dimensions $14 \times 14 \times 5$.

4. *Convolution Layer:* This convolution uses 10 kernels of size $5 \times 5 \times 5$ ans stride of $(1, 1)$. There are 5 channels in the input and 10 filters are applied - giving 10 output channels. So the output here has dimensions $14 \times 14 \times 10$.

5. *Mean Pool Layer:* Mean pooling is performed again using a window of size $2 \times 2 \times 1$ and stride $(1, 1)$. Hence the output has dimensions $7 \times 7 \times 10$.

6. *Fully connected Layer:* The $7 \times 7 \times 10$ matrix is flattened into an array of length 490. Fully connected layer is implemented by a matrix multiplication with a weight matrix of dimensions 490×128. The output therefore, is an array of length 128.

7. *Activation Layer:* This layer squares each the value in each node of the previous layer. Hence the output is again an array of length 128.

8. *Fully connected Layer:* This layer connects the incoming 128 nodes to 10 output nodes, each node corresponding to each of the 10 classes. It is implemented by multiplying the array of length 128 with a 128×10 matrix of weights.

9. *Softmax Layer:* Softmax is applied on the 10 output nodes, it gives a probability value for each node. Each node corresponds to one digit in the range $0 - 9$. The node with the highest probability is obtained and the corresponding digit (index of the node) is the prediction.

Weighted sum propagation rule is applied in the network. Which means the basic computation at the each of the nodes in the network is a weighted sum of inputs from previous layer.
In fully connected layers,

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

In Convolution layers,

$$z_j^l = (a^{l-1} * w^l)_j + b^l = \sum_k a_{j+k-1}^{l-1} w_k^l + b^l$$

where

z_j^l is the output of node j in layer l.

w_{jk}^l is the weight of connection between node k in layer $l - 1$ to node j in layer l.

b_j^l is the bias for node j in layer l.

a_k^{l-1} is the output of node k in layer $l - 1$.

b^l is the common bias value added after the convolution.

5.1.2 Dropout

In the training network, the TensorFlow dropout function is used, with a dropout rate of 0.75. Which implies there is 75% chance the nodes are not deactivated during the training.

5.1.3 Loss Function

Softmax cross entropy function is used to calculate the loss after the result is obtianed from the softmax layer. It computes the cross entropy between the vector output of softmax layer and label obtained from the training input, (a one-hot vector of length 10). Cross-entropy between two vectors is defined as

$$L = -\sum_{i=1}^{n}(y_i \cdot log(\hat{y}_i))$$

If the output is correct, meaning if the output layer has the highest probability value at the same position where the one-hot vector has the 1, the cross entropy value will be low. This indicates that the prediction is good.

5.1.4 Optimizer

The problems with stochastic gradient descent (algorithm 1) were introduced, and to thwart those problems with learning rate and for optimizing the learning process by making the learning rate adaptive, *Adam optimizer* (algorithm 2) is used instead of gradient descent. The TensorFlow implementation of this function uses the parameters ρ_1, ρ_2, δ specified in algorithm 2 automatically, but the learning rate can be set. $\epsilon = 0.01$ is used for this implementation.

5.2 Inference Phase

The inference phase operates in the encrypted realm (i.e., all operations are on encrypted data). There are logical limitations on the types of operations that can be performed and noise based limitations on the number of operations that can be performed without corrupting the cipher text. The Fan-Vercauteren encryption scheme is used, which allows additions and multiplications on the cipher texts, and by extension, polynomials and exponentiations. Hence, the exact same network as the one used in the training phase cannot be used. Softmax operation cannot be performed on encrypted data, so it can not be used in this network. But since it is a monotonically increasing function, this operation can be skipped and the cipher texts from output of the last fully connected layer can be compared to give the prediction. Moreover, because of the increased complexity of computing circuits with many nested multiplication operations in the convolution layer, it is optimal to limit the depth of multiplications. By using some tweaks to the algorithms and some optimizations, the above model is made to work on encrypted data to give the same results as the plain text version corresponding to the training network would give.

5.2.1 Encoding Scheme

From the description of HE scheme used, it is clear that there is a mismatch between the atomic constructs in the learning model (Real numbers) and the ones in the HE scheme (polynomials in R_t). An encoding scheme is needed to map one to the other, at the same time preserving the homomorphic operations. Following the notation introduced in the preliminaries section, a message m needs to be converted to its polynomial representation $\mathring{m}(x)$. The encoding scheme followed in this project is as follows:

Write the real number to be encrypted in its b-bit binary representation, such that $m = \sum_{n=0}^{b-1} a_n 2^n$. $\mathring{m}(x)$ is then constructed by replacing 2 with x so as to give $\mathring{m}(x) = \sum_{n=0}^{2^{d-1}-1} a_n x^n \in R_t$ (recall from notation section that 2^d is the degree of the cyclotomic polynomial ϕ_{2^d} where $R_t = \mathbb{Z}_q[x]/\phi_{2^d}(x)$. This way decoding becomes very simple (for the integral part), m can be recovered just by evaluating $\mathring{m}$ at 2. ($m = \mathring{m}(2)$).

However, in the network the values are not integers, but rational numbers with an integral and fractional part. So they are broken down and their integral part is encoded as described, and the fractional part is moved to the highest degree part of the polynomial with the signs of the co-efficients changed. In the implementation, 64 lowest co-efficients of the polynomial are reserved for the integral part and 32 highest for the fractional part. Increasing this increases the precision of the fractional part in encoding but for this implementation, 32 suffices. For example, the rational 26.75 would be represented as the polynomial $-1x^{1023} - 1x^{1022} + 1x^4 + 1x^3 + 1x^1$. Decoding the fractional part is not as trivial as computing $\overset{\circ}{m}(2)$, but can be done by appropriately ignoring the negative signs and shifting the co-efficients back. e.g., $-1x^{1023} - 1x^{1022} \to x^{1024-(1)} + x^{1024-(2)} \to x^{(-1)} + x^{(-2)}$ and then $m = 2^{-1} + 2^{-2} = 0.75$

5.2.2 Parameter Selection

Deciding the parameters to be used for the encryption scheme plays a major role in determining the correctness of results and the efficiency of network. There are two main factors in deciding the parameters. One is the noise growth consideration, and the other is efficiency. The implemented Encryption scheme is a practical somewhat homomorphic encryption scheme, which means it cannot perform arbitrary computations on encrypted data. Instead, in each stage the cipher texts have a specific quantity called the *invariant noise budget*, measured in bits. This noise budget in a freshly encrypted cipher text is determined by the encryption parameters. As operations are kept being performed on the cipher texts, the noise budget keeps getting consumed. Multiplications increase the noise significantly more than additions. Noise budget consumption is compounding in sequential multiplications. The network requires consecutive multiplications in the convolutional and fully connected layers. Hence, the most significant factor in choosing the parameters is the multiplicative depth of the arithmetic circuit the cipher texts go through in these layers. Once the noise budget in the cipher texts reaches zero, the noise spills over into the bits that contain the message, thereby corrupting the message and making it impossible to recover on decrypting. Therefore it is extremely important to choose the parameters to be just large enough to avoid this, but not too large that it becomes extremely computationally expensive and not practical anymore. As discussed in the explanation of the FV scheme :

- Plain text messages belong to the ring $R_t = \mathbb{Z}_t[x]/\phi_{2^d}(x)$ where $\phi_{2^d}(x) = x^{2^{d-1}} + 1$, the 2^d-th cyclotomic polynomial, and is called *Polynomial modulus.* This is one of the important parameters. For $d = 12$, $\phi_{2^d}(x) = x^{2048} + 1$. As it makes it easier to picture the polynomial being used, 2^{d-1} will be referred to as the polynomial modulus here after, instead of d.

- $\mathbb{Z}_t$ denotes the set of integers modulo t. The co-efficients in the plain text polynomial belong to this set. This parameter will henceforth be called *Plain modulus.*

- While the plain text space is the polynomial ring R_t, the cipher text space is the Cartesian product of the two polynomial rings $(R_q \times R_q)$. The parameter q will be referred to as *Coefficient modulus*

The parameters and how they effect these factors are explained in this section. However, several parameter choices have been experimented with, and several trade-offs are considered, which will be elaborated in the experiments and results section.

5.2.2.1 Polynomial Modulus (2^{d-1}) :

The polynomial modulus is always a power of 2 cyclotomic polynomial. This parameter mainly affects the security level of the scheme, because the secret is a uniform random draw from R_2, so it is a 2^{d-1} binary vector sampled for the polynomial co-efficients. The larger this parameter, the more secure the secret key, and the more secure is the scheme. However, this is the polynomial that the cipher texts are reduced modulo of. So a higher polynomial modulus also means bigger cipher text sizes. The consequence is that the operations become slower, and only consume more memory, thereby making the operations inefficient. The usual choices for this parameter include 2^{d-1} values of 1024, 2048, 4096, 8192, 16384, 32768. Lower values make the scheme insecure, and higher values make it inefficient. As there already are very large number of computations to perform in the network, higher values are avoided.

5.2.2.2 Coefficient Modulus (q) :

The cipher text coefficient modulus is the most important factor influencing the noise budget in a freshly encrypted cipher text. Bigger coefficient modulus means bigger noise budget. However, a large value of this parameter also means lower security level for the scheme. To compensate that and make the scheme secure again,the polynomial modulus needs to simultaneously increased, which thereby reduces efficiency. Guidelines for selecting parameters for 128-bit and 192-bit security levels are recommended in the white paper [50]. The SEAL library contains functions that automatically generate the appropriate co-efficient modulus for the given polynomial modulus and the required security level, following these guidelines. So, the choice of this parameter is actually restricted and is determined directly by the polynomial modulus.

5.2.2.3 Plain Modulus (t) :

The plain modulus can be any positive integer. This parameter determines the size of the plain texts. Together, plain modulus and coefficient modulus determine the noise budget in a freshly encrypted cipher text as :

$$noise\ budget\ in\ fresh\ encryption \simeq log_2(q/t)\ bits$$

The plain modulus also affects the consumption of noise budget in homomorphic operations as :

$$noise\ budget\ consumption\ in\ multiplication \simeq log_2(t)\ bits$$

Looking at these, it would seem intuitive to set the plain modulus as low as possible. But it is also to be considered that the co-efficients in the plain texts need to lie in the range $\left(\frac{-t}{2}, \frac{t}{2}\right]$. So if during the multiple operations the coefficients go beyond this range, the expected results are no longer obtained. *Therefore it is important to keep track of the values in different stages of the network and make sure t is large enough to handle them, yet small enough to make sure the noise budget does not get filled up in the process either.*

5.2.2.4 Noise Standard Deviation (σ)

As described in the LWE and FV scheme sections, the error terms in the encryption process as sampled from a Discrete Gaussian distribution, called the error distribution χ. The Discrete Gaussian distribution over integers assigns a probability proportional to $e^{\frac{-\pi x^2}{\sigma^2}}$ to each $x \in \mathbb{Z}$, where σ^2 is the variance of the distribution. As the variance increases the width of the distribution increases and that means a wider range of noise terms could be chosen. However the error distribution χ is itself not as simple as just sampling co-efficients according to the discrete Gaussian distribution [21]. Therefore it was considered best to use the recommended value for this parameter.

5.2.3 Prediction Network

The input to this network is the 28×28 matrix of cipher texts, each corresponding to the encryption of the encoded pixel value. The model parameters are obtained after training the network, in form of weights and biases for each layer. The structure of the network is almost the same as the training network except the final softmax layer is omitted from this network. So the output would be 10 encrypted values from the final fully connected layer. Although the structure of the network is the same, the implementation is very different as all the complex operations such as convolution need to be reduced to homomorphic evaluations.

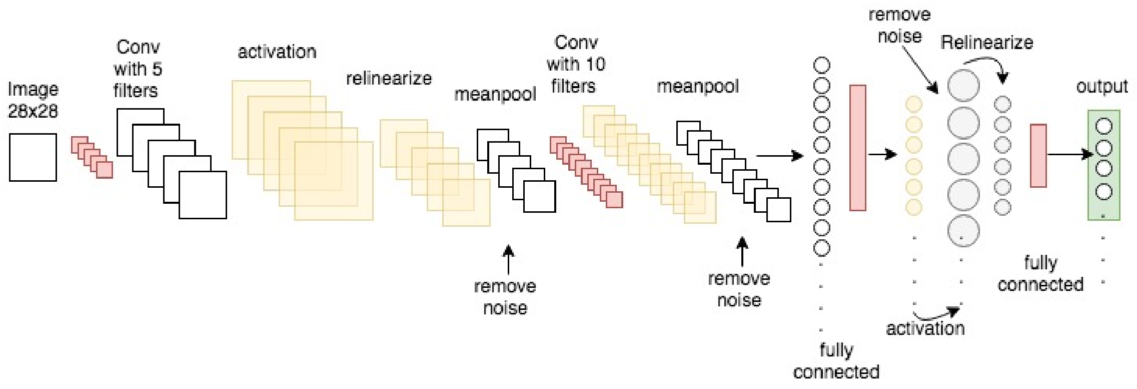


Figure 5.1: Inference Network Architecture

5.2.3.1 Meanpool and Activation Layers :

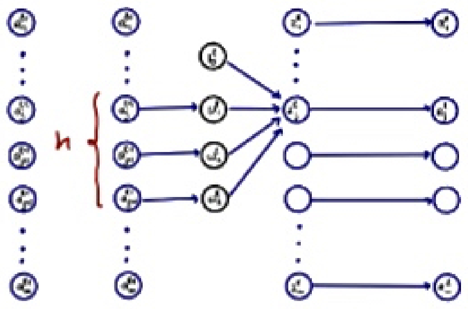
For meanpooling layers and activation layers, the transition into encrypted realm is more straightforward. In the mean pool layer, the average of 4 values is found. Therefore the four cipher texts are homomorphically added together and multiplied by 0.25. In the activation layers, the cipher texts are homomorphically multiplied with themselves. This is the only step in the network where it is required to perform multiplication with two cipher texts, making the activation layers the most noise budget consuming parts of the network.

5.2.3.2 Convolution Layers :

As described in the preliminaries section, a convolution operation is a repeated multiplication of the kernel matrix as it moves along, with pieces of the input matrix. Naively, the output matrix of convolution can be computed for one for one dimension as :

$$z_j^l = (a^{l-1} * w^l)_j + b^l = \sum_k a_{j+k-1}^{l-1} w_k^l + b^l$$

So in our first convolution layer, where the input matrix is of dimensions 28×28, and there are 5 kernels of dimensions 5×5 and the stride is (1,1), a naive implementation of convolution would mean ~5120 matrix operations($(28+5-1) \times (28+5-1) \times 5$). Each matrix operation is a multiplication of corresponding elements in two 5×5 matrices followed by adding them all together. Which means 25 multiplications and addition of 25 values. This leads to 5120×25 multiplications and 5210×24 additions on the cipher texts, in just one layer. So this implementation would be extremely inefficient and would require very loopy code which is especially bad for homomorphic evaluations. Therefore it is necessary



to optimize the convolution operation. For this, the technique used by TensorFlow's convolution function is followed and implemented as follows:

- *Input* to the convolution function is the input matrix *(input height, input width, input channels)* and the kernel matrix *(kernel height, kernel width, input channels, output channels)*. In both of the convolutional layers, the stride is $(1,1)$. Hence, zero padding is used so as to make the output height and width equal to those of the input.
- Before doing the convolution, the input is padded with zeros appropriately. In any of the dimensions (eg, height), if the input size channels is n_i, stride is s and kernel size is k, then the output size n_o is given by $\lceil \frac{n_i}{s} \rceil$. For this, a padding p needs to be added such that $\lceil \frac{n_i+p+k-1}{s} \rceil = n_o$. In both the layers in this network, we have $s = 1$, making $n_o = n_i$ and n_i, k are always integers. Therefore, we have $p = k - 1$
- The filter is first flattened into a 2-D matrix with shape *(kernel height * kernel width * input channels, output channels)*. In this case, in the first layer a $(25,5)$ matrix is obtained.
- Patches of shape *(batch, out height, out width, filter height * filter width * in channels)* are extracted from the input matrix. e.g., In the first layer patches of shape $(1,28,28,25)$ is obtained.
- Dot product between the filter matrix and a reshaped image patch is calculated for each patch. e.g., $(1,28,28,25)$ patch is reshaped to $(784,25)$ matrix and dot-product with the $(25,5)$ matrix is computed.

By following this, a reduction in number of steps from $\sim 130,000$ multiplications to $\sim 90,000$ multiplications is obtained, which may not be of much significance for operations with plain texts, but with homomorphic evaluations, where each multiplication takes $\sim 0.37ms$, it becomes useful.

5.2.3.3 Fully Connected Layers :

Fully connected layers are implemented as matrix multiplication between matrices of sizes *(1,incoming nodes)* and *(incoming nodes, outgoing nodes)* to give a matrix of size *(outgoing nodes, 1)*. The matrix multiplication is implemented using homomorphic additions and multiplications.

A point to note for all layers involving weights and biases: Since the weights and biases are obtained from a training network that runs on unencrypted data, they are just rational numbers, neither encoded nor encrypted. One approach to multiply/add these with cipher texts would be to encrypt them and then perform homomorphic multiplication/addition. But since encryption takes up time, and in the HE scheme, multiplying two cipher texts adds significantly more noise when compared to multiplying a cipher text with an unencrypted rational, the weights and biases are left unencrypted and the plain text version of homomorphic multiplication/addition functions is used. The only layer involving multiplying two cipher texts together is the activation layers, as it cannot be avoided in that layer.

5.2.4 Relinearization

It is clear from how the multiplication works in the FV scheme that fresh cipher texts have a size of 2, but on multiplying two cipher texts, the result has a size bigger than the size of both of them. More specifically, if the two cipher texts have size M and N, then the product cipher text will have size $M + N - 1$. This makes the further computations on the cipher texts very inefficient if the sizes keep growing.

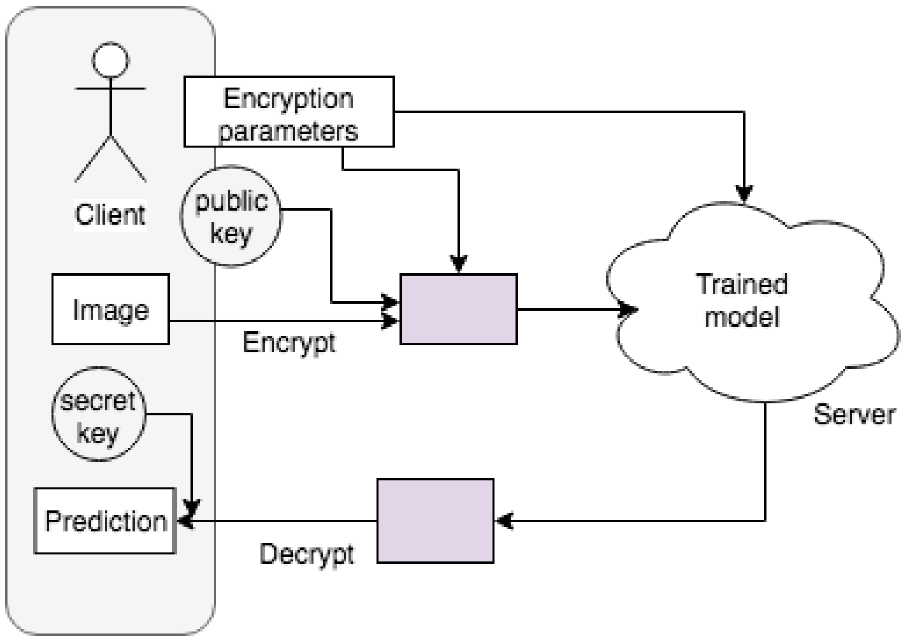
Homomorphic operations on large cipher texts are computationally much more expensive. To explain this concretely - Normal integers require 4 bytes of memory in the computer. But a cipher text built using this scheme (e.g., with $\Phi_{2^d} = x^{4096}+1$ and $q = 2^{128}$), will occupy ~65,536 bytes. (4096 co-efficients of 128 bit integers). This is just the size of fresh encryptions. On multiplication, the size of cipher texts increases. Specifically, homomorphic multiplication on cipher texts of sizes M and N requires $O(M \times N)$ polynomial multiplications, and homomorphic addition requires $O(M + N)$ additions. In order to make it less computationally expensive, the size of the cipher texts needs to be reduced back to the size of fresh encryptions, as this is the smallest possible size of cipher texts. This is done using relinearization function available in SEAL library.

In the implementation of this network, multiplying cipher texts together occurs only twice, i.e, in the two activation layers. Note that all the other multiplications are actually with the weights, which are plain texts. So relinearization is done after squaring the cipher texts to have a positive impact on performance.

5.2.5 Noise Handling

It can be seen, in the results section, that some of the computations consume a very large noise budget very rapidly. One way to avoid spill over of noise into the message is to increase the noise budget. It can be done so by increasing the cipher text modulus q but it effects the efficiency very drastically (as can be seen in the results section). Therefore instead, it is better to remove the noise using other techniques such as re-encrypting the cipher texts. In [14] Gentry introduced the scheme where the decryption circuit could be squashed and evaluated homomorphically. Thus, if the encrypted version includes a hint about the secret key, the decryption circuit can be homomorphically evaluated and the obtained values are re-encrypted. This results in a ‘fresh’ cipher text with initial noise budget. Since this is not trivial to implement during the time frame of the project, the real decryption is performed and the cipher texts are re-encrypted. This can be done by the client on his side, or by creating a re-encryption object to send to server along with evaluator and others. The first would be more secure, but would required communication over-head. Because of that, this step is tried to be avoided as much as possible. It is applied when noise budget gets almost completely consumed in the final steps of the network. This allows the handling of noise without having to make the process extremely inefficient by increasing parameters.

5.3 Flow and Summary of the Entire Process



The protocol is defined as follows:

- The Server obtains a model for predicting hand written digits using the training part of the MNIST dataset. The training network described above is implemented in TensorFlow, and the weights and biases are extracted. The client has the image, which he encodes as plain text polynomials and then encrypts using encryption parameters he chooses, using the FractionalEncoder and Encryptor objects of the SEAL library's python wrapper of their implementation of the FV encryption scheme.
- The server needs to know these parameters in order to work on the encrypted image, but should not know the secret key used for decryption. The client generates public and secret keys, builds the SEAL evaluator, and other objects that the server needs using these parameters, and sends them to the server, holding back the secret key.
- The server uses the extracted weights and biases, and runs the inference network built using homomorphic evaluation functions (also included in the SEAL library) on the encrypted image.
- At the end of inference phase, the server cannot decrypt the result to give the prediction, neither can it run the softmax layer and obtain the final prediction. But this works in favor of the client because if the server could run the softmax layer or compare the final layer outputs, the server could infer what the prediction is, without having to decrypt the result.(As it would be the index of the output node with the highest value). Therefore, the server sends the 10 values from the output layer back to the client, which inherently serves as a masking for the prediction.
- The client then decrypts the array of 10 values it received from the server. Since softmax is a monotonically increasing function, the client can then get the prediction as the index corresponding to the highest of these ten values.
- *Thus the client can utilize a model that it cannot access on the server and get the predictions, while the server does the all the computation, and still stays oblivious of both the input image and the prediction.*

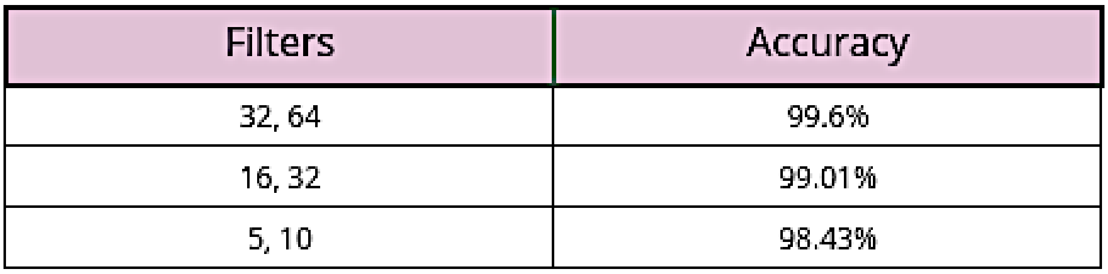
Chapter 6

Empirical Results

The key considerations in the analysis of the results are *accuracy* and *efficiency*. While accuracy is solely determined by the choices made in the architecture and parameters of the network, efficiency is largely effected by parameters of both the network and the encryption scheme. The network was trained using the MNIST data set of handwritten digits. The dataset consists of 60,000 images, where each image is a pixel array, represented by its grey-level. The training part of this data-set consists of 50,000 images, and the remaining used for testing. Training (on unencrypted MNIST training set) was done on Google colab, using Google's GPU accelerator. Inference network was tested on MacOS with 1.6 GHz Intel core i5 processor and 16 GB 1600 MHz DDR3 memory. The details of the training and inference networks presented in the implementation section is the final optimal version. However similar versions of this network were experimented with, mostly aiming to improve the efficiency and accuracy of the prediction network. Although the choice of the network was theoretically justified in the previous sections, this section presents the empirical results that support those claims. Details of the results of these experiments are presented hereunder.

6.1 Effect of Network Architecture

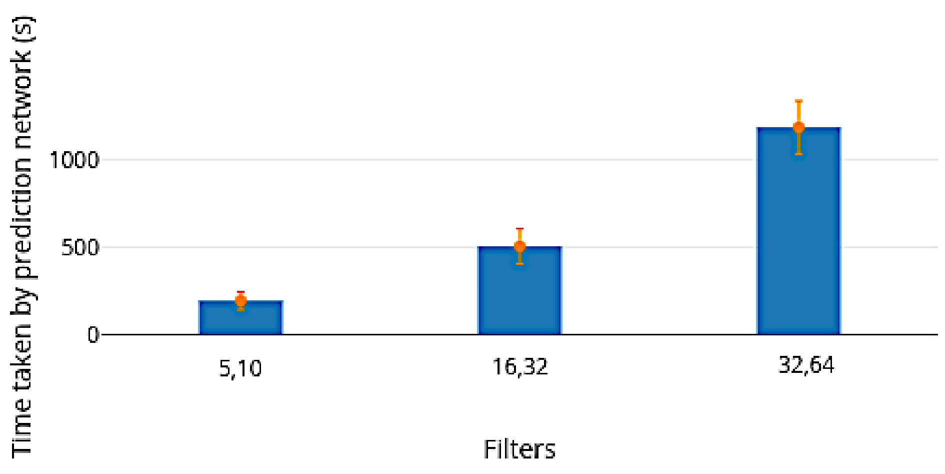
The best networks for the handwritten image detection problem have 9 layers (an additional activation layer after the second convolution), 32 filters in the first convolution layer and 64 filters in the second. The network architecture could be modified in two ways to increase the performance of the network - decrease the number of layers, or decrease the number of filters in the convolution layers. Networks with (32,64), (16,32) and (5,10) filters are tried out and accuracy, time taken and RAM consumption by the inference network are recorded.



Filters	Accuracy
32, 64	99.6%
16, 32	99.01%
5, 10	98.43%

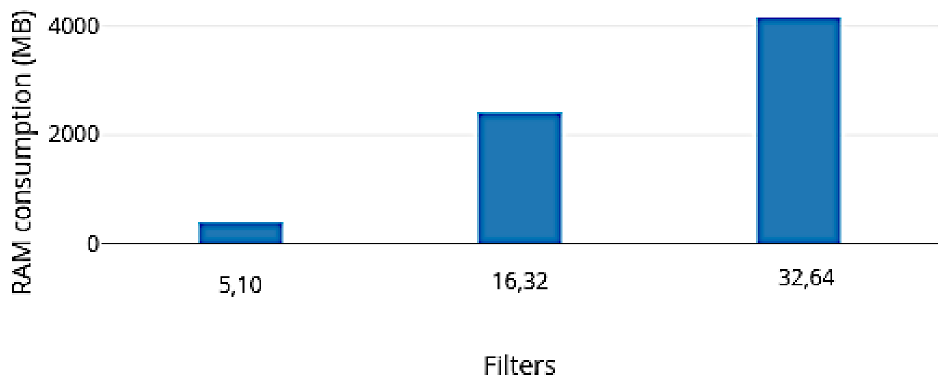
With Squared activation function and polynomial modulus 1024

Figure 6.1: Effect of additional filters on accuracy



With Squared activation function and polynomial modulus 1024

Figure 6.2: Effect of additional filters on time taken



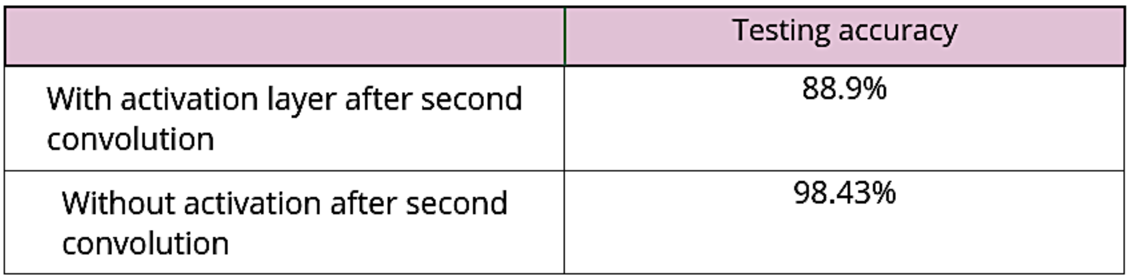
With Squared activation function and polynomial modulus 1024

Figure 6.3: Effect of additional filters on memory consumption

Although the addition of filters increases the accuracy of the network, the efficiency and practicality of drops significantly, as seen from the memory consumption and time taken. The network can be sped up using better hardware but it consumes ~ 3Gb RAM, with the smallest possible value of polynomial modulus itself. Hence it was decided that 5 and 10 filters was the better option.

NOTE: This particular experiment was run using polynomial modulus 1024 due to RAM limitations in the hardware available, as higher values were consuming more than 8Gb. These networks *did not* yield correct results as the cipher texts get corrupted, due to extremely low noise budget in them. However, the RAM consumption statistics are still meaningful and the inferences can be extrapolated to the higher polynomial modulus values.

Another major distinction in our network is that a different activation is used - the square function. One problem caused because of this is that unlike ReLU, sigmoid, or other usual activation functions, its derivative is unbounded. This potentially causes some strange behavior in training while running gradient descent. This leads to the values in the back propagation blowing up, or over fitting. When the network overfits, the testing accuracy goes down, even with high training accuracy.



	Testing accuracy
With activation layer after second convolution	88.9%
Without activation after second convolution	98.43%

With 5,10 filters and squared activation function

Figure 6.4: Effect of additional activation layer on accuracy

This issue can be partially resolved by having convolution layers without activation functions [46]. This allows us to remove the additional activation layer after the second convolution layer with confidence. It also works in our favor as it removes a very computationally expensive and noise budget consuming step.

6.2 Effect of Activation Function

The regular implementations of CNN for image recognition problem use non-linear activation functions like ReLU, Tanh etc, as covered in the preliminaries section. In order to achieve non-linearity in the network in this case, either of these two things can be done :

- Approximate the standard activation functions with polynomials using Taylor series.
- Use other simple non linear but less common activation functions like the square activation function and make adjustments to the network.

The first approach: ReLU can be approximated using a Taylor series around the point 0 of a smooth approximation of the ReLU function called softplus : $\ln(1+e^x)$

$$\ln(1+e^x) = \ln(2) + \frac{x}{2} + \frac{x^2}{8} - \frac{x^4}{192} + \frac{x^6}{2880} - \frac{17x^8}{645120} + \frac{31x^{10}}{14515200} + O(x^{12})$$

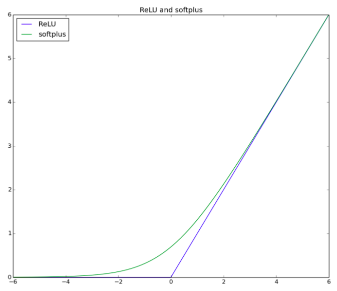


Figure 6.5: ReLU and softplus plots

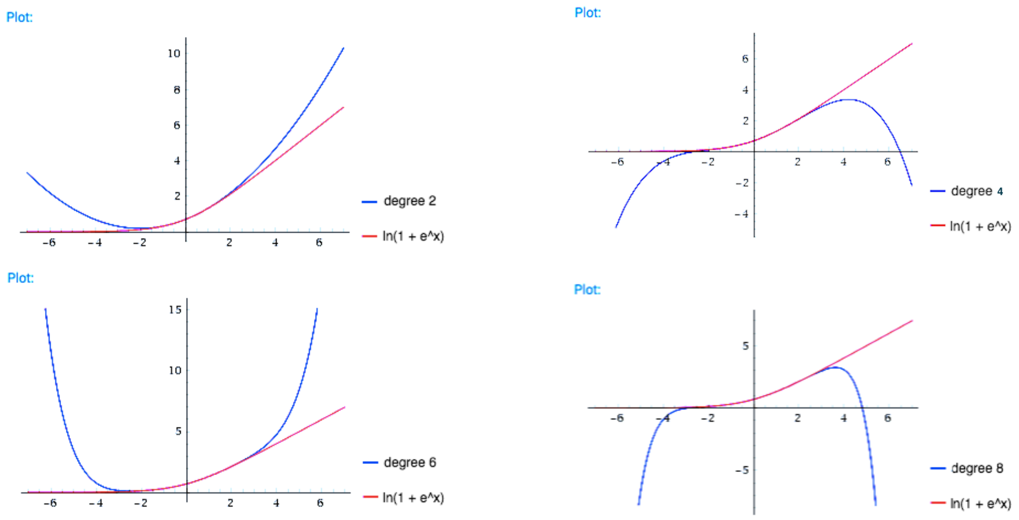


Figure 6.6: Polynomial approximations of softplus

As seen from figure 6.6, to get a better approximation, very high degree polynomials are needed, and even then, the approximations are good only within a range. e.g., degree 8 polynomial approximates the fun only in the range $[-4, 4]$. But the values in the network do not necessarily fall within this range, thus rendering the approximation not very good. Also, the best version of the network using only squared function gives 98.43% accuracy, which is not very far behind the state of the art CNNs using ReLU. due to these two reasons, this option was not explored further in this project.

6.3 Effect of Polynomial Modulus

In the implementation section, the effect of polynomial modulus, and the fact that cipher text coefficient modulus is directly dependent on it was explained. As expected, the time taken by the network increases with an increasing polynomial modulus, as seen in figure 6.7.

But as the cipher text co-efficient modulus increases with it, the noise budget increases,

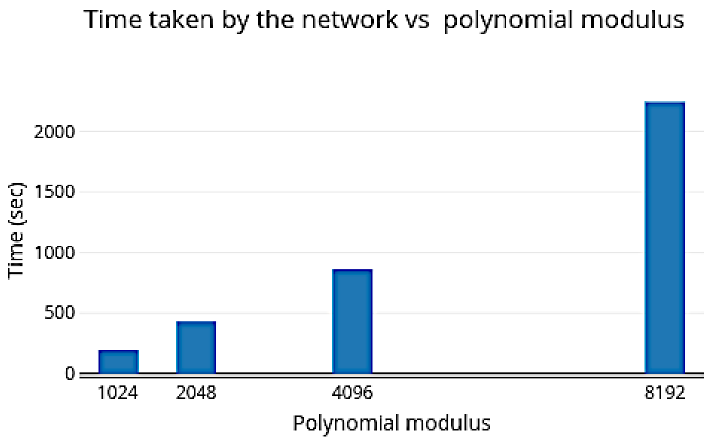


Figure 6.7: Effect of polynomial modulus on time taken by network

as can be seen from the figure 6.8

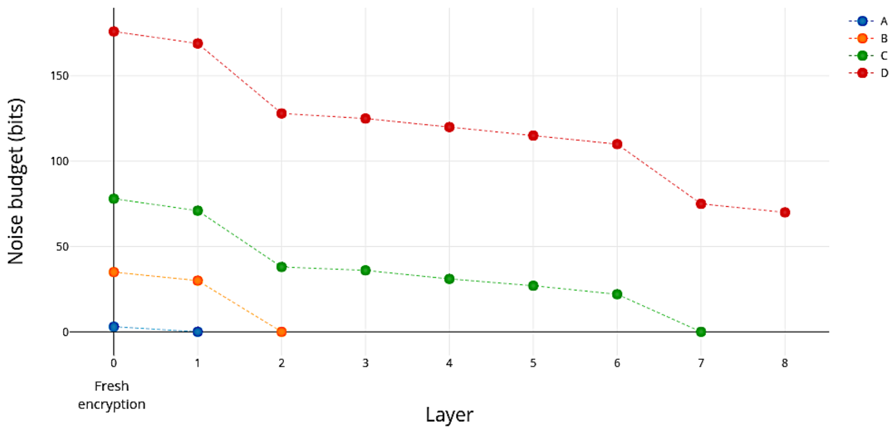


Figure 6.8: Effect of cipher text co-efficient modulus on noise budget

SEAL library does not allow setting of cipher text co-efficient modulus directly. A seal library function that automatically sets q given the polynomial modulus and security level is used in this implementation. The results in figure 6.8 correspond to polynomial modulus 1024(A), 2048(B), 4096(C) and 8192(D) with 128-bit security level.

From figure 6.8 It can be seen that the noise budget in the fresh encryptions with 1024, 2048 modulus is smaller than the noise budget consumption of a single activation layer. So it is impossible to use these values, and obtain correct results. The minimum polynomial modulus required is 4096, using which the noise budget drops to 0 only in the second last layer, so the cipher texts need to be re-encrypted at this stage, only one time. Higher value (8192) can give correct results till the end without having to do this, but as seen from timing results (figure 6.7), this is highly inefficient. Since higher values are inefficient and lower values do not yield correct results, 4096 is chosen as the optimal polynomial modulus value.

6.4 Effect of Plain Text Modulus

Having fixed the polynomial modulus and thereby the cipher text co-efficient modulus, different values of t were experimented with, and the results are shown in figure 6.9. They are in keeping with the inference made in the earlier section that increasing plain text modulus decreases the noise budget. For values of t less than 18, the decryptions do not

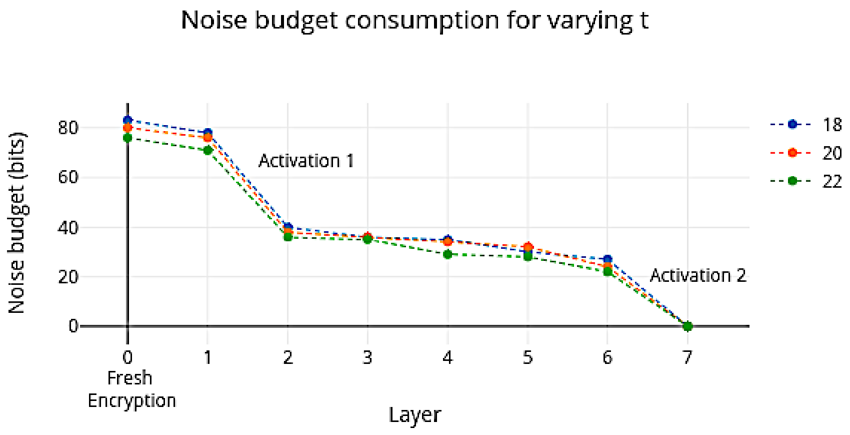


Figure 6.9: Effect of plain text co-efficient modulus on noise budget

give correct results, as the values of the co-efficients go beyond the range $(-9, 9]$ and then getting reduced modulo 18, is it not possible to recover the correct values. As the values above 18 reduce the noise budget, 18 is chosen as the optimal value for the plain text modulus.

6.5 The Final Network and Results

- CNN with 2 convolution layers with 5, 10 filters each.
- Two square activation functions, one after the first convolution and one after the first fully connected layer (figure 5.1).
- Polynomial modulus : 4096, matching cipher text coefficient modulus for 128-bit security.
- plain text modulus 18.

accuracy of network : 98.43%

As expected, convolutions are the most time consuming steps because of the high number of operations, but activations are the most noise budget consuming steps as they involve multiplying cipher texts together.

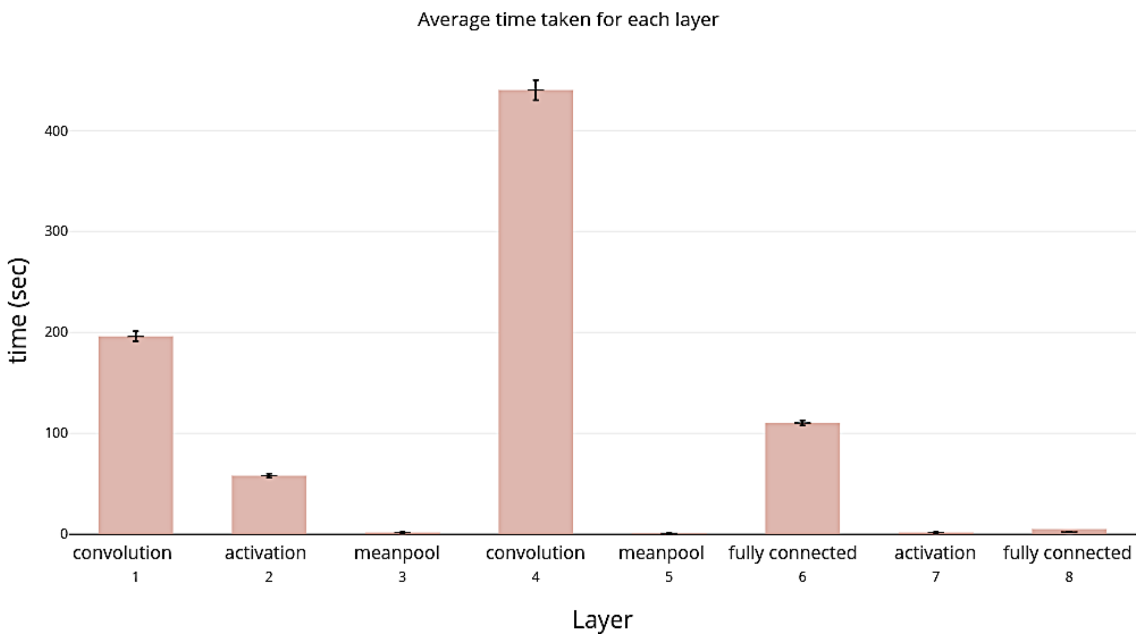


Figure 6.10: Time taken by different layers of the network

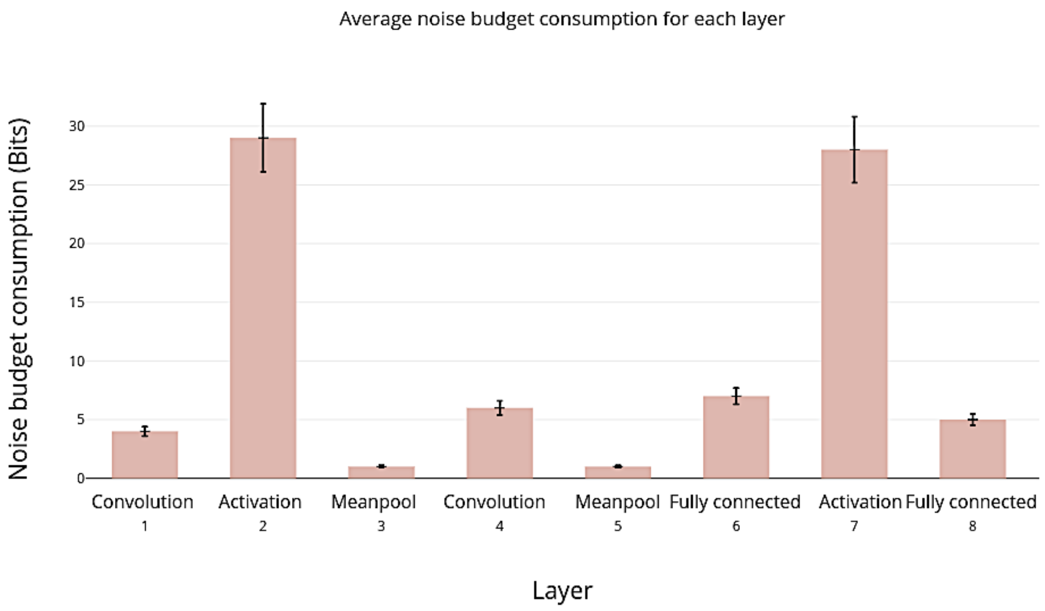


Figure 6.11: Noise budget consumption by different layers of the network

Chapter 7

Conclusion and Possible Future Work

In the present day, CNNs provide the best image classification systems and are being used extensively in several areas such as Computer Vision etc. However, sometimes predictions need to be made on sensitive data. For example, to analyze the x-ray images in a hospital etc, where privacy is a major concern. So there has been increasing interest towards making machine learning privacy friendly. This involves study of privacy in both the phases - during the training phase, with a concept called differential privacy and during the inference phase, using encryption. Training on encrypted data is possible, with polynomial activation functions, as the loss function is polynomial too, and back-propagation could in theory be implemented using additions and multiplications only. However, this has not been implemented yet (for CNNs), as it would involve efficiency based challenges (training a neural network even on plain text is very slow without access to sophisticated hardware. Add homomorphic evaluations to that, it would become very impractically slow).

This project is a study and work towards privacy preserving classification only in the inference phase. Several approaches to solving this problem can be seen in the literature, including Multi-party Computation, Oblivious Networks etc. However, in this project, use of Homomorphic encryption is the solution chosen, to study and implement. The main difference and advantage of this solution is that the client does not need to be active and interact with the prediction service as it works. Also, as there in no intermediate exchange of results, the server does not have to worry about leaking information about its model during the process. The client does not have to do any computations itself, other than encrypting and decrypting.

This network is similar to the one described in the *Cryptonets* paper, the major difference being that the prediction network is developed using a different HE scheme. Convolution is also implemented slightly differently. This version of the network is able to achieve an accuracy of 98.43%, as compared to 99% by cryptonets. However the time taken cannot be compared because of the significant difference in the hardware used. Cryptonets network takes ~250s to apply the network, using single Intel Xeon E5-1620 CPU running at 3.5GHz, and 16GB of RAM [46], while this network takes ~900s, using the hardware specified in the results section. A proper comparison of times taken could not be made using same hardware because Cryptonets is not an open-source code.

The main challenge is to handle the trade off between the three requirements - privacy, accuracy and efficiency.

To respect the privacy requirement, we use Homomorphic encryption and evaluations, which become inefficient when the algorithm has a high multiplicative depth. So the aim is to reduce the depth of evaluations without compromising the accuracy much. Network architecture was selected keeping this in mind. To achieve comparable accuracy to the non-private versions, appropriate activation functions need to be selected. The problem with the current activation function was discussed in the results section. The use of squared function led to having of one layer less than the state-of-art network architectures, and reducing the accuracy a little. But trying to increase the accuracy by adding filters etc, makes the algorithm very inefficient. Trying to increase the security of HE scheme by increasing parameters also makes the process inefficient by increasing cipher text sizes.

Accuracy could be improved by using ReLU in the training phase and polynomial approximations of ReLU in the inference phase, but the approximations are only good within a certain range. So, possible future work could include coming up with some sort of normalization technique that would reduce the values to this range before the activation layers and enable us to apply the approximations to solve this problem.

Another important factor in the implementation is the proper selection of encryption parameters to respect both the security and efficiency requirements. In this case, one set of parameters was found where the network could work without getting the cipher texts corrupted, but only until the penultimate layer (figure 6.8 and figure 6.9), after which, the cipher texts needed to be re-encrypted to increase the noise budget. Another possible direction of future work could involve finding better ways to do this, finding better encoding schemes that can allow smaller polynomials and more noise budget etc.

In conclusion, the method followed in this project has advantages such as protecting the data being submitted to cloud applications for inference. But the main limitation is the performance overhead and the fact that homomorphic encryption restricts the set of arithmetic operations supported. This introduces additional constraints in the design and architecture of the network. This results in slightly lower accuracy but extremely low efficiency when compared to its non-private versions.

References

[1] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[2] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction.* MIT press, 1998.

[3] T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised learning," in *The elements of statistical learning*, Springer, 2009, pp. 485–585.

[4] V. N. Vapnik, "An overview of statistical learning theory," *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 988–999, 1999.

[5] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[8] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659*, 2017.

[9] L. V. Fausett *et al.*, *Fundamentals of neural networks: architectures, algorithms, and applications.* Prentice-Hall Englewood Cliffs, 1994, vol. 3.

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[12] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, pp. 1139–1147.

[13] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 1999, pp. 223–238.

[14] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*, 09. Stanford University Stanford, 2009, vol. 20.

[15] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2010, pp. 24–43.

[16] D. Stehlé and R. Steinfeld, “Making ntru as secure as worst-case problems over ideal lattices,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2011, pp. 27–47.

[17] Z. Brakerski and V. Vaikuntanathan, “Fully homomorphic encryption from ring-lwe and security for key dependent messages,” in *Annual cryptology conference*, Springer, 2011, pp. 505–524.

[18] V. Lyubashevsky, C. Peikert, and O. Regev, “On ideal lattices and learning with errors over rings,” *Journal of the ACM (JACM)*, vol. 60, no. 6, p. 43, 2013.

[19] O. Regev, “On lattices, learning with errors, random linear codes, and cryptography,” *Journal of the ACM (JACM)*, vol. 56, no. 6, p. 34, 2009.

[20] C. Peikert *et al.*, “A decade of lattice cryptography,” *Foundations and Trends® in Theoretical Computer Science*, vol. 10, no. 4, pp. 283–424, 2016.

[21] J. Fan and F. Vercauteren, “Somewhat practical fully homomorphic encryption.,” *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[22] A. C.-C. Yao, “How to generate and exchange secrets,” in *Foundations of Computer Science, 1986., 27th Annual Symposium on*, IEEE, 1986, pp. 162–167.

[23] S. Goldwasser and S. Micali, “Probabilistic encryption & how to play mental poker keeping secret all partial information,” in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, ACM, 1982, pp. 365–377.

[24] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[25] R. Cramer, R. Gennaro, and B. Schoenmakers, “A secure and optimally efficient multi-authority election scheme,” *European transactions on Telecommunications*, vol. 8, no. 5, pp. 481–490, 1997.

[26] D. Boneh, E.-J. Goh, and K. Nissim, “Evaluating 2-dnf formulas on ciphertexts,” in *Theory of Cryptography Conference*, Springer, 2005, pp. 325–341.

[27] M. Fellows and N. Koblitz, “Combinatorial cryptosystems galore!” *Contemporary Mathematics*, vol. 168, pp. 51–51, 1994.

[28] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, “Towards the science of security and privacy in machine learning,” *arXiv preprint arXiv:1611.03814*, 2016.

[29] Z. Brakerski and V. Vaikuntanathan, “Efficient fully homomorphic encryption from (standard) lwe,” *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[30] A. López-Alt, E. Tromer, and V. Vaikuntanathan, “On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption,” in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, ACM, 2012, pp. 1219–1234.

[31] L. J. Aslett, P. M. Esperança, and C. C. Holmes, “A review of homomorphic encryption and software tools for encrypted statistical machine learning,” *arXiv preprint arXiv:1508.06574*, 2015.

[32] N. P. Smart and F. Vercauteren, “Fully homomorphic encryption with relatively small key and ciphertext sizes,” in *International Workshop on Public Key Cryptography*, Springer, 2010, pp. 420–443.

[33] C. Gentry, S. Halevi, and N. P. Smart, “Homomorphic evaluation of the aes circuit,” in *Advances in cryptology–crypto 2012*, Springer, 2012, pp. 850–867.

[34] H. Perl, M. Brenner, and M. Smith, “Poster: An implementation of the fully homomorphic smart-vercauteren crypto-system,” in *Proceedings of the 18th ACM conference on Computer and communications security*, ACM, 2011, pp. 837–840.

[35] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, “Manual for using homomorphic encryption for bioinformatics,” *Proceedings of the IEEE*, vol. 105, no. 3, pp. 552–567, 2017.

[36] C. Dwork, “Differential privacy: A survey of results,” in *International Conference on Theory and Applications of Models of Computation*, Springer, 2008, pp. 1–19.

[37] R. Shokri and V. Shmatikov, “Privacy-preserving deep learning,” in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, ACM, 2015, pp. 1310–1321.

[38] F. Bu, Y. Ma, Z. Chen, and H. Xu, “Privacy preserving back-propagation based on bgv on cloud,” in *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*, IEEE, 2015, pp. 1791–1795.

[39] O. Goldreich, “Secure multi-party computation,” *Manuscript. Preliminary version*, vol. 78, 1998.

[40] M. Barni, C. Orlandi, and A. Piva, “A privacy-preserving protocol for neural-network-based computation,” in *Proceedings of the 8th workshop on Multimedia and security*, ACM, 2006, pp. 146–151.

[41] C. Orlandi, A. Piva, and M. Barni, “Oblivious neural network computing via homomorphic encryption,” *EURASIP Journal on Information Security*, vol. 2007, no. 1, p. 037 343, 2007.

[42] L. J. Aslett, P. M. Esperança, and C. C. Holmes, “Encrypted statistical machine learning: New privacy preserving methods,” *arXiv preprint arXiv:1508.06845*, 2015.

[43] T. Graepel, K. Lauter, and M. Naehrig, “Ml confidential: Machine learning on encrypted data,” in *International Conference on Information Security and Cryptology*, Springer, 2012, pp. 1–21.

[44] Y. Qi and M. J. Atallah, “Efficient privacy-preserving k-nearest neighbor search,” in *Distributed Computing Systems, 2008. ICDCS’08. The 28th International Conference on*, IEEE, 2008, pp. 311–319.

[45] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, “Machine learning classification over encrypted data.,” in *NDSS*, 2015.

[46] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, “Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy,” in *International Conference on Machine Learning*, 2016, pp. 201–210.

[47] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, “Improved security for a ring-based fully homomorphic encryption scheme,” in *IMA International Conference on Cryptography and Coding*, Springer, 2013, pp. 45–64.

[48] T. Lepoint and M. Naehrig, “A comparison of the homomorphic encryption schemes fv and yashe,” in *International Conference on Cryptology in Africa*, Springer, 2014, pp. 318–335.

[49] Y. LeCun, “The mnist database of handwritten digits,” *http://yann. lecun. com/exdb/mnist/*, 1998.

[50] M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, J. Hoffstein, K. Lauter, S. Lokam, D. Moody, T. Morrison, *et al.*, “Security of homomorphic encryption,” *HomomorphicEncryption. org, Redmond WA, Tech. Rep*, 2017.

The references have been prepared using the Modern Language Association of America (MLA) format, *MLA Handbook, Eighth Edition.*

Appendix A

Code Repository and Demo Video

Source Code

The project source code is available at:
`https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2017/sxg744.git`.

The training network (train_net_5f.py) is implemented using TensorFlow and run using Google Colab, and the backpropagation, Optimizer etc used in the network are implemented using predefined TensorFllow functions. The network code itself is original. As for the inference phase, It is written using the SEAL library's open source Python wrapper 'PySEAL'.
`https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library/`. The Encryption scheme's operations used in the network - e.g., Encrypt, Decrypt, KeyGen, etc are all pre-implemented in the library. It has several dependencies, e.g., pybind11, cppimport etc. To run the code without having to install the dependencies(all but XQuartz, which will be required for display on MacOS), the docker command in the Encrypted_NN folder can be run as listed in the 'README.md' file of the repository.

The convolution optimization implemented in the prediction network is an algorithm borrowed from TensorFlow but implemented as part of this project using homomorphic evaluations. All the code within the Encrypted_NN/Predictor folder has been developed as part of this project and is original. The other folders are SEAL library files.

Demo Video

A highly sped up version of the video demonstrating the working of the prediction network is available at:
`https://youtu.be/6EapT7HAvFA`