Privacy Preserving Deep Learning

Simon McDonnell

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2019

Abstract

Deep learning models provide state-of-the-art results in many domains. These models are data intensive as they require large amounts of data to learn effectively. In recent years there has been increased concern over models using data that contains personal information, especially in fields where this information is particularly sensitive, such as medicine. A possible solution is to run deep learning on encrypted data, either during training or inference. This work explores using deep learning in conjunction with homomorphic encryption during the inference stage, and evaluates whether it is practical for use in real world applications. This would allow models to make predictions while protecting the privacy of the data.

The performance of models operating on encrypted data was evaluated across a number of different data sets that cover both regression and classification tasks. Models were successfully trained that achieved strong performance on encrypted data, however, there was significant variation in model performance depending on network design choices. It was found that using the Tanh activation function, standardising the outputs, and applying L2 regularisation to the weights provided the best performance across data sets. The biggest challenge for practical use in real world applications is the run time of the models operating on encrypted data, which is significantly longer than models operating on unencrypted data. However, if operating on a small data set or fast inference is not a necessity, these models can make effective predictions while protecting the privacy of the data.

Acknowledgements

I would like to thank my supervisors, Yongxin Yang and Timothy Hospedales, for proposing an interesting project and providing guidance throughout. It was a great pleasure to work with you both. I would also like to thank my parents, girlfriend, and friends for their support over the year.

Table of Contents

Chapter 1

Introduction

Deep learning models have provided state-of-the-art results in many domains and are employed by many companies in order to learn from data they have collected. These models are particularly data intensive as they require large amounts of data in order to learn effectively. The data collected commonly includes some private information which can cause a number of issues in relation to data protection and privacy. For example, a company may have collected a large amount of data involving their customers' spending habits, and wish to outsource the analysis of this data to a third party. The company outsourcing the data would like to protect their data from being stolen or altered. There has also been increasing concern in recent years over the collection of large amounts of personal data, with fears that this data will be misused or leaked. This is an especially important concern in fields where the data is particularly sensitive, such as medicine. A possible solution to these problems is to run deep learning on encrypted data during the training or inference stage. This work will explore using encrypted data during the inference stage, with a pre-trained model already provided. This would allow deep learning models to make predictions on new data effectively while also protecting the privacy of the data.

One approach to get meaningful results from encrypted data is to use homomorphic encryption. Homomorphic encryption allows calculations to be performed on the data while encrypted. This would allow machine learning models to operate on encrypted data to obtain encrypted results, which can be decrypted later. However, there are a number of challenges to overcome when combining deep learning and homomorphic encryption. Firstly, homomorphic encryption only supports addition and multiplication operations. This causes a problem when constructing neural networks as these models usually involve non-linear activation functions. Therefore, functions such as ReLU [15] will need to be approximated using polynomials. Secondly, there are challenges with computation accuracy. In order to encrypt a message, a small amount of noise is added. Decryption relies on accurately removing this noise to some degree of rounding error. However, when performing many repeated multiplications and additions the magnitude of the noise will grow. Consequently, removing noise after calculations becomes more error prone. This introduces a limit to the depth of the network, as a network with many layers will contain many multiplications and additions, resulting in problems with accuracy when decrypting. Different settings of certain encryption parameters can allow trade offs between the noise tolerance and security of the scheme. Further details of the encryption scheme will be provided in Section 2.2.

This work implements a Python wrapper that builds on the Simple Homomorphic Encryption Library (SEAL) [18] and PySEAL [13] for constructing neural networks that make predictions on encrypted data and explores using deep learning in conjunction with homomorphic encryption during inference. Recent papers such as CryptoNets [8] and CryptoDL [9] had success combining deep learning and homomorphic encryption to make predictions for image classification. These networks were evaluated on classic data sets such as MNIST [14] and CIFAR-10 [12]. This work will investigate the performance of neural networks on six different data sets from the UCI Machine Learning Repository [17], covering both regression and classification tasks. For each data set, the performance and speed of the networks will be evaluated for both encrypted and unencrypted data. The aim is to discover whether using encrypted data during inference is an effective way to protect the privacy of the data across a range of different machine learning tasks, and is practical for use in real world applications.

1.1 Contributions

- A Python wrapper building on SEAL and PySEAL to encrypt data and perform common neural network operations on encrypted data. These operations include matrix multiplication, vector addition, and the application of activation functions. The wrapper will aid others in exploring and implementing deep learning models that perform inference on encrypted data.
- An evaluation of how different neural network architectures perform during inference on six different UCI data sets, for both regression and classification tasks. The performance and speed of models operating on encrypted data is evaluated. In addition, a comparison of the effects of different activation functions, batch normalisation, and data processing techniques on the performance of models is conducted.
- Optimal model design choices for achieving the best performance on encrypted data across a range of different regression and classification tasks.

1.2 Outline

- **Chapter 2** - Background descriptions for neural networks and homomorphic encryption, detailing what they are, how they work, and some of the issues that arise when combining these two fields. This chapter also covers previous work done, exploring approaches taken to using homomorphic encryption in conjunction with deep learning and the design choices made in these approaches. This includes approximation of activation functions and the application of batch normalisation.

- **Chapter 3** - Description of work done to lay the foundation for experiments to be carried out. This chapter describes how the Python wrapper was built and the capabilities of this wrapper, detailing the different functions available. This chapter also describes how the different activation functions were approximated using polynomials and outlines various methods for achieving optimal performance when using these approximations, such as batch normalisation, output standardisation, and L2 weight regularisation.
- **Chapter 4** - Design of experiments. This includes descriptions of the different data sets and the network architectures used for each. Details of how each data set is evaluated for both unencrypted and encrypted data is discussed, including the environment setup and security parameters used during experiments. This chapter also provides baseline results for each data set in order to provide a frame of reference for later experiments.
- **Chapter 5** - Results of experiments. For each data set, the results of models using unencrypted and encrypted data during the inference stage are summarised. Tables of results and graphs are analysed to explore the benefits and drawbacks of different model choices and the effects these choices have when operating on encrypted data.
- **Chapter 6** - Discussion of results, providing an overall summary of model performance for different design choices.
- **Chapter 7** - Conclusions and lessons learnt, including future work.

Chapter 2

Background

2.1 Neural Networks

In this work, the term 'neural network' refers to artificial feed-forward neural networks. Neural networks are a type of machine learning model that provide state-of-the-art performance for many tasks. Figure 2.1 shows the structure of a neural network with three layers. Below are some definitions:

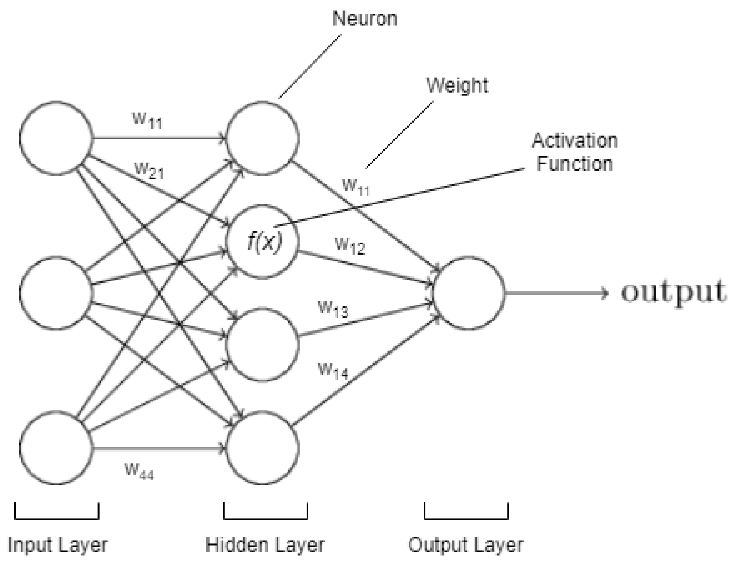


Figure 2.1: Neural Network with one hidden layer. Image source - Nielsen [16].

- **Neuron** - The basic building block of a neural network. Each neuron receives a certain number of inputs and applies a function to these inputs, outputting the result.
- **Layer** - An arrangement of neurons. For example, Figure 2.1 has 3 layers: input, hidden, and output.

- **Hidden Layer** - A hidden layer is any layer that is not the input or output layer. There can be any number of hidden layers. Figure 2.1 has 1 hidden layer.
- **Activation Function** - A function that is applied to the inputs of a neuron. For example, ReLU is a type of activation function: $f(z) = max(0, z)$.
- **Weights** - Parameters learned by the model. Every input into a neuron has a weight associated with it, which the model can learn through training in order to represent functions.
- **Training** - The stage at which the network tries to learn optimal values for the weights. This involves exposure to a training set of labelled examples and an iterative update process called gradient descent. Training is often done incrementally, with parameters updated after processing a batch of inputs.
- **Testing / Inference** - The stage at which the network makes predictions for data it has not seen before. This involves making predictions on a previously held-out test set of labelled examples, which the network does not have access to during the training stage.

2.2 Homomorphic Encryption

2.2.1 Overview

Encryption is a method of encoding a message in such a way that only those who are authorised can view the original message. However, traditional encryption schemes do not retain relationships when operations such as addition or multiplication are performed on the encrypted message. This makes these schemes unusable in machine learning models as computations performed on encrypted data will not be reflected in the underlying message. Homomorphic encryption is a method that allows computations to be performed on the underlying message while keeping the message encrypted. Encryption is performed by adding a small amount of noise to the message. When operations such as addition or multiplication are carried out, the noise in the encrypted message is increased. This increase is more significant in multiplications than in addition. Decryption relies on accurately removing noise after calculations have been performed to some degree of rounding error. If many addition and multiplication operations are carried out, decrypting the message becomes more difficult as the amount of noise has significantly increased.

There are different types of homomorphic encryption schemes that have different properties. Somewhat Homomorphic Encryption (SHE) is a method that allows for both addition and multiplication to be carried out on the underlying message while keeping the data encrypted, however, there is a limit to the number of these operations that can be performed due to the growth of noise. The first Fully Homomorphic Encryption (FHE) scheme was presented in 2009 by Gentry [7] and supports an arbitrary number of addition and multiplication operations. This is possible due to a method called bootstrapping. Bootstrapping reduces the size of the noise in the encrypted message, however, this is a computationally expensive process which makes it unfit for practical use in real systems. There have been many improvements to the FHE scheme since its introduction to make it more efficient [1, 2, 6] and this has resulted in Levelled Homomorphic Encryption (LHE) schemes. LHE schemes require that the number of operations to be carried out is known in advance so that the encryption parameters of the scheme can be set accordingly. LHE does not use the bootstrapping operation and therefore is significantly faster than FHE. This trade off allows homomorphic encryption to be practical when used in real systems.

2.2.2 Description of Method

SEAL uses the Brakerski/Fan-Vercauteren (BFV) [6] scheme for performing homomorphic operations. This is an LHE scheme and represents the plaintext (message) and ciphertext (encrypted message) as polynomials, with all homomorphic operations being carried out on these polynomials. Therefore, in order to encrypt the plaintext, it must first be encoded as a polynomial. LHE schemes require encryption parameters to be set before computations are carried out. There are three main parameters that must be considered:

- **Polynomial Modulus** - Specifies the maximum degree of polynomials in the scheme and affects the size of the ciphertext. Increasing this value improves the security level of the scheme while also increasing the computation time.
- **Plaintext Coefficient Modulus** - Specifies the maximum value of coefficients in the plaintext polynomial. It also affects the size of the plaintext and the noise tolerance in newly encrypted ciphertexts.
- **Ciphertext Coefficient Modulus** - Specifies the maximum value of coefficients in the ciphertext polynomial. Increasing this value allows for greater noise tolerance, but also decreases the security of the scheme.

Encoding numbers as polynomials begins by first performing a base expansion. For example, 24.75 can be expanded in base $b = 2$ in the following manner:

$$24.75 = 2^4 + 2^3 + 2^{-1} + 2^{-2}.$$

This expansion is then encoded as a polynomial, with the new coefficients of this polynomial chosen as follows:

- **If base b is odd** - New coefficients are integers in range $\left[-\frac{b-1}{2}, \frac{b-1}{2}\right]$
- **If base b is even** - New coefficients are integers in range $\left[-\frac{b}{2}, \frac{b-1}{2}\right]$
- **If base b = 2** - New coefficients are 0 or 1.

The degree of the polynomial is specified before computations are carried out by setting a value for the polynomial modulus. For example, if the polynomial modulus is set to $x^{1024} + 1$, then the degree of the polynomial encoding will be 1023. The number will then be encoded as follows:

$$24.75 = -1x^{1023} + -1x^{1022} + 1x^4 + 1x^3$$

The fractional part of the number is represented by the highest degree part of the polynomial with negative coefficients, and the integer part of the number is represented in the lowest degree part of the polynomial with positive coefficients. Once numbers are encoded as polynomials, plaintext and ciphertext polynomials maintain the following representation throughout computations:

- **Plaintext** - Represented as a polynomial in the space $R_t^n = \mathbb{Z}_t[x]/(x^n+1)$, where t is the plaintext coefficient modulus and x^n+1 is the polynomial modulus. This indicates that the polynomial will have a degree less than n and integer coefficients modulo t.
- **Ciphertext** - Represented as a polynomial in the space $R_q^n = \mathbb{Z}_q[x]/(x^n+1)$, where q is the ciphertext coefficient modulus and x^n+1 is the polynomial modulus. This indicates that the polynomial will have a degree less than n and integer coefficients modulo q.

Operations performed on polynomials are done modulo the polynomial modulus and the coefficient modulus. For example, two ciphertext polynomials are multiplied as follows, with polynomial modulus x^4+1 and ciphertext coefficient modulus, 3:

$$\begin{aligned}(2x^2+2)(x^2+1) &= 2x^4+4x^2+2\\ &= x^2.\end{aligned}$$

The term $2x^4$ becomes -2 when calculations are performed modulo the polynomial modulus, due to the fact that $x^4 \equiv -1(\text{mod } x^4+1)$. The term $4x^2$ becomes x^2 when calculations are performed modulo the coefficient modulus, due to the fact that $4 \equiv 1(\text{mod } 3)$.

The next stage is to generate a private and public key pair. The plaintext is encrypted using the public key, and can only be decrypted using the corresponding private key.

- **Private key** - A private key s is generated by sampling a random polynomial from R^n with coefficients of either -1, 0, or 1.
- **Public key** - Firstly, two polynomials are drawn, $a \in R_q^n$ and an error/noise polynomial $e \in R^n$, whose coefficients are drawn from a discrete Gaussian distribution. The public key is then defined as the pair of polynomials:

  $$pk = ([-as+e]_q, a)$$

  The polynomial arithmetic is done modulo the polynomial modulus, n, and the coefficient arithmetic done modulo the ciphertext coefficient modulus, q.
- **Encryption** - Two more error polynomials are drawn, $e_1, e_2 \in R^n$, with coefficients from a discrete Gaussian distribution. Another polynomial is drawn, $u \in R^n$, with coefficients of either -1, 0, or 1. The plaintext, $m \in R_t^n$, is then

encrypted using both parts of the public key. This ciphertext is also represented as a pair of polynomials as follows:

$$ct = ([pk_0 u + e_1 + qm/t]_q, [pk_1 u + e_2]_q)$$

We can see that the message is present in the ciphertext only with a scaling. This is why addition and multiplication are possible in homomorphic encryption, as the ciphertext contains only a scaled version of the original message with the other terms existing to hide the message. These other terms can then be removed using the private key.

- **Decryption** - Decrypting the ciphertext is done using the private key s as follows, with the symbol $\lfloor\rceil$ representing rounding to the nearest integer:

$$\begin{aligned} m' &= \left[\left\lfloor \frac{t}{q}[ct_0 + ct_1 s]_q \right\rceil\right]_t \\ &= \left[\left\lfloor \frac{t}{q}[-aus + eu + e_1 + \frac{q}{t}m + aus + e_2 s]_q \right\rceil\right]_t \\ &= \left[\left\lfloor \frac{t}{q}[eu + e_1 + \frac{q}{t}m + e_2 s]_q \right\rceil\right]_t \end{aligned}$$

We can see that the original message m is present with a scaling, and the other terms are error polynomials. As long as the error polynomials are not too large, they will be removed with the rounding operation $\lfloor\rceil$ and we will be left with the original message.

2.2.3 Encryption libraries

In order to perform homomorphic encryption this work uses the Simple Encrypted Arithmetic Library (SEAL), created by Microsoft Research [18]. The first version of SEAL was released in 2015, aiming to create an easy to use homomorphic encryption library. The library provides functions to carry out homomorphic computations and is written in C++. Most machine learning applications are written using the Python programming language and this led to the use of the PySEAL library provided by Lab41 [13]. This library is a fork of SEAL and provides Python APIs for SEAL functions. This makes implementing SEAL into existing machine learning work flows much simpler. There are a number of other homomorphic encryption libraries available such as HElib [19], however, SEAL provides a number of features that make it more practical for use in neural networks. For example, HElib does not provide the functionality to operate on floating point numbers, only integer computations are possible. This is in contrast to SEAL which supports floating point calculations, a feature which will be crucial when using neural networks in order to get accurate results. Another key feature that SEAL provides is the final rounding operation after decryption as seen in Section 2.2.2. HElib does not provide this rounding functionality.

2.2.4 Practical Considerations

There are a number of issues when using homomorphic encryption in neural networks. Firstly, as homomorphic encryption only supports addition and multiplication, activation functions such as ReLU will need to be approximated using polynomials. This introduces inaccuracies into network calculations that will need to be mitigated to prevent errors propagating through the rest of the network. Secondly, as neural networks involve repeated multiplications and additions as data propagates through the network, the increase of noise in the message becomes an issue. As more multiplications and addition operations are carried out, the noise in the message will become larger, making it more difficult to decrypt accurately. This introduces a limit to the depth of the networks, with deeper networks producing more unreliable results. This problem can be mitigated by setting appropriate parameters for the encryption, such as the polynomial modulus and the coefficient moduli. However, there are trade offs when choosing these encryption parameters. Improved security and tolerance for noise comes with significantly increased running times. The polynomial modulus affects the security level of the scheme, with a larger value corresponding to increased security. However, while making the scheme more secure it also makes the size of the ciphertext much larger and therefore increases the computation time of operations. The ciphertext coefficient modulus affects the noise budget of the scheme, with larger values allowing for greater noise tolerance while also decreasing the security of the scheme. Setting these parameters is a trade off that will need to be balanced in order to allow practical use of homomorphic encryption in neural networks. The official SEAL documentation [18] provides recommendations and default parameters for different levels of security.

2.3 Related Work

In 2014 Microsoft Research authored a paper [20] in which they explore how using homomorphic encryption in conjunction with neural networks could be used to protect the privacy of data during the inference stage. They discuss how using modifications to activation functions and training algorithms could allow for secure cloud-based models that provide predictions while protecting the privacy of the user's data.

In CryptoNets [8] Microsoft Research build on their previous work and evaluate the performance of homomorphic encryption with neural networks. The aim of the paper is to show that using homomorphic encryption in combination with neural networks is efficient and accurate enough for real world applications. The authors use the SEAL library for homomorphic encryption and evaluate their work using the MNIST data set. They show that their model achieves an accuracy of 98.95%. Their model is a type of neural network called a convolutional neural network, which is commonly used for analysing images. In their implementation, the network contains no activation function until the final layer. The authors comment that they didn't have a good approximation for the Sigmoid function that would work well in the encrypted space. By having only a single Sigmoid function in the final layer of the network, there is no need to provide an approximation. This is because the Sigmoid function is applied element-wise to the output of the previous fully connected layer. The output of this penultimate layer can simply be returned and decrypted, with the final processing step of applying the Sigmoid function done on the decrypted data. This approach worked well for this type of network, however, this model would not work when attempting to classify data that is not images. Internal layers with activation functions would be needed to make a network that is more general, and can be applied to many different classification and regression tasks.

CryptoDL [9] is a paper that also attempts to combine homomorphic encryption and neural networks. Again, the main aim of the paper is to show that using these two in conjunction can provide efficient and accurate enough results for real world applications. They evaluate their work on two images data sets, MNIST [14] and CIFAR-10 [12], but present an improved model compared to other similar works such as CryptoNets. The model is again a convolutional neural network, with the main improvement being the inclusion of approximated activation functions. The authors explore a number of different methods for approximating three commonly used activation functions: ReLU, Sigmoid, and Tanh. The models using the ReLU approximation proved the most successful when comparing final accuracy results on the two data sets. Their approximation involved using modified Chebyshev polynomials. The final performance of the CryptoDL model on MNIST was 99.52%.

Chabanne et al. [4] attempt to find a good approximation for ReLU in a similar fashion to CryptoDL, however, they first try to bound the region in which values lie before this ReLU layer in order to make the approximation more reliable. This is done using batch normalisation [11] which is applied before the activation function. Batch normalisation is a procedure that normalises each feature in a neuron across a batch of inputs, by subtracting the mean and dividing by the standard deviation, in order to produce a distribution of values that is approximately normal. They found that fitting a degree 4 polynomial to ReLU performed best on the MNIST data set, obtaining an accuracy of 99.3%.

In addition to work that focuses on the inference stage, there are other papers that explore protecting privacy during training. Bu et al. [3] propose a privacy preserving back propagation algorithm that operates on the cloud and uses the BGV [1] scheme. Computationally expensive operations are carried out in the cloud which improves the efficiency of back propagation learning. Zhang et al. [21] also propose an algorithm that performs back propagation on the cloud and uses BGV to protect the privacy of the data. The authors approximate the Sigmoid activation function using a polynomial in order to make secure computations compatible with the BGV scheme.

This work builds on ideas presented in previous work in order to evaluate whether using homomorphic encryption in conjunction with neural networks is practical for real world applications. Previous work has explored the performance of these models on either MNIST or the CIFAR-10, with improvements in performance measured against these data sets. However, this work evaluates the performance of neural networks across six different UCI data sets, covering both regression and classification tasks. The models are evaluated using different activation function approximations and network design choices, such as batch normalisation, output standardisation, and weight regularisation. The aim is to find an architecture that performs best across a range of different tasks.

Chapter 3

Implementation

3.1 Method

This work evaluates the performance of neural networks on six different data sets from the UCI Machine Learning Repository, covering both regression and classification tasks. The performance of the networks is evaluated for both encrypted and unencrypted data, with the aim to discover whether using encrypted data is a practical way to protect the privacy of the data during inference across a range of different machine learning tasks. All training will therefore be done on unencrypted data, with encrypted data used during the test/inference stage once the networks have been trained. For each data set, the investigation will be carried out as follows:

- Process and clean data, splitting into training and test sets.
- Train a neural network on the training set and save the weights of the model.
- Obtain predictions from the network on the test set.
- Re-build the trained network using the saved weights and the Python wrapper of PySEAL, allowing for encrypted operations.
- Encrypt the test set and use the re-built network to obtain encrypted predictions on this data.
- Decrypt the predictions and compare with the predictions made on the unencrypted test set.

3.2 Python Wrapper

In order to carry out training and inference on unencrypted data, any deep learning library can be used. This work uses Keras [5]. However, to create networks that can perform inference on data that has been encrypted, Keras cannot be used. As detailed in Section 2.2.2, this is because in order to perform homomorphic operations on the data, it first needs to be represented as a polynomial. As mentioned in Section 2.2.3, we will be using PySEAL in order to perform these operations. PySEAL provides Python APIs for SEAL functions and allows us to encode values as polynomials and encrypt them. It also allows us to perform operations such as addition and multiplication on these polynomials. However, in order to construct a neural network, additional functionality needed to be implemented, which resulted in the implementation of a Python wrapper of PySEAL. This includes encoding matrices, encrypting matrices, performing matrix operations such as the dot product and matrix addition, and performing element-wise operations on encrypted data such as the ReLU activation function. The wrapper also allows the setting of the encryption parameters, such as the polynomial and coefficient moduli. The wrapper can be imported and provides the following methods:

- **Constructor** - encodes numpy arrays as polynomials and optionally encrypts them. This allows matrices to be encoded or encrypted.
- **dot** - performs the dot product between an encrypted and encoded matrix.
- **add (+)** - overriding of Python's add operation to allow encoded/encrypted matrix addition.
- **relu** - element-wise application of the ReLU activation function on encrypted matrices.
- **sigmoid** - element-wise application of the Sigmoid activation function on encrypted matrices.
- **tanh** - element-wise application of the Tanh activation function on encrypted matrices.
- **values** - decrypts a matrix and returns values.

3.3 Activation Function Approximation

Homomorphic encryption does not support non-linearities, meaning activation functions such as ReLU will need to be approximated in order to work in an encrypted version of the network. In CryptoDL [9] the authors attempt to find an optimal approximation for the ReLU, Sigmoid, and Tanh activation functions. They discovered that using modified Chebyshev polynomials was the most successful method of approximating these functions. Following from [9] the ReLU, Sigmoid, and Tanh activation functions were approximated by fitting Chebyshev polynomials. Each activation function is defined as follows:

- $ReLU(z) = max(0, z)$
- $Sigmoid(z) = \frac{1}{1+e^{-z}}$
- $Tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

Chebyshev polynomials are a set of orthogonal polynomials with respect to the weight $\frac{1}{\sqrt{1-x^2}}$ over the range [-1, 1]. They can be defined by the following recurrence relation:

$$\begin{aligned}
T_0(x) &= 1, \\
T_1(x) &= x, \\
T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x).
\end{aligned}$$

$$Where$$

$$T_n(x) = \cos\left(n \arccos\left(x\right)\right).$$

We can then express each activation function, $f(x)$, in the following form, where u is some transformation that maps x to the range [-1, 1]:

$$f(x) = \sum_k c_k T_k(u) = c_0 + c_1 T_1(u) + ... + c_k T_k(u).$$

The coefficients, c, are then solved for by finding the least squares fit to each activation function. The least squares fit finds the coefficients that minimise the error E for function points y:

$$E = \sum_i (y_i - f(x_i))^2.$$

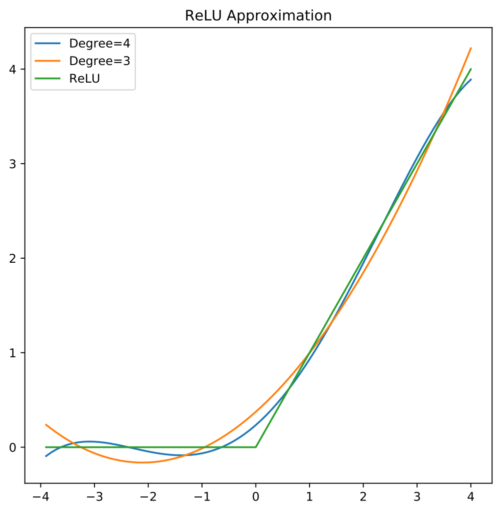
Once the coefficients c have been found, the coefficients a can be found for a polynomial in the form:

$$f(x) = \sum_k a_k x^k.$$

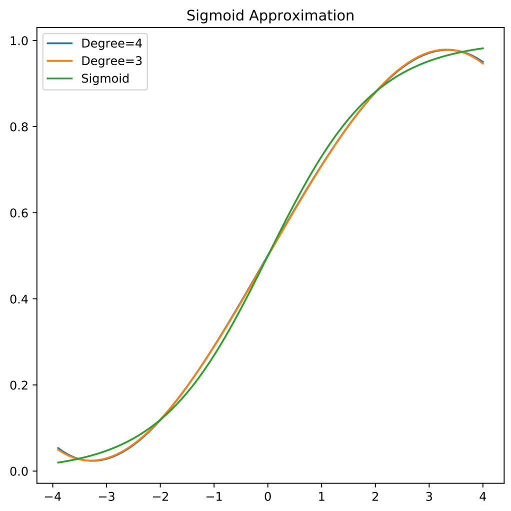
Figure 3.1 shows the approximations of each activation function using Chebyshev polynomials of both degree 3 and 4. We can see that a polynomial of degree 4 provides the best approximation for the ReLU activation function, providing a better fit than an approximation of degree 3. However, for the Sigmoid and Tanh activation functions, polynomials of degree 3 and 4 provide almost identical approximations. Therefore during experiments the ReLU function will be approximated using the degree 4 polynomial, while Sigmoid and Tanh will use the degree 3 approximation. This is done to minimise the number of computations performed. Recalling from Section 2.2, increased computations results in increased noise in the ciphertext. Therefore, proceeding with a lower degree approximation will help reduce the number of computations, keeping the noise increase to a minimum and improving run time.

3.4 Batch Normalisation

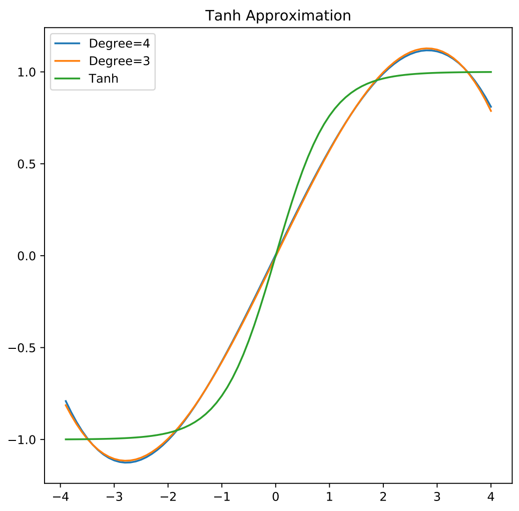
In order for the output of the activation function approximation to be as accurate as possible, the values before the function need to be bounded. Chabanne et al. [4] in-



(a) ReLU



(b) Sigmoid

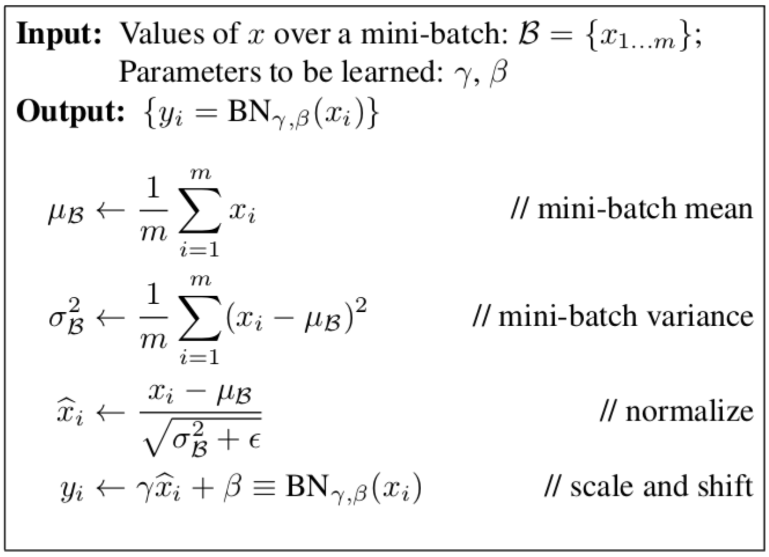


(c) Tanh

Figure 3.1: Activation function approximations.

troduce a batch normalisation layer before applying the ReLU approximation in order to bound values in a tight range. This allows for fitting a better approximation of the activation function, as the values are in a smaller range. Batch normalisation is a procedure that normalises each feature in a neuron across a batch of inputs, by subtracting the mean and dividing by the standard deviation, in order to produce a distribution of values that is approximately normal. There are also additional parameters, γ (scale) and β (shift), that are learned during training. This is because if only normalisation is carried out at each layer, this may change what the layer can represent. Normalising the values before the activation function will result in most values being close to 0.

Therefore, these scale and shift parameters are included so that the transformation can represent the identity transform if needed, allowing the network's representation abilities to remain unhindered. Figure 3.2 shows the procedure for batch normalisation.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 3.2: Batch Normalisation: Algorithm 1 from [11]

Implementing batch normalisation in encrypted neural networks would introduce extra computations, which could have the effect of significantly increasing the noise in the ciphertext and the run time of the computation. In FHE-compatible Batch Normalisation for Privacy Preserving Deep Learning [10], the authors propose a reformulation of the batch normalisation layer during the inference stage that absorbs it into the previous layer of weights and reduces the overall number of operations. They show that when they make this alteration the run time is significantly improved with no loss of accuracy. The reformulation is shown in Equation 1, operating on the data, x, the previous layer's weights and biases, W and b, and the estimates of the mean and variance over all training batches, μ_T and σ_T^2:

$$\begin{aligned}
BN &= \gamma * \left(\frac{(W * x + b) - \mu_T}{\sqrt{\sigma_T^2 + \varepsilon}} \right) + \beta \\
&= \gamma * \left(\frac{(W * x) + (b - \mu_T)}{\sqrt{\sigma_T^2 + \varepsilon}} \right) + \beta \\
&= \gamma * \left(\frac{(W * x)}{\sqrt{\sigma_T^2 + \varepsilon}} + \frac{(b - \mu_T)}{\sqrt{\sigma_T^2 + \varepsilon}} \right) + \beta \\
&= \left(\frac{(\gamma * W * x)}{\sqrt{\sigma_T^2 + \varepsilon}} + \frac{\gamma * (b - \mu_T)}{\sqrt{\sigma_T^2 + \varepsilon}} \right) + \beta \\
&= \left(\frac{(\gamma * W)}{\sqrt{\sigma_T^2 + \varepsilon}} \right) * x + \left(\frac{\gamma * (b - \mu_T)}{\sqrt{\sigma_T^2 + \varepsilon}} + \beta \right) \qquad (1) \\
&= W_{new} * x + b_{new}.
\end{aligned}$$

3.5 Output Standardisation and Regularisation

In addition to batch normalisation, there are other methods for keeping the values low after the first layer. Another approach is to standardise the outputs of the training set before training the models so that they have zero mean and unit variance. By following this approach and in addition, applying L2 regularisation to the weights of the models, all values in the network are kept low in magnitude. Once predictions have been made, they can be scaled back to their original magnitudes. This approach can also be used in combination with batch normalisation, and experiments in the following sections will apply these two techniques in different combinations to observe the outcomes. Note that output standardisation cannot be used for classification tasks as the outputs of the model come from the final activation function, Sigmoid or Softmax.

L2 regularisation involves adding an extra term to the loss function. If the original loss function is defined as $L(y, y')$, where y represents the true values and y' the predicted values, L2 regularisation adds a term involving the weights w of the model as follows:

$$L_{new}(y, y') = L(y, y') + \lambda \sum_{n=1}^{N} w_n^2.$$

The loss function now includes a term which sums the square of all the weight values in the model, and in order to minimise the loss function $L_{new}(y, y')$ the weights in the model must be kept at low values. The regularisation parameter, λ, allows for control over how much to penalise the weights. Weight regularisation also has the added benefit of helping to prevent overfitting.

Chapter 4

Design of Experiments

4.1 Data Sets

The performance of the neural networks is evaluated on six different data sets from the UCI Machine Learning Repository: 3 regression and 3 classification.

- **Abalone** (regression) - Predict the age of an Abalone from physical measurements.
- **Concrete** (regression) - Predict the compressive strength of concrete given various measurements.
- **Real Estate** (regression) - Predict the house price of unit area given various features.
- **Iris** (classification) - Predict the type of Iris plant given various measurements. There are 3 classes in which to classify plants.
- **Bank Notes** (classification) - Predict whether a bank note is genuine or forged given various features.
- **Ecoli** (classification) - Predict the cellular localisation site of ecoli given various features. There are 8 classes.

Data set	**Training instances**	**Test instances**
Abalone	3759	418
Concrete	927	103
Real Estate	331	83
Iris	126	24
Bank Notes	1234	138
Ecoli	268	68

Table 4.1: Number of instances in each data set.

4.2 Network Architecture

The neural networks used for each of the data sets have the same layer structure but with variable inputs, hidden neurons, and outputs depending on the required task. Each network has 3 layers: 1 input, 1 hidden, and 1 output layer. An example of such an architecture can be seen in Figure 2.1. Table 4.2 shows the number of neurons per layer for each data set. The number of hidden neurons was chosen by experimenting with various settings and evaluating the model's performance on unencrypted data.

Data set	No. Inputs (features)	No. Hidden Neurons	No. Outputs
Abalone	10	2	1
Concrete	8	8	1
Real Estate	5	5	1
Iris	4	4	3
Bank Notes	4	4	1
Ecoli	7	8	8

Table 4.2: Network architectures for each data set.

4.3 Experiments

Experiments are set up in order to evaluate the networks for both encrypted and unencrypted data, and compare the predictions made by models operating on each. The experiments compare the final performance of the models for their given tasks, regression or classification, and also evaluates other aspects of these models. The following experiments are conducted:

- **Speed** - A comparison of the run time is carried out. This measurement only takes into account the time taken for network operations and excludes all data processing time, encrypting of data, and decrypting of data. Figure 4.1 shows the point at which the timer is started and stopped.
- **Bounding values** - Comparison of the performance of batch normalisation and output standardisation with L2 weight regularisation at bounding values in a tight region before hidden layer activation function.
- **Activation function** - Comparison of the performance of models using different activation function approximations.

Regression and classification tasks have different methods of performance evaluation:

Regression:

- **Mean Sum Squared Error (MSSE)** - Comparison of the final real-valued output of each model. This is calculated as follow:

$$MSSE = \frac{1}{N}\sum_{n=1}^{N}(y_n - y'_n)^2$$

Where N is the number of data points, y_n is the true value of the n^{th} data point, and y'_n is the predicted value of the n^{th} data point. MSSE is a metric that is sensitive to outliers, as predictions that vary greatly from their true value will contribute their squared difference to the cost. This property is useful when comparing predictions made by models operating on unencrypted and encrypted data.

Classification:

- **Accuracy** - Comparison of the final classification accuracy of each model.
- **MSSE** - Comparison of final layer values before applying the activation function to classify. A comparison is done here as decryption is performed before the final activation function is applied.

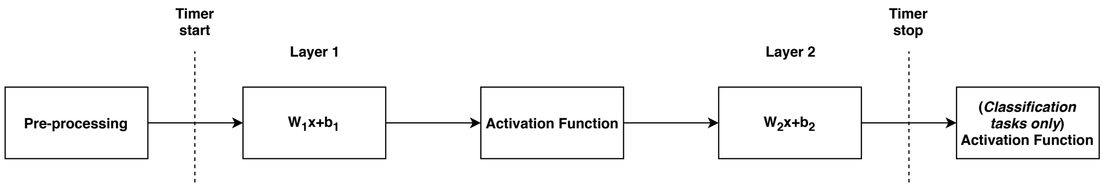


Figure 4.1: Method for evaluating speed of networks.

4.4 Baseline Models

In order to provide some context to the performance of each of the models, baseline results are summarised in Tables 4.3 and 4.4. The baseline regression models are simple linear regression models. The baseline classification models make predictions by taking into the account the training set's class distribution.

Data set	Baseline MSSE
Abalone	4.81
Concrete	107.46
Real Estate	45.82

Table 4.3: Baseline results for regression data sets.

Data set	Baseline Accuracy
Iris	0.29
Bank Notes	0.52
Ecoli	0.37

Table 4.4: Baseline results for classification data sets.

4.5 Experimental Conditions

Experiments were conducted in a virtual machine using the following hardware and software configurations:

- **Operating System** - Ubuntu 16.04
- **CPU** - Intel Core i5 (5257U)
- **Memory** - 4GB

The homomorphic encryption scheme is a levelled scheme (LHE), and as such the encryption parameters need to be set in advance of any experiments carried out. The official SEAL documentation [18] provides recommendations and default parameters for different levels of security. Parameters were chosen in a manner that allowed computations to be completed accurately and securely in networks with 3 layers, while keeping the size of the ciphertext and hence the run time of the models to a minimum. The parameters used for experiments are as follows:

- **Polynomial modulus**: $x^{4096} + 1$
- **Ciphertext coefficient modulus**: 2^{109}
- **Plaintext coefficient modulus**: 2^{16}

Chapter 5

Experiment Results and Analysis

5.1 Abalone

			Unencrypted MSSE			Encrypted MSSE		
Model No.	Batch Norm	Stand. Output	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	✓	4.90	4.80	**4.62**	13.74	5.32	5.11
2	✓	✗	5.00	4.92	4.96	204.54	8.84	5.79
3	✗	✓	4.84	5.36	4.97	**5.04**	5.37	5.27
4	✗	✗	5.04	5.01	5.17	6.50	5.37	6.82

Table 5.1: Abalone MSSE results.

			Unencrypted Time (s)			Encrypted Time (s)		
Model No.	Batch Norm	Stand. Output	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	✓	0.000275	0.000283	0.000327	101	62	63
2	✓	✗	0.0000732	0.0000932	0.000146	102	62	63
3	✗	✓	0.0000663	0.0000970	0.000209	101	78	62
4	✗	✗	0.0000737	0.000103	0.000144	102	64	69

Table 5.2: Abalone run time results.

Tables 5.1 and 5.2 show the results of mean sum squared error (MSSE) and run time experiments on the Abalone data set. When operating on unencrypted data, we can see that Model 1 provides the best performance when using the Tanh activation function. However, for encrypted data, Model 3 with ReLU provides the best performance. Model 3 uses output standardisation and L2 weight regularisation, and the predictions made by this model can be seen in Figure 5.1. We can see that the predictions made by the model operating on unencrypted and encrypted data are very similar, with only minor differences between predictions for the age of an Abalone. This is in contrast to the worst performing model, Model 2 with ReLU, which can be seen in Figure 5.2.

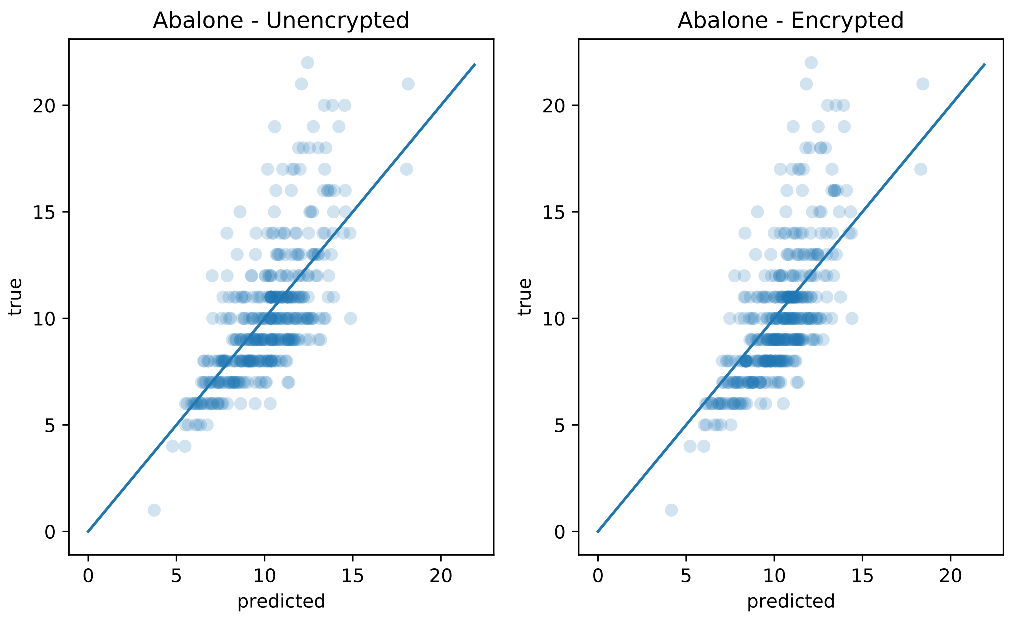


Figure 5.1: Predictions made on unencrypted and encrypted data by Model 3 + ReLU.

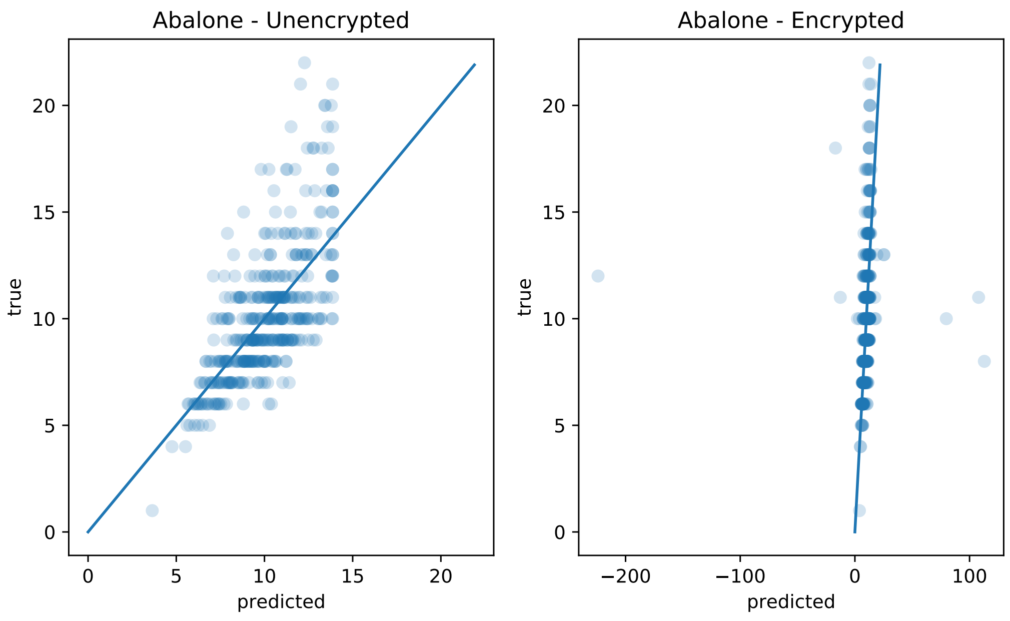
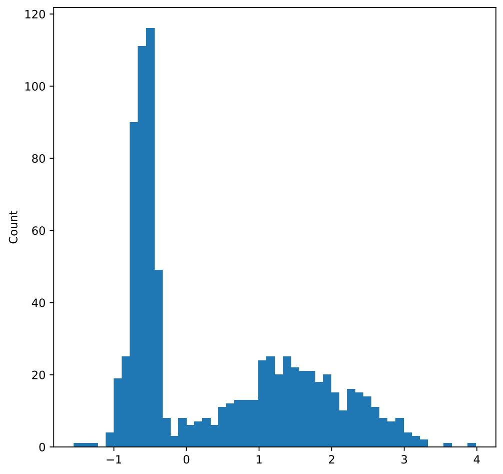


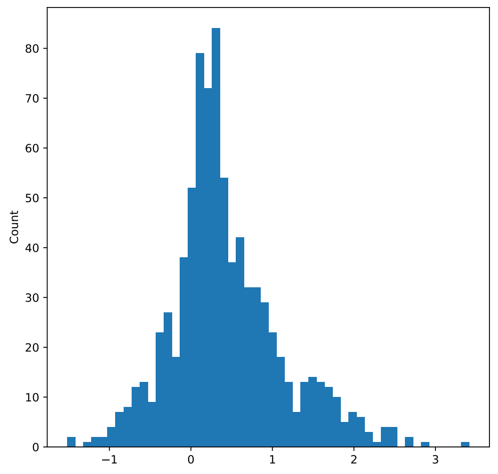
Figure 5.2: Predictions made on unencrypted and encrypted data by Model 2 + ReLU.

Model 2 uses batch normalisation to bound values in a tight range before the application of the activation function in the hidden layer. We can see from Figure 5.2 that while many of the encrypted predictions are close to the true values, there are a small number of outliers with large magnitude that cause an increased MSSE. There are a few different factors that can affect the performance of networks operating on encrypted data, such as the accuracy of the activation function approximation and the range that values lie in the hidden layer.

Figure 5.3 shows the distribution of values in the hidden neurons before the activation function is applied, with Models 2 and 3 shown in Figures 5.3a and 5.3b respectively. We can see in Figure 5.3a that after batch normalisation is applied, many of the values in the hidden neurons are negative. This distribution of values is caused by the scale and shift operations, discussed in Section 3.4, that are applied after normalising. This can cause a problem when the activation function is applied. The ReLU approximation in Figure 3.1a shows that for negative values, the function does not map directly to 0 as the original function. These small errors can propagate through the network and become amplified by large weights, resulting in outliers of large magnitude. In contrast, we can see that in Figure 5.3b most values in the hidden layer neurons are positive, resulting in more accurate results once the ReLU approximation has been applied.



(a) Model 2 + ReLU.



(b) Model 3 + ReLU.

Figure 5.3: Distribution of values in hidden layer before activation function.

Despite Model 3 with ReLU providing the best performance, we can see that the performance of all models can be made much more reliable by using the Sigmoid or Tanh activation function. Models using these activation functions achieve low MSSE scores on predictions across all models, and therefore allow for more flexibility in model design.

A key result to note is the run time of each of the models. The run time of networks operating on encrypted data is significantly longer (x100,000) than networks operating on unencrypted data. This is due to the fact that data is represented as high degree polynomials under the homomorphic encryption scheme, and all operations such as addition and multiplication are performed on these polynomials. The size of these polynomials can be adjusted to reduce the run time but changes will also affect the security level of the scheme and may cause inaccuracies when decrypting. The run time is also significantly affected by the choice of activation function. Models using the Sigmoid or Tanh activation function have a much shorter run time on encrypted data than those using ReLU. This is because ReLU was approximated using a degree 4 polynomial, while Sigmoid and Tanh were approximated using degree 3 polynomials. Therefore there are fewer addition and multiplication operations, resulting in a much faster run time. These results highlight that the activation function approximation is a large bottleneck in computation time.

5.2 Real Estate

			Unencrypted MSSE			Encrypted MSSE		
Model No.	Batch Norm	Stand. Output	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	✓	39.8	40.3	43.0	**35.7**	40.5	45.6
2	✓	✗	50.5	68.9	51.6	9598806.9	479.5	70.3
3	✗	✓	38.3	38.0	**36.5**	156.2	38.6	38.8
4	✗	✗	42.2	67.5	51.0	590484.2	3191.9	78.7

Table 5.3: Real Estate MSSE results.

			Unencrypted Time (s)			Encrypted Time (s)		
Model No.	Batch Norm	Stand. Output	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	✓	0.0000644	0.00549	0.00291	58	28	27
2	✓	✗	0.0000536	0.000077	0.000101	60	28	27
3	✗	✓	0.0000656	0.0000782	0.0000963	58	29	27
4	✗	✗	0.000121	0.000109	0.000107	54	30	28

Table 5.4: Real Estate run time results.

Tables 5.3 and 5.4 show the results of MSSE and run time experiments on the Real Estate data set. Model 3 with Tanh provides the best performance on unencrypted data, while Model 1 with ReLU provides the best performance on encrypted data. Surprisingly, Model 1 performs better on encrypted data than any of the models on unencrypted data. This is not a result that we would expect to see, as we assume that when approximations are introduced into the network that performance will suffer as a result.

Figure 5.4 shows the predictions made by Model 1 on unencrypted and encrypted data. We can see that the predictions are very similar, with no outliers of significant magnitude. While Model 1 with ReLU provides the best performance on encrypted data, we can see that models using the ReLU activation function have the most inconsistent results across model designs. It is the most sensitive to changes in layers such as batch normalisation. In contrast to this, models using the Tanh activation function remain relatively consistent.

Table 5.3 shows that the worst performing model is Model 2 with ReLU, and the predictions made by this model can be seen in Figure 5.5. We can see that most of the

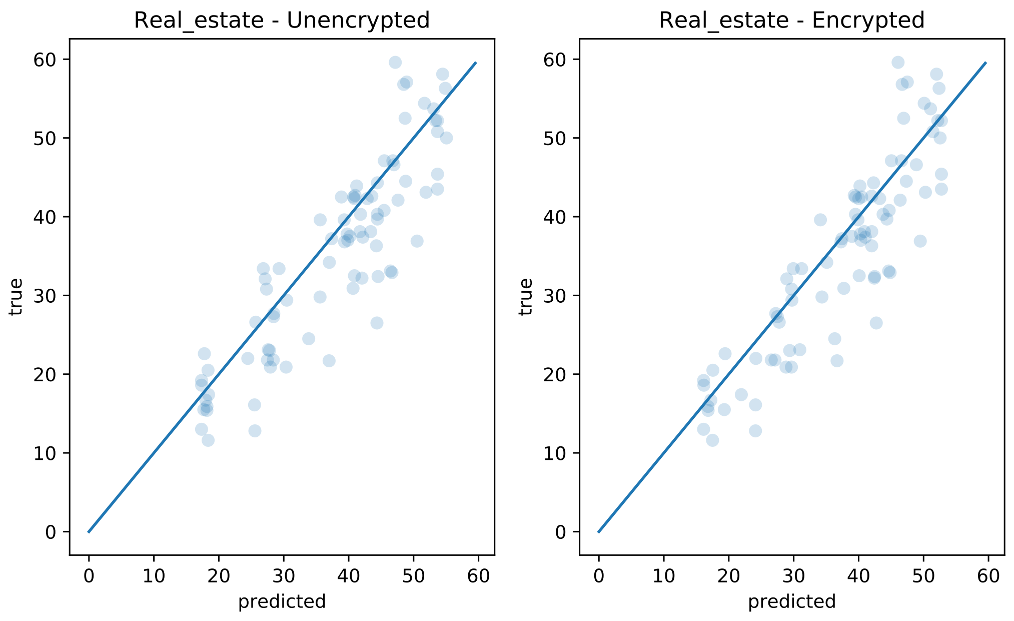


Figure 5.4: Predictions made on unencrypted and encrypted data by Model 1 + ReLU.

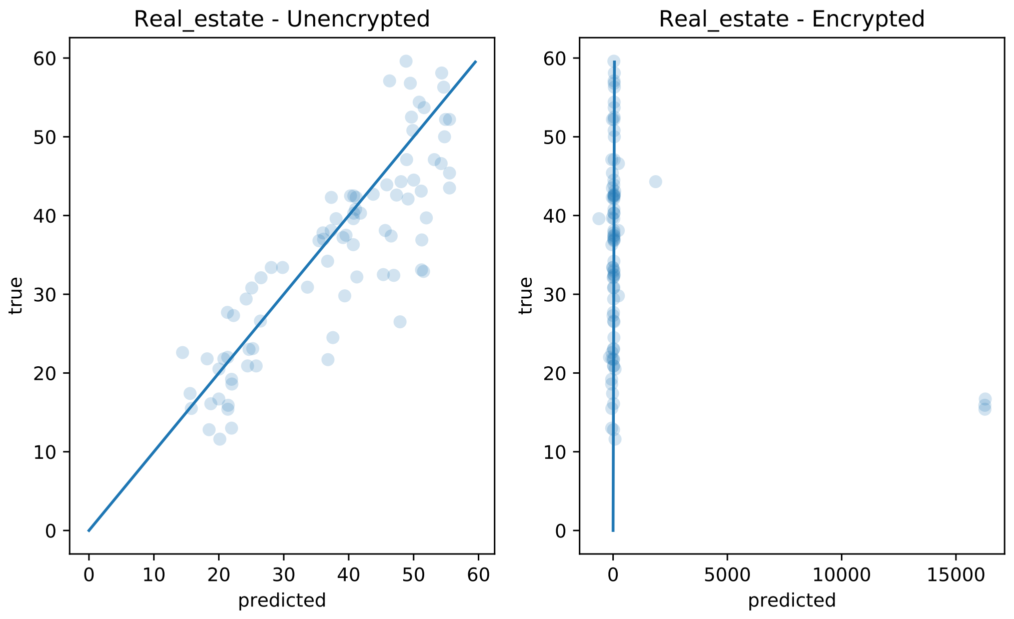
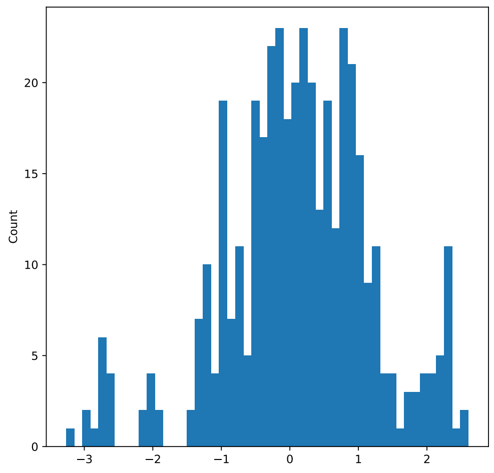


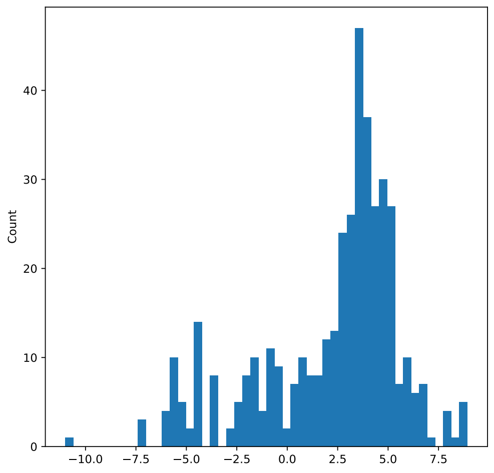
Figure 5.5: Predictions made on unencrypted and encrypted data by Model 2 + ReLU.

predictions lie in the same range as the true values, however, there are a few outliers with significant magnitude which cause the MSSE score of this model to be much larger.

Figures 5.6a and 5.6b show the distribution of values in the hidden layer neurons for



(a) Model 1 + ReLU



(b) Model 2 + ReLU

Figure 5.6: Distribution of values in hidden layer before activation function.

Models 1 and 2 respectively. We can see that the distribution of values for Model 1 is centred around 0, with the values staying within the activation function approximation range of [-4, 4]. However, we can see that the distribution of values for Model 2 is centred at approximately 3.75, with values covering a large range and extending outside the range of activation function approximation. These results show that when using batch normalisation, values may not be bounded effectively. Additionally, when the weights are not regularised, errors caused by the application of the activation function can be amplified and propagated through the network.

5.3 Concrete

			Unencrypted MSSE			Encrypted MSSE		
Model No.	Batch Norm	Stand. Output	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	✓	56.7	56.2	49.1	592.6	8943.3	704.1
2	✓	✗	107.6	50.2	50.5	57686.8	598207.7	30659.2
3	✗	✓	51.7	69.0	56.1	2137.5	**333.3**	655.4
4	✗	✗	62.7	48.5	**45.3**	57703339.6	4242539.7	328243.9

Table 5.5: Concrete MSSE results.

Tables 5.5 and 5.6 show the results of MSSE and run time experiments on the Concrete data set. Model 4 with Tanh provides the best performance on unencrypted data, while Model 3 with Sigmoid provides the best performance on encrypted data. Model 3 uses output standardisation and L2 weight regularisation, and the predictions made by this model can be seen in Figure 5.7. We can see that most of the predictions made

			Unencrypted Time (s)			Encrypted Time (s)		
Model No.	Batch Norm	Stand. Output	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	✓	0.0000932	0.00424	0.00241	107	58	59
2	✓	✗	0.0000663	0.000152	0.000157	117	59	58
3	✗	✓	0.0000619	0.000143	0.000127	123	58	57
4	✗	✗	0.0000691	0.0001	0.000147	117	58	57

Table 5.6: Concrete run time results.

on encrypted data lie close to the true values, however, there are a small number of outliers that have larger magnitudes than all other predictions. Despite this being the best performing model on encrypted data, the MSSE score does not come close to the score on unencrypted data. This highlights that while making predictions on encrypted data works well for most data sets, for some it can be more difficult to get good results.

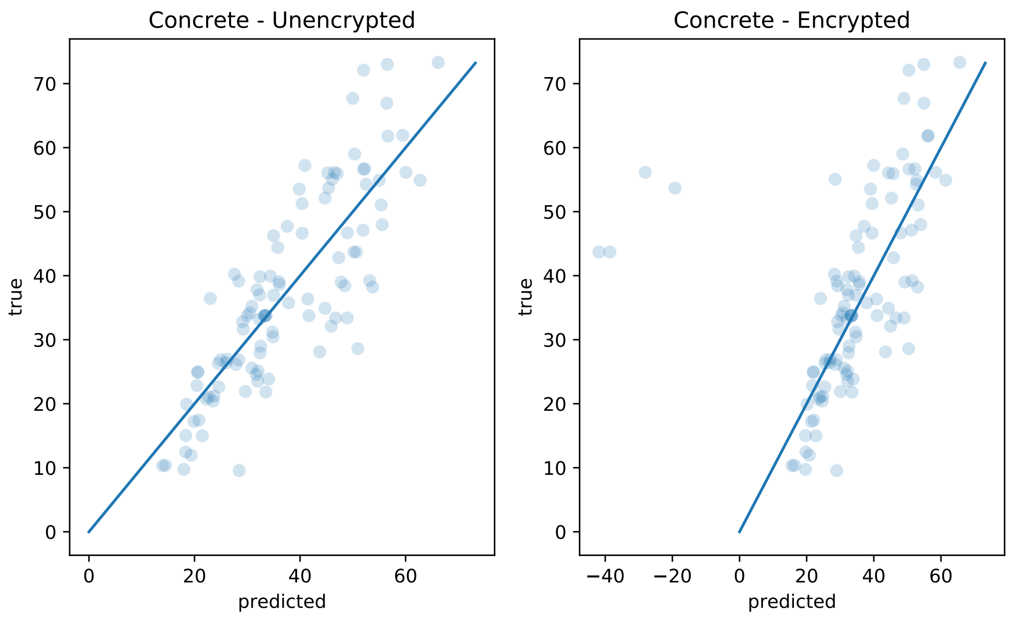


Figure 5.7: Predictions made on unencrypted and encrypted data by Model 3 + Sigmoid.

The second best performing model is Model 1 with ReLU. The predictions made by this model can be seen in Figure 5.8. Model 1 uses batch normalisation, output standardisation, and L2 weight regularisation. We can see that most of the predictions are close to the true values, however, the outliers have a much larger magnitude.

Figure 5.9 shows the distribution of values in the hidden neurons before the activation function is applied, with Models 1 and 3 shown in Figures 5.9a and 5.9b respectively. We can see that for both models there are a small number of values in the hidden layer that fall outside the activation function approximation range. The activation function is approximated in the interval [-4, 4] and therefore when the function is applied this

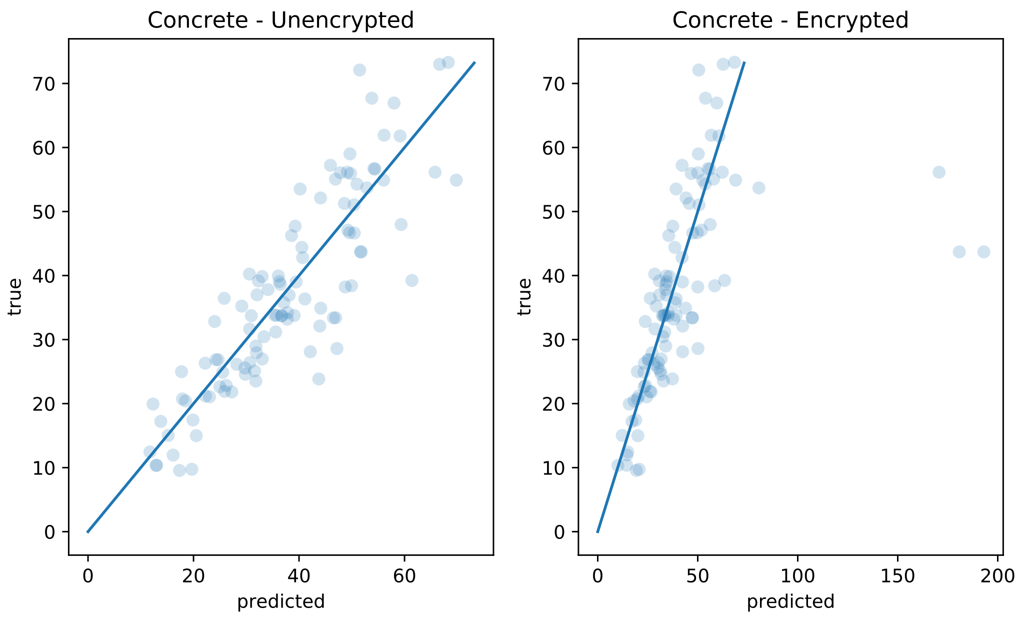
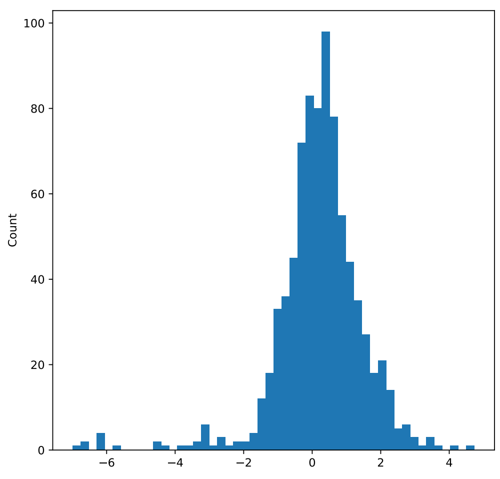
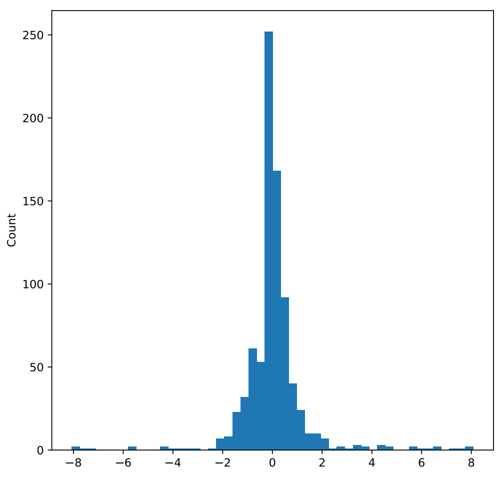


Figure 5.8: Predictions made on unencrypted and encrypted data by Model 1 + ReLU.



(a) Model 1 + ReLU



(b) Model 3 + Sigmoid

Figure 5.9: Distribution of values in hidden layer before activation function.

causes inaccurate results.

Again we can see the run time of models operating on encrypted data far exceeds the time taken for unencrypted data. Models using the Sigmoid or Tanh activation function are also significantly faster than models using ReLU, in some cases completing computations in under half the time.

5.4 Iris

		Unencrypted Accuracy			Encrypted Accuracy		
Model No.	Batch Norm	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	0.958	0.917	0.958	0.958	0.875	0.958
2	✗	0.833	0.875	0.958	0.875	0.875	**1.0**

Table 5.7: Iris Accuracy results.

		Unencrypted Time (s)			Encrypted Time (s)		
Model No.	Batch Norm	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	0.0000403	0.00698	0.0132	11.1	7.6	6.6
2	✗	0.00238	0.0000629	0.0000606	11.1	6.9	6.6

Table 5.8: Iris run time results.

Tables 5.7 and 5.8 show the accuracy and run time results on the Iris data set. Model 2 with Tanh provides the best performance for encrypted data with a perfect accuracy score of 1.0, which surprisingly performs better than all models operating on unencrypted data. The breakdown of classifications made be Model 2 can be seen in Figure 5.10.

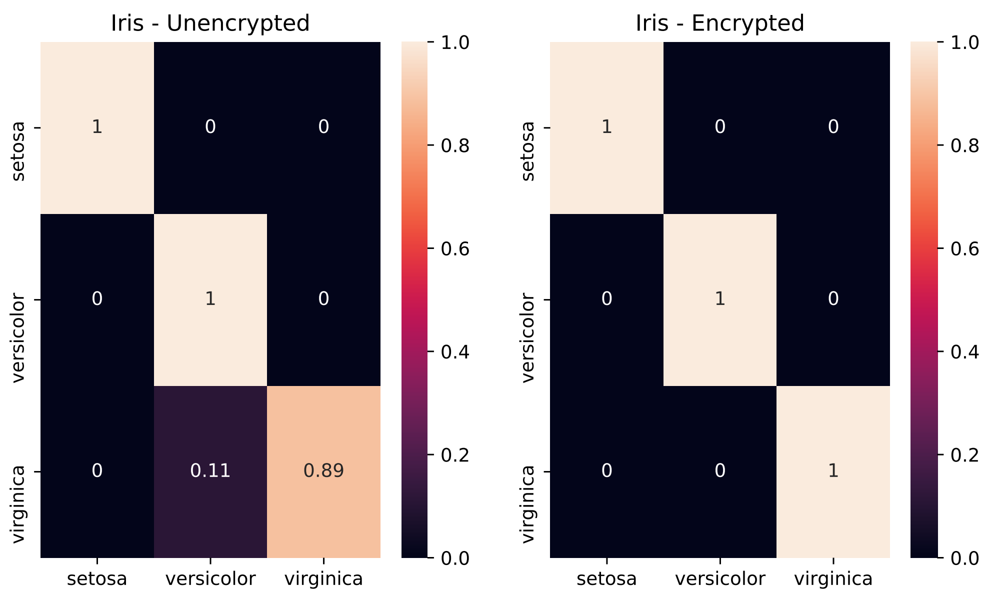
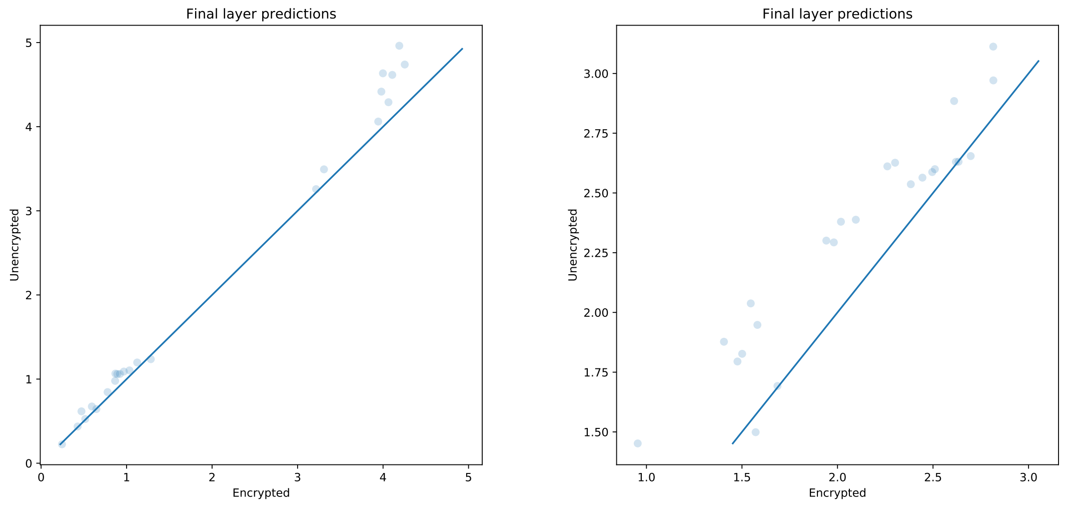


Figure 5.10: Confusion matrices of predictions made on unencrypted and encrypted data by Model 2 + Tanh.

Figure 5.10 shows that the classification accuracy for the ’setosa’ and ’versicolor’ classes remained the same for both unencrypted and encrypted data, however, the model’s accuracy for ’virginica’ increased from 0.89 to 1.0 on encrypted data. This significant increase in accuracy is in part due to the smaller size of the Iris data set, with only 24 test instances. Therefore, slight changes in classifications can have a large impact on accuracy. The Softmax function outputs the probability that each data point is of a certain class. Slight changes in the magnitude of final layer network values can have less impact in classification models for this reason. As long as changes don’t causes data points to cross classification thresholds, the results will remain the same.



(a) Model 2 + ReLU

(b) Model 2 + Tanh

Figure 5.11: Final layer values before application of Softmax activation function for classification.

In order to classify data points, the final layer values are passed through a Softmax activation function in order to provide the probability of an instance being from each class. The final layer values before the application of the Softmax function are shown in Figure 5.11. The final layer values of Model 2 with ReLU can be seen in Figure 5.11a. We can see that the values are split into approximately two clusters. This may make it difficult to classify data points into three classes. The final layer values of Model 2 with Tanh can be seen in Figure 5.11b, and in contrast to ReLU, the values are more spread out. This may make it easier to classify data points into three classes, causing the improvement in accuracy.

5.5 Bank Notes

Tables 5.9 and 5.10 show the accuracy and run time results on the Bank Notes data set. Models 1 and 2 perform equally well on encrypted data when using the ReLU or Tanh activation function. However, when models employ the Sigmoid activation

		Unencrypted Accuracy			Encrypted Accuracy		
Model No.	Batch Norm	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	0.986	0.986	0.978	0.978	0.920	0.978
2	✗	0.978	0.964	0.978	0.978	0.964	0.978

Table 5.9: Bank Notes Accuracy results.

		Unencrypted Time (s)			Encrypted Time (s)		
Model No.	Batch Norm	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	0.0000587	0.00343	0.00231	62	37	36
2	✗	0.00314	0.000145	0.000118	62	39	35

Table 5.10: Bank Notes run time results.

function the accuracy on encrypted data decreases. This is interesting as Model 1 with Sigmoid achieves the joint highest accuracy on unencrypted data. The breakdown of classifications made by Model 1 with ReLU can be seen in Figure 5.12

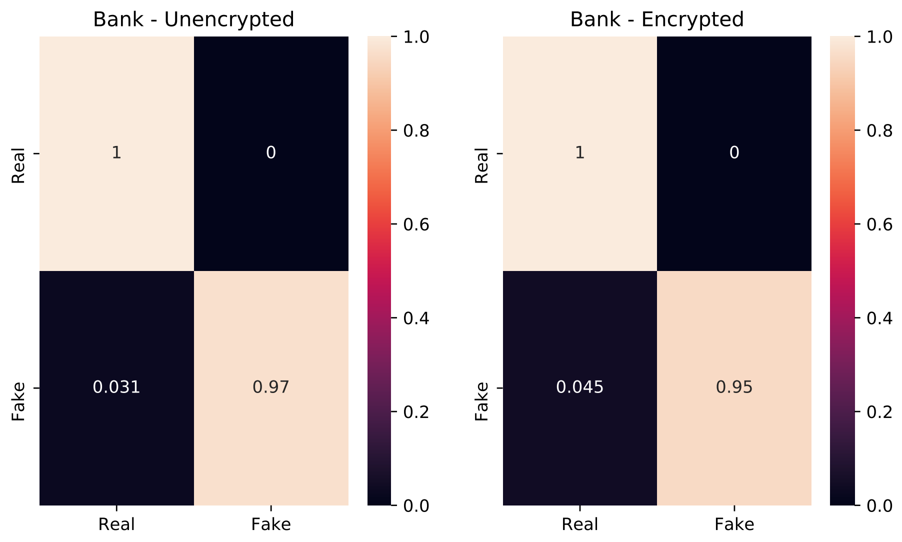
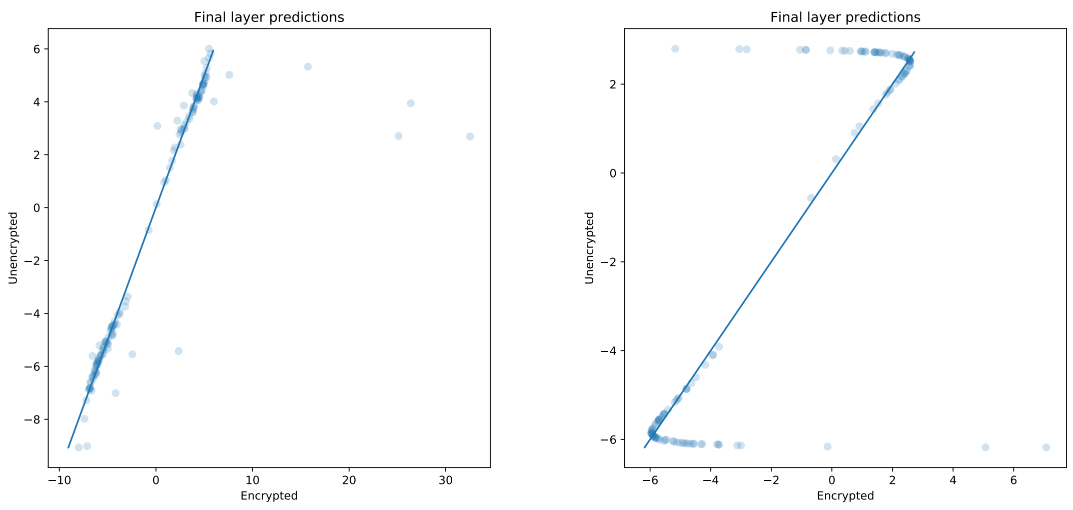


Figure 5.12: Confusion matrices of predictions made on unencrypted and encrypted data by Model 1 + ReLU.

Figures 5.13a and 5.13b show the final layer values for Model 1 and 2 respectively. We can see that by using batch normalisation the magnitude of the encrypted values becomes much larger for both models, varying greatly from the unencrypted values. However, for Model 1 with Relu, this does not affect the classification results, as all values greater than 0 will be classified the same. This is because classification is done using the Sigmoid activation function. However, we can see that Model 2 with Sigmoid has final layer values that are negative when they should be positive. This causes a



(a) Model 1 + ReLU (b) Model 1 + Sigmoid

Figure 5.13: Final layer values before application of Sigmoid activation function for classification.

change in classification and results in a lower overall accuracy.

5.6 Ecoli

		Unencrypted Accuracy			Encrypted Accuracy		
Model No.	Batch Norm	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	0.809	0.838	0.824	0.779	0.662	0.809
2	✗	0.867	0.853	**0.912**	0.853	0.838	**0.912**

Table 5.11: Ecoli Accuracy results.

		Unencrypted Time (s)			Encrypted Time (s)		
Model No.	Batch Norm	ReLU	Sigmoid	Tanh	ReLU	Sigmoid	Tanh
1	✓	0.00435	0.0119	0.00691	69	44	43
2	✗	0.0000694	0.000118	0.000122	69	43	47

Table 5.12: Ecoli run time results.

Tables 5.11 and 5.12 show the accuracy and run time results on the Ecoli data set. Model 2 with Tanh provides the best performance for both unencrypted and encrypted data. The breakdown of classifications made by Model 2 can be seen in Figure 5.14. We can see that the model classifies most data points consistently across unencrypted and encrypted data, however, there are small changes in two classes. This does not affect the overall performance of the model on both types of data.

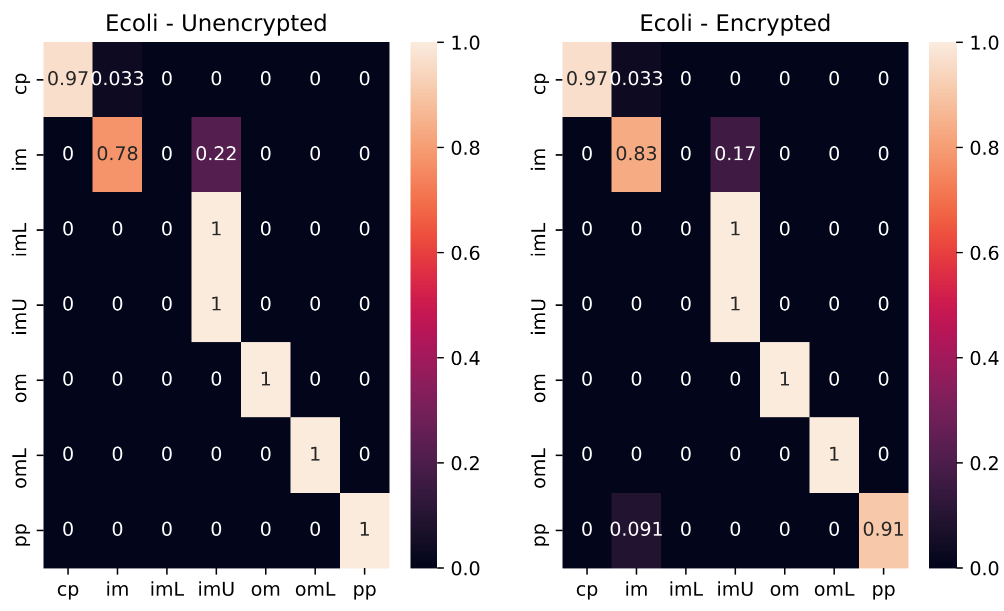
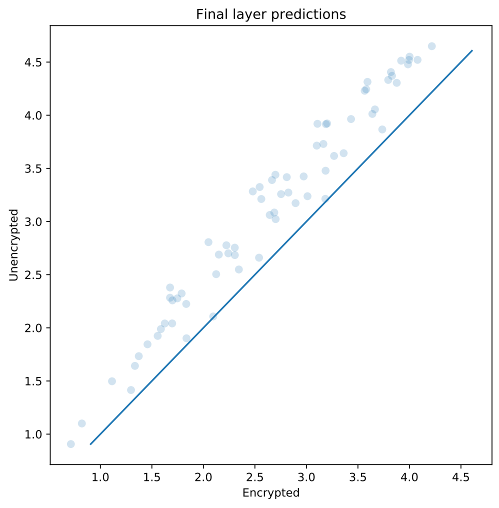
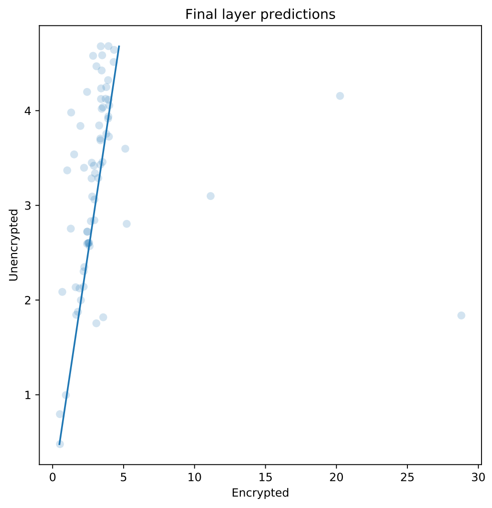


Figure 5.14: Confusion matrices of predictions made on unencrypted and encrypted data by Model 2 + Tanh.

Figures 5.15a and 5.15b show the final layer values of Model 2 and Model 1 respectively. We can see in Figure 5.15a that final layer values are slightly lower on encrypted data, however, they are consistent with the unencrypted values. This is in contrast to Model 1 with Sigmoid, where there are a number of values that have a larger magnitude when operating on encrypted data. It is these differences in magnitude that cause the drop in accuracy on encrypted data.



(a) Model 2 + Tanh



(b) Model 1 + Sigmoid

Figure 5.15: Final layer values before application of Softmax activation function for classification.

Chapter 6

Discussion

It is clear from Chapter 5 that using homomorphic encryption in conjunction with deep learning can produce good results. However, these results can be unreliable and often require different methods of processing in order to get the best performance. In general, all models perform best when values in the network are kept low, especially in the hidden layers. This is important because the activation functions are accurately approximated in a small range, [-4, 4]. If the activation function is applied to values that lie outside this range, the results will not be reliable.

This work evaluated two different methods for keeping network values at low magnitudes: batch normalisation and output standardisation with L2 weight regularisation. Batch normalisation is a technique that has been implemented and evaluated in numerous previous works. However, when implementing this feature across a range of different tasks, its ability to improve performance is inconsistent. This inconsistency is likely due to the scale and shift operations that are applied after normalisation. This can increase the magnitude of values in the hidden layer outside the range in which there is a good approximation for the activation function. A common problem that occurred across all data sets was when a model used batch normalisation with no weight regularisation. This resulted in errors, caused by the application of the activation function, being amplified by large weights and propagated through the network, causing inaccurate results.

For regression tasks, the results suggest that a reliable approach for achieving good performance is to standardise the outputs and apply L2 regularisation to the weights. This approach produced better results than using batch normalisation across all regression data sets and for all activation functions. This improvement in performance is due to values being kept at low magnitudes when passing through the network. This aids the accuracy of the activation function approximation and prevents errors from being amplified as they propagate through the network. It is clear that this method should be applied to all regression tasks. For the Real Estate data set, it was found that using both batch normalisation and output standardisation with L2 regularisation produced the best performing model. Therefore, experimenting with adding batch normalisation alongside output standardisation may be a good approach.

Using encrypted data during inference may be more reliable in classification tasks than regression tasks. Small changes in the magnitude of final layer values will usually not affect the classification results once the Sigmoid or Softmax function is applied. This is because class assignments will only change if the final layer value crosses a certain threshold. For example, if using a Sigmoid function to classify into two classes, such as in the Bank Notes data set, class assignment for a data point will only change if the final layer value changes from positive to negative, or negative to positive. This was seen in Figure 5.13.

The choice of activation function used in each model also affected the performance. The best performing activation function is different across data sets, however, we can see that some are more consistent across network design choices. The results show that ReLU is particularly sensitive to design choices, such as whether to include batch normalisation. This is especially noticeable in the Real Estate data set, where the MSSE value becomes extremely large when using batch normalisation. Sigmoid also exhibits sensitivity to design choices for some data sets, such as Bank Notes and Ecoli, but to a lessor extent than ReLU. The most stable activation function appears to be Tanh, as the range of MSSE and accuracy scores is much smaller across all data sets. Tanh performs well across all data sets and is much less sensitive to model design choices, suggesting that Tanh is the best activation function to use for reliable results on encrypted data.

There are also instances where the models produced results that were not expected. For the Bank Notes and Real Estate data sets some models performed better on encrypted data than unencrypted data. This is a surprising result, as we would expect to see the performance decrease when function approximations are introduced. However, in some cases the inaccuracy of the approximation can work to the model's advantage, producing values that help the model make better predictions.

The results show that models operating on encrypted data can produce competitive results when compared to models operating on unencrypted data, sometimes even outperforming the latter. However, this was not the case for the Concrete data set. Models operating on encrypted Concrete data did not perform well, with predictions that had significantly larger MSSE scores than models operating on unencrypted data. The models that produced the lowest MSSE scores performed well on the majority of the data, however, a small number of outliers with much larger magnitude caused increased MSSE scores. This highlights the difficulty in training models to make predictions on encrypted data, as it can be hard to prevent outliers of this nature for all data sets. Further testing with different network architectures must be done in these cases in order to find a model that is reliable.

An important aspect to note of models operating on encrypted data is the run time of operations. The run time is increased from operating almost instantly, to the point where models may take multiple minutes to complete computations. This is the biggest drawback of operating on encrypted data, as these run times can become too large for practical use in real world applications. In order to reduce these run times, implementing models that use the Sigmoid or Tanh activation function appears to be a significantly better choice. These activation functions can be approximated accurately using degree 3 polynomials, in contrast to ReLU, which requires a degree 4 polynomial. When using the Sigmoid or Tanh activation function the run time of the models can be significantly reduced, in some cases completing operations in under half the amount of time as ReLU. The degree 4 approximation of ReLU requires more additions and multiplications than both Sigmoid and Tanh, and therefore has a detrimental impact on the run time of the models. This highlights that the application of the activation function is a significant bottleneck in the computation.

Chapter 7

Conclusion

7.1 Summary

This work explored the use of deep learning in conjunction with homomorphic encryption to make predictions on encrypted data and evaluated whether this approach is practical for use in real world applications. This was done by evaluating the performance of models operating on encrypted data during the inference stage and comparing this performance to models operating on unencrypted data. This was done for six different UCI data sets that covered both regression and classification tasks, and compared different performance metrics including accuracy and speed.

This work first explored homomorphic encryption (HE) and some of the properties of different HE schemes. The plaintext and ciphertext are represented as polynomials, with encryption involving adding noise to the plaintext. Decryption later relies on accurately removing this noise to some degree of rounding error. The practical considerations involved in HE were also introduced, as performing many multiplication and addition operations causes the noise to increase and can cause errors in accuracy when attempting to decrypt. Additionally, HE schemes do not support non-linearities. This means that activation functions need to be approximated using polynomials.

A number of approaches were then introduced that address the practical considerations involved with applying HE in neural networks. Chebyshev polynomials were used to approximate each activation function. ReLU was approximated using a degree 4 polynomial, while Sigmoid and Tanh were both approximated using a degree 3 polynomial. All activation functions were approximated in the range [-4, 4]. In order to achieve accurate results from applying this approximation, values before the function would need to be bounded in this range. Two different methods were introduced in order to acheive this: batch normalisation and output standardisation with L2 regularisation. Following from previous work [10], batch normalisation operations were absorbed into the first layer of weights. This reduced the total number of operations and ensured better compatibility with HE. The effectiveness of both approaches was evaluated during experiments.

Experiments were carried out on six different data sets and compared the performance of models operating on encrypted and unencrypted data during the inference stage. It was found that models operating on encrypted data performed competitively when compared to those operating on unencrypted data. Some models were able to achieve a similar or better performance than their unencrypted counterparts. However, there was also significant variation in model performance depending on choices of activation function and inclusion of batch normalisation or output standardisation. It was found that the inclusion of batch normalisation negatively impacted performance across data sets. The results suggest that output standardisation with L2 regularisation is a better approach for bounding values in a tight range before the application of the activation function approximation. It was also found that the Tanh activation function provides the most consistent results across different model design choices, and is less sensitive to the inclusion of batch normalisation. The results also showed that models operating on encrypted data have significantly longer run times, making this the greatest obstacle to overcome for use in real world applications. It was noted, however, that Tanh and Sigmoid models completed operations much faster than ReLU, in some cases in under half the run time of ReLU. This was due to the degree 3 polynomials used to approximate these functions, in contrast to ReLU's degree 4.

In summary, using homomorphic encryption in conjunction with deep learning can provide good results. However, the models can be somewhat temperamental. Using the Tanh activation function and output standardisation with L2 regularisation on the weights is recommended to get the best results across data sets. With some tuning and experimentation it is possible to get a model that performs well on encrypted data. The biggest hindrance to practical use in real world applications is the run time of these models, which take significantly longer than models operating on unencrypted data. However, if operating on small amounts of data or in instances where fast inference is not a necessity, these models can make effective predictions while protecting the privacy of the data.

7.2 Future Work

Future work could experiment with deeper network architectures and evaluate their performance as the depth of the network increases. These new networks could build on the optimal design choices discovered in this work, such as using the Tanh activation function, standardising the output, and applying L2 regularisation on the weights. Experiments could also be done using different parameters for encryption, evaluating how the run time and accuracy are affected by different settings of the polynomial modulus and coefficient moduli.

Bibliography

[1] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325, 2012.

[2] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology*, pages 868–886, 2012.

[3] Fanyu Bu, Yu Ma, Zhikui Chen, and Han Xu. Privacy preserving backpropagation based on bgv on cloud. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 1791–1795. IEEE, 2015.

[4] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive*, 2017:35, 2017.

[5] François Chollet et al. Keras. `https://keras.io`, 2015.

[6] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012. `http://eprint.iacr.org/2012/144`.

[7] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC, volume 9*, pages 169–178, 2009.

[8] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

[9] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

[10] Alberto Ibarrondo and Melek Önen. Fhe-compatible batch normalization for privacy preserving deep learning. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 389–404. Springer, 2018.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[12] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). `https://www.cs.toronto.edu/~kriz/cifar.html`.

[13] Lab41. Pyseal. `https://github.com/Lab41/PySEAL`. Accessed 21 September 2018.

[14] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`. 1998.

[15] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. 27th International Conference on Machine Learning*, pages 807–814, 2010.

[16] Michael Nielsen. Neural networks and deep learning. `http://neuralnetworksanddeeplearning.com/chap1.html`. Accessed 22 January 2019.

[17] UCI Machine Learning Repository. `https://archive.ics.uci.edu/ml/datasets.html`. Accessed 30 September 2018.

[18] Microsoft Research. Simple encrypted arithmetic library (seal). `https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library/`. Accessed 21 September 2018.

[19] Shaih. Helib. `https://github.com/shaih/HElib`. Accessed 25 February 2019.

[20] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin Lauter, and Michael Naehrig. Crypto-nets: Neural networks over encrypted data. *arXiv preprint arXiv:1412.6181*, 2014.

[21] Qingchen Zhang, Laurence T Yang, and Zhikui Chen. Privacy preserving deep computation model on cloud for big data feature learning. *IEEE Transactions on Computers*, 65(5):1351–1362, 2016.