Efficient Evaluation of Activation Functions over Encrypted Data

Patricia Thaine
Department of Computer Science
University of Toronto
Toronto, Canada
pthaine@cs.toronto,edu

Sergey Gorbunov
Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
sgorbunov@uwaterloo.ca

Gerald Penn
Department of Computer Science
University of Toronto
Toronto, Canada
gpenn@cs.toronto.edu

***Abstract*—We describe a method for approximating any bounded activation function given encrypted input data. The utility of our method is exemplified by simulating it within two typical machine learning tasks: namely, a Variational Autoencoder that learns a latent representation of MNIST data, and an MNIST image classifier.**

***Index Terms*—machine learning, homomorphic encryption, privacy, security, activation function, non-polynomial function**

I. Introduction

Efficiently calculating non-linear activation functions is essential for deep learning algorithms. These are the combinations and distortions of a neural unit's inputs, using functions such as hyperbolic tangents and sigmoids, that give neural networks their real applicability. Various techniques have previously claimed to compute these functions securely over encrypted data, but use either Secure Two-Party Computation (2PC) [23], [28], distantly approximated functions [21], or special-purpose activation functions [18] as approaches. Unfortunately, the proposed approaches are not realistic in many real-world use cases.

First, in the case of 2PC-based solutions, a client must remain online to communicate with the remote server over multiple rounds and might incur substantial computational costs on small devices. More specifically, in [28], [29], methods are described for running Gaussian mixture models and hidden Markov models on encrypted data that employ an ingenious way of calculating logarithms using oblivious transfer, randomization, and homomorphic encryption. Making use of 2PC as well, [23] use garbled circuits to compute non-polynomial functions. The main limitations of these protocols are communication costs and network availability; a user's network capacity might be overwhelmed by too many calculation requests, or their device might require computations to be performed when they are offline. This could be particularly arduous for users with lower computational capacity or low Internet bandwidth.

Second, in the case for distant approximations of non-polynomial functions, Chebyshev polynomials have been used to approximate the sigmoid function within the encrypted domain [21], [22]. A drawback to these approximations is that their outputs are not bound between $[0,1]$, meaning that machine learning models relying on the sigmoid function for predicting probabilities would fail.

Finally, a special purpose activation function that has been proposed in $f(x)=x^2$, which is not guaranteed to converge.

Homomorphic encryption has also featured in machine learning protocols for fully homomorphic random forests, fully homomorphic naïve Bayes, and fully homomorphic logistic regression [5]. Secure random forests have been used to create language models [33]. [19] design low-degree polynomial algorithms that classify encrypted data. There has also been work on applying differential privacy [13] to neural networks in order to preserve input privacy [1], [26], [27], and on using GANs to train neural networks so that sensitive information cannot be inferred from the model's weights [11].

Contributions. (1) We show how to represent the value of any function over a defined and bounded interval, given encrypted input data, without needing to decrypt any intermediate values before obtaining the function's output. The encrypted representation itself is accurate to within any specified precision tolerance. Functions to which this method is applicable include sigmoid functions, Rectified Linear Units (ReLU), tanh, logarithms, sines, cosines, tangents, and encrypted lookup tables. We propose a method with also $log(N)+2$ multiplications over encrypted integers (where N is the size of the lookup table), which is a natural fit for applications within the NLP and ML domains. This differs from [32], in which a function requiring $\log(N)$ multiplications is proposed, that works for values encoded as binary strings. It also differs from the naive approach of performing lookups via polynomial interpolation. For instance, one can construct a polynomial of degree N to perform 1 of N lookups. However, multiplication in (F)HE is very "expensive" (as it introduces a lot of noise and affects the underlying parameters for FHE ciphertext size). Minimizing the degree of the polynomial and the total number of multiplications is therefore necessary for efficiency. (2) We compare our method on calculating ReLUs with the alternative activation function suggested in [18], within a Variational Autoencoder task that learns latent representations of MNIST images, as well as within an MNIST image classifier itself. For both of these tasks, we compute the losses and accuracies that result from varying the floating point precision of the inputs to the non-polynomial activation

DOI 10.1109/SPW.2019.00022

functions within both of these networks. Finally, we analyse the results of these numerical precision experiments.

II. Background

A. Homomorphic Encryption

Homomorphic encryption schemes allow for computations to be performed on encrypted data without needing to decrypt the data.

For this work, we use Brakerski-Fan-Vercauteren (BFV) encryption, a fully homomorphic scheme based upon ring-learning with errors (RLWE) [7], [14], with the encryption and homomorphic multiplication improvements of [20][1].

B. Notation and Scheme Overview

To describe the BFV scheme, we will use the same notation as [14], but, because we provide here only a brief overview of the homomorphic encryption scheme, the specific optimizations introduced in [14], [20] will not be discussed.

Let $R = \mathbb{Z}[x]/(g(x))$ be an integer ring, where $g(x) \in \mathbb{Z}[x]$ is a monic irreducible polynomial of degree d. Bold lowercase letters denote elements of R and their coefficients will be denoted by indices (e.g., $\mathbf{a} = \sum_{i=0}^{d-1} a_i \cdot x^i$). $\mathbb{Z}_q$ denotes the set of integers $(-q/2, q/2]$, where $q > 1$ is a power of 2. The secret key is called $\texttt{sk} = (1, \mathbf{s})$, where $\mathbf{s} \leftarrow R^2$. The public key is called $\texttt{pk} = ([-(\mathbf{a}\cdot\mathbf{s}+\mathbf{e})]_q, \mathbf{a})$, where $\mathbf{a} \leftarrow R_q$, $\mathbf{e} \leftarrow \chi$.

The plaintext space is taken as R_p for some integer modulus $p > 1$. Let $\Delta = \lfloor q/p \rfloor$ and denote $q \mod p$ by $r_p(q)$ then we clearly have $q = \Delta \cdot p + r_p(q)$.

- $\texttt{Encrypt(pk,}\mathbf{m}\texttt{)}$: message $m \in R_p$, $\mathbf{p_0} = \texttt{pk}[0]$, $\mathbf{p_1} = \texttt{pk}[1]$, $\mathbf{u} \leftarrow R_2$, $\mathbf{e_1}, \mathbf{e_2} \leftarrow \chi$:
$$\texttt{ct} = \Big([\mathbf{p_0}\cdot\mathbf{u} + \mathbf{e_1} + \boldsymbol{\Delta}\cdot\mathbf{m}]_q, [\mathbf{p_1}\cdot\mathbf{u} + \mathbf{e_2}]_q\Big)$$
- $\texttt{Decrypt(sk, ct)}$: $\mathbf{s} = \texttt{sk}$, $\mathbf{c_0} = \texttt{ct}[0]$, $\mathbf{c_1} = \texttt{ct}[1]$.
$$\left[\left\lfloor \frac{\mathbf{p}\cdot[\mathbf{c_0} + \mathbf{c_1}\cdot\mathbf{s}]_q}{q} \right\rceil\right]_p$$
- $\texttt{Add}(\texttt{ct}_1, \texttt{ct}_2)$: $([\texttt{ct}_1[0] + \texttt{ct}_2[0]]_q, [\texttt{ct}_1[1] + \texttt{ct}_2[1]]_q)$
- $\texttt{Mul}(\texttt{ct}_1, \texttt{ct}_2)$: For this paper, we use component-wise multiplication, a simplified description of which is: $([\texttt{ct}_1[0]\cdot\texttt{ct}_2[0]]_q, [\texttt{ct}_1[1]\cdot\texttt{ct}_2[1]]_q)$. We omit the details for obtaining this result [14].

Using homomorphic encryption, we are able to perform linear and, depending on the encryption scheme, polynomial operations on encrypted data (i.e., multiplication, addition, or both). We can neither divide a ciphertext, nor exponentiate using an encrypted exponent. However, we can separately keep track of a numerator and a corresponding denominator. For clarity, we shall refer to the encrypted version of a value $*$ as $E(*)$.

C. Dealing with Error

We can preset our required multiplicative depth at the parameters-generation step of a protocol. The error resulting from ciphertext multiplication is reduced using the scale-invariant method [7], [14].

[1]This scheme is implemented in the PALISADE Lattice Cryptography Library https://git.njit.edu/palisade/PALISADE.

D. Optimization: Single Instruction Multiple Data (SIMD)

Single Instruction Multiple Data (SIMD) is explained in [17], [32]. Using the Chinese Remainder Theorem, an operation between two SIMD-encoded lists of encrypted numbers can be performed by the same operations between two regularly-encoded encrypted numbers[2]. We denote a SIMD-encoded encrypted vector by $E(\mathbf{a}) = E(\langle a_1, ..., a_N \rangle)$.

III. Our Setting and Security Model

We describe our setting as a few phases between a client and a remote (potentially untrusted) server.

- *Setup phase.* In our model, a client runs an FHE setup algorithm and generates a pair of public and secret keys $(\texttt{pk}, \texttt{sk})$. It sends the public key to the server and keeps the secret key private.
  A server performs a preprocessing step where it is given a collection of functions $\{f : D \to R\}$ with small bounded-size domains D and it pre-computes all of the outputs. That is, for all functions f and all inputs to the function $v \in D$, the server pre-computes f_v. It then stores a key-value mapping set $L_f := \{(v, f_v)\}$. (It may, optionally, encrypt the mapping using the client's public key to protect against future server compromises).

- *Online phase.* In the online phase, a client has an input v and it wants the server to compute $E(f_v)$ without learning anything about v. It encrypts v with $\texttt{pk}$, and sends the resulting ciphertext $E(v)$ to the server. The server performs $Eval(\texttt{pk}, E(v), L_f)$ to obtain $E(f_v)$. The server may either send the result to the client or continue to run various functions on the result to learn a model. Eventually, the server sends a ciphertext to the client who decrypts it to learn the results in the clear.

The online phase may be repeated multiple times. Alternatively, a client may batch and send a vector of inputs to the server for computation.

Correctness. For any input v and its associated encryption transmitted by the client, the server learns a valid encryption $E(f_v)$. We prove this in Section IV-A.

Security. First, we distinguish between *malicious* and *semi-honest* servers. A malicious server may arbitrarily deviate from the scheme to learn something about the inputs, while a semi-honest server follows the scheme but tries to learn something about the inputs from the execution trace alone.

We protect against a malicious server (or an adversary that compromises the server) that tries to learn information about inputs v, intermediary values of computation or the output results. Note that under the standard security of FHE, the server learns nothing about $\mathbf{x}$ (introduced in Section IV-A), no matter which functions it chooses to execute over the ciphertext $E(v)$.

Integrity. FHE does not ensure integrity of the results of the computations: that is, the server may deviate from the

[2]This technique is integrated into PALISADE's implementation of the BFV scheme.

computation of f (desired by the client) and compute other functions over the ciphertext. Hence, we must assume that the server runs the functions desired by the client honestly in order to satisfy integrity.

To summarize, our assumptions are as follows:

1) No information about the inputs provided by the client is revealed to even a *malicious server*.
2) Assuming the server is *semi-honest*, no information about the inputs is revealed, and the client learns the correct results of its desired computations.

IV. Description of the Method

A. Algorithm

First, let us look at our proposed table lookup algorithm if it were only to accept plaintext inputs.

Let $\mathbf{f} = (f(x_1), \ldots, f(x_{2^n+1}))$ be a pre-computed vector over predetermined values, where n is the base-2 logarithm of N (i.e., $n = \log(N)$). Let $v \in [1, 2^n]$ be an index in the vector $\mathbf{f}$ at which a client wishes to look up the output value of the function $f(\cdot)$, i.e., we want f_v. We describe an Algorithm 1 for obtaining this.

Let $\mathbf{x} = (v, v, \ldots, v) - (0, 1, \ldots, 2^n)$ be the result of subtracting an index vector of size $2^n + 1$ from a vector of repeated v's (which we call $\mathbf{v}$). Let s denote the number of times we have called LOOKUP, starting at 0, $\cdot$ be the dot product operation, $\times$ be component-wise multiplication of two vectors, and r_{2^s} be a function that rotates a vector to the right by 2^s elements.

Algorithm 1 Table Lookup

```
1: procedure LOOKUP($v$, $\mathbf{x}$, $s$, $\mathbf{f}$)
2:     if $s = \log(|\mathbf{x}| - 1)$ then return $\frac{\mathbf{f}\cdot\mathbf{x}}{x_v}$
3:     else return LOOKUP($v$, $\mathbf{x} \times r_{2^s}(\mathbf{x})$, $s+1$, $\mathbf{f}$)
4:     end if
5: end procedure
```

Intuition. Note that if $v = 1$, then $\mathbf{x} = \mathbf{v} - [0, 1, 2, ..., 2^n - 2, 2^n - 1, 2^n] = [1, 0, -1, ..., 3 - 2^n, 2 - 2^n, 1 - 2^n]$; if $v = 2$, then $\mathbf{x} = [2, 1, 0, -1, -2, ..., 4 - 2^n, 3 - 2^n, 2 - 2^n]$, etc.

Clearly, there will always be a 0 at the $(v+1)^{\text{th}}$ index of $\mathbf{x}$ and a 1 at the $(v)^{\text{th}}$ index of $\mathbf{x}$. Now, iterating the algorithm effectively zeroes out all coefficients in $\mathbf{x}$ in $log(|\mathbf{x}|)$ steps, except for the position at the index which the client wishes to look up. That position will have some integer value. This is why the base case of Algorithm 1 returns $\mathbf{f} \cdot \mathbf{x}$ *divided* by x_v. $\mathbf{f} \cdot \mathbf{x}$ is essentially $f_v \times x_v$, where x_v can be a value different from 1.

Consider Algorithm 2, a slight modification of Algorithm 1.

Proof of Correctness. We prove by induction on s that for all n, at step s of Algorithm 2 we have 2^s many 0's within $\mathbf{x}$ from index $(v+1)$ to index $((v+1) + 2^s) \mod (2^n + 1)$, given $v \in [1, 2^n]$, $|\mathbf{f}| = |\mathbf{x}| = 2^n + 1$, and $\mathbf{x} = \mathbf{v} - [0, 1, 2, 3, 4, ..., 2^n - 2, 2^n - 1, 2^n]$.

Algorithm 2

```
1: procedure RM($v$, $\mathbf{x}$, $s$)
2:     if $s = \log(|\mathbf{x}| - 1)$ then return $\mathbf{x}$
3:     else return RM($v$, $\mathbf{x} \times r_{2^s}(\mathbf{x})$, $s+1$)
4:     end if
5: end procedure
```

Clearly, for $s = 0$, $\mathbf{x}$ has a single 0 at index v. Let us assume that Algorithm 2 has an $\mathbf{x}$ with 2^s consecutive zeros at steps $s = 0, 1, 2, ..., n-1$. We prove that its $\mathbf{x}$ contains 2^s consecutive zeros at step $s = n$, with a single non-zero entry at index v.

At $s = n-1$, there are 2^{n-1} consecutive 0 entries in $\mathbf{x}$ and 2^{n-1} consecutive zeros in $r_{2^{n-1}}(\mathbf{x})$, since it is simply a rotation of $\mathbf{x}$. Given that the very first 0 in $\mathbf{x}$ is at index $v+1$, its last 0 is at index $((v+1) + 2^{n-1}) \mod (2^n + 1)$. This means that the very first 0 within $r_{2^{n-1}}(\mathbf{x})$ is at index $((v+1) + 2^{n-1}) \mod (2^n + 1)$, given the rotation by 2^{n-1} elements to the right. Thus, the very last 0 within $r_{2^{n-1}}(\mathbf{x})$ is at index $((v+1) + 2^{n-1} + 2^{n-1}) \mod (2^n + 1) = ((v+1) + 2 \times 2^{n-1}) \mod (2^n + 1)$. By multiplying $\mathbf{x}$ with $r_{2^{n-1}}(\mathbf{x})$, we get a vector with consecutive zeros from index $(v+1)$ all the way through $((v+1) + 2 \times 2^{n-1}) \mod (2^n + 1)$. How many consecutive zeros? Exactly $((v+1) + 2 \times 2^{n-1}) \mod (2^n + 1) - (v+1) \mod (2^n + 1) = 2^n$. Because the first of the consecutive zeros is at index $v+1$, the only non-zero value of $\mathbf{x} \times r_{2^{n-1}}(\mathbf{x})$ must be at index v.

$s = n$ is therefore the final step since, by definition, $\log(|\mathbf{x}| - 1) = n$. At this point, $\mathbf{x}$ contains 2^n consecutive zeros, with a non-zero value at index v.

We have shown that, by using the core functionality of Algorithm 1 found in Algorithm 2, we end up with a vector $\mathbf{x}$ containing zeros in all indices but one. Specifically, its only non-zero value is at index v. It is thus trivial to see that by computing the dot product of $\mathbf{f}$ and $\mathbf{x}$ and dividing the result by x_v, we obtain f_v exactly.

FHE Method. As per our threat model, v and $\mathbf{x}$ must be encrypted inputs into Algorithm 3.

Algorithm 3 Encrypted Table Lookup

```
1: procedure ELOOKUP($E(v)$, $E(\mathbf{x})$, $s$, $\mathbf{f}$)
2:     if $s = \log(|E(\mathbf{x})| - 1)$ then return $\frac{\mathbf{f}\cdot E(\mathbf{x})}{\sum_j E(\mathbf{x_j})}$
3:     else return ELOOKUP($E(v)$, $E(\mathbf{x}) \times E(r_{2^s}(\mathbf{x}))$, $s+1$, $\mathbf{f}$)
4:     end if
5: end procedure
```

In the encrypted domain, we must separately keep track of the numerator, $\mathbf{f} \cdot E(\mathbf{x})$, and the denominator, $\sum_j E(x_j)$. While `Decrypt(sk,`$\mathbf{f} \cdot E(\mathbf{x})$`)/Decrypt(sk,`$\sum_j E(x_j)$`)` might be exactly equal to f_v for smaller values of n, larger n might take us over the plaintext modulus p, making the result nonsensical. But since we can predict every possible x_v at every step s and we know each value of $\mathbf{f}$, we can solve the equation $\mathbf{f} \cdot (E(\mathbf{x}) \times$

$\mathbf{r}) = f_v \pmod{p}$ for $\mathbf{r}$, which gives us n plaintext values r_i. In practice, p can be chosen based on $\max(l, o)$, where l is the largest index and o is the upper bound on the function output value.

B. Example

Input: an encrypted number $E(v)$, a function vector $\mathbf{f}$, and a range of values (e.g., 0 to 8) with a difference of 1 between each value.
Output: $E(f_v)$
Step 1: create a vector ($\mathbf{I}$) of indices using base-one numbering, a function vector ($\mathbf{f}(\mathbf{I})$) pre-computed on values corresponding to said indices, and an encrypted vector ($E(\mathbf{v})$). Suppose $v = 4$:

$$\mathbf{I} = \begin{bmatrix}0\\1\\2\\3\\4\\5\\6\\7\\8\end{bmatrix}, \mathbf{f}(\mathbf{I}) = \begin{bmatrix}f(1)\\f(2)\\f(3)\\f(4)\\f(5)\\f(6)\\f(7)\\f(8)\\0\end{bmatrix}, E(\mathbf{v}) = E(\begin{bmatrix}4\\4\\4\\4\\4\\4\\4\\4\\4\end{bmatrix})$$

.
Step 2: subtract:

$$E(\begin{bmatrix}4\\4\\4\\4\\4\\4\\4\\4\\4\end{bmatrix}) - \begin{bmatrix}0\\1\\2\\3\\4\\5\\6\\7\\8\end{bmatrix} = E(\begin{bmatrix}4\\3\\2\\1\\0\\-1\\-2\\-3\\-4\end{bmatrix}) = E(\mathbf{x})$$

.
Step 3: shift by one and multiply:

$$E(\begin{bmatrix}4\\3\\2\\1\\0\\-1\\-2\\-3\\-4\end{bmatrix}) \times E(\begin{bmatrix}-4\\4\\3\\2\\1\\0\\-1\\-2\\-3\end{bmatrix}) = E(\begin{bmatrix}-16\\12\\6\\2\\0\\0\\2\\6\\12\end{bmatrix})$$

.
Step 4: shift by two and multiply:

$$E(\begin{bmatrix}-16\\12\\6\\2\\0\\0\\2\\6\\12\end{bmatrix}) \times E(\begin{bmatrix}6\\12\\-16\\12\\6\\2\\0\\0\\2\end{bmatrix}) = E(\begin{bmatrix}-96\\144\\-96\\24\\0\\0\\0\\0\\24\end{bmatrix})$$

Step 5: shift by four and multiply:

$$E(\begin{bmatrix}-96\\144\\-96\\24\\0\\0\\0\\0\\24\end{bmatrix}) \times E(\begin{bmatrix}0\\0\\0\\24\\-96\\144\\-96\\24\\0\end{bmatrix}) = E(\begin{bmatrix}0\\0\\0\\576\\0\\0\\0\\0\\0\end{bmatrix})$$

.
Step 6 (preamble): Since we know $\mathbf{I}$, what every possible value x_i can be, and the ring modulus p, we can pre-compute the following vectors (say $p = 65537$):

$$\begin{bmatrix}7711\\5780\\53977\\\mathbf{14450}\\53977\\5780\\7711\\56381\\0\end{bmatrix} \times \begin{bmatrix}-5040\\1440\\-720\\\mathbf{576}\\-720\\1440\\-5040\\40320\\0\end{bmatrix} = \begin{bmatrix}1 \pmod{65537}\\1 \pmod{65537}\\1 \pmod{65537}\\\mathbf{1} \pmod{\mathbf{65537}}\\1 \pmod{65537}\\1 \pmod{65537}\\1 \pmod{65537}\\1 \pmod{65537}\\0\end{bmatrix}$$

.
Step 6: premultiply:

$$\begin{bmatrix}7711\\5780\\53977\\14450\\53977\\5780\\7711\\56381\\0\end{bmatrix} \times E(\begin{bmatrix}0\\0\\0\\576\\0\\0\\0\\0\\0\end{bmatrix}) = E(\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix})$$

.
Step 7: postmultiply:

$$E(\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}) \cdot \begin{bmatrix}f(1)\\f(2)\\f(3)\\f(4)\\f(5)\\f(6)\\f(7)\\f(8)\\0\end{bmatrix} = E(f(4))$$

.

The table lookup step has linear-time online complexity at runtime.

V. Numerical Analysis

A. Encoding Variables

The very first step in converting an algorithm to an FHE-friendly form is to make the data amenable to FHE operations. This includes transforming floating point numbers into approximations, such as by computing an approximate rational representation for them [19].

We follow the method suggested in [5] by choosing the number of decimal places to be retained based on a desired level of accuracy ϕ, then multiplying the data by 10^{ϕ} and rounding to the nearest integer.

Though straightforward, tests have yet to be run on which values of ϕ would lead to decent accuracies for various tasks. This work makes a contribution in this space.

B. *Numerical Precision Experiments*

To determine how well our method might do in practice, we simulate the floating point precision limitations of the BFV scheme by varying the precision of the inputs into the ReLU and sigmoid activation functions of both a Variational Autoencoder and a handwritten digit classifier. We compare our method for calculating ReLUs to the alternative activation function suggested in [18] for the same handwritten digit classification task. Instead of ReLUs, [18] uses a square activation function ($f(x) = x^2$).

It is worth highlighting that in [19] the authors mention: "[...] functions such as trigonometric functions or the exponential function are not D-polynomial, which rules out methods like exact logistic or probit regression and non-linear neural networks which rely on the evaluation of sigmoidal functions, in particular bounded sigmoid functions which are hard to approximate with polynomials." Furthermore, in [18] the authors mention that they "don't have a good way of dealing with the sigmoid function in the encrypted realm." Seeing as our method can be used to calculate the sigmoid function, it can be applied during the learning phase of an encrypted neural network.

In our experiments, we use the plaintext modulus $p = 14942209$ and set the scheme's parameters to correspond to over a 256-bit security level, given the values presented in [2]. This means that it would take over 2^{256} computations to crack the key.

Using the modulus trick in place of true division, it takes 5219.1 ms to run Algorithm 3 on an Intel Core i-7-8650U CPU @ 1.90GHz and 16GB RAM using an $\mathbf{f}$ of size 5, 10797.3 ms for one of size 9, and 28605.6 ms for one of size 17, 49698.4 ms for one of size 33, 108988 ms for one of size 65, and 237488 ms for one of size 129 [3].

Variational Autoencoder (VAE). We use the PyTorch VAE example to learn a latent representation of MNIST images, which employs an architecture inspired by [24]. It is made up of an encoder that has one fully connected layer with 784 input features and a ReLU activation function, followed by two fully connected layers, each with 400 input features, that produce the means and variances of a 20-dimensional Gaussian distribution. The decoder is made up of a fully connected layer with a ReLU activation function, followed by another fully connected layer with 400 input features with a sigmoid activation function. The results are shown in Table I.

MNIST Image Classification. We test the classification accuracy of the simple Pytorch MNIST model, described in Table II. Table III shows the results of integrating our method within the model. Replacing the ReLUs with a square activation function resulted in 10% accuracy, perhaps because, as mentioned in [18], the square activation function's derivative is unbounded, and so prevents the model from converging during training. In [18], a novel neural architecture for image classification is proposed expressly for the square activation function, obtaining 99% accuracy on the MNIST dataset. Since most prevalent neural network architectures do not use square activation functions, our method may be a more practical option than devising a new alternative model for each task in which a square function either explodes or overfits.

C. *Analysis*

What exactly happens between $\phi = 0$ and $\phi = -1$ for rounding to lead to significantly worse results? Of course, as we lower the value of ϕ, the range of *distinct* input values to the activation function decreases as well. A typical example of this effect, as observed in both the VAE and MNIST image classification experiments, can be seen in Figure 1. We cannot overemphasize how superb the news is for homomorphic encryption enthusiasts that the loss increases minimally when allowing 52 (at $\phi = 0$) distinct numbers to be encoded rather than 549301760 (at $\phi = 5$) as inputs to the VAE decoder's sigmoid function. $\phi = -1$ seems to be the breaking point, at which stage only 6 distinct values are used. To put this into context, if we know that the range of potential inputs to a particular activation function is from -40 to 12, we need to have a lookup table containing only $2^6 + 1 = 65$ values. For reference, at $\phi = 0$, the VAE encoder's ReLU function takes in an aggregate of 27 different input values over the 10 epochs (from -17 to 9), its decoder's ReLU takes in 16 (from -7 to 8), the MNIST image classifier's forward algorithm's first ReLU takes in 22 (from -8 to 13), its second ReLU takes in 65 (from -33 to 31), and its third ReLU takes in 59 (from -21 to 27).

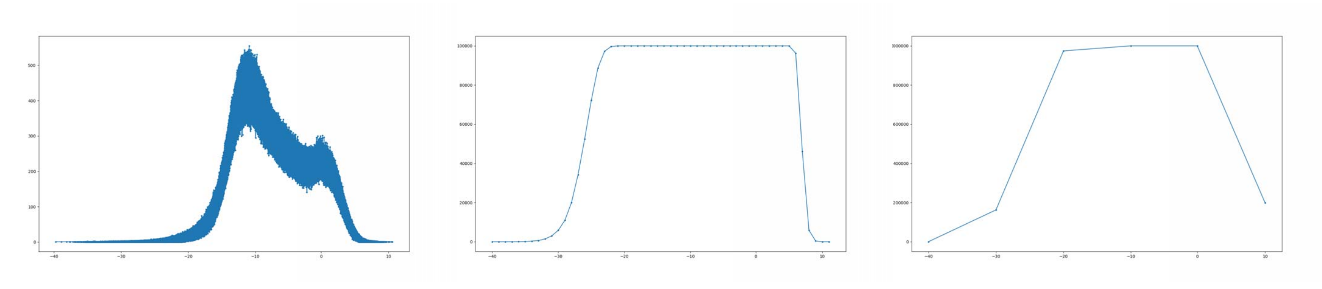
VI. Related Work

Our algorithm can be qualified as a private information retrieval (PIR) protocol. Prior work on PIR using HE is presented in [16], [9], and [12]. [9] propose a single server PIR protocol which relies on both a somewhat homomorphic encryption scheme and a symmetric encryption scheme in order to function. Specifically, the symmetric secret key is encrypted using the homomorphic encryption key and the query index is compared with the indices embedded within the entries of the encrypted database through homomorphic evaluation. This is an example of how to achieve very low communication costs for querying, at the expense of a far

[3]As far as we know, PALISADE does not allow for the rotation of ciphertexts for which the plaintext is unknown, since automorphism keys are first generated for a specific plaintext and are only usable on its corresponding ciphertext. Our runtimes are therefore a result of rotations performed through the matrix multiplications of SIMD-encoded values. In all likelihood, regular ciphertext rotation (without the overhead of matrix multiplications in which only one value within most of the encrypted SIMD-encoded vectors is useful) would significantly improve efficiency.

	original	*1.0e5	*1.0e4	*1.0e3	*1.0e2	*1.0e1	*1.0e0	*1.0e-1
1r	105.8954	105.8795	105.8053	105.796	105.8069	105.7872	106.6232	164.9262
1t	105.8954	105.8554	105.8638	105.8022	105.7999	105.7899	107.8547	182.7708
2r	105.8954	105.8962	105.8831	105.9115	105.8593	105.8793	114.4415	221.438
2t	105.8954	105.8444	105.7312	105.678	105.7479	105.934	190.439	221.438
3r	105.8954	105.8466	105.8598	105.8646	105.836	105.8159	116.906	543.4274
3t	105.8954	105.8021	105.7998	105.825	105.825	106.0815	180.8667	543.4274
4r	107.3935	107.3934	107.3902	107.3664	107.2444	107.6297	114.7207	221.438
4t	107.3935	107.3944	107.3932	107.3943	107.4063	107.3695	145.1698	221.438
5r	107.3935	107.3913	107.3913	107.37	107.37	107.6602	116.7702	543.4274
5t	107.3935	107.3935	107.3932	107.3928	107.3503	107.4812	146.6633	543.4274

TABLE I: Resulting losses for the VAE on MNIST. (Xr) and (Xt) mean the input values were approximated using rounding and truncation, respectively. The results shown are from: **(1)** approximating the inputs to the sigmoid activation function; **(2)** approximating the inputs to both ReLU activation functions; **(3)** approximating the inputs to both ReLU activation functions and the sigmoid activation function; **(4)** replacing the ReLU activation functions with a square activation function and approximating the inputs to square functions; **(5)** replacing the two ReLU activation functions by the square activation function and approximating the inputs to the two square functions as well as to the sigmoid activation function.
The column headers refer to the precision of the input values (e.g., *1.0e5 means $\phi = 5$).



(a) $\phi = 5$ (b) $\phi = 0$ (c) $\phi = -1$

Fig. 1: Histograms of the aggregate number of distinct values over 10 epochs input into the VAE decoder's sigmoid function for various values of ϕ. The x-axis represents the input values and the y-axis represents the quantity of inputs with that value. In (a) there are 549301760 many distinct values, in (b) there are 52, and in (c) there are 6.

Layer Type
10 Conv 5×5
Max-Pooling
ReLU
20 Conv 5×5
Dropout, $p = 0.5$
Max-Pooling
ReLU
50 Fully Connected
ReLU
Dropout
20 Fully Connected
Log Softmax

TABLE II: Description of the MNIST image classifier.

higher computational cost. [16] and [12]'s PIR protocols use polynomials of degree $\log(N) + 1$. While we could frame our algorithm in terms of a polynomial operation, our method regardless requires $\log(N) + 2$ multiplications. We leave a more systematic comparison of these methods for future work.

Work on secure search using homomorphic encryption (e.g., [3] and [4]) is similar to PIR, but instead of using indices to access data at a particular location, the task is to seek a particular value within a dataset. It is easy to see that our method can be adapted to secure search, given a dataset at which no value occurs at more than one index.

To the best of our knowledge, PIR and secure search have not been proposed as methods for computing approximations of non-polynomial functions within the encrypted domain.

VII. Conclusion

We described a novel approach of approximating the value of any function on an interval over which it is defined and bounded, given encrypted input data, without needing to decrypt any intermediate values before obtaining the function's output. This approach can be incorporated into a privacy-preserving neural network. Numerical analysis can be performed in order to optimize our approach for different machine learning tasks and architectures. Furthermore, it makes sense to use a symmetric key encryption scheme with our method, which would likely lower the runtime. It would also be useful to calculate performance metrics for our method when implemented with the popular CKKS [10] homomorphic encryption scheme.

Acknowledgments

We would like to thank Sean Robertson for his feedback on the experiments.

	original	***1.0e5**	***1.0e4**	***1.0e3**	***1.0e2**	***1.0e1**	***1.0e0**	***1.0e-1**
Loss (Rounded)	0.0524	0.0531	0.0535	0.0542	0.0541	0.0534	0.0642	2.301
# Correct (Rounded)	9839	9835	9838	9836	9832	9841	9807	1135
Loss (Truncated)	0.0524	0.0526	0.0535	0.0548	0.0526	0.0542	2.3011	2.3011
# Correct (Truncated)	9839	9838	9834	9828	9836	9829	1135	1135

TABLE III: Resulting losses and number of correct classifications of 10000 test set images from MNIST with the inputs to the three ReLU activation functions approximated at various precisions. The column headers refer to the precision of the input values (e.g., *1.0e5 means $\phi = 5$).

References

[1] Abadi, Martin, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. "Deep learning with differential privacy." In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 308-318. ACM, 2016.

[2] Albrecht, Martin, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi et al. "Homomorphic Encryption Standard." (2018).

[3] Akavia, Adi, Dan Feldman, and Hayim Shaul. "Secure Search via Multi-Ring Fully Homomorphic Encryption." IACR Cryptology ePrint Archive 2018 (2018): 245.

[4] Akavia, Adi, Craig Gentry, Shai Halevi, and Max Leibovich. "Setup-Free Secure Search on Encrypted Data: Faster and Post-Processing Free."

[5] Aslett, Louis JM, Pedro M. Esperana, and Chris C. Holmes. "Encrypted statistical machine learning: new privacy preserving methods." arXiv preprint arXiv:1508.06845 (2015).

[6] Boneh, Dan, Eu-Jin Goh, and Kobbi Nissim. "Evaluating 2-DNF formulas on ciphertexts." In Theory of Cryptography Conference, pp. 325-341. Springer, Berlin, Heidelberg, 2005.

[7] Brakerski, Zvika. "Fully homomorphic encryption without modulus switching from classical GapSVP." In Advances in cryptologycrypto 2012, pp. 868-886. Springer, Berlin, Heidelberg, 2012.

[8] Brakerski, Zvika, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping." ACM Transactions on Computation Theory (TOCT) 6, no. 3 (2014): 13.

[9] Brakerski, Zvika, and Vinod Vaikuntanathan. "Fully homomorphic encryption from ring-LWE and security for key dependent messages." In Annual cryptology conference, pp. 505-524. Springer, Berlin, Heidelberg, 2011.

[10] Cheon, Jung Hee, Andrey Kim, Miran Kim, and Yongsoo Song. "Homomorphic encryption for arithmetic of approximate numbers." In International Conference on the Theory and Application of Cryptology and Information Security, pp. 409-437. Springer, Cham, 2017.

[11] Coavoux, Maximin, Shashi Narayan, and Shay B. Cohen. "Privacy-preserving Neural Representations of Text." arXiv preprint arXiv:1808.09408 (2018).

[12] Dorz, Yarkn, Berk Sunar, and Ghaith Hammouri. "Bandwidth efficient PIR from NTRU." In International Conference on Financial Cryptography and Data Security, pp. 195-207. Springer, Berlin, Heidelberg, 2014.

[13] Dwork, Cynthia. "Differential privacy." Encyclopedia of Cryptography and Security (2011): 338-340.

[14] Fan, Junfeng, and Frederik Vercauteren. "Somewhat Practical Fully Homomorphic Encryption." IACR Cryptology ePrint Archive 2012 (2012): 144.

[15] Fontaine, Caroline, and Fabien Galand. "A survey of homomorphic encryption for nonspecialists." EURASIP Journal on Information Security 1 (2009): 41-50.

[16] Gentry, Craig, and Dan Boneh. A fully homomorphic encryption scheme. Vol. 20, no. 09. Stanford: Stanford university, 2009.

[17] Gentry, Craig, Shai Halevi, and Nigel P. Smart. "Fully homomorphic encryption with polylog overhead." In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 465-482. Springer, Berlin, Heidelberg, 2012.

[18] Gilad-Bachrach, Ran, 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, pp. 201-210. 2016.

[19] Graepel, Thore, Kristin Lauter, and Michael Naehrig. "ML confidential: Machine learning on encrypted data." In International Conference on Information Security and Cryptology, pp. 1-21. Springer, Berlin, Heidelberg, 2012.

[20] Halevi, Shai, Yuriy Polyakov, and Victor Shoup. "An improved RNS variant of the BFV homomorphic encryption scheme." In Cryptographers Track at the RSA Conference, pp. 83-105. Springer, Cham, 2019.

[21] Hesamifard, Ehsan, Hassan Takabi, and Mehdi Ghasemi. "CryptoDL: Towards Deep Learning over Encrypted Data." In Annual Computer Security Applications Conference (ACSAC 2016), Los Angeles, California, USA. 2016.

[22] Hesamifard, Ehsan, Hassan Takabi, Mehdi Ghasemi, and Rebecca N. Wright. "Privacy-preserving machine learning as a service." Proceedings on Privacy Enhancing Technologies 2018, no. 3 (2018): 123-142.

[23] Juvekar, Chiraag, Vinod Vaikuntanathan, and Anantha Chandrakasan. "GAZELLE: A Low Latency Framework for Secure Neural Network Inference." In 27th USENIX Security Symposium (USENIX Security 18), pp. 1651-1669. 2018.

[24] Kingma, Durk P., Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. "Semi-supervised learning with deep generative models." In Advances in neural information processing systems, pp. 3581-3589. 2014.

[25] Paillier, Pascal. "Public-key cryptosystems based on composite degree residuosity classes." In International Conference on the Theory and Applications of Cryptographic Techniques, pp. 223-238. Springer, Berlin, Heidelberg, 1999.

[26] Papernot, Nicolas, Martn Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. "Semi-supervised knowledge transfer for deep learning from private training data." arXiv preprint arXiv:1610.05755 (2016).

[27] Papernot, Nicolas, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and lfar Erlingsson. "Scalable private learning with PATE." arXiv preprint arXiv:1802.08908 (2018).

[28] Pathak, Manas, Shantanu Rane, Wei Sun, and Bhiksha Raj. "Privacy preserving probabilistic inference with hidden Markov models." In 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5868-5871. IEEE, 2011.

[29] Pathak, Manas A., and Bhiksha Raj. "Privacy-preserving speaker verification and identification using gaussian mixture models." IEEE Transactions on Audio, Speech, and Language Processing 21, no. 2 (2013): 397-406.

[30] Rivest, Ronald L., Len Adleman, and Michael L. Dertouzos. "On data banks and privacy homomorphisms." Foundations of secure computation 4, no. 11 (1978): 169-180.

[31] Rothblum, Ron D. "On the circular security of bit-encryption." In Theory of Cryptography Conference, pp. 579-598. Springer, Berlin, Heidelberg, 2013.

[32] Smart, Nigel P., and Frederik Vercauteren. "Fully homomorphic SIMD operations." Designs, codes and cryptography 71, no. 1 (2014): 57-81.

[33] Su, Yi, Frederick Jelinek, and Sanjeev Khudanpur. "Large-scale random forest language models for speech recognition." In Eighth Annual Conference of the International Speech Communication Association. 2007.