Zachary Zanussi

Supervised Text Classification with Leveled Homomorphic Encryption

Zachary Zanussi*[1], Benjamin Santos[1], Saeid Molladavoudi[1]

[1] Statistics Canada, Ottawa, Canada.

Abstract:

Privacy concerns are a barrier to applying remote analytics, including machine learning, on sensitive data via the cloud. In this work, we use a leveled fully Homomorphic Encryption scheme to train an end-to-end supervised machine learning algorithm to classify texts while protecting the privacy of the input data points. We train our single-layer neural network on a large simulated dataset, providing a practical solution to a real-world multi-class text classification task. To improve both accuracy and training time, we train an ensemble of such classifiers in parallel using ciphertext packing.

Keywords: Privacy Preservation, Machine Learning, Encryption

1. Introduction:

Real time statistical products require access to real time data, however, access to real time unencrypted transactional data may provide exposure to confidentiality threats and cybersecurity attacks. Even with secure cyber environments that are resistant to outside threats, data is still at risk of misuse from insiders with proper data access clearances.

In recent years, there has been a significant growth in emerging privacy enhancing computation techniques that would potentially provide protection against such attacks while enabling analytics, including Machine Learning (ML) tasks such as training and inference. Among the existing techniques, fully Homomorphic Encryption (HE) is a prominent candidate to address the privacy-related issues that arise in ML-as-a-service scenarios where the ML tasks, such as *training* and *inference*, are delegated to an untrusted service provider, such as the cloud.

HE schemes are asymmetric crypto-systems that support homomorphic evaluation of arbitrary computable functions on encrypted data. More specifically, HE allows arbitrary arithmetic operations, such as addition and multiplication, to be performed on the encrypted data. That is, one may apply these operations on encrypted data to obtain an encryption of the result of the corresponding cleartext operation. The *leveled* HE schemes support a predetermined number of operations, in particular multiplication, based on the target circuit.

A client seeks to train a neural network on sensitive data by outsourcing it to a cloud computer. We assume this cloud is *semi-honest*, or honest-but-curious, so that it will follow any protocol we assign it but it will attempt to learn anything it can in the process. The privacy goal is that no new information leaks from the client's private dataset to the server, beyond what can be inferred from the trained model. After training, the cloud can either perform predictions for the client, or can return the model to the client for local, unencrypted inference. Despite many advances and improvements in HE schemes, they have not been widely used in computationally intensive ML tasks such as model training.

The high memory requirements for text classification are compounded by the computational inflation HE imposes. Much of the work on this topic in the recent years has been transforming these encrypted learning systems into practical algorithms that balance data security with

reasonable computation cost.

In this paper, we utilize the leveled HE scheme of Cheon, Kim, Kim & Song (2017) to train a single layer neural network and make predictions for the ML text classification task. The data we use in this proof-of-concept is a publicly available dataset of product descriptions that we classify into an internationally recognized retail product classification system. We propose a privacy-preserving classifier that is *secure* and *practical*. We first review some preliminaries.

Homomorphic Encryption. Focused research over the past decade has transformed Homomorphic Encryption (HE) into a full-fledged encryption scheme, capable of performing computations on sensitive data without sacrificing privacy. Research teams are using it to perform machine learning, statistical analysis, and more. In fact, systems using HE are becoming advanced enough to be deployed in real-world systems; for example, Raisaro, Troncoso-Pastoriza, Misbach, Sousa et al. (2018).

The HE cloud computation paradigm is an asymmetric scheme that proceeds as follows. A data source encrypts their data using someone's public key; it is now impossible to access it unless you hold the appropriate secret key. This data can now be safely transferred to the cloud, even through insecure communication channels. Upon receipt, the cloud can perform the desired calculations homomorphically on the encrypted values, even though it cannot read the original input or any intermediate or final results. These final encrypted results can be sent to the holder of the secret key (typically the client, or possibly a third party), who can then decrypt and read them. While there are multiple schemes that implement HE on numerical data, we only consider the scheme of Cheon, Kim, Kim, and Song as it is designed for calculations on real numbers. Due to the approximate nature of its calculations this scheme is perfect for ML analysis. We have made extensive use of the CKKS scheme, so we describe the basics presently.

CKKS Encryption Scheme. The Cheon–Kim–Kim–Song (CKKS) homomorphic encryption scheme is designed for floating-point arithmetic, which makes it ideal for storing the weight matrices of a neural network. In the proceeding, we give a high-level description of the scheme's interface; the interested reader should refer to Cheon, Kim, Kim & Song (2017) for a description of the underlying mathematical structures and algorithms.

Suppose x is a vector of real numbers; we would like to encrypt x and perform homomorphic operations on it. Using a public key, we can encrypt this entire vector into a single *ciphertext*, denoted by [**x**]. That we encrypt a *vector* of values is a non-trivial detail; as we will see, the way values are organized within the vector can affect performance and is referred to as *packing*. We can think of the data encrypted in a ciphertext [**x**] as a vector of values of the form $x = (x_0, x_1, \cdots, x_k)$. Each coordinate of this vector is called a *slot*.

Given two ciphertexts [**x**] and [**y**], we have access to homomorphic addition $\oplus$ and multiplication $\otimes$ operations. These operations are performed *slot-wise*; that is, the homomorphic sum [**x**] $\oplus$ [**y**] results in a ciphertext [**x + y**] that encrypts a vector that is the component-wise sum of the vectors x and y. The same is true for homomorphic multiplication. This highlights the importance of the packing structure; values need to be correctly aligned during calculations.

We also have access to an operation called *rotation*. Simply put, we may rotate all the values in x left or right by any number of slots. This relatively costly operation allows us to interact values within a ciphertext with themselves; for example, repeatedly adding [**x**] to its rotation results in a ciphertext **total**([**x**]) that encrypts the total $\sum_i x_i$ in every slot.

Every ciphertext exists on a level, and multiplications consume levels. This puts an upper bound on the multiplicative depth of circuits that can be performed. Further, factors and summands must be on the same level in order to perform operations between them.

Text Classification. Natural Language Processing (NLP) as a field has benefited immensely from the development of various ML algorithms in the previous decades. Text classification is a task in NLP that takes unstructured input text and attempts to place it into one or more classes. Applications include spam detection and sentiment analysis. Different approaches (see Kowsari, Meimandi, Heidarysafa, Mendu et al. (2019)) have found success in a variety of text classification problems. Many of them, such as deep neural networks or recurrent neural networks such as long short-term memory networks, require deep circuits that have many multiplications and are thus unsuitable for use with leveled HE.

On the other hand, simpler approaches such as bag-of-words and low-depth neural networks provide cheaper solutions for the same text classification tasks at the cost of disregarding context, grammar and word order. The bag-of-words model involves encoding a vector based on the presence of words in the text. Even though simple, these approaches have been successfully used as feature generation tools in information retrieval and document classification. These simple techniques have provided an effective and HE-friendly approach to our supervised text classification problem.

2. Methodology:

In this section, we describe the methods we followed to implement our text classification protocol, including the structure of the ensemble network and the intricacies introduced by incorporating HE.

Network Structure. From prior experience in classifying text data, including the data that we are targeting, a single layer network and a bag of words encoding is often sufficient for an acceptable model performance. Working with such a shallow network is beneficial from a computational complexity standpoint, especially in a leveled HE scheme such as CKKS.

To maximize performance with our leveled scheme, we train an *ensemble*. In an ensemble model $\mathcal{M}$, a number $S > 1$ of submodels $\mathcal{M}_s$ are each trained separately, and at prediction time these models vote to determine the ensemble prediction. We packed several submodels into a single ciphertext, which effectively trains multiple submodels in parallel. Each of our submodels is a single layer neural network composed of multiplication by a weight matrix W_s. We opt not to use an activation function, as the small accuracy gain was not found to be worth the computational cost of including one.

Our training set consists of N pairs $\{x, y\}$, where $x \in \mathbb{R}^M$ and y is a one-hot encoded vector in $\mathbb{R}^L$ representing one of L classes. Each model $\mathcal{M}_s$ consists of an $L \times M$ weight matrix W_s, randomly initialized with small real values. The weight matrix is multiplied by a data vector x to obtain a vector of *logits* $z = \mathcal{M}_s(x) = W_s x \in \mathbb{R}^L$. The ensemble of submodels performs a soft vote $\mathcal{M}(x) = \sum_s \mathcal{M}_s(x)$ and then selects the index of the largest logit to obtain the ensemble prediction.

We use Mean Squared Error as our loss function, and to perform model updates we compute

$$W_s^{(t+1)} = W_s^{(t)} - \lambda\left(W_s^{(t)} x_i - y_i\right) \cdot {x_i}^T. \tag{1}$$

We have accelerated our gradient descent protocol using a protocol due to Nesterov (2004).

Encrypted Gradient Descent Protocol. We present pseudo-code for our encrypted gradient descent protocol for a single submodel $\mathcal{M}_s$ in Algorithm 1. The ensemble training procedure is a straightforward extension of this protocol. Data is encrypted in the form $[\mathbf{x}]$ where x is a vector in $\mathbb{R}^M$, and the corresponding output labels are $y \in \{1, \ldots, L\}$. The labels would typically be one-hot encoded into a single vector, but in the encrypted protocol they are instead encrypted into a series of ciphertexts $\{[\mathbf{y}]_l\}_{l=1}^L$, where $[\mathbf{y}]_l$ encrypts a vector of M ones if $y = l$ and zeros

otherwise. The $L \times M$ weight matrix W_s is encrypted row-wise into ciphertexts $\{[\mathbf{W}_s]_l\}_{l=1}^{L}$.

Algorithm 1: Encrypted Training Procedure

input : encrypted training dataset, saved to file
learning rate λ and momentum coefficient γ
output: trained weights, $[\mathbf{W}_s]_l$

1 **for** *each update and* $l < L$ **do**
2 momentum look-forward, $[\mathbf{W}_s]_l = [\mathbf{W}_s]_l - \gamma[\mathbf{v}]_l$;
3 load in data X;
4 **for** *each* $[\boldsymbol{x}]$ *in* X **do**
5 propagate forward, $[\mathbf{z}]_l = \mathbf{total}([\mathbf{W}_s]_l \otimes [\mathbf{x}])$;
6 subtract true labels, $[\mathbf{dz}]_l = [\mathbf{z}]_l - [\mathbf{y}]_l$;
7 compute gradient, $[\mathbf{dW}_s]_l^x = [\mathbf{dz}]_l \otimes [\mathbf{x}]$;
8 accumulate gradient, $[\mathbf{dW}_s]_l$ += $[\mathbf{dW}_s]_l^x$;
9 multiply by learning rate, $[\mathbf{dW}_s]_l = \frac{\lambda}{N}[\mathbf{dW}_s]_l$;
10 update weights, $[\mathbf{W}_s]_l = [\mathbf{W}_s]_l - [\mathbf{dW}_s]_l$;
11 update momentum, $[\mathbf{v}]_l = \gamma[\mathbf{v}]_l + [\mathbf{dW}_s]_l$;
12 preparation for next update;

In the ML literature, the term "epoch" typically refers to one pass over the dataset, which often involves several model updates over batches of data. In the ciphertext model, we are not limited by the number of epochs but rather the number of times the model is updated–see line 10 in Algorithm 1. Thus, we measure the training progress of our model in terms of *updates*, rather than epochs.

Data. Statistics Canada collects real-time data from major retailers for a variety of data products. These data, which include some identifiers, a product description, and the transaction price, are commonly referred to as "scanner data", named after the price scanners used to ring a customer through checkout. This is a very valuable data source, used in the generation of, among others, the Consumer Price Index (Statistics Canada. (2021)). The organization treats this data as sensitive and endeavors to protect the privacy of it and the retailers that provide it.

The first step to processing this data is to classify the product descriptions into an internationally standardized system of product codes called North American Product Classification System (NAPCS) codes. This system is used to classify different types of products for comparison. For example, one code may correspond to coffee and related products. Each entry in the scanner data needs to be assigned one of these codes, based on the product description given by the retailer. As a proof-of-concept, we replace the scanner data with a synthetic data source, which allows us to conduct experiments without fear of putting the security of the data at risk. This dataset is adapted from USDA's FoodData Central (U.S. Department of Agriculture (USDA)), and consists of 50,000 product descriptions from 5 different NAPCS codes.

The data were encoded using a word-level bag-of-words encoding. That is, each word used in the dataset was enumerated and stored in a dictionary $\mathcal{D}$, resulting in 4,030 distinct words. A product description d was encoded into a 4,030-dimensional vector v, where $v_i = 1$ if and only if the i-th word appears in d. These vectors were padded with zeros to fit into the 4,096-slot chunks during encryption.

3. Results:

In this section, we outline some of the details of the experiments we ran. First, we detail the cloud compute environment we ran on. Next, we share some of the details of the different training procedures used. Finally, we share the timing and performance results of our encrypted text classification protocol.

Compute Environment. All experiments were performed in the Microsoft Azure cloud, using a virtual machine with 32 GB of memory and 8 virtual CPUs. We have also implemented multithreading to utilize the multiple cores available. It should be noted that the cleartext model does not utilize multithreading. We note that the cost to rent a cloud computer of this strength for the time required to train our model is low enough that this sort of solution could be practical for an individual or organization in possession of sensitive data.

We use the open-source library Microsoft SEAL (Simple Encrypted Arithmetic Library) which has a native implementation of the CKKS HE scheme SEAL. The library is written in C++ and provides a simple, low-level interface for initializing keys, encrypting data, and performing the protocol.

Training. Our selection of encryption parameters allows us to perform 6 model updates. As described above, we encode every data entry to a 4,096 dimensional vector so that 4 submodels could be packed into each 16,384-slot ciphertext. Of the 50,000 entry dataset, 40,000 entries were used for training and the remainder were used for testing.

We present two methods to overcome the circuit depth restriction imposed by HE. One involves the cloud forwarding the spent model ciphertexts to the secret key holder, who can decrypt and re-encrypt before returning them. This process requires communication between the cloud and key holder several times throughout the training process. While inconvenient, we found this burden to be reasonable while giving an appreciable increase in model accuracy; the amount of data that needs to be exchanged is on the order of tens of megabytes.

We can eliminate this communication burden by widening our model. Indeed, by adding additional sets of weight ciphertexts, we can train as many submodels as we desire. For example, rather than refreshing the same set of weights four times, we can train four different sets of model ciphertexts, which trains in the same amount of time while eliminating the communication burden of refreshing. We have found this to be effective in practice, and it is closer to how ensembles are typically used in the unencrypted literature, where the number of submodels would be much larger than we consider here.

Performance Evaluation. The first step in the algorithm is to load and serialize the dataset so that it can be transferred to the cloud. In the experiments presented here, we used $S = 4$ submodels. Thus the entire dataset of 50,000 was packed into 12,500 ciphertexts, each of which takes 8 MB when serialized. In total, the training set, originally 14.9 MB, encrypts to about 78.5 GB. Encrypting and serializing this set takes 14.6 minutes.

We ran experiments testing both of our methods for maximizing our encrypted model's learning opportunities. First, we test a "tall" model, where one set of model ciphertexts, packed with 4 submodels, is repeatedly trained and refreshed. Then we have a "wide" model, where we use four sets of model ciphertexts, each packed with four submodels, each of which are trained on different subsets of the data and never require refreshing.

We compare the results obtained by the encrypted network to those of the cleartext network. The latter network is structured exactly like the former. For fairness, the same hyperparameter tuning protocol was performed for both networks, and a batch size of 1000 was used for both. We report the number of model updates that the cleartext model requires to match the maximum accuracy that our ciphertext model has obtained. We note, however, than when given unlimited epochs, the cleartext network can obtain an accuracy of 87% in about 10 minutes of training time, corresponding to about 80 epochs or 3,200 model updates. Results are listed in Table 1.

4. Discussion and conclusion:

In this paper we present a private text classification protocol using HE that provides rea-

Network	Submodels	Model Updates	Model Refreshes	Training Time	Test Accuracy
Cleartext	1	80	NA	15 s	74.3%
"Tall" Ciphertext	4	18	2	5.03 hr	74.2%
"Wide" Ciphertext	16	6×4	0	6.97 hr	74.4%

Table 1: Comparison of results between the clear- and ciphertext models. The multiplication in "Model Updates" is to emphasize the fact that there are 4 sets of model ciphertexts, each of which is updated 6 times. "Model Refreshes" refers to the number of decrypt–re-encrypt steps required in the run, as discussed above.

sonable performance for a realistic use-case. Our main goal was to investigate the feasibility of using HE in computationally intensive ML tasks, such as training a neural network while preserving the confidentiality of the input dataset. Comparing to the cleartext experiments, our results of experiments in the ciphertext domain prove that the performance degradation introduced by the inherent noise as well as the approximate computation of HE is manageable. To the best of our knowledge, our experiment is the largest encrypted text classification training problem with neural networks undertaken so far.

With techniques such as packing and multithreading we managed to train an ensemble neural network that learns from a large encrypted dataset for the supervised text classification. Our solution is secure and practical in this context with the moderate compute power and tools that are available on the cloud today. However, there are still a number of challenges that need to be addressed to improve the performance of both the evaluation results and compute time. For instance, utilizing the power of GPUs will result in a significantly improved compute time. Heuristic search methods may lead to better hyperparameter tuning, as another major issue that can lead to more effective models. Additionally, higher security parameters, such as higher polynomial modulus, can help train deeper neural networks and more epochs during training and hence, better evaluation results.

Finally, it is worth noting that HE, as a technology, has finally advanced to a point where we can take an open-source library and solve a real problem in a reasonable amount of development and compute time.

References:

Cheon, J.H.; Kim, A.; Kim, M. & Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In Advances in Cryptology–ASIACRYPT 2017, pages 409–437, Springer.

Raisaro, J.L.; Troncoso-Pastoriza, J.R.; Misbach, M.; Sousa, J.S.; Pradervand, S.; Missiaglia, E.; Michielin, O.; Ford, B. & Hubaux, J.P. (2018). Med Co: Enabling Secure and Privacy-Preserving Exploration of Distributed Clinical and Genomic Data. IEEE/ACM transactions on computational biology and bioinformatics, 16(4):1328–1341.

Kowsari, K.; Meimandi, K.J.; Heidarysafa, M.; Mendu, S.; Barnes, L.E. & Brown, D.E. (2019). Text Classification Algorithms: A Survey. CoRR, abs/1904.08067.

Nesterov, Y. (2004). Introductory lectures on convex programming volume: A Basic course. Springer.

Statistics Canada. (2021). Consumer Price Index. `https://www.statcan.gc.ca/eng/survey/business/2301`.

U.S. Department of Agriculture (USDA), A.R.S. (2020). FoodData Central: USDA Global Branded Food Products Database. `fdc.nal.usda.gov`.

SEAL (2020). Microsoft SEAL (release 3.5). `https://github.com/Microsoft/SEAL`, Microsoft Research, Redmond, WA.