Privacy Preserving Recurrent Neural Network (RNN) Prediction using Homomorphic Encryption

Robert Podschwadt; Daniel Takabi
Georgia State University

Abstract

Multiple privacy-preserving machine learning approaches have been developed to preserve the privacy of the data in machine learning applications. One such approach is using homomorphic encryption which allows for computation over encrypted data. Common deep neural networks architectures, such as convolutional and fully connected networks, have been adapted for use over encrypted data. However, there is very little work on recurrent neural networks (RNNs) and existing solutions for running RNNs over encrypted data require communication between the data owner and the machine learning service provider. In this work, we introduce parallel RNN blocks, an RNN architecture that can be run on homomorphically encrypted data without requiring client server interaction. We evaluate our architecture on a real-world dataset of online product reviews and IMDb movie review data. Our results are promising and show that we can achieve 88.8% F1 score on the product reviews. The model generalizes well to IMDb data set with 74.36% F1 score using our proposed architecture. In both cases the results are within 3 percentage points of the plaintext versions.

RNNs using Homomorphic Encryption

Homomorphic encryption (HE) schemes are like other asymmetric encryption schemes as in they have a public key pk for encrypting (Enc) data and a private or secret key sk for decryption (Dec). Additionally, HE schemes also have a so-called evaluation function, $Eval$. This evaluation function allows the evaluation of a circuit C over encrypted data without the need for decryption. Given a set of plaintexts $\{m_i\}_0^n$ and their encryption $\{c_i\}_0^n = Enc(pk, \{m_i\}_0^n)$ the circuit C can be evaluated as: $Dec(sk, Eval(pk, C, c_0, \cdots, c_n)) = C(m_0, \cdots, m_n)$. We use the CKKS[1] scheme allows supports computation on rational numbers instead of just integers. When operations are performed on the ciphertexts the noise grows and when it passes a certain threshold the ciphertext can not be decrypted correctly anymore. Multiplications add much more noise than additions. A standard RNN (**Figure 1**) with weights w, recurrent weights v and activation function f can be viewed as:

$$s_t = f(x_t \cdot w + \cdots f(x_1 \cdot w + f(x_0 \cdot w) \cdot v) \ldots \cdot v)$$

The initial input x_0 is repeatedly multiplied by the recurrent weights v and during the application of f. This repeated multiplication, and the resulting noise accumulation, is the main limiting factor for RNNs over HE data. The number of multiplications in RNNs is dependent on the length of the inputs sequence. In other architectures, such as CNNs or fully connected networks, the number of multiplications is independent from the inputs.

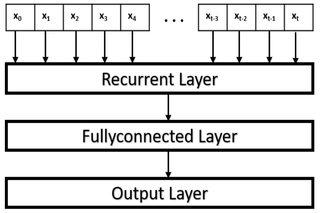

Figure 1. A Standard RNN

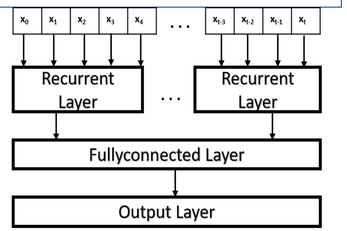

Figure 2. Our Proposed Parallel RNN Blocks Architecture

The Proposed Approach

Most operations in neural networks can performed on encrypted data despite the limitations. The only part that can not be evaluated within the HE constraints are the activation functions. We use the approach by Hesamifard[1] to approximate the Tanh activation with a degree 3 polynomial.

To deal with the depth of the network, we propose an alternative solution for running simple RNNs over encrypted data. We want to keep the strengths of RNNs but not at the cost of discarding information. We split the input sequence into multiple sub-sequences of equal length. Each of these sub-sequences is fed into a simple RNN layer. The output of the simple RNN layers is concatenated and fed into a fully connected layer, see **Figure 2**. We refer to these RNN layers as blocks or parallel blocks. There are at least two ways of setting up the simple RNNs within the blocks. Every simple RNN could have its own independent weights, or the weights could be shared between all the blocks. While no data is discarded from the input sequence there is some loss of information at the block boundaries. In a regular simple RNNs the network has the entirety of the input sequence in its internal state. In our architecture the blocks have no input from the other blocks. However, our experiments show that the fully connected layer following the concatenation layer mitigates the effect for the most part.

Text Classification using Parallel RNN Blocks

The ultimate goal is to run parallel RNN blocks on encrypted data. First, we assess the performance of our architecture on plain data. We train multiple models on a dataset of Amazon product reviews[2] and additionally test on the IMDb[3] movie reviews. The task is to perform sentiment analysis based on the user rating. We group one- and two-star reviews into the negative class and four- and five-star reviews into the positive class. Three-star reviews are filtered out. To vectorize the words we use pretrained 100-dimensional Glove[4] embeddings. We truncate reviews to 64 words. The recurrent layers consist of 128 units for both the Simple RNN and the parallel blocks. In the case of shared individual weights, we reduce the number of units in the recurrent layers to create models with a similar number of parameters. We also create a large model with individual weights, where we do not reduce the number of units. The models perform within 5-10% of the Simple RNN with Tanh baseline model (**Figure 3**). The RNN blocks models are slightly weaker at generalizing to data from a different distribution, as shown with the IMDb data. Especially the models with individual weights suffer a greater drop than the models with shared weights. As the number of splits increases (**Figure 4**) the performance of the model decreases slightly. The performance decrease is greater for models with Tanh activation.

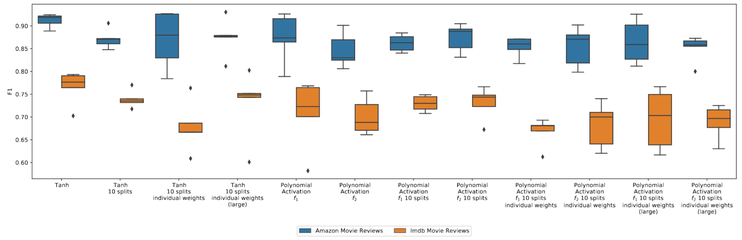

Figure 3. Performance comparison of different models and activations functions

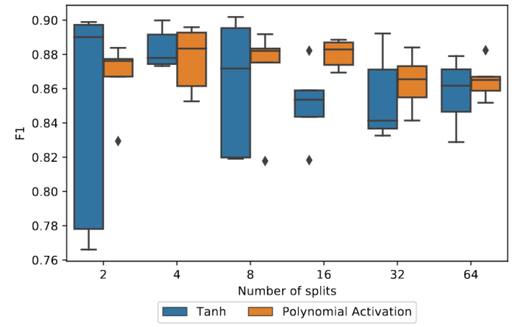

Figure 4. Impact of the number of splits on the Model Performance

RNN Blocks on Encrypted Data

On encrypted data the client needs to perform the text preprocessing, including the embedding. By using SIMD batching we can process up to 32,768 instances at once by encrypting the same dimension of multiple instance in the same ciphertext. The crypto parameters are chosen based on the of splits so that we achieve least 128-bit security. Due to memory constraints, we run models with 64, 32 and 16 splits. The resource requirements increase as the number of splits decreases. Running the model on plaintext, using the same batch sizes, shows an increase in resources required by the encrypted version of 37-109x times, see **Figure 5**.

Prior work[5] on RNN inference using CKKS relies on interactive phases for noise removal. Our architecture does not need an interactive phase. To compare our approach to interactive approaches we implement the interactive approach. For a fair comparison we use the same crypto parameters we use in the previous experiments. The interactive approach preforms slower at but requires less memory (**Figure 6**). Additionally, 2 to 3 times the data needs to be transferred on between client and server, using the interactive approach. The interactive approach needs to transfer 200 GB per batch in worst case, whereas our approach only requires 100 GB of data transfer.

Splits	Running-time (s)			RAM (GB)		
	Ptxt	Ctxt	increase	Ptxt	Ctxt	increase
64	25	1179	47x	3.2	120.5	37x
32	40	2080	52x	2.5	106.2	42x
16	97	10527	109x	3.3	215.6	65x

Figure 5. Running-time and Memory requirements on encrypted and plain data

Splits/ ia Phases	Running-time (s)			RAM (GB)		
	ia	nia	increase	ia	nia	increase
64	3076	1179	0.38x	24.1	120.5	5x
32	3895	2080	0.53x	32.8	106.2	3.2x
16	14036	10527	0.75x	145.3	215.6	1.5x

Figure 6. Comparison with interactive approaches

Contact

Robert Podschwadt
Georgia State University
Email: rpodschwadt1@student.gsu.edu

References

1. Hesamifard, Ehsan, Hassan Takabi, and Mehdi Ghasemi. "Deep neural networks classification over encrypted data." Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy. 2019.
2. Ruining He and Julian McAuley. Ups and downs: Modelling the visual evolution of fashion trends with one-class collaborative filtering. In proceedings of the 25th international conference on world wide web,pages 507–517, 2016
3. Maas, Andrew L., et al. "Learning word vectors for sentiment analysis." *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*. Association for Computational Linguistics, 2011
4. Jeffrey Pennington, Richard Socher, and Christopher D. Manning."Glove: Global vectors for word representation. In Proceedings of the2014 conference on empirical methods in natural language processing(EMNLP), pages 1532–1543, 2014
5. Robert Podschwadt and Daniel Takabi. "Classification of Encrypted Word Embeddings using Recurrent Neural Networks." In PrivateNLP @ ACM Web Search and Data Mining Conference (WSDM), pages 27–31, 2020