PolyDNN
Polynomial Representation of NN for Communication-Less SMPC Inference

Philip Derbeko(✉) and Shlomi Dolev(✉)

Department of Computer Science, Ben-Gurion University of the Negev, Beersheba, Israel
dolev@cs.bgu.ac.il

Abstract. The structure and weights of Deep Neural Networks (DNN) typically encode and contain very valuable information about the dataset that was used to train the network. One way to protect this information when DNN is published is to perform an interference of the network using secure multi-party computations (MPC). In this paper, we suggest a translation of deep neural networks to polynomials, which are easier to calculate efficiently with MPC techniques. We show a way to translate complete networks into a single polynomial and how to calculate the polynomial with an efficient and information-secure MPC algorithm. The calculation is done without intermediate communication between the participating parties, which is beneficial in several cases, as explained in the paper.

Keywords: Privacy · DNN · Data publishing · Data sharing

1 Introduction

Deep Neural Networks (DNN) are the state-of-the-art form of Machine Learning techniques these days. They are used for speech recognition, image recognition, computer vision, natural language processing, machine translation, and many other tasks. Similar to other Machine Learning (ML) methods, DNN is based on finding patterns in the data and, as such, the method embeds information about the data into a concise and generalized model. Subsequently, the sharing of the DNN model also reveals private and valuable information about the data.

In this paper, we first suggest approximating a trained neural network with a single (possibly nested) polynomial. We present a nested polynomial approach to speed up the calculation of the polynomial on a single node. The essence of the idea is to nest the polynomial approximation of each layer within the approximation of the next layer, such that a single polynomial (or arithmetic circuit) will approximate not only a single network unit, but a few layers or even

This work was partially supported by the Lynne and William Frankel Center for Computer Science, by the Rita Altura Trust Chair in Computer Science, a grant from the Ministry of Science and Technology, Israel.

the entire network. We discuss an efficient, (perfect information theoretically secure) secret-sharing MPC calculation of the polynomial calculation of DNN. Lastly, we compare the MPC calculation of the neural network itself with a calculation of polynomial representation.

Our main contribution in this research is an optimization of (communication-less) MPC calculations of a shared DNN by approximating neighboring layers by a single polynomial, and in some cases, the entire network. An additional contribution is a nesting of a multi-layer polynomial to reduce the redundant calculations of the intermediate layers.

Previous relevant research is covered in Sect. 2. Section 3 and Sect. 4 discuss polynomial approximation of DNN on a single computing node. A secure, communication-less multi-party computation, which is presented in Sect. 5. Section 6 summarizes the techniques to obtain blind execution of DNN. Empirical experiments are described in Sect. 7 and, lastly, the paper is concluded in Sect. 8. Details are excluded from this version and can be found in [6].

2 Previous Work

Distributed MPC protocols are built for fixed-point arithmetic, and many times even for limited range values. Thus, the main issue in calculating the neural network activation with secure multi-party computations algorithms is the translation of activation functions from floating-point arithmetic to fixed-point.

Approximation of neural network units' activation function with fixed-point arithmetic and without MPC was considered before in [9,10], where polynomial functions were suggested for approximation. CryptoDL [10] showed an implementation of Convolutional Neural Networks (CNN) over encrypted data using homomorphic encryption (HE). The paper has shown approximation of CNN activation functions by low-degree polynomials due to the high-performance overhead of higher degree polynomials.

A calculation of neural networks with secure multi-party computations was considered in [12]. Their experiments showed that the polynomial approximation of the sigmoid function requires at least a 10-degree polynomial, which causes a considerable performance slow-down with garbled circuit protocol. The work had a limitation for two participating parties and the algorithm was shown to be limiting in terms of performance and the practical size of the network.

CrypTFlow [11] is a system that converts TensorFlow (TF) code automatically into secure multi-party computation protocol. The most salient characteristic of CrypTFlow is the ability to automatically translate the code into MPC protocol, where the specific protocol can be easily changed and added. The optimized three-party computational protocol is specifically targeted for NN computation and speeds up the computation. This approach is similar to the holistic approach of [1].

SecureNN [15] proposed arguably the first practical three-party secure computations, both for training and for activation of DNN and CNN. The impressive performance improvement over then, state-of-the-art, results is achieved by replacing garbled circuits and oblivious transfer protocols with secret sharing protocols. The replacement also allowed information security instead of computational security. Despite being efficient, the protocols require ten communication rounds for ReLu calculation of a single unit not counting share distribution rounds.

A different approach at speeding up performance was made by [5], which concentrated on two-party protocols. The work showed a mixed protocol framework based on Arithmetic sharing, Boolean sharing, and Yao's garbled circuit (ABY). Each protocol was used for its specific ability, and the protocols are mixed to provide a complete framework for neural networks activation functions.

3 Neural Network as Polynomial Functions in a Single Node Case

We show how to approximate functions that are a typical part of DNNs, by polynomials. We focus on the most commonly used functions in neural networks.

Weighted Sum of the Unit Input. Given neuron inputs $X_1, \ldots, X_n$, the weighted sum is a multiplication of inputs with the corresponding weights $S = \sum_{i=1}^{n} w_i X_i - b$, where b is a bias of the neuron which is a polynomial of degree 1.

Common Activation Functions. Most of the research approximating DNN activation functions focused on these few common functions:
ReLu ($ReLu(x) = max(0, x)$), Leaky ReLu (similar to ReLu but $LReLu(x) = 0.01x$ if $x \leq 0$), Sigmoid $\left(\sigma(x) = \frac{1}{1+e^{-x}}\right)$, TANh $\left(tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}\right)$, SoftMax (used for multi-class prediction $\sigma(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{k} e^{x_i}}$). All those functions can be approximated with a polynomial using various different methods, for example [1, 10, 12, 13, 16]. Our optimization method is agnostic to a specific approximation method.

Differently from the most of the research approaches, which minimized the degree of the approximating polynomial, our communication-less approach allows us to use a higher degree polynomials. In our previous research [7] we have shown that 30-degree Chebyshev polynomials achieve good results.

Max and Mean Pooling. Max and Mean pooling compute the corresponding functions of a set of units. Those functions are frequently used in CNN following the convolution layers. Previous works [16] suggested replacing max-pooling with a scaled mean-pooling, which is trivially represented by a polynomial. However, this requires the replacement to be done during the training stage, while we focus on a post-training stage.

In this paper, we have used a simple and practical approximation of max function is:

$$m'(x,y) = \frac{x+y}{2} + ((x-y)^2)^{1/2}. \quad (1)$$

Notice that the function provides an approximation near any values of x and y, which is an advantage over Taylor or Chebyshev approximations, that are developed according to a specific point. Despite its simplicity, Eq. 1 provides a relatively good approximation.

Notice that using a two-variable function for the max pooling layer of k inputs requires chaining of the max functions:

$$max(x_1, x_2, \ldots x_k) = max(x_1, max(x_2, \ldots, max(x_{k-1}, x_k))).$$

Alternatively, the optimization sequence is interrupted at the max-pooling layer, which will require an MPC protocol for the max function calculation, for example [15].

4 Multiple Layers Approximation

We have discussed the approximation of DNN functions by polynomials. The approximation exists for all the common functions. This makes it possible to combine multiple layers into a single polynomial function according to the connectivity of the layers.

One example of a network that can be approximated by a single polynomial function is auto-decoder where hidden layers are dense layers with (commonly) ReLu or sigmoid activation.

The idea is to create a polynomial for the "flow" of the data in the network instead of approximating every single neural unit with a polynomial. As an example, consider the network in Fig. 1.

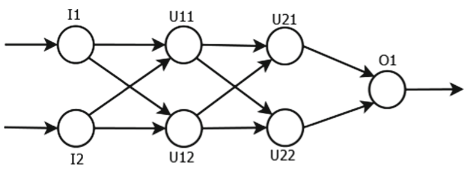

Fig. 1. A small example network with an input layer on the left, two dense hidden layers U1 and U2, and an output layer on the right consisting of a single unit. Each layer utilizes ReLu or sigmoid activation functions, or any other function that can be approximated by a polynomial.

The network consists of an input layer (I) on the left, two dense hidden layers (U_1 and U_2), and one output layer O, which is implemented by the softmax function. The units are marked as u_{li} where l is the hidden layer number and i

is the number of the unit in the layer. We assume that the activation functions of the hidden layers are ReLu (or any other function that can be approximated by a polynomial function).

Consider a unit u_{11}. It calculates the function which is approximated by the polynomial. Assume that ReLu activation functions are approximated using a polynomial of d-degree.

$$ReLu(\sum_i w_i I_i) \approx P_{11} = Pol_{11}(\sum_i w_i I_i). \tag{2}$$

Unit u_{21} receives P_{11} and P_{12} as inputs and calculates the “nested” polynomial function:

$$P_{21} = Pol_{21}(\sum_i w_i P_{1i}). \tag{3}$$

In general, assuming dense layers, the nested polynomials are defined as:

$$P_{lj} = Pol_{lj}(\sum_i w_i P_{(l-1)i}). \tag{4}$$

In this simple case, the result of networks evaluation can be calculated by evaluating two polynomials of d^2-degree: P_{21} and P_{22}, and calculating the output layer function of their output. Overall, by approximating softmax by Pol^{sm} we get the following polynomial for the entire network:

$$\begin{aligned} DNN(x) &= Pol^{sm}\,(w_1^o P_{21} + w_2^o P_2 2) \\ &= Pol^{sm}\,\big(w_1^o Pol_{21}(w_1^{21} P_{11} + w_2^{21} P_{12}) + w_2^o Pol_{22}(w_1^{22} P_{11} + w_2^{22} P_{12})\big) \\ &= Pol^{sm}\,\big(w_1^o Pol_{21}(w_1^{21} Pol_{11}(w_1^{11} I_1 + w_2^{11} I_2) + w_2^{21} Pol_{12}(w_1^{12} I_1 + w_2^{12} I_2)) \\ &\quad + w_2^o Pol_{22}(w_1^{22} Pol_{11}(w_1^{11} I_1 + w_2^{11} I_2) + w_2^{22} Pol_{12}(w_1^{12} I_1 + w_2^{12} I_2))\big) \end{aligned} \tag{5}$$

Notice that P_{11} and P_{12} were calculated twice as they are used as inputs for both U_{21} and U_{22} units.

5 Communication-Less MPC for Polynomial Calculations

The goal of MPC calculations in the considered setup is to protect the published model from exposure to participating cloud providers. The model is trained by the data provider and has two components: architecture, which includes the layout, type, and interconnection of the neural units, as well as the weights of the input, which were refined during the training of the network, i.e. back-propagation phase.

Our goal is to protect the weights that were obtained by a costly process of training. While the architecture also might hold ingenious insights, it is considered less of a secret and may be exposed to the cloud providers.

Even though the described algorithm is agnostic to the specific MPC protocol, it is better to use a protocol that can support $k > 2$ parties, provides perfect information theoretical security and is efficient for a polynomial calculations in terms of communication rounds to enable usage of high-degree polynomials.

A number of MPC protocols answer those requirements [2,4]. These MPC protocols based on Shamir secret sharing [14] can cope with a minority of semi-honest parties and even with a third of the malicious parties. BGW protocol [2] provides a perfect security and [4] provides statistical security with any desirable certainty. In our case, the input is not a multi-variable that is secret-shared, but rather the weights and coefficients of the network are the secrets.

Clear-Text Inputs. In a simpler scenario, the input is revealed to all participating parties. In this case, the secrets are the weights of the trained network. The input values are then can be considered as numerical constants for the MPC calculation and thus, communication rounds can be eliminated completely, see BGW [2] algorithm where additive "gates" are calculated locally without any communication.

Given a secret-share of coefficient a: $s = [s_1, s_2]$. The polynomial $p(x)$ can be calculated as $p(x) = p_1(x) + p_2(x)$, where $p_1(x)$ and $p_2(x)$ use the corresponding secret share.

Secret-Shared Inputs. In the second scenario, the input values are protected as well, and thus, they are distributed by the secret share. As the input values are raised to polynomial degree k, the secret share is done on the set of values: $X = [x, x^2, \ldots x^k]$. Multiplication of secret shares requires communication rounds in a general case, still when secret sharing every element of X it is possible to eliminate the communications all-together using techniques from [3] or [8].

6 Distributed Communication-Less Secure Interference for Unknown DNN

The last two sections, Sect. 4 and Sect. 5, provide all the required building blocks for communication-less MPC for common DNNs. In Sect. 4 we showed how a given, pre-trained network can be approximated with a single polynomial, in most common cases. As a side-note, as the neural network activation functions are not limited to a specific set, there might be networks that cannot be approximated. However, the majority of networks use a rather small set of functions and architectures.

Once the network is presented by a single polynomial, Sect. 5 shows that it can be calculated without a single communication round (apart from the input distribution and output gathering) when the inputs are revealed, or with half the communication rounds when the inputs are secret.

Taken together, those two results enable a somewhat surprising outcome: the data owner can train DNN models, pre-process, and share them with multiple cloud providers. The providers then can collaboratively calculate interference of the network on common or secret-shared inputs without ever communicating with each other. Thus, reducing the attack surface even further even for multi-layer networks.

7 Experiments

All tests were performed on the Fashion database of MNIST, which contains a training set of 60,000 and a testing set of 10,000 28 × 28 images of 10 fashion categories. The task is a multi-class classification of a given image. Experiments on larger datasets and different types of DNN are planned for extended version of the paper.

To solve the problem we have used a non-optimized neural network with two dense hidden layers: one of 300 units and the second one with 100 units. The output layer is a softmax layer with ten units and batch normalization layers before each activation layer.

The performed experiments were done on a pre-trained model. The model was loaded and translated into polynomial as described above automatically. This enables us to perform translation for any pre-trained network, similarly in spirit to [11]. Both the original model and polynomial representation were executed on the same inputs. The outputs are compared for different classification (divided by a total number of test inputs).

Figure 7 shows the difference in accuracy of the network with different degrees. As can be seen, the accuracy improves with the degree of the polynomial approximation, however the improvement flattens at around $d = 30$.

The computation costs are increasing linearly with the polynomial degree (data not shown), where the original ReLu is similar to $d = 1$ degree polynomial. Thus, it makes sense to choose the lowest degree that still provides consistent and accurate results.

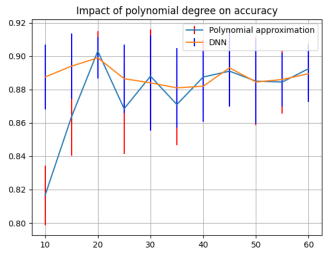

Fig. 2. Accuracy of DNN and polynomial approximation averaged over 10 runs of 500 examples each.

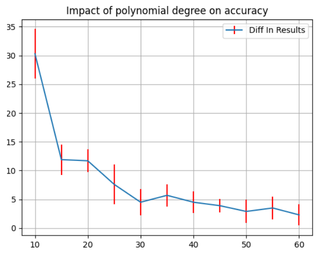

Fig. 3. Relative difference in results between polynomial approximation and the DNN model as a function of polynomial degree.

8 Conclusions

In this paper, we have presented a way to reduce and ultimately eliminate the number of communication rounds in the secure multi-party computation of DNN models. We believe that this optimization method can enable more efficient DNN calculations and further progress in the process of privacy-preserving data sharing.

The above optimization of DNN evaluation targets the inference phase, which is done after the DNN-based model is shared and distributed across cloud providers. The network is not trained anymore, but only queried by the clients. At this phase, the performance issues do not impact the data owners, which could be resource-limited end-devices, but rather are relevant for the cloud providers that have as much larger resources.

References

1. Agrawal, N., Shamsabadi, A.S., Kusner, M.J., Gascón, A.: Quotient: two-party secure neural network training and prediction. In: CCS 2019 (2019)
2. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC 1988 (1988)
3. Berend, D., Bitan, D., Dolev, S.: Polynomials whose secret shares multiplication preserves degree for 2-CNF circuits over a dynamic set of secrets. IACR Cryptol. ePrint Arch. **2019**, 1192 (2019)
4. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: STOC 1988 (1988)
5. Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
6. Derbeko, P., Dolev, S.: Polydnn: Polynomial representation of NN for communication-less SMPC inference (2021)
7. Derbeko, P., Dolev, S., Gudes, E.: Deep neural networks as similitude models for sharing big data. In: 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019, pp. 5728–5736. IEEE (2019)
8. Dolev, S., Doolman, S.: Blindly follow: sits CRT and FHE for DCLSMPC of DUFSM. Cryptology ePrint Archive, Report 2021/410 (2021). https://eprint.iacr.org/2021/410
9. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.R.: CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. In: ICML (2016)
10. Hesamifard, E., Takabi, H., Ghasemi, M.: CryptoDL: deep neural networks over encrypted data. CoRR, abs/1711.05189 (2017)
11. Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow: secure tensorflow inference (2019)
12. Mohassel, P., Zhang, Y.: SecureML: a system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 19–38, May 2017
13. Rouhani, B.D., Riazi, M.S., Koushanfar, F.: DeepSecure: scalable provably-secure deep learning. In: DAC (2018)
14. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
15. Wagh, S., Gupta, D., Chandran, N.: SecureNN: efficient and private neural network training. IACR Crypt. ePrint Arch. **2018**, 442 (2018)
16. Xie, P., Bilenko, M., Finley, T., Gilad-Bachrach, R., Lauter, K.E., Naehrig, M.: Crypto-Nets: neural networks over encrypted data. ArXiv, abs/1412.6181 (2014)