S++: A Fast and Deployable Secure-Computation Framework for Privacy-Preserving Neural Network Training

Prashanthi Ramachandran[1] Shivam Agarwal[1] Arup Mondal[1] Aastha Shah[1] Debayan Gupta[1]
[1]Ashoka University

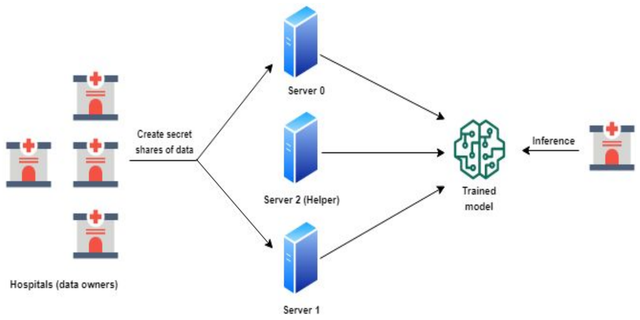

Fig. 1.: The Architecture of S++

Introduction

In this paper, we propose S++, a three-party secure computation framework for secure exponentiation, and exponentiation-based activation functions such as logistic sigmoid, tanh, and softmax. This enables us to construct three-party secure protocols for training and inference of several NN architectures such that no single party learns any information about the data. S++ is an efficient MPC-based privacy-preserving neural network training framework based on (Wagh, Gupta, and Chandran 2018) for 3PC with the activation functions logistic sigmoid, tanh, their derivatives, and softmax. In our setting, there are D (where D can be arbitrary) data owners who wish to jointly train a model over their data with the help of 3-servers. In the setup phase, these data owners create "additive secret-shares" of their input data and send one share each to 2-servers. In the computation phase, the 3-servers (the 2 primary servers and the helper server) collectively run an interactive protocol to train a neural network on the data owners' data without learning any information beyond the trained model [Ref Fig. 1].

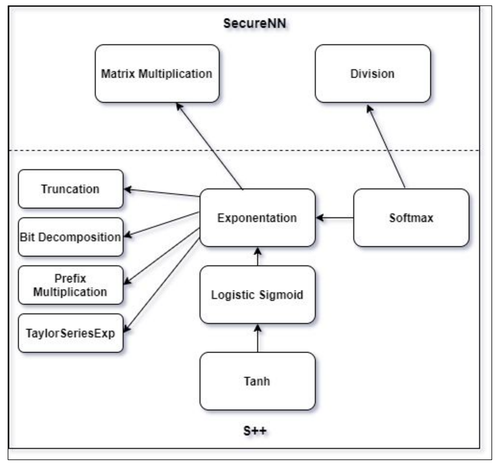

Fig 2.: Function Dependencies in S++.

Our contributions

We summarize our key contributions as follows:

• We first propose a **secure protocol for exponentiation** in the three-party setting. This protocol is based on SCALE MAMBA's (Aly et al. 2020) protocol for base 2 exponentiation.

• We describe **novel secure protocols for the logistic sigmoid, softmax, and tanh**—popular activation functions that are significantly more complex than ReLU (described by (Wagh, Gupta, and Chandran 2018))—along with their derivatives with the help of the exponentiation protocol.

The inclusion of these protocols in a secure and private setting vastly increases the **practicality** of the framework and enables people to convert their protocols into secure ones **without having to redesign** the actual internal structure of their NNs.

Motivation

The logistic sigmoid squashes a real value in [0, 1] and has been used widely over the years for its simplicity, especially in binary classification tasks with a maximum likelihood loss function. It also has significant variants like the tanh and softmax.

Why is ReLU not enough?

• For classification tasks, output layers usually use softmax or logistic sigmoid because they make for more **interpretable** values in [0, 1] unlike unbounded functions like ReLU and its variants.

• ReLU's **unbounded nature** limits its applicability in RNNs. There have been a few notable improvements to assuage this, but even in LSTMs, the tanh often gives more consistent results.

• **Improvements** such as 'leaky' tanh which performs better than tanh and ReLU and is as good as 'leaky' ReLU highlight the need for more secure protocols for such functions.

Algorithm 2 Exponentiation $\mathcal{F}_{Exp}(\{P_0, P_1\}, P_2)$

Input: P_0 and P_1 hold $\langle x\rangle_0^L$ and $\langle x\rangle_1^L$ (shares of a value x).

Output: P_0 and P_1 obtain $\langle y\rangle_j^L = \langle e^x\rangle_j^L$.

1: For $j \in \{0,1\}$, party P_j calls $\mathcal{F}_{Trunc}(\{P_0, P_1\})$ to obtain $\langle a\rangle_j^L$, which is the j^{th} share of $\lfloor x \rfloor$.

2: For $j \in \{0,1\}$, party P_j gets the fractional part of x by locally computing:

$$\langle b\rangle_j^L = \langle x\rangle_j^L - \langle a\rangle_j^L$$

3: For $j \in \{0,1\}$ party P_j invokes $\mathcal{F}_{BitDecomp}(\{P_0, P_1\})$ to obtain $(\langle c_0\rangle_j^L, \langle c_1\rangle_j^L, \cdots, \langle c_{m-1}\rangle_j^L)$: shares of $(x_0, x_1, \cdots, x_{m-1})$, where x is a m-bit number.

4: **for** $i = 0, 1, \ldots, m-1$ **do**

5: P_j for $j \in \{0,1\}$ computes $\langle v_i\rangle_j^L = e^{2^i}.(\langle c_i\rangle_j^L) + j - (\langle c_i\rangle_j^L)$.

6: **end for**

7: P_j for $j \in \{0,1\}$ invokes $\mathcal{F}_{PreMult}(\{P_0, P_1\})$ to get $\langle m\rangle_j^L$.

8: P_j for $j \in \{0,1\}$ invokes $\mathcal{F}_{Taylor}(\{P_0, P_1\})$ to get $\langle n\rangle_j^L$

9: Finally, P_j for $j \in \{0,1\}$ invokes $\mathcal{F}_{MatMul}(\{P_0, P_1\}P_2)$ with inputs $\langle m\rangle_j^L$ and $\langle n\rangle_j^L$ to obtain share $\langle m\rangle_j^L$ of:

$$y = m \times n.$$

Protocol	Dimension	Time(s)	Comm.(mb)
$\mathcal{F}_{Exp}$	64x16	0.08	0.025
	128x128	2.134	0.393
	576x20	0.882	0.276
$\mathcal{F}_{Sigmoid}$	64x16	0.252	2.58
	128x128	5.631	41.288
	576x20	2.615	29.03
$\mathcal{F}_{tanh}$	64x16	0.275	2.58
	128x128	5.32	41.288
	576x20	2.613	29.03
$\mathcal{F}_{Softmax}$	64x16	0.324	2.58
	128x128	5.438	41.288
	576x20	2.617	29.03
$\mathcal{F}^{\mathcal{D}}{}_{sigmoid}$	64x16	0.464	2.597
	128x128	8.033	41.55
	576x20	4.121	29.214
$\mathcal{F}^{\mathcal{D}}{}_{tanh}$	64x16	0.383	2.58
	128x128	4.465	41.288
	576x20	2.84	29.03
$\mathcal{F}_{taylorExp}$	64x16	0.032	0.005
	128x128	0.092	0.079
	576x20	0.427	0.055

Table 1: Benchmarks in LAN setting on i5 7th Gen

Conclusion and Future Work

Table 1 shows the performance of our functions in a LAN setting.

The addition of exponentiation allows the usage of activation functions that require exponentiation. These functions could not be used in a secure setting earlier and their addition allows a greater set of real world neural networks to be migrated directly to the secure setting.

The current version of exponentiation suffers from an *overflow* for small values that makes us unable to use the functions that call the exponentiation function in a practical setting. Ideas from the protocol suggested by (Aly and Smart 2019) can be used to avoid using a workaround that would involve increasing the size of shares to accommodate the overflow.

The future work on S++ will be focused on dealing with the overflow such that the exponentiation function can handle the magnitude of the values that it would encounter in a regular neural network.

References

Aly, A.; and Smart, N. P. 2019. Benchmarking privacy preserving scientific operations. In International Conference on Applied Cryptography and Network Security, 509–529. Springer

Catrina, O.; and Saxena, A. 2010. Secure Computation with Fixed-Point Numbers. In International Conference on Financial Cryptography and Data Security. Springer.

Wagh, S.; Gupta, D.; and Chandran, N. 2018. SecureNN: Efficient and Private Neural Network Training. IACR Cryptology ePrint Archive 2018: 442.