Privacy-Preserving Inference on Neural Networks using Homomorphic Cryptosystems

Skanda Koppula SKOPPULA@MIT.EDU
Massachusetts Institute of Technology, Cambridge, MA 02139 USA

1. Introduction

A large number of data-intensive problems require analysis of private data. For example, diagnosis using a patient's genome, stock analysis using a fund's trading history, or authentication using voice biometrics are all cases where a party may wish to outsource data analysis, but still keep their data private.

Experiencing a recent renaissance, neural networks have become the go-to predictive model, and become a mainstay of machine learning. The hype is partially deserved. In the past decade, neural networks have demonstrated state-of-the-art (near human-level) performance in a diverse set of tasks: drug discovery, image recognition, speech recognition, language translation, and speaker identification (Krizhevsky et al., 2012; Ma et al., 2015; Hinton et al., 2012; Richardson et al., 2015; Sutskever et al., 2014; Variani et al., 2014).

Perhaps just as influential has been work in the past three decades related to homomorphic encryption (Rivest et al., 1978). Many cryptosystems exhibit homomorphisms, whereby computations on the ciphertext translate to computations on the corresponding plaintexts, allowing for private, outsourced computation on encrypted data. In context of privacy-preserving inference (stated more formally in Section 2), we ask if it is it possible to evaluate a neural network on input, such that the input is private to the network evaluator.

Overview and Structure of Report

In this report, I will first overview current literature in privacy-preserving network inference, and second, propose extensions to existing protocols to enable encrypted inference on state-of-the-art network architectures [1].

The structure of this report is as follows. In Section 2, I will formalize the multi-party problem for which we aim to find a protocol. In Section 3, I will provide background on (1) operations required in neural network inference and (2) the FHE/SHE schemes used in literature and our extensions. In Section 4, I will detail the two current methods in literature to the problem. Finally, in Section 5, I will describe our proposed extensions that support of state-of-the-art networks used in practice today.

[1] The proposed extensions are ongoing work together with research group-mate Chiraag Juvekaar (chiraag@mit.edu)

2. Adversarial Model and Setup

Our model has two semi-honest parties Alice (A) and Bob (B). Alice owns some private data $\mathbb{X}$. Bob owns a proprietary network model $\mathbb{N}$ parameterized by some $\mathbb{W} = \{W_i, b_i | 1 \leq i \leq N\}$ (the weights for the N network layers). Alice and Bob wish to engage in an efficient form of secure multi-party computation to evaluate $\mathbb{X}$ on $\mathbb{N}$, without leaking information about their private data (Figure 1). At the conclusion of the MPC, only Alice should recieve the network output on her data. As is typically required, our protocol ought to be semantically secure in both directions:

1. Denote the transcript T as the set of messages from A to B. For any PPT adversary B, $\mathsf{Pr}[B(\mathbb{W}, T) = \mathbb{X}] = \mathsf{Pr}[B(\mathbb{W}) = \mathbb{X}]$. That is, Bob should not be able to learn anything about Alice's data.

2. Denote the transcript T as the set of messages from B to A, and O the output of the network. For any PPT adversary A, $\mathsf{Pr}[A(\mathbb{X}, O, T) = \mathbb{W}] = \mathsf{Pr}[B(O, \mathbb{X}) = \mathbb{W}]$. That is Alice, should not learn anything about Bob's network except for what can be derived from her inputs and the network output.

3. Preliminaries

3.1. Neural Networks

We will describe the operations necessary to perform inference on state-of-the-art networks used in practice

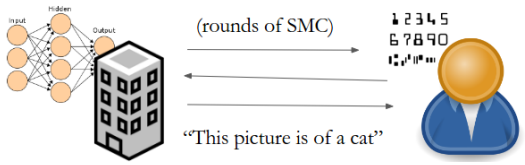

Figure 1. Bob (left) and Alice (right) wish to engage in a secure MPC such that Alice can evaluate her data on Bob's model.

today. As described in the setup, we wish to evaluate input X on network with weights $\mathbb{W}$.

A network is a series of alternating linear and non-linear operations. As shown in Figure 2, at every layer, the input passes through a weighted sum operation, and then non-linear 'activation'. For input vector X, activation operation σ, and layer weights (W_i, b_i), this is representated by $X_{i+1} = \sigma(W_i X_i + b_i)$.

For the task of classification among K-options, the final layer output X is run through the softmax function ($\mathsf{Pr}[y = j|X] = \frac{e^{X^T w_j}}{\sum_{k=1}^{K} e^{X^T w_k}}$) to normalize the output to a valid probability distribution, and obtain the probabilities that the input is each of k classes. The maximum of the probabilities is chosen as the final classification output.

In practice, a variety of activations are used:

1. **Step function**:
$$\sigma_\delta(x) = \begin{cases} 0 & \text{if } x < \delta \\ 1 & \text{if } x \geq \delta \end{cases}$$
2. **Sigmoid**: $\sigma(x) = \frac{1}{1+e^{-x}}$
3. **Rectified Linear Unit**: $\sigma(x) = \max(0, x)$
4. **Tanh**: hyperbolic tangent of input x
5. **MaxPool**: $\sigma(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n)$

Recurrent networks are also in wide use for time-dependent speech and language processing. The only difference between recurrent networks and the *feed-forward* network architecture in Figure 2, is that the output of every recurrent perceptron at time $t-1$ is passed back in alongside the input at time t to the perceptron at time t ($Y_{i+1} = \sigma(W_i[X_i|Y_i] + b_i)$). The chain of perceptrons can be on the upwards of 100 timesteps. A commonly used recurrent architecture is called *Long-Short Term Memory* (LSTM), and in our extensions, we will describe modifications that allow us to support privacy-preserving inference on LSTMs.

3.2. Multi-Party Computation

A first approach would be to apply a generic 2PC algorithm such as Goldwasser-Micali-Widgerson (GMW) or Yao's Garbled Circuits (Goldwasser et al., 1987; Huang et al., 2011). While Yao's protocol runs in constant number of rounds, symmetric-key encryption/decryptions are required for every gate. A network of L layers and N perceptrons per layer translates to $\Omega(L)$ matrix multiplies, or $\Omega(N^2 L)$ 32-bit multiplications. Every k-bit multiplier requires $O(k^2)$ gates. To evaluate a simple 5-deep, 512-wide feedforward network would require over 1.6 billion gates. Optimistically, if every gate evaluation takes 100 milliseconds, this would require 4.9 years for a single network inference, not counting in time for evaluating activations.

Similarly, GMW requires oblivious transfer (OT) for every AND-gate, which occupy a constant proportion (roughly 60%) of multiplier circuits (Schneider & Zohner, 2013). OT can be implemented efficiently using symmetric key operations (Choi et al., 2012), but we still run into similar scaling problems as before.

Fortunately, pre-trained neural networks have a fixed structure and depth. This allows us to leverage homomorphic ciphertext operations to more efficiently compute matrix multiplications and achieve practical runtimes. We describe two homomorphic crytosystems that can be used for this purpose.

3.3. Pallier

The Pallier cryptosystem is used in the prior art for oblivious network inference (Orlandi et al., 2007), as well in our extensions. The security of Pallier relies on hardness of the n-th residuosity problem and factoring. The n-th residuosity assumption states that distinguishing $z = y^n \mod n^2, z \in \mathbb{Z}^*_{n^2}$ from random $r \in \mathbb{Z}^*_{n^2}$ is hard.

The Pallier encryption function is a mapping $\mathbb{Z}_n \times \mathbb{Z}^*_N \to \mathbb{Z}^*_{N^2}$ (Paillier, 1999):

$$m, y = g^m y^n \mod n^2$$

where $g \in \mathbb{Z}^*_{N^2}$ has order tN. Random selection of y ensures that the encryption scheme is probabilistic. The plaintext message is $m \in \mathbb{Z}_N$. (g, n) is the public key. The private key λ is $\text{lcm}(p-1, q-1)$, for two large primes s.t. $pq = n$.

Decryption of ciphertext c is given by the formula:

$$m = \frac{L(c^\lambda \mod n^2)}{L(g^\lambda \mod n^2)} \mod n$$

where $L = \frac{(x-1)}{n}$. A proof of correctness can be found

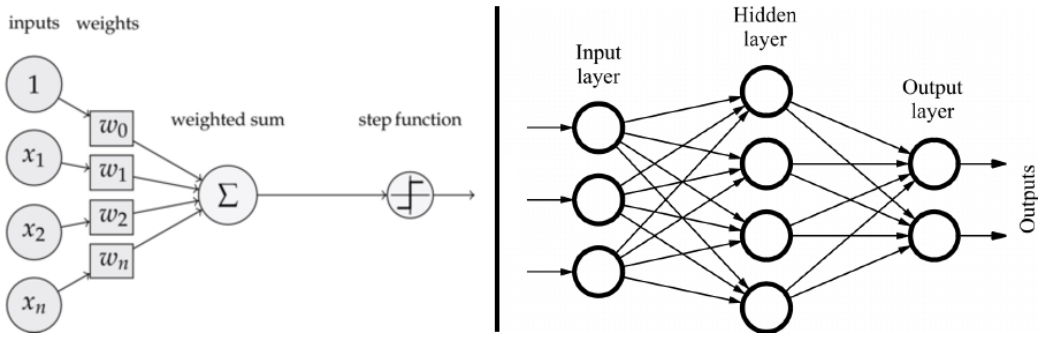

Figure 2. A. (left) A *perceptron*, the building block of a neural network. Each perceptron evaluates a weighted linear sum followed by a non-linear *activation*. B. (right) Multiple perceptrons put together to form a small neural network. Figures reproduced from (Bargen; Quiza, 2011)

.

in Section 3 of (Paillier, 1999).

Notice that Pallier ciphertexts exhibit additive homomorphism (by multiplication of ciphertexts) and by extension, constant multiplicative homomorphism. We will use this to implement safe linear weighted sums.

3.4. YASHE

Yet-Another-Somewhat-Homomorphic-Encryption scheme is a leveled FHE scheme that allows circuit multiplications and additions using ciphertexts up to a certain depth (Bos et al., 2013). This allows for more expressive operations than Pallier.

YASHE plaintext messages are elements of the ring $R_q^n := \mathbb{Z}_q[x]/(x^n + 1)$ (the set of polynomials with coefficients in $\mathbb{Z}_q$ of degree less than n). In brief, encryption maps messages in R_t^n to R_q^n using the encryption procedure:

1. Sample random polynomial $f', g \in R_q^n$.
2. Compute the secret key $f := tf' + 1$.
3. Compute the public key $h := tgf^{-1}$, using Euclid's algorithms over rings to find the ring inverse. Not all f have an inverse, so repeat from (1) if no inverse exists.
4. Encrypt a message $m \in R_t^m$ by computing $c := [\lfloor \frac{q}{t} \rfloor m + e + hs]_q$. $[a]_q$ denotes $a \mod q$ shifted to the interval $[-\frac{q}{2}, \frac{q}{2}]$. e is a randomly sampled the bounded error ring polynomial.

The correctness of decryption, $m = [\lfloor \frac{t}{q} fc \rfloor]_t$, is shown in Section 4.1 of (Bos et al., 2013). The security of the scheme depends on the hardness of the R-LWE problem (Lepoint & Naehrig, 2014).

Notice that YASHE exhibits both additive and multiplicative homomorphism, until the noise growth interferes with correctness. The multiplicative depth in particular dominates noise growth. Requiring higher multiplicative depth requires large enough q, t that can handle noise growth. Computing with extremely large q, t-sized polynomials, however, becomes computationally infeasible as q, t grows larger [2].

4. Prior Work

We describe two methods of privacy-preserving network inference in prior work: the work of (Xie et al., 2014) and (Gilad-Bachrach et al., 2016) in Section 4.1 and the work of (Orlandi et al., 2007) and (Barni et al., 2006) in Section 4.2.

4.1. Crypto-Nets with YASHE

(Xie et al., 2014) and (Gilad-Bachrach et al., 2016) approximate a neural network using a low-degree polynomial. As a low-degree polynomial (with only multiplications and additions), the authors are able to encrypt the input and apply YASHE's homomorphic properties to carry out inference.

To achieve this polynomial approximation, they apply a number of tricks, borrowed from (Livni et al., 2014). First, eschewing the complex non-linear activations used in practice, the only non-linear activation they use is a the low-degree polynomial $y = x^2$. Second, in lieu of a MaxPool layer, they apply the supremum norm approximation: $\max(x_1, \ldots, x_n) = \lim_{d\to\infty} \sum_i (x_i^d)^{\frac{1}{d}}$. In their implementation, the authors use $d = 1$.

They cap their plaintext numbers at 2^{80}, in order to avoid wrap-around in cryptosystem modulus. To con-

[2] Future work: derive an approximate noise-q, t scaling factor, with respect to depth of the neural network with plausible dimensionality, based on approximations given in (Bos et al., 2013)

vert floating point weights and inputs to elements on the ring polynomial, they apply a linear-scale mapping from the range of their minimum and maximum weights down to the range specified by 10-bit integers. Then, they encode this 10-bit integer into polynomial space as a 'constant polynomial'. As they acknowledge, this encoding is simple, but inefficient in that only one of n coefficients are used. They modify the scheme to encode multiple weights/inputs into the network as different coefficients in the polynomial, to achieve higher throughput.

Finally, as a small work-around to the limitation that partial-sums do not exceed the YASHE t-modulus, the authors employ a clever trick using CRT. They decompose every polynomial $\sum a_i x^i$ into k polynomials, such that the j-th polynomial is $\sum(a_i \mod t_j)x^i$. Each polynomial is encrypted and used in the manner described above. The Chinese Remainder Theorem guarantees that as long as the the coefficients do not grow beyond $\prod t_j = 2^{80}$, then the scheme can correctly decrypt. More information can be found in Appendix A.2. and Section 3.2.3 of (Gilad-Bachrach et al., 2016).

Note that this protocol is non-interactive; Alice only needs to send over her encrypted inputs once for Bob to be able to perform evaluation. In this setting, Bob is able to evaluate the linear layers as the multiplication of a YASHE ciphertext with Bob's plaintext weights. This allows for noise growth to occur only during evaluation of the non-linear activations.

They demonstrate the correctness of their approach by performing inference on a shallow two-layer network. In practice, networks can run up to 130+ layers, making this approach (and feasible YASHE noise/hyperparameter selection) significantly more challenging (He et al., 2016).

4.2. Oblivious Inference with Pallier

(Orlandi et al., 2007) and (Barni et al., 2006) perform weighted linear sums under the Pallier cryptosystem with an encrypted $\mathbb{X}$ and plaintext $\mathbb{W}$. Their scheme is non-interactive and supports the step threshold non-linearity. In order to evaluate a step activation, the authors rely on a homebrew 2PC.

1. After the weighted linear sum, Bob holds $\mathsf{Enc}(WX)$. Alice holds the decryption key for this value. The goal is for Bob to own $\mathsf{Enc}(\sigma_\delta(WX))$, without revealing information about W to Alice.

2. Notice that evaluation of the $\sigma_\delta(x)$ is equivalent to evaluation of $\sigma_0(x-\delta)$, which is the problem of determining the sign of $x-\delta$. Bob homomorphically computes $\mathsf{Enc}(WX-\delta)$.

3. Bob scales this value by a random $\pm a$ to form $\mathsf{Enc}(a(WX-\delta))$ (as a form of obfuscation) [3]

4. Bob sends over the randomized activation, Alice decrypts it, determines the sign, and returns corresponding encrypted step function threshold output o back to Bob [4].

5. If a was positive, $\mathsf{Enc}(\sigma_\delta(WX)) = o$. Otherwise, if a was negative, $\mathsf{Enc}(\sigma_\delta(WX)) = \mathsf{Enc}(1)o^{-1}$ (flipping the value of the activation).

Like in the previous section, the authors apply a simple linear mapping, quantizing the rational weights and into n integer buckets.

5. Extensions

In Section 5.1, we extend (Orlandi et al., 2007) to support more non-linear activations than just the step function and sigmoid, adding provable security and correctness. In Section 5.3, we also will show how to extend our scheme to support another state-of-the-art architecture, the Long-Short Term Memory network.

As in 4.2, Alice submits her data for evaluation as encrypted Pallier ciphertext. Like before, Bob then evaluates the linear layers in ciphertext space.

5.1. Generalization of Activation Evaluation

After a linear layer, Bob holds $W, \mathsf{Enc}(WX)$, Alice holds the decryption key, and the goal is for Bob to receive $\mathsf{Enc}(\sigma_\delta(WX))$. Our protocol uses generic 2PC, noting that for most of the commonly applied activations, the non-linearity circuit is very small. For example, we describe our protocol for evaluation of the rectified linear unit non-linearity, $\max(0, x)$, using Yao's Garbled Circuits:

1. Bob samples random r, computes $\mathsf{Enc}(WX+r)$, and sends this back to Alice. r is Bob's private garbled circuit input.

2. Alice decrypts, retrieving $WX+r$, which is Alice's private garbled circuit input.

[3] It is not clear, or described in the paper, how you choose an a such that there is no wrap-around plaintext in the Pallier modulus

[4] Note that this step does not ensure semantic security for non-binary activation functions, because Alice sees a decrypted activation, which can reveal partial information about W

3. Alice and Bob agree on the garbled circuit for the ReLU function $\max(0, (WX+r)-r)$. Bob garbles his input r, the circuit, and sends it to Alice for evaluation.

4. Alice uses oblivious transfer to obtain her input, and evaluates the circuit. The output of the garbled circuit is two random additive shares r_1 and r_2 of the final answer.

5. Bob provides Alice with the decryption mappings for only one of the outputs, O_1. Alice ungarbles O_1, encrypts it using her public key, and sends it alongside the garbled O_2 back to Bob.

6. Bob uses the decryption mappings to uncover O_2 and uses the Paillier additive homomorphism to add this to $\mathsf{Enc}(O_1)$, yielding desired output, $\mathsf{Enc}(\sigma_\delta(WX))$.

Unlike (Orlandi et al., 2007), at no point does Alice or Bob hold a decrypted activation or matrix product. We can show by contradiction the semantic security of the 2PC protocol, reducing it to the security of Bob's garbled inputs and the Pallier encryptions.

The garbled circuit for a ReLU layer has gate count $O(Nk)$, where N is nodes/layer of the network and k is the bit-width of weights. This reduces the quadratic time bottleneck of a generic MPC.

We can apply essentially the same protocol for other discrete activations (e.g. MaxPool, step threshold, etc.). For a MaxPool over n inputs for example, Bob can sample $r_1, \ldots, r_n$, one for each input, as his secret obfuscating input. The garbled circuit computes the maximum instead of the ReLU. Discrete or linear activations convert nicely into small garbled digital circuits as exhibited above. While it is possible to implement continuous activations with our protocol (e.g. tanh and sigmoid), the circuit sizes would no-longer be linear in the bit-width without linear approximations, because of the multiplications.

5.2. Batch Normalization

Batch normalization (BN) is a layer introduced by (Ioffe & Szegedy, 2015) to speed up training. The layer is liberally used in most state-of-the-art networks today. BN scales the output of every layer to zero-mean, unit variance in order to reduce inter-layer covariate shift, and ensure that changes in early layers during training do not affect the performance of later layers. We can lump the per feature normalization into the next layer linear multiply: instead of $Ax + b$, we compute a normalized $A(\frac{x-\mu}{\sigma}) + b = \frac{A}{\sigma}x + (\frac{A*\mu}{\sigma} + b) = A'x + b'$. We are able to support batch normalization with no modification to our protocol.

5.3. Long-Short Term Memory

A widely-used variant of the recurrent neural network is the Long-Short Term Memory network [5], illustrated in Figure 3. While a perceptron in a recurrent neural network carries state unperturbed from one time slice to the next, the LSTM learns what to 'forget' and what to 'add to memory' on every new time-slice. The LSTM carries memory state C_t across time, and outputs a value h_t on every timestep. We define the LSTM's forget mask, f, the input mask, i, and the output mask, o, as follows:

$$f = \sigma(W_f[h_{t-1}|x_t] + b_f)$$

$$i = \sigma(W_i[h_{t-1}|x_t] + b_i)$$

$$o = \sigma(W_o[h_{t-1}|x_t] + b_o)$$

The proposed contributions to the memory state are evaluated to be $\widetilde{C} = \sigma(W_C[h_{t-1}|x_t] + b_C)$. Now, we combine these variables to produce the new memory state and the new output (Hochreiter & Schmidhuber, 1997; Olah, 2015):

$$C_t = f_t C_{t-1} + i_t \widetilde{C}, \quad h_t = o_t \sigma(C_t)$$

Unfortunately, with our cryptographic construction in 5.1, we are unable to support this architecture because of the multiplication between the input/forget/output masks and the memory states. This would require multiplicative homomorphism which is not supported under Pallier. We modify the LSTM construction to circumvent the limitation.

Specifically, notice that to capture the idea of 'forgetting' memory and masking inputs, we only need subtractions and additions if the state C is always within a fixed constant range. For example, instead of the elementwise multiplication $f_t C_{t-1} | 0 \leq f_{t,i} \leq 1$, we can compute $\text{ReLU}(C_{t-1} - f_t)$. If we keep $0 \leq f_{t,i}, C_{t-1,i} \leq 1$, this has the same effect of cancelling out elements of C_{t-1} which the network wishes to forget. We can enforce this bounds constraints by applying the appropriate non-linear activation after every update. Our updated LSTM perceptron, using no ciphertext multiplications, is as follows:

$$i, f, o = \text{Sigmoid}(W_{i,f,o}[h_{t-1}|x_t] + b_{i,f,o})$$

[5] With the scheme introduced in Section 5.1, we can evaluate simple recurrent neural networks,described at the end of Section 3

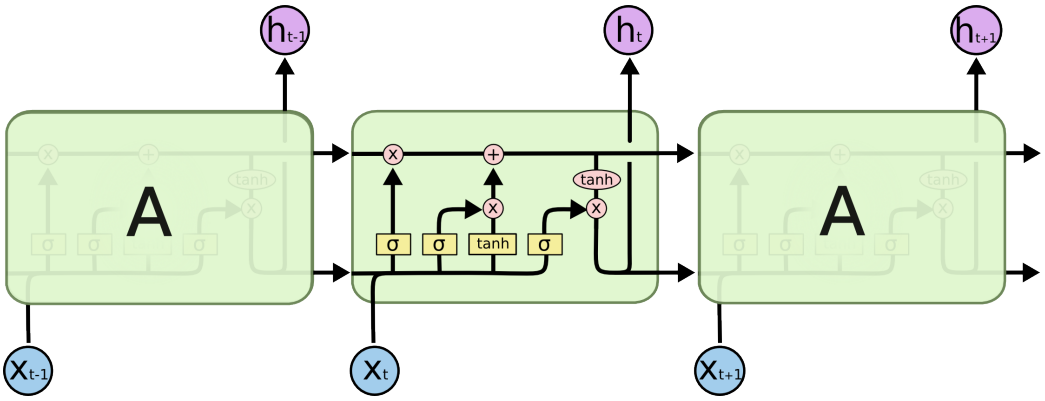

Figure 3. Long-Short Term Memory Perceptron that allows the network to keep state across time-varying input chains. Reproduced from (Olah, 2015)

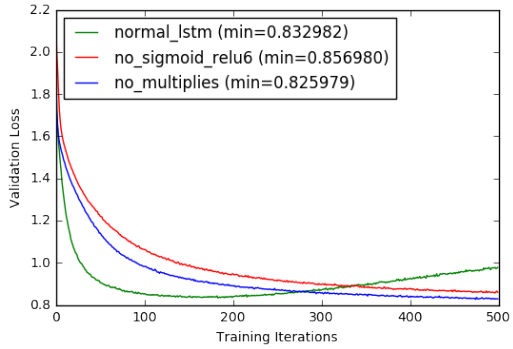

Figure 4. The testing loss of a normal LSTM, and two variants of our proposed modification that allows inference using Pallier ciphertexts. We notice that while our variants take longer to train, they are competitive in their accuracy of inference. `no_multiplies` refers to the our original LSTM modification with sigmoid activations. `no_sigmoid_relu6` is the `no_multiplies` construction with sigmoids replaced by ReLU6's.

$$C_t = \text{Sigmoid}(C_{t-1} - f_t + \widetilde{C} + i_t)$$

$$h_t = \text{Sigmoid}(C_t - o_t)$$

An alternative to using a sigmoid to enforce scaling constraints is to use Tensorflow's in-built ReLU6 $= \min(\max(x, 0), 1)$, which is easier to implement as a garbled digital circuit.

We demonstrate that these changes do not significantly effect the accuracy of inference, after training the network. We trained an LSTM to produce written English, and evaluated the loss on a held-out test set as training progressed, as shown in Figure 4. From this we see that our modifications take longer to train, but are simpler, and just as accurate, as the original LSTM construction. Most importantly, our modification supports evaluation on Pallier ciphertext.

6. Conclusion and Future Work

We have presented an overview of methods pertaining to the problem of privacy-preserving evaluation of neural networks. We described two existing solutions from our literature search, and expanded on this with our own joint contributions. Our extensions circumvent the accuracy degradations associated with the polynomial approximations in (Xie et al., 2014) and (Gilad-Bachrach et al., 2016), while allowing a larger variety of network architectures than (Orlandi et al., 2007) and (Barni et al., 2006).

As part of future work, we intend to derive more rigorous bounds on the runtime of each algorithm for a fixed feedforward architecture and fixed weight bit-width. Additionally, we hope to implement and benchmark the latency, throughput, and accuracy of our extensions, as compared to prior art.

References

Bargen, Danilo. Programming a perceptron in python. URL https://blog.dbrgn.ch/2013/3/26/perceptrons-in-python/.

Barni, Mauro, Orlandi, Claudio, and Piva, Alessandro. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia and security*, pp. 146–151. ACM, 2006.

Bos, Joppe W, Lauter, Kristin E, Loftus, Jake, and Naehrig, Michael. Improved security for a ring-based fully homomorphic encryption scheme. In *IMA Int. Conf.*, pp. 45–64. Springer, 2013.

Choi, Seung Geol, Hwang, Kyung-Wook, Katz, Jonathan, Malkin, Tal, and Rubenstein, Dan. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces.

In *Cryptographers Track at the RSA Conference*, pp. 416–432. Springer, 2012.

Gilad-Bachrach, Ran, Dowlin, Nathan, Laine, Kim, Lauter, Kristin, Naehrig, Michael, and Wernsing, John. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pp. 201–210, 2016.

Goldwasser, Shafi, Micali, Silvio, and Wigderson, Avi. How to play any mental game, or a completeness theorem for protocols with an honest majority. In *Proc. of the Nienteenth Annual ACM STOC*, volume 87, pp. 218–229, 1987.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George E, Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Huang, Yan, Evans, David, Katz, Jonathan, and Malka, Lior. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, volume 201, 2011.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Lepoint, Tancrede and Naehrig, Michael. A comparison of the homomorphic encryption schemes fv and yashe. In *International Conference on Cryptology in Africa*, pp. 318–335. Springer, 2014.

Livni, Roi, Shalev-Shwartz, Shai, and Shamir, Ohad. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pp. 855–863, 2014.

Ma, Junshui, Sheridan, Robert P, Liaw, Andy, Dahl, George E, and Svetnik, Vladimir. Deep neural nets as a method for quantitative structure–activity relationships. *Journal of chemical information and modeling*, 55(2):263–274, 2015.

Olah, Chris. Understanding lstm networks, 2015. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Orlandi, Claudio, Piva, Alessandro, and Barni, Mauro. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security*, 2007:18, 2007.

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, 1999.

Quiza, Ramon. Computational methods and optimization. *Machining of hard materials*, 2011.

Richardson, Fred, Reynolds, Douglas, and Dehak, Najim. Deep neural network approaches to speaker and language recognition. *IEEE Signal Processing Letters*, 22(10):1671–1675, 2015.

Rivest, Ronald L, Adleman, Len, and Dertouzos, Michael L. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11): 169–180, 1978.

Schneider, Thomas and Zohner, Michael. Gmw vs. yao? efficient secure two-party computation with low depth circuits. In *International Conference on Financial Cryptography and Data Security*, pp. 275–292. Springer, 2013.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Variani, Ehsan, Lei, Xin, McDermott, Erik, Moreno, Ignacio Lopez, and Gonzalez-Dominguez, Javier. Deep neural networks for small footprint text-dependent speaker verification. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 4052–4056. IEEE, 2014.

Xie, Pengtao, Bilenko, Misha, Finley, Tom, Gilad-Bachrach, Ran, Lauter, Kristin, and Naehrig, Michael. Crypto-nets: Neural networks over encrypted data. *arXiv preprint arXiv:1412.6181*, 2014.