Certified Private Inference on Neural Networks via Lipschitz-Guided Abstraction Refinement*

Edoardo Manino[1 [0000−0003−0028−5440]], Bernardo Magri[1 [0000−0003−4537−7023]], Mustafa A. Mustafa[1,2 [0000−0002−8772−8023]], Lucas C. Cordeiro[1 [0000−0002−6235−4272]]

[1] The University of Manchester, Department of Computer Science, Oxford Road, Manchester M13 9PL, United Kingdom
[2] imec-COSIC, KU Leuven, Kasteelpark Arenberg 10, B-3001, Belgium

Abstract. Private inference on neural networks requires running all the computation on encrypted data. Unfortunately, neural networks contain a large number of non-arithmetic operations, such as ReLU activation functions and max pooling layers, which incur a high latency cost in their encrypted form. To address this issue, the majority of private inference methods replace some or all of the non-arithmetic operations with a polynomial approximation. This step introduces approximation errors that can substantially alter the output of the neural network and decrease its predictive performance. In this paper, we propose a Lipschitz-Guided Abstraction Refinement method (LiGAR), which provides strong guarantees on the global approximation error. Our method is iterative, and leverages state-of-the-art Lipschitz constant estimation techniques to produce increasingly tighter bounds on the worst-case error. At each iteration, LiGAR designs the least expensive polynomial approximation by solving the dual of the corresponding optimization problem. Our preliminary experiments show that LiGAR can easily converge to the optimum on medium-sized neural networks.

Keywords: Privacy-Preserving Machine Learning · Homomorphic Encryption· Deep Neural Networks · Lipschitz Constant · Polynomial Approximation · Abstract Interpretation

1 Introduction

The rise of very large neural network models has made it attractive to offload all the related computation to remote servers in the cloud [27]. While this configuration eases the computational burden on the client side, it opens the door to privacy concerns, as the user is forced to send their private data to a third-party machine [22]. An ideal solution would involve a private inference scheme (see Figure 1), where the server evaluates the neural network on encrypted data and the client decrypts the result afterwards.

* This work is funded by the EPSRC grant EP/T026995/1 entitled "EnnCore: End-to-End Conceptual Guarding of Neural Architectures" under *Security for all in an AI enabled society.*

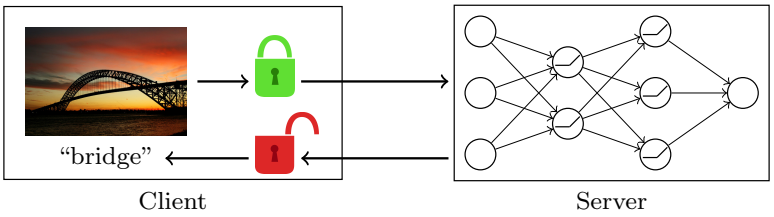


Fig. 1: Private inference requires running a neural network on encrypted data.

In order to realise such vision, each operation performed by the neural network needs to be adapted for operating on ciphertext. Homomorphic encryption schemes enable this translation as follows. For a given encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, we say that $\mathcal{E}$ is additively homomorphic if for two ciphertexts $c_1 \leftarrow \mathsf{Enc}(m_1)$ and $c_2 \leftarrow \mathsf{Enc}(m_2)$, one can compute $c_3 = c_1 + c_2$ where $\mathsf{Dec}(c_3) = m_1 + m_2$. One example of such scheme is due to Paillier [21]. Similarly, we say that $\mathcal{E}$ is multiplicative homomorphic if for two ciphertexts $c_1 \leftarrow \mathsf{Enc}(m_1)$ and $c_2 \leftarrow \mathsf{Enc}(m_2)$, one can compute $c_3 = c_1 \cdot c_2$ where $\mathsf{Dec}(c_3) = m_1 \cdot m_2$. The RSA encryption scheme [23] is of this flavor. A Fully Homomorphic Encryption (FHE) scheme is one that combines both operations. In particular, it allows for a client to compute an encryption of its secret input x and a server to homomorphically evaluate any circuit C over the ciphertext; as an output the server produces an encryption of the evaluation of $C(x)$ while it never learns anything about x. The first construction of FHE by Gentry [7] is based on the hardness of lattices problems and it is extremely inefficient. Since then, a lot of work has been done trying to bring FHE closer to the practical realm [2, 3, 15].

In this regard, neural networks contain a significant number of operations that cannot be represented as a combination of additions and multiplications: e.g. ReLU, hyperbolic tangent, sigmoid, softmax and maxpool [1]. In order to avoid the computational cost of directly executing these in FHE, existing private inference methods propose to approximate them with polynomials [4, 18, 16, 14, 13]. Such polynomial conversion introduces some approximation error at every layer of the neural network. For some inputs, the error can accumulate and drastically distort the output of the network [6]. While most existing works claim that the approximated network has similar predictive accuracy to the original one, some authors express the need for stronger guarantees on the approximation error [6]. To the best of our knowledge, this need has not been met so far. The only exception is the theoretical analysis in [14], which proves that the output error is bounded, but does not provide a constructive algorithm to compute it.

In this paper, we address this research gap by designing neural network approximations with formal guarantees on the maximum output error. Essentially, we recognise that this research objective is fundamentally a problem of equivalence verification. As such, we leverage state-of-the-art verification techniques, and we adapt them to the polynomial approximation setting. More specifically, our contributions are the following:

- we propose an abstraction technique to represent the polynomial error;
- we adapt state-of-the-art reachability and Lipschitz estimation techniques to the architecture of our abstracted network;
- we derive a closed-form solution to the problem of optimal error allocation;
- we introduce our Lipschitz-Guided Abstraction Refinement (LiGAR) method to iteratively minimise the over-estimation caused by our abstraction;
- we show that LiGAR can easily converge on medium-sized neural networks;
- we discuss the practical limitations of a worst-case method like LiGAR.

All the code for the experiments in this paper is publicly available at [17].

2 Related Work

The literature on privacy-preserving machine learning is vast. Here, we only cover the main research trends on the topic of private inference.

Neural Architecture Search (NAS). Since the main bottleneck is executing non-arithmetic operations on encrypted data, some works propose to select neural architectures that contain the bare minimum of them. Delphi [18] uses NAS to select which ReLUs we can approximate with degree-2 polynomials. The rest is replaced with garbled circuits to avoid accuracy drop. Similarly, SAFENet [16] uses NAS to replace a subset of the ReLUs with polynomials of degree in $[0, 3]$. Other methods include a finetuning phase after altering the network architecture: DeepReDuce [9] uses pruning and distillation techniques, while SNL [5] replaces some ReLUs with the identity function before re-training.

Low-degree approximations. Earlier approaches aimed at replacing all non-arithmetic operations with low-degree polynomials. CryptoNets [8] replaces all activation functions with x^2. Similarly, the authors of [4] use degree-2 polynomials but includes batch normalisation layers to keep the input distribution centred with the polynomial approximation. As discussed in [6], these low-degree approximation only allow for shallow neural networks, as it is very challenging to fine-tune the approximations without incurring in the gradient explosion problem.

High-degree approximations. An alternate approach is approximating non-arithmetic operations with high-degree polynomials [14]. This way, the approximated network does not require any fine-tuning. The major obstacle to such goal is obtaining stable high-degree approximations of the sign operator [12]. From it, a number of popular non-arithmetic operations can be derived, including ReLU and max-pooling. A recent example of this line of research is [13], which focuses on the challenge applying fully-homomorphic encryption to large-scale networks.

3 Preliminaries

3.1 Neural Network Definitions and Notation

In this paper, we focus on feedforward fully-connected neural networks with ReLU activation functions. This neural architecture encapsulates all the chal-

lenges in designing private inference methods. Extension of our method to general neural networks is possible, but we leave it for future work.

Definition 1 (ReLU Layer). *Define the Rectified Linear Unit (ReLU) as the function* $\sigma(z_i) = \max(z_i, 0)$ *with potential* $z_i \in \mathbb{R}$*. A ReLU layer is a mapping* $h : \mathbb{R}^n \to \mathbb{R}^n$ *that applies* σ *element-wise, i.e.* $(h(z))_i = \sigma(z_i)$*.*

Definition 2 (Affine Layer). *An affine layer is a mapping* $g : \mathbb{R}^m \to \mathbb{R}^n$ *such that* $g(z) = Wz+b$ *where* $W \in \mathbb{R}^{n\times m}$ *is a matrix and* $b \in \mathbb{R}^n$ *is a column vector.*

Definition 3 (Feedforward Fully-Connected ReLU Neural Network). *A feedforward fully-connecte ReLU neural network* $f(x)$ *is the composition of an alternating sequence of affine and ReLU layers:*

$$y = f(x) = g_k \circ h_{k-1} \circ g_{k-1} \circ \cdots \circ h_1 \circ g_1(x) \tag{1}$$

3.2 Univariate Polynomial Approximation

The ReLU function is non-arithmetic and thus necessitates replacement. This is a classic problem of univariate polynomial approximation [26], with the goal of minimising the maximum error over a finite interval $[\underline{z}, \overline{z}]$:

$$p_d^* = \arg\min_{p_d} \left\{ \max_{z\in[\underline{z},\overline{z}]} \left\{ |p_d(z) - \sigma(z)| \right\} \right\} \tag{2}$$

where d is the polynomial degree. The solution of Equation 2 can be computed via Remez's algorithm, of which many implementations exist (e.g. [20]); we present an example of it in Figure 3a. Crucially, univariate polynomial approximation becomes numerically unstable as the degree d grows into the hundreds. The authors of [12] address this issue by decomposing the minimax polynomial into a composition of lower-degree ones.

4 LiGAR: Lipschitz-Guided Abstraction Refinement

We define the problem of finding the optimal polynomial approximation of a neural network $f(x)$ as follows:

Definition 4 (Optimal Polynomial Approximation). *Given a (possibly infinite) set of inputs* $\mathcal{X}$ *and a target output error* δ_y*, find the least computationally expensive polynomial approximations* $p_{d_i}^* \approx \sigma(z_i)$ *of each ReLU function* i *in* $f(x)$*, such that the maximum output error satisfies* $||f(x) - \hat{f}(x)||_\infty \leq \delta_y$*, where all the ReLUs in* $\hat{f}(x)$ *have been replaced with their respective approximations.*

Here, we interpret the least expensive computation objective in terms of minimising the sum of all degrees $d_{tot} = \sum_i d_i$. However, we note that other choices are possible (e.g. multiplication depth in [13]). These are correlated with the polynomial degree, but not equivalent to it.

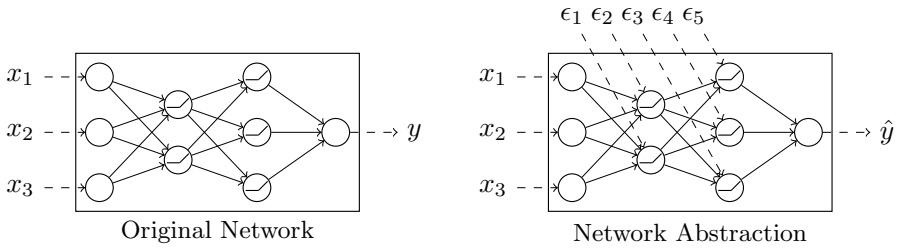


Fig. 2: We abstract away the details of the polynomial approximation by introducing an additional error input ϵ_i to the output of each ReLU activation.

4.1 Approximation Error Abstraction

Substituting a ReLU activation functions with a polynomials introduces some input-dependent approximation error $\epsilon(z_i) = |\sigma(z_i) - p^*_{d_i}(z_i)|$. In order to facilitate the solution of the optimisation problem in Definition 4, we propose to remove the dependency of $\epsilon(z_i)$ on z_i by letting its value vary freely in the interval $\epsilon_i \in [-\delta_i, +\delta_i]$. In this way, we can isolate the contribution of each approximation $p^*_{d_i}$ on the output error (see Section 4.4).

We present a high-level picture of our abstraction in Figure 2. The abstracted network uses the same weights, biases and activations of the original network. However, a new input ϵ_i is added to the output of each activation function:

Definition 5 (Polynomial Neural Network Abstraction). *Call $\hat{f}(x)$ the polynomial approximation of neural network $f(x)$, such that $|\sigma(z_i) - p_{d_i}(z_i)| \leq \delta_i$ for each ReLU i with potentials $z_i \in [\underline{z}_i, \overline{z}_i]$. Define $f(x, \epsilon)$ as the abstraction of $\hat{f}(x)$ if each ReLU layer h_ℓ of the abstraction takes the following form:*

$$\left(\hat{h}_\ell(z)\right)_i = \sigma(z_i) + \epsilon_i \tag{3}$$

where $\epsilon_i \in [-\delta_i, +\delta_i]$ and $z_i \in [\underline{z}_i, \overline{z}_i]$ for all i and $x \in \mathcal{X}$.

4.2 Potential Range Estimation

Note that estimating the range of the potentials z_i for each ReLU neuron i corresponds to the classic reachability problem in neural network verification [29, 25]. In our experiments, we use the FastLin bound propagation technique in [28] to compute linear bounds on the output of the affine layers g_ℓ. Differently from the standard definition therein, the bounds depend on both the original input vector x and the error vector ϵ:

$$\underline{S}_\ell x + \underline{U}_\ell \epsilon + \underline{v}_\ell \leq g_\ell(x, \epsilon) \leq \overline{S}_\ell x + \overline{U}_\ell \epsilon + \overline{v}_\ell \tag{4}$$

Note that the matrices $\underline{U}_\ell$ and $\overline{U}_\ell$ have non-zero entries only in the columns $j \in [1, \sum_{k=0}^{\ell-1} n_k]$, where n_k is the number of ReLU neurons in layer k. Furthermore, we derive the concrete range of the ReLU potentials $(g_\ell(x, \epsilon))_i \in [\underline{z}_i, \overline{z}_i]$ by maximising (resp. minimising) the right-hand side (resp. left-hand side) of Equation 4 over both $x \in \mathcal{X}$ and $\epsilon_i \in [-\delta_i, +\delta_i]$.

4.3 Local Lipschitz Constant Estimation

In order to compute the optimal polynomial approximation of each ReLU, we need both the potential range $[\underline{z}_i, \overline{z}_i]$ and the optimal approximation error ϵ_i (see Section 4.4). In this regard, we can inform our decision by estimating the impact of each ϵ_i on the output error. We achieve this goal by computing the local Lipschitz constant of ϵ_i:

Definition 6 (Lipschitz Constant). *The local Lipschitz constant of ϵ_i is:*

$$L_i^{\infty} = \max_{x,\epsilon,\epsilon'} \frac{|f(x,\epsilon) - f(x,\epsilon')|_{\infty}}{|\epsilon_i - \epsilon_i'|_{\infty}} \tag{5}$$

where $x \in \mathcal{X}$, $\epsilon_j, \epsilon_j' \in [-\delta_j, +\delta_j]$ for all j and $\epsilon_j = \epsilon_j'$ for all $j \neq i$.

In other terms, Definition 6 quantifies the maximum rate of change in the output as we modify one single error input ϵ_i. Thanks to it, we can bound the total approximation error as follows:

$$||f(x) - \hat{f}(x)||_{\infty} \leq \max_{\epsilon} \left\{ ||f(x) - f(x,\epsilon)||_{\infty} \right\} \leq \sum_i L_i^{\infty} \max_{\epsilon_i} \{|\epsilon_i|\} \tag{6}$$

Computing the Lipshitz constant of a neural network is highly non-trivial, and it is currently an area of active research [24, 10]. In our experiments, we adapt the algorithm in [24] to the architecture of our abstracted neural network, by extracting the Jacobian of ϵ from the intermediate results of the backward pass.

4.4 Optimal Error Allocation

Recall that our objective is approximating the neural network $f(x)$ in such a way that the output error never exceed a given margin δ_y (see Definition 4). With the Lipschitz estimates from Section 4.3, we can decide how to optimally distribute the local approximation error ϵ_i as follows:

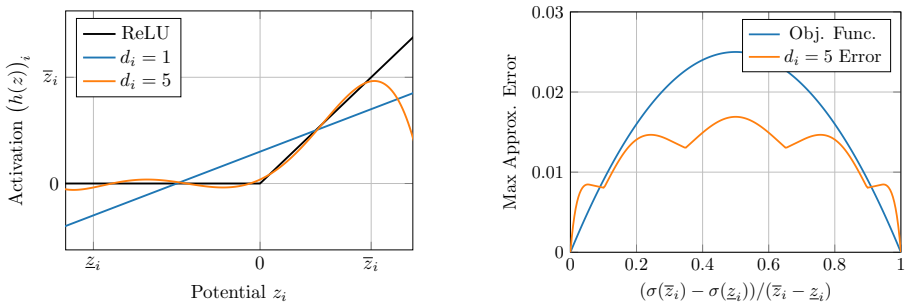
$$\text{Minimise} \quad d_{tot} = \sum_i d_i(\epsilon_i, \underline{z}_i, \overline{z}_i) \tag{7}$$

$$\text{Subject to} \quad \sum_i L_i^{\infty} \epsilon_i \leq \delta_y \tag{8}$$

$$\text{And} \quad 0 \leq \epsilon_i \leq \delta_i, \quad \forall i \tag{9}$$

where ϵ_i is a shorthand for $\max_{\epsilon_i}\{|\epsilon_i|\}$ from Equation 6, i.e. the maximum approximation error of each ReLU function in $[\underline{z}_i, \overline{z}_i]$.

Crucially, the objective function estimates the inference cost of replacing each activation i with a polynomial of degree d_i. We give an example of such polynomial in Figure 3a. Clearly, the degree is a monotonically increasing function in the size of the potential range $[\underline{z}_i, \overline{z}_i]$, and monotonically decreasing in the required precision ϵ_i. Furthermore, we know that the minimax approximation



(a) Minimax polynomial in $[\underline{z}_i, \overline{z}_i]$.

(b) Minimax error vs $C_i d_i^{-1}$ objective.

Fig. 3: Approximating a ReLU activation with polynomials (3a) and the resulting error (3b). Note that the example on the left has $(\sigma(\overline{z}_i) - \sigma(\underline{z}_i))/(\overline{z}_i - \underline{z}_i) = 0.4$.

$p^*_{d_i}(z_i)$ of degree d_i of any continuous non-differentiable function $\sigma(z_i)$ guarantees a maximum approximation error $\epsilon_i \propto O(d^{-1})$ [26]. As a result, we can compute the exact relation between ϵ_i and $d_i = 1$ and extend it to any $d_i > 1$:

$$\epsilon_i \approx \frac{C_i}{d_i} \qquad \text{where} \qquad C_i = \frac{1}{2}\left(\sigma(\underline{z}_i) - \underline{z}_i \frac{\sigma(\overline{z}_i) - \sigma(\underline{z}_i)}{\overline{z}_i - \underline{z}_i}\right) \tag{10}$$

which is exact for $d_i = 1$ and overestimated for $d_i > 1$, unless the constant C_i is very close to 0 or 1 (see example for $d_i = 5$ in Figure 3b). We show how to derive the corresponding optimal solution in Appendix A.

4.5 Abstraction Refinement Cycle

Now, we can discuss the key idea behind our LiGAR method. When we compute the Lipschitz constants $L - i^\infty$ and potential ranges $[\underline{z}_i, \overline{z}_i]$, we need to specify a domain $\epsilon_i \in [-\delta_i, +\delta_i]$ for the approximation error. If we choose a domain that is too large, our estimates will be conservative. If we choose it too small, the values of ϵ_i will also be small, thus yielding a larger cost C_i/ϵ_i.

We solve this issue by making our abstraction-optimisation process iterative (see Algorithm 1). More specifically, we start with a large user-defined error domain (Line 2). Then, we estimate the potential ranges and Lipschitz estimates accordingly (Lines 5-10). Next, we solve the dual optimisation problem (Line 11), which gives us the optimal error allocation *given the current estimates*. With this result, we can tighten the error domain (Line 12) and repeat the process.

Note that the error domain is always kept larger than any of the solutions found so far, and smaller than all error domains that did not activate the constraint $\epsilon_i^* \leq \delta_i$. In our experiments, LiGAR converges after 20-30 iterations. Finally, we use an indirect definition for the input domain in Lines 5-10. That is, akin to many neural network verification settings [19], we select a finite set of concrete inputs $\mathcal{X}$, and consider a norm-∞ ball of size δ_x around each of them.

Algorithm 1 LiGAR

Input: network abstraction $f(x, \epsilon)$, finite input set $\mathcal{X}$, input domain δ_x, error domain δ_ϵ, output precision δ_y, convergence threshold h.
Output: optimal error allocation ϵ^*, potential ranges $\underline{z}, \overline{z}$.

$t \leftarrow 0$
$\delta_i(1) \leftarrow \delta_\epsilon, \forall i$ ▷ Largest error domain
repeat
  $t \leftarrow t + 1$
  for all $x \in \mathcal{X}$ **do**
    $\underline{z}(x), \overline{z}(x) \leftarrow \texttt{ComputePotentials}(f, x, \delta_x, \delta_i(t))$ ▷ Forward pass
    $L^\infty(x) \leftarrow \texttt{ComputeLipschitz}(f, \underline{z}, \overline{z})$ ▷ Backward pass
  end for
  $C_i \leftarrow C(\min_x\{\underline{z}_i(x)\}, \max_x\{\overline{z}_i(x)\}), \forall i$ ▷ Cost coefficients
  $L_i^\infty \leftarrow \max_x\{L_i^\infty(x)\}, \forall i$ ▷ Lipschitz estimates
  $\epsilon^*(t), v(t) \leftarrow \texttt{OptimalAllocation}(C, L^\infty, \delta_i(t), \delta_y)$ ▷ Solve dual problem
  $\delta_i \leftarrow \frac{1}{2}(\max_t\{\epsilon_i^*(t)\} + \min_{t:\epsilon_i^*(t)<\delta_i(t)}\{\epsilon_i^{max}(t)\}), \forall i$ ▷ Tighter error domain
until $v(t-1) - v(t) \leq h$ ▷ Stop if no progress
$t^* \leftarrow \arg\min_t\{v(t)\}$
return $\epsilon^*(t^*)$, $v(t^*)$ ▷ Return best allocation

5 Empirical Analysis

Here, we present our preliminary experimental results. Our main goal is twofold: demonstrate the behaviour of LiGAR on medium-size networks; gather evidence on the practical utility of worst-case approximations for private inference.

5.1 Experimental Setting

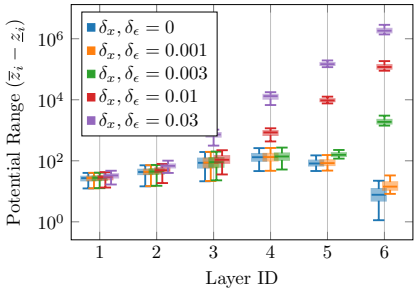
Neural Network Model. We select the MNIST (fc) image classification network from VNN-COMP'21.[3] This network has six hidden layer containing 256 neurons each. The inputs are grey-scaled images with 784 pixels, whose values are scaled to the interval $[0, 1]$. The outputs are ten scores that represent the log-likelihood of the image containing a hand-written digit between zero and nine [11].

LiGAR Parameters. For reasons of time, we run LiGAR with $\mathcal{X}$ containing only 1000 random samples from the MNIST training set [11]. We set the initial error domain to $\delta_\epsilon = 0.1$, the convergence threshold to $h = 0.01$ and the output precision to $\delta_y = 0.01$, unless otherwise specified.

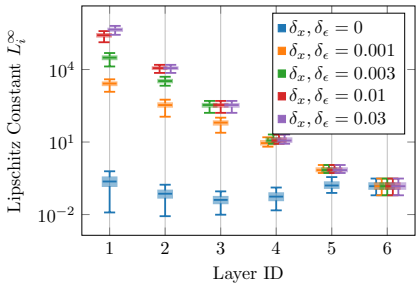
5.2 Experimental Results

Potential Ranges and Lipschitz Constants. Computing worst-case guarantees on the potential ranges and Lipschitz constants incur a cost regarding over-estimation. In Figure 4, we show what happens when we increase the input δ_x

[3] https://github.com/stanleybak/vnncomp2021/tree/main/benchmarks

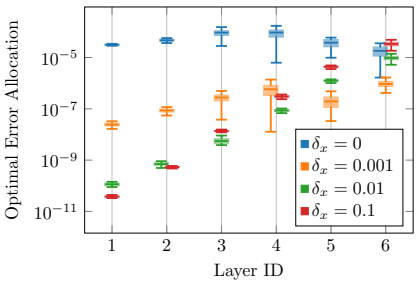


(a) Potential ranges.

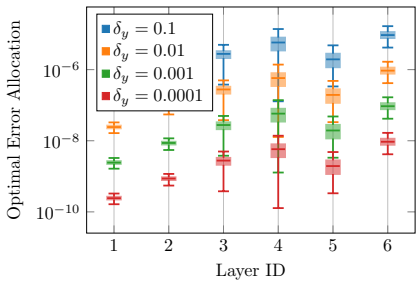


(b) Lipschitz constants.

Fig. 4: Potential ranges (Fig. 4a) and Lipschitz constants (Fig. 4b) on the MNIST (fc) network. The values become more conservative as δ_x and δ_ϵ increase.



(a) Input domain (with $\delta_y = 0.01$).



(b) Output precision (with $\delta_x = 0.001$).

Fig. 5: Optimal error allocation for different input domain sizes (Fig. 5a) and output precision requirements (Fig. 5b) on the MNIST (fc) network. ReLU neurons have been omitted in their linear state (always active or inactive).

and error δ_ϵ domains. As expected, the bound propagation yields fairly tight estimates for the potential range $\overline{z}_i - \underline{z}_i$ over the first layers of the network until the over-approximation caused by the FastLin abstraction technique begins to accumulate. Similarly, Lipschitz constants are tight for the final few layers but get increasingly worse as the backward estimation procedure reaches earlier layers. Note that LiGAR can iteratively reduce the over-estimation caused by δ_ϵ, but not the one caused by δ_x.

LiGAR Error Allocation. We report the optimal error allocation computed by LiGAR in Figure 5. Note how increasing the input domain δ_x forces LiGAR to be more conservative in its estimates. Interestingly, this conservative behaviour is more pronounced for earlier layers. We believe this is caused by the corresponding over-estimation of the Lipschitz constants (see Figure 4b). In constrast, the error allocation is proportional to the output precision δ_y. This is not surprising, since δ_y appears only in one constraint of the optimization problem (see Equation 8).

Linearly Removable ReLU Activations. It has been claimed in the literature that many ReLU activations in a given network can be replaced by a linear function [5]. We test this claim by checking how many ReLUs are either always active or always inactive according to LiGAR's potential range estimates. The results in Table 1 show that the number of ReLUs we can remove quickly falls to zero as we cover a larger portion of the input space by increasing δ_x.

Table 1: Number of ReLUs in the always-active or always-inactive state.

MNIST (fc)	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6
$\delta_x = 0$	23	11	53	81	238	162
$\delta_x = 0.001$	23	11	53	76	161	0
$\delta_x = 0.01$	17	7	0	0	0	0
$\delta_x = 0.1$	0	0	0	0	0	0

Inference-Time Output Error. LiGAR guarantees that the output approximation error is never greater than δ_y. Here, we estimate the practical performance of the approximated network by measuring both the maximum observed output error and the change in predictive accuracy. Note that due to numerical instability in existing univariate polynomial approximation libraries (see Section 3.2) we report only the result for the abstracted network. In Table 2 we show results for both the 1000-sample LiGAR train set $\mathcal{X}$ (see Section 5.1), and the full 10000-sample MNIST test set. Note how the accuracy matches the original network, and the maximum error is at least two orders of magnitude smaller than the LiGAR guarantee $\delta_y = 0.01$. This shows that using worst-case estimates in designing neural network approximations leads to very conservative solutions.

Table 2: Output error and inference accuracy of the abstracted network.

	LiGAR Train Set		Test Set	
	$\lvert f(x) - f(x,\epsilon)\rvert$	Accuracy	$\lvert f(x) - f(x,\epsilon)\rvert$	Accuracy
Original $f(x)$	-	0.980	-	0.969
$\delta_x = 0$	2.64×10^{-4}	0.980	7.26×10^{-4}	0.969
$\delta_x = 0.001$	4.90×10^{-6}	0.980	5.67×10^{-6}	0.969
$\delta_x = 0.01$	3.36×10^{-5}	0.980	4.00×10^{-5}	0.969
$\delta_x = 0.1$	1.26×10^{-4}	0.980	1.30×10^{-4}	0.969

6 Conclusions

In this paper, we propose the Lipschitz-Guided Abstraction Refinement (LiGAR) method, which can compute the optimal neural network approximation with a given worst-case output error. We show that LiGAR converges quickly on medium-sized networks and could in principle scale to larger ones. At the same time, the worst-case guarantees come at the price of conservative solutions. We believe that this is a major obstacle for the adoption of worst-case methods like LiGAR for the purpose of private inference.

References

1. Bishop, C.M., Nasrabadi, N.M.: Pattern recognition and machine learning. Springer (2006)
2. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011. pp. 97–106. IEEE Computer Society (2011). https://doi.org/10.1109/FOCS.2011.12, https://doi.org/10.1109/FOCS.2011.12
3. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Rogaway, P. (ed.) Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6841, pp. 505–524. Springer (2011). https://doi.org/10.1007/978-3-642-22792-9_29, https://doi.org/10.1007/978-3-642-22792-9_29
4. Chabanne, H., de Wargny, A., Milgram, J., Morel, C., Prouff, E.: Privacy-preserving classification on deep neural network. Cryptology ePrint Archive, Paper 2017/035 (2017), https://eprint.iacr.org/2017/035, https://eprint.iacr.org/2017/035
5. Cho, M., Joshi, A., Reagen, B., Garg, S., Hegde, C.: Selective network linearization for efficient private inference. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S. (eds.) Proceedings of the 39th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 162, pp. 3947–3961. PMLR (17–23 Jul 2022), https://proceedings.mlr.press/v162/cho22a.html
6. Garimella, K., Jha, N.K., Reagen, B.: Sisyphus: A cautionary tale of using low-degree polynomial activations in privacy-preserving deep learning (2021)
7. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009. pp. 169–178. ACM (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440
8. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 201–210. PMLR, New York, New York, USA (20–22 Jun 2016), https://proceedings.mlr.press/v48/gilad-bachrach16.html
9. Jha, N.K., Ghodsi, Z., Garg, S., Reagen, B.: Deepreduce: Relu reduction for fast private inference. In: Meila, M., Zhang, T. (eds.) Proceedings of the 38th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 139, pp. 4839–4849. PMLR (18–24 Jul 2021), https://proceedings.mlr.press/v139/jha21a.html
10. Laurel, J., Yang, R., Ugare, S., Nagel, R., Singh, G., Misailovic, S.: A general construction for abstract interpretation of higher-order automatic differentiation. Proc. ACM Program. Lang. **6**(OOPSLA2) (oct 2022). https://doi.org/10.1145/3563324, https://doi.org/10.1145/3563324
11. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
12. Lee, E., Lee, J.W., No, J.S., Kim, Y.S.: Minimax approximation of sign function by composite polynomial for homomorphic comparison. IEEE Trans-

actions on Dependable and Secure Computing **19**(6), 3711–3727 (2022). https://doi.org/10.1109/TDSC.2021.3105111
13. Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., No, J.S.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. IEEE Access **10**, 30039–30054 (2022). https://doi.org/10.1109/ACCESS.2022.3159694
14. Lee, J., Lee, E., Lee, J.W., Kim, Y., Kim, Y.S., No, J.S.: Precise approximation of convolutional neural networks for homomorphically encrypted data (2021)
15. Lin, W.K., Mook, E., Wichs, D.: Doubly efficient private information retrieval and fully homomorphic ram computation from ring lwe. Cryptology ePrint Archive (2022)
16. Lou, Q., Shen, Y., Jin, H., Jiang, L.: {SAFEN}et: A secure, accurate and fast neural network inference. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=Cz3dbFm5u-
17. Manino, E., Magri, B., Mustafa, M.A., Cordeiro, L.: Certified Private Inference on Neural Networks via Lipschitz-Guided Abstraction Refinement (May 2023). https://doi.org/10.5281/zenodo.7897618, https://doi.org/10.5281/zenodo.7897618
18. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: A cryptographic inference service for neural networks. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 2505–2522. USENIX Association (Aug 2020), https://www.usenix.org/conference/usenixsecurity20/presentation/mishra
19. Müller, M.N., Brix, C., Bak, S., Liu, C., Johnson, T.T.: The third international verification of neural networks competition (vnn-comp 2022): Summary and results (2022)
20. Pachón, R., Trefethen, L.N.: Barycentric-remez algorithms for best polynomial approximation in the chebfun system. BIT Numerical Mathematics **49**(4), 721–741 (2009). https://doi.org/10.1007/s10543-009-0240-1, https://doi.org/10.1007/s10543-009-0240-1
21. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology — EUROCRYPT '99. pp. 223–238. Springer Berlin Heidelberg (1999)
22. Pulido-Gaytan, B., Tchernykh, A., Cortés-Mendoza, J.M., Babenko, M., Radchenko, G., Avetisyan, A., Drozdov, A.Y.: Privacy-preserving neural networks with homomorphic encryption: Challenges and opportunities. Peer-to-Peer Networking and Applications **14**(3), 1666–1691 (2021). https://doi.org/10.1007/s12083-021-01076-8, https://doi.org/10.1007/s12083-021-01076-8
23. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978). https://doi.org/10.1145/359340.359342, https://doi.org/10.1145/359340.359342
24. Shi, Z., Wang, Y., Zhang, H., Kolter, J.Z., Hsieh, C.J.: Efficiently computing local lipschitz constants of neural networks via bound propagation. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems. vol. 35, pp. 2350–2364. Curran Associates, Inc. (2022), https://proceedings.neurips.cc/paper_files/paper/2022/file/0ff54b4ec4f70b3ae12c8621ca8a49f4-Paper-Conference.pdf
25. Tran, H.D., Manzanas Lopez, D., Musau, P., Yang, X., Nguyen, L.V., Xiang, W., Johnson, T.T.: Star-based reachability analysis of deep neural networks. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) Formal Methods – The Next 30 Years. pp. 670–686. Springer International Publishing, Cham (2019)

26. Trefethen, L.N.: Approximation Theory and Approximation Practice, Extended Edition. SIAM (2019)
27. Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., Rellermeyer, J.S.: A survey on distributed machine learning. ACM Comput. Surv. **53**(2) (mar 2020). https://doi.org/10.1145/3377454, https://doi.org/10.1145/3377454
28. Weng, L., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Daniel, L., Boning, D., Dhillon, I.: Towards fast computation of certified robustness for ReLU networks. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 5276–5285. PMLR (10–15 Jul 2018), https://proceedings.mlr.press/v80/weng18a.html
29. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018), https://proceedings.neurips.cc/paper_files/paper/2018/file/d04863f100d59b3eb688a11f95b0ae60-Paper.pdf

A Closed-Form Dual Solution

The objective function $d_{tot} \approx \sum_i C_i/\epsilon_i$ is convex, and it is possible to derive its global minimum in closed form as follows. First, let us write the Lagrangian of our optimization problem:

$$\mathcal{L}(\epsilon, \mu, \lambda) = \sum_i \frac{C_i}{\epsilon_i} + \sum_i \mu_i(\epsilon_i - \delta_i) + \lambda(\sum_i L_i^\infty \epsilon_i - \delta_y) \tag{11}$$

where we omit the constraints $\epsilon_i \geq 0$ for simplicity. Now, we can derive the minimum of the Lagrangian over ϵ by equating its derivative to zero:

$$\epsilon_i^* = \arg\min_\epsilon \{\mathcal{L}(\epsilon, \mu, \lambda)\} = \sqrt{\frac{C_i}{\mu_i + \lambda L_i^\infty}} \tag{12}$$

for all i, which already satisfies the implicit constraint $\epsilon_i \geq 0$. Finally, the values of the dual variables $\mu_i \geq 0$ and $\lambda \geq 0$ can be found by maximizing ϵ_i^*. This operations requires satisfying the following system of equations:

$$\sum_i \sqrt{\frac{C_i}{\mu_i + \lambda L_i^\infty}} L_i^\infty = \delta_y \tag{13}$$

$$\mu_i = \max\left(\frac{C_i}{(\epsilon_i^*)^2} - \lambda L_i^\infty, 0\right) \tag{14}$$

which can be done iteratively by first solving Equation 13 with a binary search, and then computing Equation 14. In our experience, very few iterations are required until convergence.