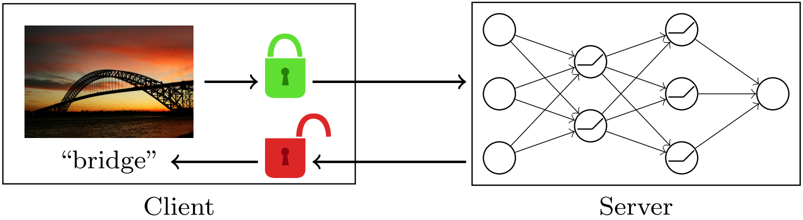
Certified Private Inference on Neural Networks via Lipschitz-Guided Abstraction Refinement

Edoardo Manino   Bernardo Magri   Mustafa A. Mustafa
Lucas C. Cordeiro

University of Manchester (UK)
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*

Private inference for neural networks



An ideal model for third-party AI services

- The user sends out encrypted data
- The provider never sees the plaintext
- The user deciphers the NN output locally
- But how?

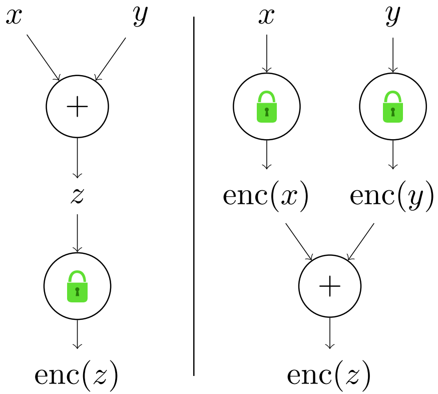
Fully-homomorphic encryption

Main idea

- enc($x + y$) =
- enc(x)+enc(y)
- for all x, y

Current FHE

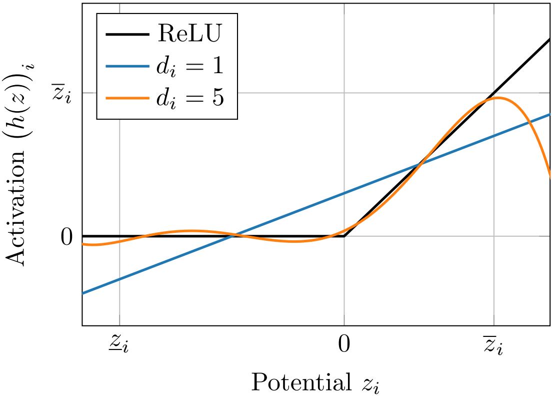
- "Fast" with $+, *$
- Slow otherwise
- (garbled circuits)



Application to neural networks is non-trivial

- How do we do activation functions with only $+$ and $*$?
- The whole NN needs to be a large polynomial!

Polynomial activations (example)



General issues with polynomial activations

- The polynomial is "stable" in a limited range only
- Higher-degree polynomials are more expensive to compute

Existing private inference schemes

Retrain from scratch

- E.g. CryptoNets [2016]
- Uses x^2 activations
- Gradient instability

Approximate & fine-tune

- E.g. DeepReDuce, SNL
- Low-degree poly activations
- Escaping activation problem

Neural architecture search

- E.g. Delphi, SAFENet
- Keep a few ReLUs
- Replace the rest
- Requires garbled circuits

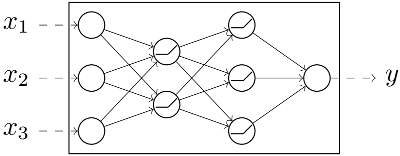
Post-training approximation

- E.g. Lee's work [2021-2022]
- High-degree poly activations
- **Equivalence problem**

Our research goal

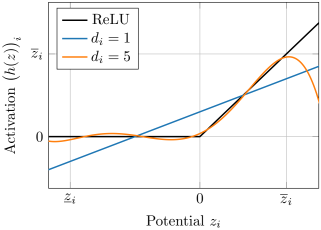
Setting

- To deploy a NN for private inference
- Replace activations with polynomials
- (post-training)



Objective

- Keep degree small
- Given target error*
- (fast & equivalent)



*Provide worst-case guarantees on the output error

Output error (1): average case vs worst case

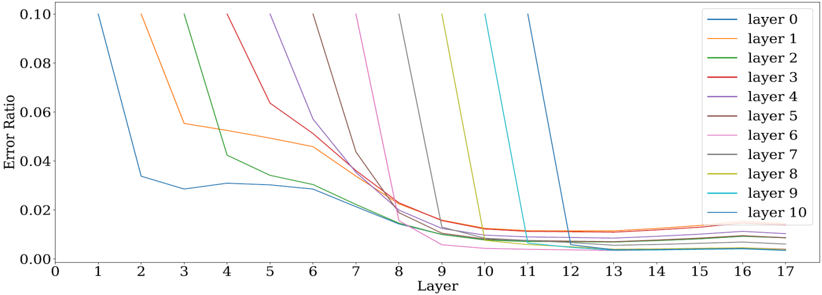


Figure: Attenuation of injected noise on a VGG-19 net trained on CIFAR-10. A curve starts at the layer where a scaled Gaussian noise is injected to its input (from Arora *et al.* ICML 2018).

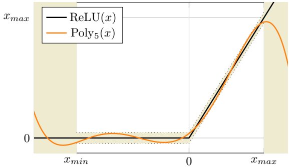
Polynomial activations inject approximation error everywhere

- For most inputs x, the approximation errors cancel out
- However, we want to minimise $\max_x |f(x) - \hat{f}(x)|$

Output error (2): polynomial approximation abstraction

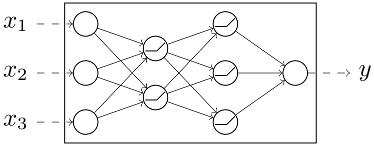
Input-independent guarantee

- $|\mathrm{p}(x) - \mathrm{act}(x)| \in [-\delta, \delta]$
- for any $x \in [x_{min}, x_{max}]$

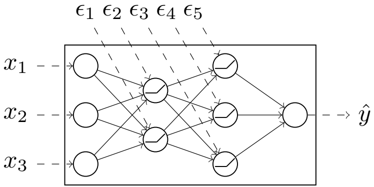


Abstract the approximation

- For each activation i
- Add input $\epsilon_i \in [-\delta_i, \delta_i]$



Original Network



Network Abstraction

Output error (3): potential range estimation

The abstraction is valid

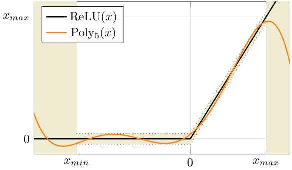
- If the potentials
- are $x_i \in [x^i_{min}, x^i_{max}]$
- For all activations i

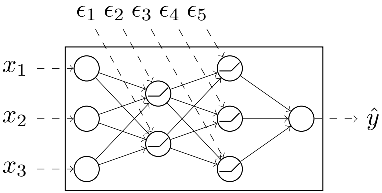
The potential range

- Depends on both:
- The global input x
- And previous ϵ_j

Forward bound prop.

- $x \in \mathcal{X}, \epsilon_i \in [-\delta_i, \delta_i]$
- Use a SOTA method

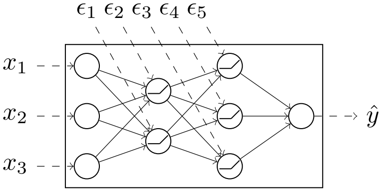




Output error (4): Lipschitz constant estimation

What we have so far

- Start from $x \in \mathcal{X}$
- Bound propagation
- Add $\epsilon_i \in [-\delta_i, \delta_i]$
- As we go forward



What is the impact of each ϵ_i on the output error $|\hat{y} - y|$?

- We need to compute the local Lipschitz constant L_i^∞
- Which guarantees $|\hat{y} - y| \leq \sum_i L_i^\infty \max |\epsilon_i|$
- Use SOTA methods to compute the Lipschitz constants:
- e.g. Shi *et al.*, NeurIPS 2022 or Laurel *et al.*, OOPSLA 2022

Optimisation (1): formalising the problem

Setting

- To deploy a NN for private inference
- Replace activations with polynomials

Objective

- Keep poly $\deg_{tot}$ small
- Given target error* δ_y
- *worst-case guarantees

We can finally formalise it as an optimisation problem:

$$\text{Minimise} \quad \deg_{tot} = \sum_i \deg_i(\epsilon_i, x^i_{min}, x^i_{max}) \tag{1}$$

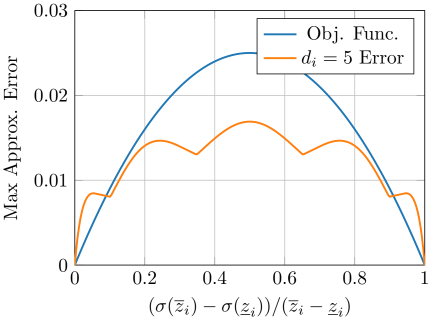
$$\text{Subject to} \quad \sum_i L_i^\infty \epsilon_i \leq \delta_y \tag{2}$$

$$\text{And} \quad 0 \leq \epsilon_i \leq \delta_i, \quad \forall i \tag{3}$$

Optimisation (2): closed-form objective function

Degree vs error

- Minimax approx.
- of continuous
- activation func.
- $\epsilon_i \approx C_i / \mathrm{deg}_i^{-1}$



The optimisation problem is convex!

$$\text{Minimise} \quad \mathrm{deg}_{tot} = \sum_i \frac{C_i(x^i_{min}, x^i_{max})}{\epsilon_i} \tag{4}$$

$$\text{Subject to} \quad \sum_i L_i^\infty \epsilon_i \leq \delta_y, \quad 0 \leq \epsilon_i \leq \delta_i \tag{5}$$

LiGAR (1): algorithmic challenges

The optimisation problem is convex, but...

$$\text{Minimise} \quad \deg_{tot} = \sum_i \frac{C_i(x^i_{min}, x^i_{max})}{\epsilon_i} \tag{6}$$

$$\text{Subject to} \quad \sum_i L_i^\infty \epsilon_i \leq \delta_y, \quad 0 \leq \epsilon_i \leq \delta_i \tag{7}$$

Over-estimated coefficients

- L_i^∞, x^i_{min} and x^i_{max}
- depend on the domain δ_i
- via bound propagation

Coefficient tightness

- Ideally, we set δ_i small
- to tighten $L_i^\infty, x^i_{min}, x^i_{max}$
- but also let ϵ_i be large!

LiGAR (2): our iterative solution

$\{\delta_i\} \leftarrow \text{LargeGuess}()$

Repeat for $t \in [0, \infty]$

$\{x^i_{min}, x^i_{max}\} \leftarrow \text{PotentialRanges}(f, \mathcal{X}, \{\delta_i\})$

$\{L^\infty_i\} \leftarrow \text{LipschitzConstants}(f, \{x^i_{min}, x^i_{max}\})$

$\{C_i\} \leftarrow \text{CostCoefficients}(\{x^i_{min}, x^i_{max}\})$

$\{\epsilon_i\}, v \leftarrow \text{OptimalPolyError}(\{C_i, L^\infty_i, \delta_i\}, \delta_y,)$

$\{\delta_i\} \leftarrow \text{TightenDomains}(\{\delta_i, \epsilon_i\})$

Until $v(t-1) - v(t) < \text{threshold}$

Return $\epsilon(t^*)$ where $t^* = \arg\min_t v(t)$

Given

- f neural net
- $\mathcal{X}$ input dom.
- δ_y output err.

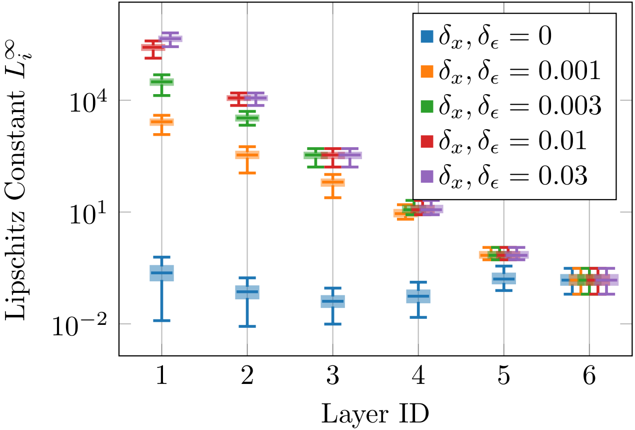
Estimated

- δ_i max err.
- L^∞_i Lipschitz
- x_{min} low pot.
- x_{max} up pot.
- C_i cost coeff.

Optimised

- ϵ_i actual err.
- v obj. value

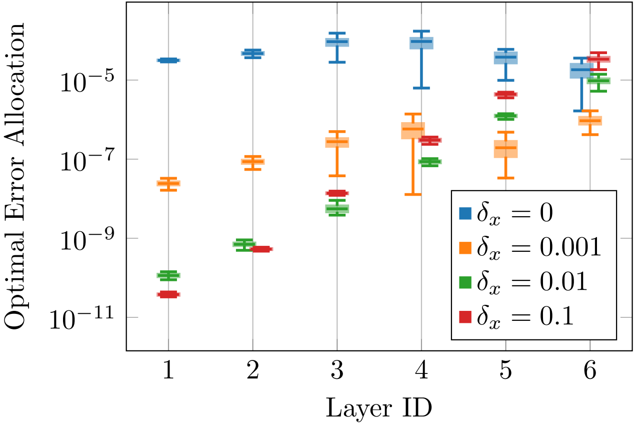
Results (1): effects of polynomial approximation error



$\delta_x, \delta_\epsilon$ measure the size of the input and error domains

- Smaller domains yield tighter estimates (esp. early layers)
- LiGAR may run 20-30 iterations to tighten δ_ϵ for each i

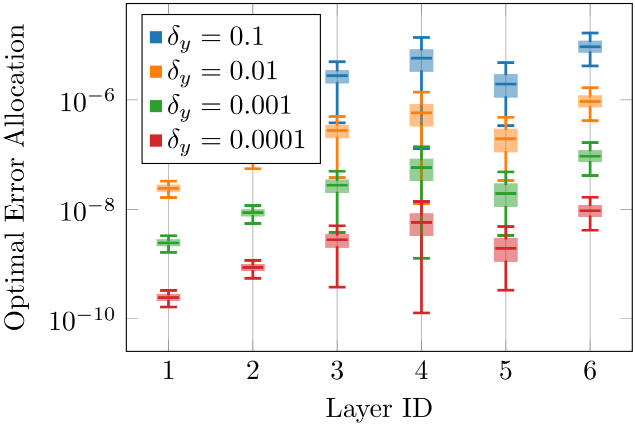
Results (2): effects of worst-case guarantees



Pure sampling $\delta_x = 0$ vs robust estimates $\delta_x > 0$

- To get guarantees, early layers will be conservative
- Sampled estimates yield uniform polynomial approximation

Results (3): effects of output error requirements



δ_y is the output error guarantee (design requirement)

- The optimisation constraint is $\sum_i L_i^\infty \epsilon_i \leq \delta_y$
- Linear effect on the optimal allocation of polynomial error

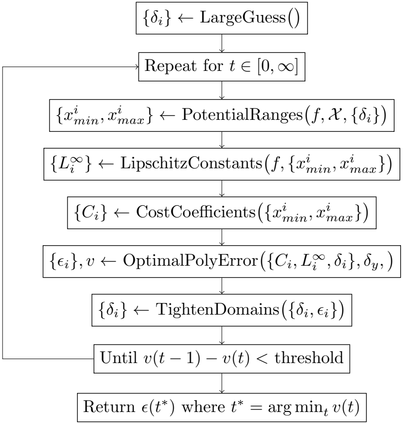
Discussion (1): drawbacks of worst-case design

Precision vs speed

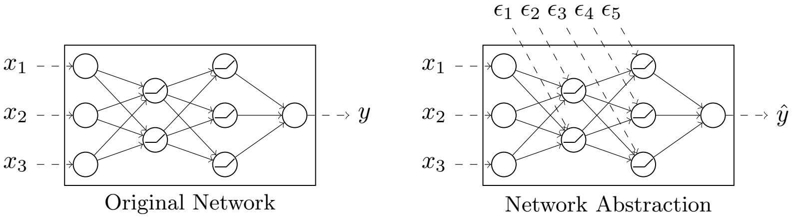
- Potential ranges
- Lipschitz constants
- are slow to compute
- and over-estimated

LiGAR equivalence

- We guarantee
- $\max_x ||f(x) - \hat{f}(x)||_\infty$
- which is not equal to
- classification accuracy



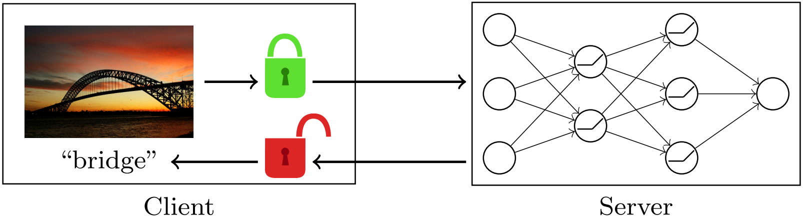
Discussion (2): generality of LiGAR's error abstraction



Any kind of neuron-specific error injection is possible!

- Applicable to: quantised neural network design
- Applicable to: robustness against weight perturbation
- Collaboration potential ;-)

Discussion (3): future of private inference?



Inference on Encrypted Data is Hard

- FHE schemes are order of magnitudes slower than plaintext
- Privacy vs speed tradeoff may not be worth it
- E.g. just focus on compressed NNs for edge computing?
- Time will tell...

Summary

Private inference on neural networks

- Run the NN on encrypted inputs
- Possible with FHE and polynomial activations
- Can we guarantee output equivalence?

LiGAR: Lipschitz-Guided Abstraction Refinement

- Polynomial error abstraction via neuron noise injection
- Iterative algorithmic design alternates between
- Estimating potential ranges and Lipschitz constants
- Minimising the polynomial degree of each activation

Any question?