Verification of Neural Networks' Local Differential Classification Privacy

Roie Reshef, Anan Kabaha, Olga Seleznova, and Dana Drachsler-Cohen

Technion, Haifa, Israel
{sror@campus.,anan.kabaha@campus.,olga.s@,ddana@ee.}technion.ac.il

Abstract. Neural networks are susceptible to privacy attacks. To date, no verifier can reason about the privacy of individuals participating in the training set. We propose a new privacy property, called *local differential classification privacy (LDCP)*, extending local robustness to a differential privacy setting suitable for black-box classifiers. Given a neighborhood of inputs, a classifier is LDCP if it classifies all inputs the same regardless of whether it is trained with the full dataset or whether any single entry is omitted. A naive algorithm is highly impractical because it involves training a very large number of networks and verifying local robustness of the given neighborhood separately for every network. We propose `Sphynx`, an algorithm that computes an abstraction of all networks, with a high probability, from a small set of networks, and verifies LDCP directly on the abstract network. The challenge is twofold: network parameters do not adhere to a known distribution probability, making it difficult to predict an abstraction, and predicting too large abstraction harms the verification. Our key idea is to transform the parameters into a distribution given by KDE, allowing to keep the over-approximation error small. To verify LDCP, we extend a MILP verifier to analyze an abstract network. Experimental results show that by training only 7% of the networks, `Sphynx` predicts an abstract network obtaining 93% verification accuracy and reducing the analysis time by $1.7 \cdot 10^4$x.

1 Introduction

Neural networks are successful in various tasks but are also vulnerable to attacks. One kind of attacks that is gaining a lot of attention is privacy attacks. Privacy attacks aim at revealing sensitive information about the network or its training set. For example, membership inference attacks recover entries in the training set [56,43,37,75,11,35,40], model inversion attacks reveal sensitive attributes of these entries [22,23], model extraction attacks recover the model's parameters [64], and property inference attacks infer global properties of the model [24]. Privacy attacks have been shown successful even against platforms providing a limited access to a model, including black-box access and a limited number of queries.

Such restricted access is common for platforms providing machine-learning-as-a-service (e.g., Google's Vertex AI[1], Amazon's ML on AWS[2], and BigML[3]).

A common approach to mitigate privacy attacks is *differential privacy* (DP) [18]. DP has been adopted by numerous network training algorithms [1,10,44,8,28]. Given a privacy level, a DP training algorithm generates the same network with a similar probability, regardless of whether a particular individual's entry is included in the dataset. However, DP is not an adequately suitable privacy criterion for black-box classifiers, for two reasons. First, it poses a too strong requirement: it requires that the training algorithm returns the same network (i.e., assign the same score to every class), whereas black-box classifiers are considered the same if they predict the same class (i.e., assign the *maximal* score to the same class). Second, DP can only be satisfied by randomized algorithms, adding noise to the computations. Consequently, the accuracy of the resulting network decreases. The amount of noise is often higher than necessary because the mathematical analysis of differentially private algorithms is highly challenging and thus practically not tight (e.g., it often relies on compositional theorems [18]). Thus, network designers often avoid adding noise to their networks. This raises the question: *What can a network designer provide as a privacy guarantee for individuals participating in the training set of a black-box classifier?*

We propose a new privacy property, called *local differential classification privacy (LDCP)*. Our property is designed for black-box classifiers, whose training algorithm is not necessarily DP. Conceptually, it extends the local robustness property, designed for adversarial example attacks [62,27], to a "deterministic differential privacy" setting. Local robustness requires that the network classifies all inputs in a given neighborhood the same. We extend this property by requiring that the network classifies all inputs in a given neighborhood the same *regardless of whether a particular individual's entry is included in the dataset.*

Proving that a network is LDCP is challenging, because it requires to check the local robustness of a very large number of networks: $|\mathcal{D}| + 1$ networks, where $\mathcal{D}$ is the dataset, which is often large (e.g., $>$ 10k entries). To date, verification of local robustness [31,25,63,58,65,71,42,54], analyzing a single network, takes non-negligible time. A naive, accurate but highly unscalable algorithm checks LDCP by training every possible network (i.e., one for every possibility to omit a single entry from the dataset), checking that all inputs in the neighborhood are classified the same network-by-network, and verifying that all networks classify these inputs the same. However, this naive algorithm does not scale since training and analyzing thousands of networks is highly time-consuming.

We propose `Sphynx` (**S**afety **P**rivacy analyzer via **Hy**per-**N**etworks) for determining whether a network is LDCP at a given neighborhood (Figure 1). `Sphynx` takes as inputs a network, its training set, its training algorithm, and a neighborhood. Instead of training $|\mathcal{D}|$ more networks, `Sphynx` computes a *hyper-network* abstracting these networks with a high probability (under several conditions).

[1] https://cloud.google.com/vertex-ai
[2] https://aws.amazon.com/machine-learning/
[3] https://bigml.com/

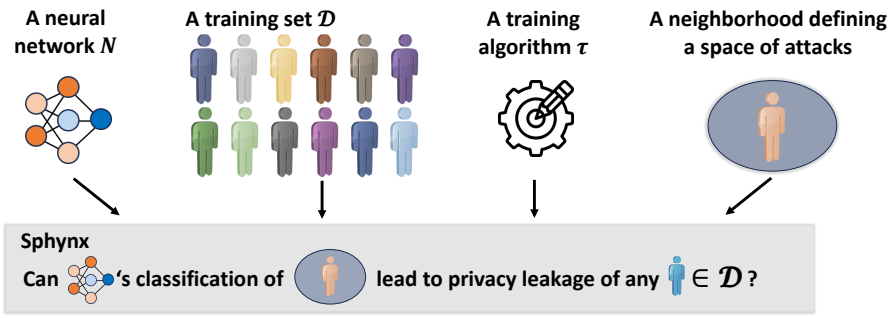

Fig. 1: Sphynx checks leakage of individuals' entries at a given neighborhood.

A hyper-network abstracts a set of networks by associating to each network parameter an interval that contains the respective values in all networks. If Sphynx would train all networks, computing the hyper-network would be straightforward. However, Sphynx's goal is to reduce the high time overhead and thus it does not train all networks. Instead, it predicts a hyper-network from a small number of networks. Sphynx then checks LDCP at the given neighborhood directly on the hyper-network. These two ideas enable Sphynx to obtain a practical analysis time. The main challenges in predicting a hyper-network are: (1) network parameters do not adhere to a known probability distribution and (2) the inherent trade-off between a sound abstraction, where each parameter's interval covers all values of this parameter in every network (seen and unseen) and the ability to succeed in verifying LDCP given the hyper-network. Naturally, to obtain a sound abstraction, it is best to consider large intervals for each network parameter, e.g., by adding noise to the intervals (like in adaptive data analysis [17,29,5,15,16,21,52,68]). However, the larger the intervals the harder it is to verify LDCP, because the hyper-network abstracts many more (irrelevant) networks. To cope, Sphynx transforms the network parameters into a distribution given by kernel density estimation (KDE), allowing to predict the intervals without adding noise.

To predict a hyper-network, Sphynx executes an iterative algorithm. In every iteration, it samples a few entries from the dataset. For each entry, it trains a network given all the dataset except this entry. Then, given all trained networks, Sphynx predicts a hyper-network, i.e., it computes an interval for every network parameter. An interval is computed by transforming every parameter's observed values into a distribution given by KDE, using normalization and the Yeo-Johnson transformation [77]. Then, Sphynx estimates whether the hyper-network abstracts every network with a high probability, and if so, it terminates.

Given a hyper-network, Sphynx checks LDCP directly on the hyper-network. To this end, we extend a local robustness verifier [63], relying on mixed-integer linear programming (MILP), to analyze a hyper-network. Our extension replaces the equality constraints, capturing the network's affine computations, with inequality constraints, since our network parameters are associated with intervals and not real numbers. To mitigate an over-approximation error, we propose two

approaches. The first approach relies on preprocessing to the network's inputs and the second one relies on having a lower bound on the inputs.

We evaluate `Sphynx` on data-sensitive datasets: Adult Census [13], Bank Marketing [41] and Default of Credit Card Clients [76]. We verify LDCP on three kinds of neighborhoods for checking safety to label-only membership attacks [11,35,40], adversarial example attacks in a DP setting (like [34,46]), and sensitive attributes (like [22,23]). We show that by training only 7% of the networks, `Sphynx` predicts a hyper-network abstracting an (unseen) network with a probability of at least 0.9. Our hyper-networks obtain 93% verification accuracy. Compared to the naive algorithm, `Sphynx` provides a significant speedup: it reduces the training time by 13.6x and the verification time by $1.7 \cdot 10^4$x.

2 Preliminaries

In this section, we provide the necessary background.

Neural network classifiers We focus on binary classifiers, which are popular for data-sensitive tasks. As example, we describe the data-sensitive datasets used in our evaluation and their classifier's task (Section 6). Adult Census [13] consists of user records of the socioeconomic status of people in the US. The goal of the classifier is to predict whether a person's yearly income is higher or lower than 50K USD. Bank Marketing [41] consists of user records of direct marketing campaigns of a Portuguese banking institution. The goal of the classifier is to predict whether the client will subscribe to the product or not. Default of Credit Card Clients [76] consists of user records of demographic factors, credit data, history of payments, and bill statements of credit card clients in Taiwan. The goal of the classifier is to predict whether the default payment will be paid in the next month. We note that our definitions and algorithms easily extend to non-binary classifiers. A binary classifier N maps an input, a user record, $x \in \mathcal{X} \subseteq [0,1]^d$ to a real number $N(x) \in \mathbb{R}$. If $N(x) \geq 0$, we say the classification of x is 1 and write $\text{class}(N(x)) = 1$, otherwise, it is -1, i.e., $\text{class}(N(x)) = -1$. We focus on classifiers implemented by a fully-connected neural network. This network consists of an input layer followed by L layers. The input layer x_0 takes as input $x \in \mathcal{X}$ and passes it as is to the next layer (i.e., $x_{0,k} = x_k$). The next layers are functions, denoted $f_1, f_2, \ldots, f_L$, each taking as input the output of the preceding layer. The network's function is the composition of the layers: $N(x) = f_L(f_{L-1}(\cdots(f_1(x))))$. A layer m consists of neurons, denoted $x_{m,1}, \ldots, x_{m,k_m}$. Each neuron takes as input the outputs of all neurons in the preceding layer and outputs a real number. The output of layer m is the vector $(x_{m,1}, \ldots, x_{m,k_m})^T$ consisting of all its neurons' outputs. A neuron $x_{m,k}$ has a weight for each input $w_{m,k,k'} \in \mathbb{R}$ and a bias $b_{m,k} \in \mathbb{R}$. Its function is the composition of an affine computation, $\hat{x}_{m,k} = b_{m,k} + \sum_{k'=1}^{k_{m-1}} w_{m,k,k'} \cdot x_{m-1,k'}$, followed by an activation function computation, $x_{m,k} = \sigma(\hat{x}_{m,k})$. Activation functions are typically non-linear functions. In this work, we focus on the ReLU activation function, $\text{ReLU}(\hat{x}) = \max(0, \hat{x})$. We note that, while we focus on

fully-connected networks, our approach can extend to other architectures, e.g., convolutional networks or residual networks. The weights and biases of a neural network are determined by a training process. A training algorithm $\mathcal{T}$ takes as inputs a network (typically, with random weights and biases) and a labelled training set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq \mathcal{X} \times \{-1, +1\}$. It returns a network with updated weights and biases. These parameters are computed with the goal of minimizing a given loss function, e.g., binary cross-entropy, capturing the level of inaccuracy of the network. The computation typically relies on iterative numerical optimization, e.g., stochastic gradient descent (SGD).

Differential privacy (DP) DP focuses on algorithms defined over arrays (in our context, a dataset). At high-level, an algorithm is DP if for any two inputs differing in a single entry, it returns the same output with a similar probability. Formally, DP is a probabilistic privacy property requiring that the probability of returning different outputs is upper bounded by an expression defined by two parameters, denoted ϵ and δ [18]. Note that this requirement is too strong for classifiers providing only black-box access, which return only the input's classification. For such classifiers, it is sufficient to require that the classification is the same, and there is no need for the network's output (the score of every class) to be the same. To obtain the DP guarantee, DP algorithms add noise, drawn from some probability distribution (e.g., Laplace or Gaussian), to the input or their computations. That is, DP algorithms trade off their output's accuracy with privacy guarantees: the smaller the DP parameters (i.e., ϵ and δ) the more private the algorithm is, but its outputs are less accurate. The accuracy loss is especially severe in DP algorithms that involve loops in which every iteration adds noise. The loss is high because (1) many noise terms are added and (2) the mathematical analysis is not tight (it typically relies on compositional theorems [18]), leading to adding a higher amount of noise than necessary to meet the target privacy guarantee. Nevertheless, DP has been adopted by numerous network training algorithms [1,10,44,8,28]. For example, one algorithm adds noise to every gradient computed during training [1]. Consequently, the network's accuracy decreases significantly, discouraging network designers from employing DP. To cope, we propose a (non-probabilistic) privacy property that (1) only requires the network's classification to be the same and (2) can be checked even if the network has not been trained by a DP training algorithm.

Local robustness Local robustness has been introduced in response to adversarial example attacks [62,27]. In the context of network classifiers, an adversarial example attack is given an input and a space of perturbations and it returns a perturbed input that causes the network to misclassify. Ideally, to prove that a network is robust to adversarial attacks, one should prove that *for any valid input*, the network classifies the same under any valid perturbation. In practice, the safety property that has been widely-studied is *local robustness*. A network is locally robust at *a given input* if perturbing the input by a perturbation in a given space does not cause the network to change the classification. Formally, the space of allowed perturbations is captured by a *neighborhood* around the input.

Definition 1 (Local Robustness). *Given a network N, an input $x \in \mathcal{X}$, a neighborhood $I(x) \subseteq \mathcal{X}$ containing x, and a label $y \in \{-1, +1\}$, the network N is* locally robust at $I(x)$ with respect to y *if* $\forall x' \in I(x).\ class(N(x')) = y$.

A well-studied definition of a neighborhood is the ϵ-*ball* with respect to the L_∞ norm [31,25,63,58,65,71,42,54]. Formally, given an input x and a bound on the perturbation amount $\epsilon \in \mathbb{R}^+$, the ϵ-ball is: $I^\infty_\epsilon(x) = \{x' \mid \|x' - x\|_\infty \leq \epsilon\}$. A different kind of a neighborhood captures *fairness* with respect to a given sensitive feature $i \in [d]$ (e.g., gender) [6,38,69,53]: $I^S_i(x) = \left\{x' \mid \bigwedge_{j \in [d] \setminus \{i\}} x'_j = x_j\right\}$.

3 Local Differential Classification Privacy

In this section, we define the problem of verifying that a network is locally differentially classification private (LDCP) at a given neighborhood.

Local differential classification privacy (LDCP) Our property is defined given a classifier N, trained on a dataset $\mathcal{D}$, and a neighborhood I, defining a space of attacks (perturbations). It considers N private with respect to an individual's entry (x', y') if N classifies all inputs in I the same, whether N has been trained with (x', y') or not. If there is discrepancy in the classification, the attacker may exploit it to infer information about x'. Namely, if the network designer cared about a single individual's entry (x', y'), our privacy property would be defined over two networks: the network trained with (x', y') and the network trained without (x', y'). Naturally, the network designer wishes to show that the classifier is private for every (x', y') participating in $\mathcal{D}$. Namely, our property is defined over $|\mathcal{D}| + 1$ networks. The main challenge in verifying this privacy property is that the training set size is typically very large (>10k entries). Formally, our property is an extension of local robustness to a differential privacy setting suitable for classifiers providing black-box access. It requires that the inputs in I are classified the same by *every* network trained on $\mathcal{D}$ or trained on $\mathcal{D}$ except for any single entry. Unlike DP, our definition is applicable to any network, even those trained without a probabilistic noise. We next formally define our property.

Definition 2 (Local Differential Classification Privacy). *Given a network N trained on a dataset $\mathcal{D} \subseteq \mathcal{X} \times \{-1, +1\}$ using a training algorithm $\mathcal{T}$, a neighborhood $I(x) \subseteq \mathcal{X}$ and a label $y \in \{-1, +1\}$, the network N is* locally differentially classification private (LDCP) *if (1) N is locally robust at $I(x)$ with respect to y, and (2) for every $(x', y') \in \mathcal{D}$, the network of the same architecture as N trained on $\mathcal{D} \setminus \{(x', y')\}$ using $\mathcal{T}$ is locally robust at $I(x)$ with respect to y.*

Our privacy property enables to prove safety against privacy attacks by defining a suitable set of neighborhoods. For example, several membership inference attacks [11,35,40] are label-only attacks. Namely, they assume the attacker has a set of inputs and they can query the network to obtain their classification. To prove safety against these attacks, given a set of inputs $X \subseteq \mathcal{X}$, one has to check LDCP for every neighborhood defined by $I^\infty_0(x)$ where $x \in X$. To prove safety

against attacks aiming to reveal sensitive features (like [22,23]), given a set of inputs $X \subseteq \mathcal{X}$, one has to check that the network classifies the same regardless of the value of the sensitive feature. This can be checked by checking LDCP for every neighborhood defined by $I_i^S(x)$ where $x \in X$ and i is the sensitive feature.

Problem definition We address the problem of determining whether a network is locally differentially classification private (LDCP) at a given neighborhood, while minimizing the analysis time. A definite answer involves training and verifying $|\mathcal{D}| + 1$ networks (the network trained given all entries in the training set, and for each entry in the training set, the network trained given all entries except this entry). However, this results in a very long training time and verification time (even local robustness verification takes a non-negligible time). Instead, our problem is to provide an answer which is correct with a high probability. This problem is highly challenging. On the one hand, the fewer trained networks the lower the training and verification time. On the other hand, determining an answer, with a high probability, from a small number of networks is challenging, since network parameters do not adhere to a known probabilistic distribution. Thus, our problem is not naturally amenable to a probabilistic analysis.

Prior work Prior work has considered different aspects of our problem. As mentioned, several works adapt training algorithms to guarantee that the resulting network satisfies differential privacy [1,10,44,8,28]. However, these training algorithms tend to return networks of lower accuracy. A different line of research proposes algorithms for *machine unlearning* [36,26,9], in which an algorithm retrains a network "to forget" some entries of its training set. However, these approaches do not guarantee that the forgetting network is equivalent to the network obtained by training without these entries from the beginning. It is also more suitable for settings in which there are multiple entries to forget, unlike our differential privacy setting which focuses on omitting a single entry at a time. Several works propose verifiers to determine local robustness [45,30,19,48,73,55,31]. However, these analyze a single network and not a group of similar networks. A recent work proposes a proof transfer between similar networks [67] in order to reduce the verification time of similar networks. However, this work requires to explicitly have all networks, which is highly time consuming in our setting. Other works propose verifiers that check robustness to data poisoning or data bias [61,39,12]. These works consider an attacker that can manipulate or omit up to several entries from the dataset, similarly to LDCP allowing to omit up to a single entry. However, these verifiers either target patch attacks of image classifiers [61], which allows them to prove robustness without considering every possible network, or target decision trees [39,12], relying on predicates, which are unavailable for neural networks. Thus, neither is applicable to our setting.

4 Our Approach

In this section, we present our approach for determining whether a network is locally differentially classification private (LDCP). Our key idea is to *predict an*

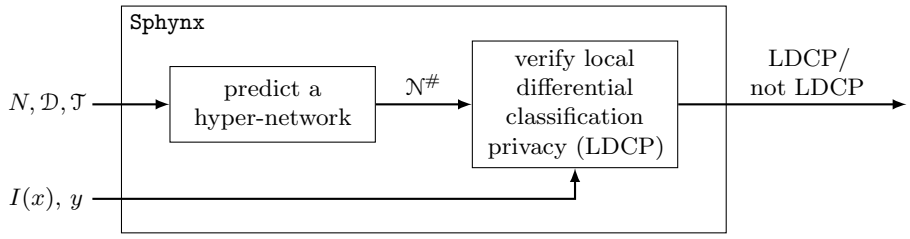

Fig. 2: Given a network classifier N, its training set $\mathcal{D}$ and its training algorithm $\mathcal{T}$, `Sphynx` predicts an abstract hyper-network $\mathcal{N}^{\#}$. It then checks whether $\mathcal{N}^{\#}$ is locally robust at $I(x)$ with respect to y to determine whether N is LDCP.

abstraction of all concrete networks from a small number of networks. We call the abstraction a *hyper-network*. Thanks to this abstraction, `Sphynx` does not need to train all concrete networks and neither verify multiple concrete networks. Instead, `Sphynx` first predicts an abstract hyper-network, given a network N, its training set $\mathcal{D}$ and its training algorithm $\mathcal{T}$, and then verifies LDCP directly on the hyper-network. This is summarized in Figure 2. We next define these terms.

Hyper-networks A *hyper-network* abstracts a set of networks $\mathcal{N} = \{N_1, \ldots, N_K\}$ with the same architecture and in particular the same set of parameters, i.e., weights $\mathcal{W} = \{w_{1,1,1}, \ldots, w_{L,d_L,d_{L-1}}\}$ and biases $\mathcal{B} = \{b_{1,1}, \ldots, b_{L,d_L}\}$. The difference between the networks is the parameters' values. In our context, $\mathcal{N}$ consists of the network trained with the full dataset and the networks trained without any single entry: $\mathcal{N} = \{\mathcal{T}(N, \mathcal{D})\} \cup \{\mathcal{T}(N, \mathcal{D} \setminus \{(x', y')\}) \mid (x', y') \in \mathcal{D}\}$. A *hyper-network* is a network $\mathcal{N}^{\#}$ with the same architecture and set of parameters, but the domain of the parameters is not $\mathbb{R}$ but rather an abstract domain $\mathcal{A}$. As standard, we assume the abstract domain corresponds to a lattice and is equipped with a concretization function γ. We focus on a non-relational abstraction, where each parameter is abstracted independently. The motivation is twofold. First, non-relational domains are computationally lighter than relational domains. Second, the relation between the parameters is highly complex because it depends on a long series of optimization steps (e.g., SGD steps). Thus, while it is possible to bound these computations using simpler functions (e.g., polynomial functions), the over-approximation error would be too large to be useful in practice.

Formally, given a set of networks $\mathcal{N}$ with the same architecture and set of parameters $\mathcal{W} \cup \mathcal{B}$ and given an abstract domain $\mathcal{A}$ and a concretization function $\gamma : \mathcal{A} \to \mathcal{P}(\mathbb{R})$, a *hyper-network* is a network $\mathcal{N}^{\#}$ with the same architecture and set of parameters, where the parameters range over $\mathcal{A}$ and satisfy the following:

$$\forall N' \in \mathcal{N} : \left[\forall w_{m,k,k'} \in \mathcal{W} : w^{N'}_{m,k,k'} \in \gamma\left(w^{\#}_{m,k,k'}\right) \wedge \forall b_{m,k} \in \mathcal{B} : b^{N'}_{m,k} \in \gamma\left(b^{\#}_{m,k}\right)\right]$$

where $w^{N'}_{m,k,k'}$ and $b^{N'}_{m,k}$ are the values of the parameters in the network N' and $w^{\#}_{m,k,k'}$ and $b^{\#}_{m,k}$ are the values of the parameters in the hyper-network $\mathcal{N}^{\#}$.

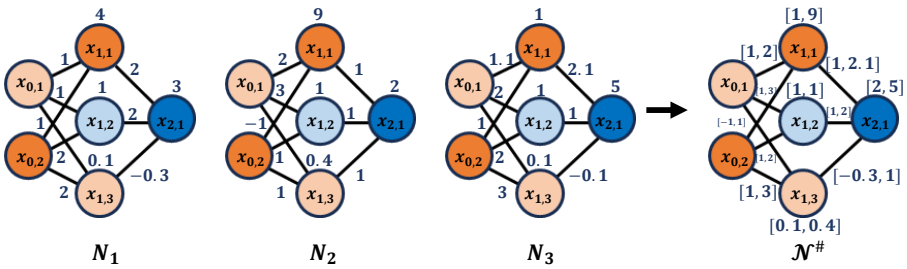

Fig. 3: Three networks, N_1, N_2, N_3, and their interval hyper-network $\mathcal{N}^\#$.

Interval abstraction In this work, we focus on the interval domain. Namely, the abstract elements are intervals $[l, u]$, with the standard meaning: an interval $[l, u]$ abstracts all real numbers between l and u. We thus call our hyper-networks *interval hyper-networks*. Figure 3 shows an example of an interval hyper-network. An interval corresponding to a weight is shown next to the respective edge, and an interval corresponding to a bias is shown next to the respective neuron. For example, the neuron $x_{1,2}$ has two weights and a bias whose values are: $w^\#_{1,2,1} = [1, 3]$, $w^\#_{1,2,2} = [1, 2]$, and $b^\#_{1,2} = [1, 1]$. Computing an interval hyper-network is straightforward if all concrete networks are known. However, computing all concrete networks defeats the purpose of having a hyper-network. Instead, `Sphynx` predicts an interval hyper-network with a high probability (Section 5.1).

Checking LDCP given a hyper-network Given an interval hyper-network $\mathcal{N}^\#$ for a network N, a neighborhood $I(x)$ and a label y, `Sphynx` checks whether N is LDCP by checking whether $\mathcal{N}^\#$ is locally robust at $I(x)$ with respect to y. If $\mathcal{N}^\#$ is robust, then N is LDCP at $I(x)$, with a high probability. Otherwise, `Sphynx` determines that N is not LDCP at $I(x)$. Note that this is a conservative answer since N is either not LDCP or that the abstraction or the verification lose too much precision. `Sphynx` verifies local robustness of $\mathcal{N}^\#$ by extending a MILP verifier [63] checking local robustness of a neural network (Section 5.2).

5 Sphynx: Safety Privacy Analyzer via Hyper-Networks

In this section, we present `Sphynx`, our system for verifying local differential classification privacy (LDCP). As described, it consists of two components, the first component predicts a hyper-network, while the second one verifies LDCP.

5.1 Prediction of an Interval Hyper-Network

In this section, we introduce `Sphynx`'s algorithm for predicting an interval hyper-network, called `PredHyperNet`. `PredHyperNet` takes as inputs a network N, its training set $\mathcal{D}$, its training algorithm $\mathcal{T}$, and a probability error bound α, where $\alpha \in (0, 1)$ is a small number. It returns an interval hyper-network $\mathcal{N}^\#$ which,

Algorithm 1: `PredHyperNet` $(N, \mathcal{D}, \mathcal{T}, \alpha)$

Input: a network N, a training set $\mathcal{D}$, a training algorithm $\mathcal{T}$, an error bound α.
Output: an interval hyper-network.

1 nets $\leftarrow \{N\}$
2 entr $\leftarrow \emptyset$
3 $\mathcal{N}^{\#}_{prev} \leftarrow \bot$
4 $\mathcal{N}^{\#}_{curr} \leftarrow N^{\#}$
5 **while** $\Sigma_{w \in \mathcal{W} \cup \mathcal{B}} \mathbb{1} \left[1 - J([l^w_{curr}, u^w_{curr}], [l^w_{prev}, u^w_{prev}]) \leq R\right] < M \cdot |\mathcal{W} \cup \mathcal{B}|$ **do**
6 $\mathcal{N}^{\#}_{prev} \leftarrow \mathcal{N}^{\#}_{curr}$
7 **for** k *iterations* **do**
8 $(x, y) \leftarrow \text{Random}\,(\mathcal{D} \setminus \text{entr})$
9 entr $\leftarrow$ entr $\cup \{(x, y)\}$
10 nets $\leftarrow$ nets $\cup \{\mathcal{T}\,(N, \mathcal{D} \setminus \{(x, y)\})\}$
11 **for** $w \in \mathcal{W} \cup \mathcal{B}$ **do**
12 $\mathcal{V}_w \leftarrow \{w^{N'} \mid N' \in \text{nets}\}$
13 $w^{\mathcal{N}^{\#}_{curr}} \leftarrow$ `PredInt` $\left(\mathcal{V}_w, \frac{\alpha}{|\mathcal{W} \cup \mathcal{B}|}\right)$
14 **return** $\mathcal{N}^{\#}_{curr}$

with probability $1 - \alpha$ (under certain conditions), abstracts an (unseen) concrete network returned by $\mathcal{T}$ given N and $\mathcal{D} \setminus \{(x, y)\}$, where $(x, y) \in \mathcal{D}$. The main idea is to predict an abstraction for every network parameter from a small number of concrete networks. To minimize the number of networks, `PredHyperNet` executes iterations. An iteration trains k networks and predicts an interval hyper-network using all trained networks. If the intervals' distributions have not converged to the expected distributions, another iteration begins. We next provide details.

`PredHyperNet`'s algorithm `PredHyperNet` (Algorithm 1) begins by initializing the set of trained networks `nets` to N and the set of entries `entr`, whose corresponding networks are in `nets`, to $\emptyset$. It initializes the previous hyper-network $\mathcal{N}^{\#}_{prev}$ to $\bot$ and the current hyper-network $\mathcal{N}^{\#}_{curr}$ to the interval abstraction of N, i.e., the interval of a network parameter $w \in \mathcal{W} \cup \mathcal{B}$ (a weight or a bias) is $[w^N, w^N]$. Then, while the stopping condition (Line 5), checking convergence using the Jaccard distance as described later, is `false`, it runs an iteration. An iteration trains k new networks (Line 7–Line 10). A training iteration samples an entry (x, y), adds it to `entr`, and runs the training algorithm $\mathcal{T}$ on (the architecture of) N and $\mathcal{D} \setminus \{(x, y)\}$. The trained network is added to `nets`. `PredHyperNet` then computes an interval hyper-network from `nets` (Line 11–Line 13). The computation is executed via `PredInt` (Line 13) independently on each network parameter w. `PredInt`'s inputs are all observed values of w and a probability error bound α', which is α divided by the number of network parameters.

Interval prediction: overview `PredInt` predicts an interval for a parameter w from a (small) set of values $\mathcal{V}_w = \{w_1, \ldots, w_K\}$, obtained from the concrete

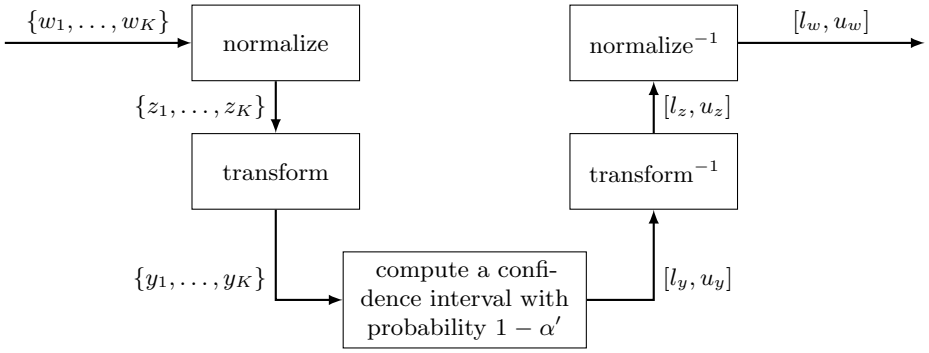

Fig. 4: The flow of `PredInt` for predicting an interval for a network parameter.

networks, and given an error bound α'. If it used interval abstraction, w would be mapped to the interval defined by the minimum and maximum in $\mathcal{V}_w$. However, this approach cannot guarantee to abstract the unseen concrete networks, since we aim to rely on a number of concrete networks significantly smaller than $|\mathcal{D}|$. Instead, `PredInt` defines an interval by predicting the minimum and maximum of w, over all its values in the (seen and unseen) networks. There has been an extensive work on estimating statistics, with a high probability, of an unknown probability distribution (e.g., expectation [17,29,5,15,16,21,52,68]). However, `PredInt`'s goal is to estimate the *minimum* and *maximum* of unknown samples from an unknown probability distribution. The challenge is that, unlike other statistics, the minimum and maximum are highly sensitive to outliers. To cope, `PredInt` transforms the unknown probability distribution of w into a known one and then predicts the minimum and maximum. Before the transformation, `PredInt` normalizes the values in $\mathcal{V}_w$. Overall, `PredInt`'s operation is (Figure 4): (1) it normalizes the values in $\mathcal{V}_w$, (2) it transforms the values into a known probability distribution, (3) it predicts the minimum and maximum by computing a confidence interval with a probability of $1 - \alpha'$, and (4) it inverses the transformation and normalization to fit the original scale. We note that executing normalization and transformation and then their inversion does not result in any information loss, because they are bijective functions. We next provide details.

Transformation and normalization `PredInt` transforms the values in $\mathcal{V}_w$ to make them seem as if drawn from a known probability distribution. It employs the Yeo-Johnson transformation [77], transforming an unknown distribution into a Gaussian distribution. This transformation has the flexibility that the input random variables can have any real value. It has a parameter λ, whose value is determined using maximum likelihood estimation (MLE) (i.e., λ maximizes the likelihood that the transformed data is Gaussian). It is defined as follows:

$$T_\lambda(z) = \left\{ \begin{array}{lll} \frac{(1+z)^\lambda - 1}{\lambda}, & \lambda \neq 0, z \geq 0; & \log(1+z), \quad \lambda = 0, z \geq 0 \\ -\frac{(1-z)^{2-\lambda} - 1}{2-\lambda}, & \lambda \neq 2, z < 0; & -\log(1-z), \quad \lambda = 2, z < 0 \end{array} \right\}$$

`PredInt` employs several adaptations to this transformation to fit our setting better. First, it requires $\lambda \in [0, 2]$. This is because if $\lambda < 0$, the transformed outputs have an upper bound $\frac{1}{-\lambda}$ and if $\lambda > 2$, they have a lower bound $-\frac{1}{\lambda-2}$. Since our goal is to predict the minimum and maximum values, we do not want the transformed outputs to artificially limit them. Second, the Yeo-Johnson transformation transforms into a Gaussian distribution. Instead, we transform to a distribution given by kernel density estimation (KDE) with a Laplace kernel, which is better suited for our setting (Section 6). We describe KDE in Appendix A. Third, the Yeo-Johnson transformation operates best when given values centered around zero. However, training algorithms produce parameters of different scales and centered around different values. Thus, before the transformation, `PredInt` normalizes the values in $\mathcal{V}_w$ to be centered around zero and have a similar scale as follows: $z_i \leftarrow \frac{w_i - \mu}{\Delta}$, $\forall i \in [K]$. Consequently, `PredInt` is invariant to translation and scaling and is thus more robust to the computations of the training algorithm $\mathcal{T}$. There are several possibilities to define the normalization's parameters, μ and Δ, each is based on a different norm, e.g., the L_1, L_2 or L_∞ norm. In our setting, a normalization based on the L_1 norm works best, since we use a Laplace kernel. Its center point is $\mu = \text{median}(w_1, \ldots, w_K)$ and its scale is the centered absolute first moment: $\Delta = \frac{1}{K} \sum_{i=1}^{K} |w_i - \mu|$.

Interval prediction After the transformation, `PredInt` has a cumulative distribution function (CDF) for the transformed values: $F_y(v) = \mathbb{P}\{y \leq v\}$. Given the CDF, we compute a confidence interval, defining the minimum and maximum. A confidence interval, parameterized by $\alpha' \in (0, 1)$, is an interval satisfying that the probability of an unknown sample being inside it is at least $1 - \alpha'$. It is defined by: $[l_y, u_y] = \left[F_y^{-1}\left(\frac{\alpha'}{2}\right), F_y^{-1}\left(1 - \frac{\alpha'}{2}\right)\right]$. Since we wish to compute an interval hyper-network $\mathcal{N}^{\#}$ abstracting an unseen network with probability $1 - \alpha$ and since there are $|\mathcal{W} \cup \mathcal{B}|$ parameters, we choose $\alpha' = \frac{\alpha}{|\mathcal{W} \cup \mathcal{B}|}$ (Line 13). By the union bound, we obtain confidence intervals guaranteeing that the probability that an unseen concrete network is covered by the hyper-network is at least $1 - \alpha$.

Stopping condition The goal of `PredHyperNet`'s stopping condition is to identify when the distributions of the intervals computed for $\mathcal{N}^{\#}_{curr}$ have converged to their expected distributions. This is the case when the intervals have not changed significantly in $\mathcal{N}^{\#}_{curr}$ compared to $\mathcal{N}^{\#}_{prev}$. Formally, for each network parameter w, it compares the current interval to the previous interval by computing their Jaccard distance. Given the previous and current intervals of w, I_{prev} and I_{curr}, the Jaccard Index is: $J(I_{curr}, I_{prev}) = \frac{I_{curr} \sqcap I_{prev}}{I_{curr} \sqcup I_{prev}}$. For any two intervals, the Jaccard distance $1 - J(I_{curr}, I_{prev}) \in [0, 1]$, such that the smaller the distance, the more similar the intervals are. If the Jaccard distance is below a ratio R (a small number), we consider the interval of w as converged to the expected CDF. If at least $M \cdot 100\%$ of the hyper-network's intervals have converged, we consider that the hyper-network has converged, and thus `PredHyperNet` terminates.

5.2 Verification of a Hyper-Network

In this section, we explain how `Sphynx` verifies that an interval hyper-network is locally robust, in order to show that the network is LDCP. Our verification extends a local robustness verifier [63], designed for checking local robustness of a (concrete) network, to check local robustness given an interval hyper-network. This verifier encodes the verification task as a mixed-integer linear program (MILP), which is submitted to a MILP solver, and provides a sound and complete analysis. We note that we experimented with extending incomplete analysis techniques to our setting (interval analysis and DeepPoly [58]) to obtain a faster analysis. However, the over-approximation error stemming both from these techniques and our hyper-network led to a low precision rate. The challenge with extending the verifier by Tjeng et al. [63] is that both the neuron values and the network parameters are variables, leading to quadratic constraints, which are computationally heavy for constraint solvers. Instead, `Sphynx` relaxes the quadratic constraints. To mitigate an over-approximation error, we propose two approaches. We begin with a short background and then describe our extension.

Background The MILP encoding by Tjeng et al. [63] precisely captures the neurons' affine computation as linear constraints. The ReLU computations are more involved, because ReLU is non-linear. For each ReLU computation, a boolean variable is introduced along with four linear constraints. The neighborhood is expressed by constraining each input value by an interval, and the local robustness check is expressed by a linear constraint over the network's output neurons. This encoding has been shown to be sound and complete. To scale the analysis, every neuron is associated with a real-valued interval. This allows to identify ReLU neurons whose computation is linear, which is the case if the input's interval is either non-negative or non-positive. In this case, the boolean variable is not introduced for this ReLU computation and the encoding's complexity decreases.

Extension to hyper-networks To extend this verifier to analyze hyper-networks, we encode the affine computations differently because network parameters are associated with intervals and not real numbers. A naive extension replaces the parameter values in the linear constraints by variables, which are bounded by intervals. However, this leads to quadratic constraints and significantly increases the problem's complexity. To keep the constraints linear, one option is to introduce a fresh variable for every multiplication of a neuron's input and a network parameter. However, such variable would be bounded by the interval abstracting the multiplication of the two variables, which may lead to a very large over-approximation error. Instead, we rely on the following observation: if the input to every affine computation is non-negative, then the abstraction of the multiplication does not introduce an over-approximation error. This allows us to replace the constraint of each affine variable $\hat{x}$ (defined in Section 2), previously captured by an equality constraint, with two inequalities providing a lower and upper bound on $\hat{x}$. Formally, given lower and upper bounds of the weights and biases at the matrices l_W and u_W and the vectors l_b and u_b, the affine variable $\hat{x}$

is bounded by:

$$l_W \cdot x + l_b \leq \hat{x} \leq u_W \cdot x + u_b$$

To guarantee that the input to every affine computation x is non-negative our analysis requires (1) preprocessing of the network's inputs and (2) a non-negative lower bound to every activation function's output. The second requirement holds since the MILP encoding of ReLU explicitly requires a non-negative output. If preprocessing is inapplicable (i.e., $x \in \mathcal{X} \not\subseteq [0,1]^d$) or the activation function may be negative (e.g., leaky ReLU), we propose another approach to mitigate the precision loss, given a lower bound $l_x \leq x$. Given lower and upper bounds for the weights and biases l_W, u_W, l_b, and u_b, we can bound the output by:

$$l_W \cdot x + l_b - (u_W - l_W) \cdot \max(0, -l_x) \leq \hat{x} \leq u_W \cdot x + u_b + (u_W - l_W) \cdot \max(0, -l_x)$$

We provide a proof in Appendix B. Note that each bound is penalized by $(u_W - l_W) \cdot \max(0, -l_x) \geq 0$, which is an over-approximation error term for the case where the lower bound l_x is negative.

5.3 Analysis of Sphynx

In this section, we discuss the correctness of `Sphynx` and its running time. Proofs are provided in Appendix B.

Correctness Our first lemma states the conditions under which `PredInt` computes an abstraction for the values of a single network parameter with a high probability.

Lemma 1. *Given a parameter w and an error bound α', if the observed values $w_1, \ldots, w_K$ are IID and suffice to predict the correct distribution, and if there exists $\lambda \in [0, 2]$ such that the distribution of the transformed normalized values $y_1, \ldots, y_K$ is similar to a distribution given by KDE with a Laplace kernel (using the bandwidth defined in Appendix A), then `PredInt` computes a confidence interval containing an unseen value w_i, for $i \in \{K+1, \ldots, |\mathcal{D}|+1\}$, with a probability of $1 - \alpha'$.*

Note that our lemma does not make any assumption about the distribution of the observed values $w_1, \ldots, w_K$. Next, we state our theorem pertaining the correctness of `PredHyperNet`. The theorem states that when the stopping condition identifies correctly when the observed values have converged to the correct distribution, then the hyper-network abstracts an unseen network with the expected probability.

Theorem 1. *Given a network N, its training set $\mathcal{D}$, its training algorithm $\mathcal{T}$, and an error bound α, if R is close to 0 and M is close to 1, then `PredHyperNet` returns a hyper-network abstracting an unseen network with probability $1 - \alpha$.*

Our next theorem states that our verifier is sound and states when it is complete. Completeness means that if the hyper-network is locally robust at the given neighborhood, `Sphynx` is able to prove it.

Theorem 2. *Our extension to the MILP verifier provides a sound analysis. It is also complete if all inputs to the affine computations are non-negative.*

By Theorem 2 and Definition 2, if `Sphynx` determines that the hyper-network is locally robust, then the network is LDCP. Note that it may happen that the network is LDCP, but the hyper-network is not locally robust due to the abstraction's precision loss.

Running time The running time of `Sphynx` consists of the running time of `PredHyperNet` and the running time of the verifier. The running time of `PredHyperNet` consists of the networks' training time and the time to compute the hyper-networks (the running time of the stopping condition is negligible). The training time is the product of the number of networks and the execution time of the training algorithm $\mathcal{T}$. The time complexity of `PredInt` is $O(K^2)$, where $K = |\texttt{nets}|$, and thus the computation of a hyper-network is: $O\left(K^2 \cdot |\mathcal{W} \cup \mathcal{B}|\right)$. Since `PredHyperNet` runs $\frac{K}{k}$ iterations, overall, the running time is $O\left(|\mathcal{T}| \cdot K + \frac{K^3}{k} \cdot |\mathcal{W} \cup \mathcal{B}|\right)$. In practice, the second term is negligible compared to the first term ($|\mathcal{T}| \cdot K$). Thus, the fewer trained networks the faster `PredHyperNet` is. The running time of the verifier is similar to the running time of the MILP verifier [63], verifying local robustness of a single network, which is exponential in the number of ReLU neurons whose computation is non-linear (their input's interval contains negative and positive numbers). Namely, `Sphynx` reduces the verification time by a factor of $|\mathcal{D}|+1$ compared to the naive algorithm that verifies robustness network-by-network.

6 Evaluation

We implemented `Sphynx` in Python[4]. Experiments ran on an Ubuntu 20.04 OS on a dual AMD EPYC 7713 server with 2TB RAM and 8 NVIDIA A100 GPUs. The hyper-parameters of `PredHyperNet` are: $\alpha = 0.1$, the number of trained networks in every iteration is $k = 400$, the thresholds of the stopping condition are $M = 0.9$ and $R = 0.1$. We evaluate `Sphynx` on the three data-sensitive datasets described in Section 2: Adult Census [13] (Adult), Bank Marketing [41] (Bank), and Default of Credit Card Clients [76] (Credit). We preprocessed the input values to range over $[0, 1]$ as follows. Continuous attributes were normalized to range over $[0, 1]$ and categorical attributes were transformed into two features ranging over $[0, 1]$: $\cos\left(\frac{\pi}{2} \cdot \frac{i}{m-1}\right)$ and $\sin\left(\frac{\pi}{2} \cdot \frac{i}{m-1}\right)$, where m is the number of categories and i is the category's index $i \in \{0, \ldots, m-1\}$. Binary attributes were transformed with a single feature: 0 for the first category and 1 for the second one. While one hot encoding is highly popular for categorical attributes, it has also been linked to reduced model accuracy when the number of categories is high [74,51]. However, we note that the encoding is orthogonal to our algorithm. We consider three fully-connected networks: 2×50, 2×100, and 4×50, where the first number is

[4] Code is at: https://github.com/Robgy/Verification-of-Neural-Networks-Privacy

the number of intermediate layers and the second one is the number of neurons in each intermediate layer. Our network sizes are comparable to or larger than the sizes of the networks analyzed by verifiers targeting these datasets [69,38,4]. All networks reached around 80% accuracy. The networks were trained over 10 epochs, using SGD with a batch size of 1024 and a learning rate of 0.1. We used L_1 regularization with a coefficient of 10^{-5}. All networks were trained with the same random seed, so `Sphynx` can identify the maximal privacy leakage (allowing different seeds may reduce the privacy leakage since it can be viewed as adding noise to the training process). We remind that `Sphynx`'s challenge is intensified compared to local robustness verifiers: their goal is to prove robustness of a single network, whereas `Sphynx`'s goal is to prove that privacy is preserved over a very large number of concrete networks: 32562 networks for Adult, 31649 networks for Bank and 21001 networks for Credit. Namely, the number of parameters that `Sphynx` reasons about is the number of parameters of all $|\mathcal{D}| + 1$ concrete networks. Every experiment is repeated 100 times, for every network, where each experiment randomly chooses dataset entries and trains their respective networks.

Performance of `Sphynx` We begin by evaluating `Sphynx`'s ability to verify LDCP. We consider three kinds of neighborhoods, each is defined given an input x, and the goal is to prove that all inputs in the neighborhood are classified as a label y:

1. *Membership*, $I(x) = \{x\}$: safety to label-only membership attacks [11,35,40].
2. *DP-Robustness*, $I^{\infty}_{\epsilon}(x) = \{x' \mid \|x' - x\|_{\infty} \leq \epsilon\}$, where $\epsilon = 0.05$: safety to adversarial example attacks in a DP setting (similarly to [34,46]).
3. *Sensitivity*, $I^{S}_{i}(x) = \left\{x' \mid \bigwedge_{j \in [d] \setminus \{i\}} x'_j = x_j\right\}$, where the sensitive feature is $i =$*sex* for Adult and Credit and $i =$*age* for Bank: safety to attacks revealing sensitive attributes (like [22,23]). We note that sensitivity is also known as *individual fairness* [30].

For each dataset, we pick 100 inputs for each of these neighborhoods. We compare `Sphynx` to the naive but most accurate algorithm that trains all concrete networks and verifies the neighborhoods' robustness network-by-network. Its verifier is the MILP verifier [63] on which `Sphynx`'s verifier builds. We let both algorithms run on all networks and neighborhoods. Table 1 reports the confusion matrix of `Sphynx` compared to the ground truth (computed by the naive algorithm):

- True Positive (TP): the number of neighborhoods that are LDCP and that `Sphynx` returns they are LDCP.
- True Negative (TN): the number of neighborhoods that are not LDCP and `Sphynx` returns they are not LDCP.
- False Positive (FP): the number of neighborhoods that are not LDCP and `Sphynx` returns they are LDCP. A false positive may happen because of the probabilistic abstraction which may miss concrete networks that are not locally robust at the given neighborhood.
- False Negative (FN): the number of neighborhoods that are LDCP but `Sphynx` returns they are not LDCP. A false negative may happen because the hyper-network may abstract spurious networks that are not locally robust at the given neighborhood (an over-approximation error).

Table 1: Sphynx's confusion matrix.

Dataset	Network	Membership				DP-Robustness				Sensitivity			
		TP	TN	FP	FN	TP	TN	FP	FN	TP	TN	FP	FN
Adult	2×50	93	0	0	7	75	21	0	4	85	10	0	5
	2×100	82	1	0	17	54	39	0	7	75	10	0	15
	4×50	93	0	0	7	11	86	3	0	1	97	1	1
Bank	2×50	100	0	0	0	100	0	0	0	100	0	0	0
	2×100	99	0	0	1	98	1	0	1	99	0	0	1
	4×50	81	0	0	19	22	62	9	7	2	71	8	19
Credit	2×50	93	0	0	7	91	0	0	9	92	0	0	8
	2×100	100	0	0	0	91	2	2	5	100	0	0	0
	4×50	91	0	0	9	2	95	3	0	0	96	4	0

Results show that Sphynx's average accuracy is 93.3% (TP+TN). The FP rate is 1.1% and at most 9%. The FN rate (i.e., the over-approximation error) is 5.5%. Results further show how private the different networks are. All networks are very safe to label-only membership attacks. Although Sphynx has several false negative results, it still allows the user to infer that the networks are very safe to such attack. For DP-Robustness, results show that some networks are fairly robust (e.g., Bank 2×50 and 2×100), while others are not (e.g., Bank 4×50). For Sensitivity, results show that Sphynx enables the user to infer what networks are sensitive to the sex/age attribute (e.g., Credit 4×50) and what networks are not (e.g., Credit 2×100). An important characteristic of Sphynx's accuracy is that the false positive and false negative rates do not lead to inferring a wrong conclusion. For example, if the network is DP-robust, Sphynx proves LDCP (TP+FP) for significantly more DP-robustness neighborhoods than the number of DP-robustness neighborhoods for which it does not prove LDCP (TN+FN). Similarly, if the network is not DP-robust, Sphynx determines for significantly more DP-robustness neighborhoods that they are not LDCP (TN+FN) than the number of DP-robustness neighborhoods that it proves they are LDCP (TP+FP).

Table 2 compares the execution time of Sphynx to the naive algorithm. It shows the number of trained networks (which is the size of the dataset plus one for the naive algorithm, and $K = |nets|$ for Sphynx), the overall training time on a single GPU in hours and the verification time of a single neighborhood (in hours for the naive algorithm, and in *seconds* for Sphynx). Results show the two strengths of Sphynx: (1) it reduces the training time by 13.6x, because it requires to train only 7% of the networks and (2) it reduces the verification time by $1.7 \cdot 10^4$x. Namely, Sphynx reduces the execution time by four orders of magnitude compared to the naive algorithm. The cost is the minor decrease in Sphynx's accuracy (<7%). That is, Sphynx trades off precision with scalability, like many local robustness verifiers do [49,2,70,71,7,25,58,55,57,42].

Table 2: Training and verification time of `Sphynx` and the naive algorithm.

Dataset	Network	Trained networks		GPU training time		Verification time	
		naive	`Sphynx`	naive	`Sphynx`	naive	`Sphynx`
		$\lvert\mathcal{D}\rvert+1$	K	hours	hours	hours	seconds
Adult	2×50	32562	2436	44.64	3.33	2.73	0.35
	2×100	32562	2024	46.56	2.89	5.24	0.72
	4×50	32562	1464	52.37	2.35	0.33	0.87
Bank	2×50	31649	2364	41.04	3.06	2.73	0.35
	2×100	31649	2536	41.76	3.34	5.60	0.69
	4×50	31649	1996	49.41	3.11	1.3	1.19
Credit	2×50	21001	2724	18.72	2.42	2.1	0.35
	2×100	21001	2234	19.21	2.04	3.6	0.64
	4×50	21001	1816	22.67	1.96	0.08	0.75

Table 3: `PredInt` vs. the interval abstraction and several variants.

	Int. Abs.	-Transform	-Normalize	-KDE	`PredInt`
Weight abstraction rate	19.60	55.25	22.89	48.10	70.61
Miscoverage	7.20	5.42	1.5×10^{-6}	2.13	2.49
Overcoverage	1	1.12	299539	1.62	2.80

Ablation study We next study the importance of `Sphynx`'s steps in predicting an interval hyper-network. Recall that `PredInt` predicts an interval for a given network parameter by running a normalization and the Yeo-Johnson transformation and transforming it into a distribution given by KDE. We compare `PredInt` to interval abstraction, mapping a set of values into the interval defined by their minimum and maximum, and to three variants of `PredInt`: (1) -*Transform*: does not use the normalization or the transformation and directly estimates the density with KDE, (2) -*Normalize*: does not use the normalization but uses the transformation, (3) -*KDE*: transforms into a Gaussian distribution (as common), and thus employs a normalization based on the L_2 norm. We run all approaches on the 2×100 network, trained for Adult. Table 3 reports the following:

- Weight abstraction rate: the average percentage of weights whose interval provides a (sound) abstraction. Note that this metric is not related to Lemma 1, guaranteeing that the value of a network parameter of a *single* network is inside its predicted interval with probability $1-\alpha'$. This metric is more challenging: it measures how many values of a given network parameter, over $|\mathcal{D}|+1$ networks, are inside the corresponding predicted interval.
- Miscoverage: measures how much the predicted intervals need to expand to be an abstraction. It is the average over all intervals' miscoverage. The interval

Table 4: `PredHyperNet` vs. the interval abstraction variant (using K networks).

Dataset	Network	Network abstraction rate		Overcoverage	
		Int. Abs.	`Sphynx`	Int. Abs.	`Sphynx`
Adult	2×50	77.04	94.55	1.00	4.52
	2×100	55.20	92.82	1.00	2.80
	4×50	58.24	92.22	1.00	2.83
Bank	2×50	80.06	93.89	1.00	2.54
	2×100	73.21	90.13	1.00	3.21
	4×50	74.80	90.47	1.00	1.93
Credit	2×50	84.07	95.98	1.00	15.07
	2×100	67.16	90.59	1.00	3.68
	4×50	73.14	93.19	1.00	3.53

miscoverage is the ratio of the size of the difference between the optimal interval and the predicted interval and the size of the predicted interval.
- Overcoverage: measures how much wider than necessary the predicted intervals are. It is the geometric average over all intervals' overcoverage. The interval overcoverage is the ratio of the size of the join of the predicted interval and the optimal interval and the size of the optimal interval.

Results show that the weight abstraction rate of the interval abstraction is very low and `PredInt` has a 3.6x higher rate. Results further show that `PredInt` obtains a very low miscoverage, by less than 2.9x compared to the interval abstraction. As expected, these results come with a cost: a higher overcoverage. An exception is the interval abstraction which, by definition, does not have an overcoverage. Results further show that the combination of normalization, transformation, and KDE improve the weight abstraction rate of `PredInt`. Next, we study how well `PredHyperNet`'s hyper-networks abstract an (unseen) concrete network with a probability ≥ 0.9. We compare to a variant that replaces `PredInt` by the interval abstraction, computed using the K concrete networks reported in Table 2. Table 4 reports the network abstraction rate (the average percentage of concrete networks abstracted by the hyper-network) and overcoverage. Results show that `PredHyperNet` obtains a very high network abstraction rate and always above $1 - \alpha = 0.9$. In contrast, the variant obtains lower network abstraction rate with a very large variance. As before, the cost is the over-approximation error.

7 Related Work

Network abstraction Our key idea is to abstract a set of concrete networks (seen and unseen) into an interval hyper-network. Several works rely on network abstraction to expedite verification of a single network. The goal of the abstraction is to generate a smaller network, which can be analyzed faster, and that preserves

soundness with respect to the original network. One work proposes an abstraction-refinement approach that abstracts a network by splitting neurons into four types and then merging neurons of the same type [20]. Another work merges neurons similarly but chooses the neurons to merge by clustering [3]. Other works abstract a neural network by abstracting its network parameters with intervals [47] or other abstract domains [60]. In contrast, `Sphynx` abstracts a large set of networks, seen and unseen, and proves robustness for all of them.

Robustness verification `Sphynx` extends a MILP verifier [63] to verify local robustness given a hyper-network. There are many local robustness verifiers. Existing verifiers leverage various techniques, e.g., over-approximation [49,2,70], linear relaxation [71,7,25,58,55,57,42], simplex [31,32,20], mixed-integer linear programming [63,33,59], and duality [14,50]. A different line of works verifies robustness to small perturbations to the network parameters [72,66]. These works assume the parameters' perturbations are confined in a small L_∞ ϵ-ball and compute a lower and upper bounds on the network parameters by linearly bounding their computations. In contrast, our network parameters are not confined in an ϵ-ball, and our analysis is complete if the inputs are (or processed to be) non-negative.

Adaptive data analysis `PredHyperNet` relies on an iterative algorithm to predict the minimum and maximum of every network parameter. Adaptive data analysis deals with estimating statistics based on data that is obtained iteratively. Existing works focus on statistical queries computing the expectation of functions [17,29,15,16,21,52,68] or low-sensitivity queries [5].

8 Conclusion

We propose a privacy property for neural networks, called local differential classification privacy (LDCP), extending local robustness to the setting of differential privacy for black-box classifiers. We then present `Sphynx`, a verifier for determining whether a network is LDCP at a given neighborhood. Instead of training all networks and verifying local robustness network-by-network, `Sphynx` predicts an interval hyper-network, providing an abstraction with a high probability, from a small number of networks. To predict the intervals, `Sphynx` transforms the observed parameter values into a distribution given by KDE, using the Yeo-Johnson transformation. `Sphynx` then verifies LDCP at a neighborhood directly on the hyper-network, by extending a local robustness MILP verifier. To mitigate an over-approximation error, we rely on preprocessing to the network's inputs or on a lower bound for them. We evaluate `Sphynx` on data-sensitive datasets and show that by training only 7% of the networks, `Sphynx` predicts a hyper-network abstracting any concrete network with a probability of at least 0.9, obtaining 93% verification accuracy and reducing the verification time by $1.7 \cdot 10^4$x.

Acknowledgements We thank the anonymous reviewers for their feedback. This research was supported by the Israel Science Foundation (grant No. 2605/20).

References

1. Abadi, M., Chu, A., Goodfellow, I.J., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep learning with differential privacy. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. ACM (2016)
2. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri, S.: Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In: McKinley, K.S., Fisher, K. (eds.) Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI (2019)
3. Ashok, P., Hashemi, V., Kretínský, J., Mohr, S.: Deepabstract: Neural network abstraction for accelerating verification. In: Automated Technology for Verification and Analysis - 18th International Symposium, ATVA. Lecture Notes in Computer Science, Springer (2020)
4. Baharlouei, S., Nouiehed, M., Beirami, A., Razaviyayn, M.: Rényi fair inference. In: 8th International Conference on Learning Representations, ICLR. OpenReview.net (2020)
5. Bassily, R., Nissim, K., Smith, A.D., Steinke, T., Stemmer, U., Ullman, J.R.: Algorithmic stability for adaptive data analysis. In: Wichs, D., Mansour, Y. (eds.) Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC. ACM (2016)
6. Bastani, O., Zhang, X., Solar-Lezama, A.: Probabilistic verification of fairness properties via concentration. Proc. ACM Program. Lang. **3**(OOPSLA) (2019)
7. Boopathy, A., Weng, T., Chen, P., Liu, S., Daniel, L.: Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI. AAAI Press (2019)
8. Bu, Z., Dong, J., Long, Q., Su, W.J.: Deep learning with gaussian differential privacy. CoRR **abs/1911.11607** (2019)
9. Cao, Y., Yang, J.: Towards making systems forget with machine unlearning. In: IEEE Symposium on Security and Privacy, SP. IEEE Computer Society (2015)
10. Chamikara, M.A.P., Bertók, P., Khalil, I., Liu, D., Camtepe, S., Atiquzzaman, M.: Local differential privacy for deep learning. IEEE Internet Things J. (2020)
11. Choquette-Choo, C.A., Tramèr, F., Carlini, N., Papernot, N.: Label-only membership inference attacks. In: Meila, M., Zhang, T. (eds.) Proceedings of the 38th International Conference on Machine Learning, ICML. Proceedings of Machine Learning Research, PMLR (2021)
12. Drews, S., Albarghouthi, A., D'Antoni, L.: Proving data-poisoning robustness in decision trees. In: Donaldson, A.F., Torlak, E. (eds.) Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI. ACM (2020)
13. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
14. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli, P.: A dual approach to scalable verification of deep networks. In: Globerson, A., Silva, R. (eds.) Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI. AUAI Press (2018)

15. Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., Roth, A.: Generalization in adaptive data analysis and holdout reuse. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems (2015)
16. Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., Roth, A.: The reusable holdout: Preserving validity in adaptive data analysis. Science (2015)
17. Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., Roth, A.L.: Preserving statistical validity in adaptive data analysis. In: Servedio, R.A., Rubinfeld, R. (eds.) Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC. ACM (2015)
18. Dwork, C., Roth, A.: The algorithmic foundations of differential privacy. Found. Trends Theor. Comput. Sci. **9**(3-4) (2014)
19. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Kumar, K.N. (eds.) Automated Technology for Verification and Analysis - 15th International Symposium, ATVA, Proceedings. Lecture Notes in Computer Science, Springer (2017)
20. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: Computer Aided Verification - 32nd International Conference, CAV, Proceedings, Part I. Lecture Notes in Computer Science, Springer (2020)
21. Feldman, V., Steinke, T.: Generalization for adaptively-chosen estimators via stable median. In: Kale, S., Shamir, O. (eds.) Proceedings of the 30th Conference on Learning Theory, COLT. Proceedings of Machine Learning Research, PMLR (2017)
22. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Ray, I., Li, N., Kruegel, C. (eds.) Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM (2015)
23. Fredrikson, M., Lantz, E., Jha, S., Lin, S.M., Page, D., Ristenpart, T.: Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In: Fu, K., Jung, J. (eds.) Proceedings of the 23rd USENIX Security Symposium. USENIX Association (2014)
24. Ganju, K., Wang, Q., Yang, W., Gunter, C.A., Borisov, N.: Property inference attacks on fully connected neural networks using permutation invariant representations. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS. ACM (2018)
25. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: IEEE Symposium on Security and Privacy, SP. IEEE Computer Society (2018)
26. Goel, S., Prabhu, A., Kumaraguru, P.: Evaluating inexact unlearning requires revisiting forgetting. CoRR **abs/2201.06640** (2022)
27. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR, Conference Track Proceedings (2015)
28. Ha, T., Dang, T.K., Dang, T.T., Truong, T.A., Nguyen, M.T.: Differential privacy in deep learning: An overview. In: Lê, L., Dang, T.K., Minh, Q.T., Toulouse, M., Draheim, D., Küng, J. (eds.) International Conference on Advanced Computing and Applications, ACOMP. IEEE Computer Society (2019)

29. Hardt, M., Ullman, J.R.: Preventing false discovery in interactive data analysis is hard. In: 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS. IEEE Computer Society (2014)
30. John, P.G., Vijaykeerthy, D., Saha, D.: Verifying individual fairness in machine learning models. In: Adams, R.P., Gogate, V. (eds.) Proceedings of the Thirty-Sixth Conference on Uncertainty in Artificial Intelligence, UAI. Proceedings of Machine Learning Research, AUAI Press (2020)
31. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kuncak, V. (eds.) Computer Aided Verification - 29th International Conference, CAV, Proceedings, Part I. Lecture Notes in Computer Science, Springer (2017)
32. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.W.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV, Proceedings, Part I. Lecture Notes in Computer Science, Springer (2019)
33. Lazarus, C., Kochenderfer, M.J.: A mixed integer programming approach for verifying properties of binarized neural networks. In: Espinoza, H., McDermid, J.A., Huang, X., Castillo-Effen, M., Chen, X.C., Hernández-Orallo, J., hÉigeartaigh, S.Ó., Mallah, R., Pedroza, G. (eds.) Proceedings of the Workshop on Artificial Intelligence Safety co-located with the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI). CEUR Workshop Proceedings, CEUR-WS.org (2021)
34. Lécuyer, M., Atlidakis, V., Geambasu, R., Hsu, D., Jana, S.: Certified robustness to adversarial examples with differential privacy. In: 2019 IEEE Symposium on Security and Privacy, SP. IEEE (2019)
35. Li, Z., Zhang, Y.: Membership leakage in label-only exposures. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS: ACM SIGSAC Conference on Computer and Communications Security. ACM (2021)
36. Liu, Y., Ma, Z., Liu, X., Ma, J.: Learn to forget: Machine unlearning via neuron masking. IEEE Transactions on Dependable and Secure Computing (2022)
37. Long, Y., Bindschaedler, V., Wang, L., Bu, D., Wang, X., Tang, H., Gunter, C.A., Chen, K.: Understanding membership inferences on well-generalized learning models. CoRR **abs/1802.04889** (2018)
38. Mazzucato, D., Urban, C.: Reduced products of abstract domains for fairness certification of neural networks. In: Dragoi, C., Mukherjee, S., Namjoshi, K.S. (eds.) Static Analysis - 28th International Symposium, SAS. Lecture Notes in Computer Science, Springer (2021)
39. Meyer, A.P., Albarghouthi, A., D'Antoni, L.: Certifying robustness to programmable data bias in decision trees. In: Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems NeurIPS (2021)
40. Monreale, A., Naretto, F., Rizzo, S.: Agnostic label-only membership inference attack. In: Li, S., Manulis, M., Miyaji, A. (eds.) Network and System Security. Springer Nature Switzerland (2023)
41. Moro, S., Cortez, P., Rita, P.: A data-driven approach to predict the success of bank telemarketing. Decision Support Systems **62** (2014)
42. Müller, C., Serre, F., Singh, G., Püschel, M., Vechev, M.T.: Scaling polyhedral neural network verification on gpus. In: Smola, A., Dimakis, A., Stoica, I. (eds.) Proceedings of Machine Learning and Systems, MLSys. mlsys.org (2021)

43. Nasr, M., Shokri, R., Houmansadr, A.: Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In: IEEE Symposium on Security and Privacy, SP. IEEE (2019)
44. Nasr, M., Shokri, R., Houmansadr, A.: Improving deep learning with differential privacy using gradient encoding and denoising. CoRR **abs/2007.11524** (2020)
45. Pham, L.H., Sun, J.: Verifying neural networks against backdoor attacks. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV, Proceedings, Part I. Lecture Notes in Computer Science, Springer (2022)
46. Phan, N., Thai, M.T., Hu, H., Jin, R., Sun, T., Dou, D.: Scalable differential privacy with certified robustness in adversarial learning. In: Proceedings of the 37th International Conference on Machine Learning, ICML. Proceedings of Machine Learning Research, vol. 119. PMLR (2020)
47. Prabhakar, P., Afzal, Z.R.: Abstraction based output range analysis for neural networks. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems NeurIPS (2019)
48. Pulina, L., Tacchella, A.: Challenging SMT solvers to verify neural networks. AI Commun. **25**(2) (2012)
49. Qin, C., Dvijotham, K.D., O'Donoghue, B., Bunel, R., Stanforth, R., Gowal, S., Uesato, J., Swirszcz, G., Kohli, P.: Verification of non-linear specifications for neural networks. In: 7th International Conference on Learning Representations, ICLR. OpenReview.net (2019)
50. Raghunathan, A., Steinhardt, J., Liang, P.: Certified defenses against adversarial examples. In: 6th International Conference on Learning Representations, ICLR, Conference Track Proceedings. OpenReview.net (2018)
51. Rodríguez, P., Bautista, M.Á., Gonzàlez, J., Escalera, S.: Beyond one-hot encoding: Lower dimensional target embedding. Image Vis. Comput. **75** (2018)
52. Rogers, R., Roth, A., Smith, A.D., Srebro, N., Thakkar, O., Woodworth, B.E.: Guaranteed validity for empirical approaches to adaptive data analysis. In: Chiappa, S., Calandra, R. (eds.) The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS. Proceedings of Machine Learning Research, PMLR (2020)
53. Ruoss, A., Balunovic, M., Fischer, M., Vechev, M.T.: Learning certified individually fair representations. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems, NeurIPS (2020)
54. Ryou, W., Chen, J., Balunovic, M., Singh, G., Dan, A.M., Vechev, M.T.: Scalable polyhedral verification of recurrent neural networks. In: Silva, A., Leino, K.R.M. (eds.) Computer Aided Verification - 33rd International Conference, CAV, Proceedings, Part I. Lecture Notes in Computer Science, Springer (2021)
55. Salman, H., Yang, G., Zhang, H., Hsieh, C., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS (2019)
56. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: IEEE Symposium on Security and Privacy, SP. IEEE Computer Society (2017)
57. Singh, G., Ganvir, R., Püschel, M., Vechev, M.T.: Beyond the single neuron convex barrier for neural network certification. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural

Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS (2019)
58. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. POPL (2019)
59. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: Boosting robustness certification of neural networks. In: 7th International Conference on Learning Representations, ICLR. OpenReview.net (2019)
60. Sotoudeh, M., Thakur, A.V.: Abstract neural networks. In: Pichardie, D., Sighireanu, M. (eds.) Static Analysis - 27th International Symposium, SAS. Lecture Notes in Computer Science, vol. 12389. Springer (2020)
61. Sun, Y., Usman, M., Gopinath, D., Pasareanu, C.S.: VPN: verification of poisoning in neural networks. In: Isac, O., Ivanov, R., Katz, G., Narodytska, N., Nenzi, L. (eds.) Software Verification and Formal Methods for ML-Enabled Autonomous Systems - 5th International Workshop, FoMLAS 2022, and 15th International Workshop, NSV. Lecture Notes in Computer Science, vol. 13466. Springer (2022)
62. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: Bengio, Y., LeCun, Y. (eds.) 2nd International Conference on Learning Representations, ICLR, Conference Track Proceedings (2014)
63. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: 7th International Conference on Learning Representations, ICLR. OpenReview.net (2019)
64. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 16. USENIX Association (2016)
65. Tran, H., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using imagestars. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV, Proceedings, Part I. Lecture Notes in Computer Science, Springer (2020)
66. Tsai, Y., Hsu, C., Yu, C., Chen, P.: Formalizing generalization and adversarial robustness of neural networks to weight perturbations. In: Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems, NeurIPS (2021)
67. Ugare, S., Singh, G., Misailovic, S.: Proof transfer for fast certification of multiple approximate neural networks. Proc. ACM Program. Lang. **6**(OOPSLA1) (2022)
68. Ullman, J.R., Smith, A.D., Nissim, K., Stemmer, U., Steinke, T.: The limits of post-selection generalization. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS (2018)
69. Urban, C., Christakis, M., Wüstholz, V., Zhang, F.: Perfectly parallel fairness certification of neural networks. Proc. ACM Program. Lang. **4**(OOPSLA) (2019)
70. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS (2018)
71. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C., Kolter, J.Z.: Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network

robustness verification. In: Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems, NeurIPS (2021)
72. Weng, T., Zhao, P., Liu, S., Chen, P., Lin, X., Daniel, L.: Towards certificated model robustness against weight perturbations. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI. AAAI Press (2020)
73. Wu, H., Ozdemir, A., Zeljic, A., Julian, K., Irfan, A., Gopinath, D., Fouladi, S., Katz, G., Pasareanu, C.S., Barrett, C.W.: Parallelization techniques for verifying neural networks. In: Formal Methods in Computer Aided Design, FMCAD. IEEE (2020)
74. Ye, A.: Stop one-hot encoding your categorical variables. (2020), https://medium.com/analytics-vidhya/stop-one-hot-encoding-your-categorical-variables-bbb0fba89809
75. Ye, J., Maddi, A., Murakonda, S.K., Bindschaedler, V., Shokri, R.: Enhanced membership inference attacks against machine learning models. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS. ACM (2022)
76. Yeh, I.C., Lien, C.h.: The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert systems with applications (2009)
77. Yeo, I.K., Johnson, R.A.: A new family of power transformations to improve normality or symmetry. Biometrika (2000)

A Kernel Density Estimation

As described, `PredInt` adapts the Yeo-Johnson transformation to transform an unknown distribution into a distribution given by kernel density estimation (KDE) with a Laplace kernel. We next provide a short background and explain how `PredInt` uses KDE. KDE is a kernel smoothing technique, estimating an unknown distribution using a weighted average of neighboring samples. The weight is determined by a *kernel*, which is a probability density function $g(\cdot)$ with mean 0. KDE is parameterized by a positive *bandwidth* h, determining the scale of the kernel. The density estimation function is: $\hat{f}_y(v) = \frac{1}{Kh}\sum_{i=1}^{K} g\left(\frac{v-y_i}{h}\right)$. The advantages of KDE is that it can capture complex densities, and that it converges to the true density when $K \to \infty$ and $h \to 0$. Many popular kernel functions are bounded, making them unsuitable for our setting, where the goal is to predict the minimum and maximum values. We thus focus on unbounded kernels. The Gaussian kernel is unbounded, however it is challenging for computing the confidence interval since the decaying rate varies. We thus rely on the Laplace kernel $g(x) = \frac{1}{2}e^{-|x|}$, also known as the exponential kernel. It is especially suitable for our setting because it has a heavy tail, resulting in a wider interval, leaving room for unseen outliers. Additionally, computing its confidence interval can be done efficiently. Technically, the confidence interval is: $[l_y, u_y] = \left[F_y^{-1}\left(\frac{\alpha'}{2}\right), F_y^{-1}\left(1-\frac{\alpha'}{2}\right)\right] = \left[-h\log\left(\frac{1}{K}\sum_{i=1}^{K} e^{-\frac{y_i}{h}}\right) - h\cdot\log\left(\frac{1}{\alpha'}\right), h\log\left(\frac{1}{K}\sum_{i=1}^{K} e^{\frac{y_i}{h}}\right) + h\cdot\log\left(\frac{1}{\alpha'}\right)\right]$,

where F is the CDF of the kernel density estimator and y_i-s are the transformed values. Note that this is a non-symmetric interval which is wider than the interval that we would obtain if we assumed the Laplace distribution (without KDE), in which case the interval is symmetric and equal to: $\left[\text{median}(y_1 \ldots, y_K) - h \cdot \log\left(\frac{1}{\alpha'}\right), \text{median}(y_1 \ldots, y_K) + h \cdot \log\left(\frac{1}{\alpha'}\right)\right]$. The bandwidth h is a hyper-parameter, but it cannot be computed using MLE because as h decreases, the likelihood increases, not considering any neighboring samples. Instead, the bandwidth is usually chosen to minimize the error between the real (unknown) distribution and a distribution given by KDE. However, this results in a too small bandwidth leading to a too small interval, which will not abstract well. On the other hand, choosing a too large bandwidth leads to a too large over-approximation error. To balance, `PredInt` sets h to be the centered absolute first moment: $h = \frac{1}{K}\sum_{i=1}^{K} |y_i - \mu_y|$, where $\mu_y = \text{median}\{y_1, \ldots, y_K\}$.

B Proofs

In this section, we provide proofs.

Lemma 1. *Given a parameter w and an error bound α', if the observed values $w_1, \ldots, w_K$ are IID and suffice to predict the correct distribution, and if there exists $\lambda \in [0, 2]$ such that the distribution of the transformed normalized values $y_1, \ldots, y_K$ is similar to a distribution given by KDE with a Laplace kernel (using the bandwidth defined in Appendix A), then `PredInt` computes a confidence interval containing an unseen value w_i, for $i \in \{K+1, \ldots, |\mathcal{D}|+1\}$, with a probability of $1 - \alpha'$.*

Proof. Since the normalization and transformation are bijective functions, for $[l_w, u_w]$ to contain an unseen value w_i, for $i \in \{K+1, \ldots, |\mathcal{D}|+1\}$, with a probability of $1 - \alpha'$, it suffices to show that $[l_y, u_y]$ satisfies this requirement for y_i. For $[l_y, u_y]$ to satisfy this requirement, by the computation of a confidence interval, three conditions have to hold (1) the transformed values $y_1, \ldots, y_K$ have to be IID, (2) they have to be sufficient to predict the correct distribution, and (3) `PredInt` has to have their distribution. Because the normalization and transformation are bijective functions, conditions (1) and (2) stem from the assumption that $w_1, \ldots, w_K$ satisfy conditions (1) and (2). To have the distribution of $y_1, \ldots, y_K$ (condition (3)), `PredInt` requires the Yeo-Johnson transformation to succeed. While there is no guarantee, similarly to [77], this transformation succeeds if its inputs $z_1, \ldots, z_K$ are IID, sufficient to predict the correct distribution and there exists $\lambda \in [0, 2]$ such that the distribution of the transformed normalized values $y_1, \ldots, y_K$ is *similar* to a distribution given by KDE with a Laplace kernel. The inputs $z_1, \ldots, z_K$ are practically IID because $w_1, \ldots, w_K$ are IID and the normalization is a bijective function (the only source of dependence is that given $K-1$ normalized samples, the last normalized sample can be inferred). The inputs are sufficient to predict the correct distribution because we assume $w_1, \ldots, w_K$ are sufficient. Lastly, the requirement about λ is one of the lemma's assumptions.

Theorem 1. *Given a network N, its training set $\mathcal{D}$, its training algorithm $\mathcal{T}$, and an error bound α, if R is close to 0 and M is close to 1, then* `PredHyperNet` *returns a hyper-network abstracting an unseen network with probability $1-\alpha$.*

Proof. We show that the conditions of Lemma 1 hold, for every weight w and for $\alpha' = \frac{\alpha}{|\mathcal{W} \cup \mathcal{B}|}$, and thus by the union bound, it follows that the hyper-network provides an abstraction with a probability of $1-\alpha$. Let w be a network parameter. First, the observed values $w_1, \ldots, w_K$ are IID since each is computed independently by the training algorithm $\mathcal{T}$ when given the training set without a random sample. Second, the number of observed values is sufficient to predict the correct distribution, because of our stopping condition. By its definition, if the stopping condition is true, R is close to 0, and M is close to 1, then all weights' distributions have converged to their expected distribution. Third, the lemma requires that there exists $\lambda \in [0,2]$ such that the distribution of the transformed normalized values $y_1, \ldots, y_K$ is *similar* to a distribution given by KDE with a Laplace kernel (using the bandwidth defined in Appendix A). This holds in practice because, given IID samples that suffice to predict the correct distribution, KDE provides a good estimation for an unknown distribution and constraining $\lambda \in [0,2]$ practically does not affect this estimation.

Theorem 2. *Our extension to the MILP verifier provides a sound analysis. It is also complete if all inputs to the affine computations are non-negative.*

Proof. Consider an affine variable $\hat{x} = W \cdot x + b$. Assume its input x is non-negative. Given the lower and upper bounds on the weights and biases, because x is non-negative and by interval arithmetic, $\hat{x}$ is bounded in $[l_W \cdot x + l_b, u_W \cdot x + u_b]$. Moreover, this interval is tight, since the lower bound is obtained when $W = l_W$ and $b = l_b$ and the upper bound is obtained when $W = u_W$ and $b = u_b$. The ReLU encoding is identical to [63] and thus soundly and precisely captures its computation. Similarly, the neighborhood's encoding and the local robustness check's encoding are identical to [63] and are thus sound and complete. If x may be negative but has a lower bound $l_x \leq x$, then our encoding bounds $\hat{x}$ in $[l_W \cdot x + l_b - (u_W - l_W) \cdot \max(0, -l_x), u_W \cdot x + u_b + (u_W - l_W) \cdot \max(0, -l_x)]$. This bound is sound because we can write $\hat{x} = W \cdot x + b = W \cdot (x - l_x) + W \cdot l_x + b$. Then, we rewrite this expression as follows: $W \cdot (x - l_x) + W \cdot \max\{0, l_x\} - W \cdot \max\{0, -l_x\} + b$. Note that $x - l_x \geq 0$, $\max\{0, l_x\} \geq 0$, and $-\max\{0, -l_x\} \leq 0$. Thus, this expression is upper bounded by: $u_W \cdot (x - l_x) + u_W \cdot \max\{0, l_x\} - l_W \cdot \max\{0, -l_x\} + u_b$. By rearranging it, we obtain the upper bound: $u_W \cdot x + u_b + u_W \cdot (\max\{0, l_x\} - l_x) - l_W \cdot \max\{0, -l_x\} = u_W \cdot x + u_b + (u_W - l_W) \cdot \max\{0, -l_x\}$. Similarly, we obtain the lower bound.