Formal Methods in Computer-Aided Design 2020

Selecting Stable Safe Configurations for Systems Modelled by Neural Networks with ReLU Activation

Franz Brauße
Department of Computer Science
University of Manchester, UK
franz.brausse@manchester.ac.uk

Zurab Khasidashvili
Product Enablement Solutions Group
Intel Israel Development Center
zurab.khasidashvili@intel.com

Konstantin Korovin
Department of Computer Science
University of Manchester, UK
konstantin.korovin@manchester.ac.uk

***Abstract*—Combining machine learning with constraint solving and formal methods is an interesting new direction in research with a wide range of safety critical applications. Our focus in this work is on analyzing Neural Networks with Rectified Linear Activation Function (`NN-ReLU`). The existing, very recent research works in this direction describe multiple approaches to satisfiability checking for constraints on `NN-ReLU` output. Here we extend this line of work in two orthogonal directions: We propose an algorithm for finding configurations of `NN-ReLU` that are (1) safe and (2) stable. We assume that the inputs of the `NN-ReLU` are divided into existentially and universally quantified variables, where the former represent the parameters for configuring the `NN-ReLU` and the latter represent (possibly constrained) free inputs. We are looking for (1) values of the configuration parameters for which the `NN-ReLU` output satisfies a given constraint for any legal values of the input variables (the safety requirement); and (2) such that the entire family of configurations with configuration variable values close to a safe configuration is also safe (the stability requirement). To our knowledge this is the first work that proposes SMT-based algorithms for searching safe and stable configuration parameters for systems modelled using neural networks. We experimentally evaluate our algorithm on `NN-ReLUs` trained on a set of real-life datasets originating from an industrial CAD application at Intel.**

I. Introduction

Neural Networks (`NN`) are widely used in modeling real life systems and processes, including safety critical ones. Formal analysis of `NN` models is therefore becoming increasingly important for exploration, validation and optimisation of complex systems, and for a much wider range of applications. Multiple recent research works have partly addressed this emerging need: they propose satisfiability checking algorithms for the constraints defined by an `NN-ReLU` and by inequality constraints on its inputs and outputs by encoding this problem into Satisfiability Modulo Theories (SMT) [3] or Mixed-Integer Linear Programming (MILP) [28], [4], [16], [6], [9], [17]. In this work, by a constraint we will mean a Boolean combination of inequality constraints on the inputs of a `NN-ReLU` or its outputs. Given an `NN-ReLU` and constraints on inputs most of these algorithms can verify whether output constraints are satisfied and provide a tight over-approximation (guarantee) on the outputs' range if required; the former capability is called *robustness to adversarial examples* in [4] for classification models, and the latter is called *range estimation* in [9] for regression models; and both are studied for extensions of `NN-ReLU` called *Piecewise Linear* `NN` [25] in [6] and [9], respectively.

In this paper we are looking into a related but quite different problem. In many applications we have analog systems which do not have explicit representations and are modelled using `NNs` based on some experimental test data. An important part of such systems are parameters that are usually configured manually to obtain a desired system behaviour. In this paper we propose several algorithms for finding safe, stable and close to optimal parameters for such systems. An atomic building block in our algorithm is the capability of an SMT solver to check that the output satisfies safety constraints for inputs restricted by input constraints. The problem of finding parameter configurations is more general than the safety problem and requires solving problems with quantifier alternations which are notoriously difficult for SMT solvers.

In a nutshell, we assume that the inputs of an `NN-ReLU` are divided into two groups: the *configuration parameters* that are used to configure the system, and the regular inputs to the system for interacting with the environment. Values of the configuration parameters should be fixed before the system starts operating to perform the task it is designed and configured for, in a safe and close to optimal fashion. A *configuration* is then an assignment of values to the configuration parameters. We assume that the `NN-ReLU` output o is a numeric variable that ranges between 0 and 1, and we are looking for assignments to the configuration parameters such that for all legal values of inputs the output satisfies one or more range constraints like $o \geq 0.9$. Such an assignment is then a *safe* assignment and so is the corresponding configuration. In addition, assuming that, say, the high values of the `NN-ReLU` output are considered as a better performance, then *optimisation* in the context of this work would mean finding safe configuration of the `NN-ReLU` for a constraint $o \geq th$ with a threshold value $th > 0.9$, e.g., constraint $o \geq 0.95$. Since we do not aim to always find a maximal possible threshold for which a safe configuration exists, "close to optimal" is used in this work informally.

To avoid any confusion, we remark straightaway that by configuring a NN−ReLU we do not mean configuring the NN−ReLU training parameters themselves to aid a faster convergence of the training or to improve the modeling accuracy. Configuring parameters that control the training procedure of NNs is an important problem that in principle can be approached heuristically with the procedure proposed in this work to search for close to optimal configurations but this application is outside of the scope of this paper. In this paper we consider configuration parameters which are a part of the analog system which is modelled by a NN−ReLU.

We propose to formalise the problem of configuration selection for NN−ReLU modelled in first-order logic or quantified SMT, where the configuration parameters correspond to the existentially quantified variables and the inputs correspond to universally quantified variables. The configuration selection problem somewhat corresponds to the *Effectively Propositional (EPR)* fragment, also called the *Bernays-Schönfinkel-Ramsey* fragment, which consists of first-order formulas with no occurrences of function symbols other than constants, and which when written in prenex normal form have the quantifier prefix $\exists^*\forall^*$. EPR is a decidable fragment of pure first-order logic and very efficient solvers exist [20], and therefore multiple formal verification problems have been encoded into the EPR fragment [27], [18], [14], [15], [19].

In our encoding of the configuration selection problem for NNs we require support for reasoning with linear and non-linear functions: the theory we deal with is quantified linear real arithmetic with ReLU constraints; in addition, for the industrial application that we are dealing with, it is critical for our algorithm to support ordered categorical variables (say integers) and unordered categorical variables. One of the main contributions in this paper is a δ-decision procedure for the relevant fragment of $\exists^*\forall^*$ formulas over these domains, we call *normed GEAR fragment*.

In many real-life applications, for example the ones dealing with analog systems, the value applied to an analog pin, which we usually model as a numeric feature in machine learning and as a real number in the constraint solving world, is not the same as the value sampled by the system. There is always an error, maybe very small, in the value that is applied and in the value that is sampled, and these two errors do not need to add up to 0. Thus when configuring or verifying such a system, it is required to take this error into account. We think that this aspect has been largely neglected in the context of formal methods for quantified formulas, and we will address this problem in this work by considering *stable solutions* for the configuration problem, elaborated upon below.

Neural Networks and the ML models in general do not model the systems with a hundred percent accuracy. In fact, when training a model, it is a bad idea to build a model that exactly matches the output values in the training dataset; this is known in ML literature as *overfitting*, and is considered bad practice because such a model is unlikely to be accurate on the unseen samples (on which the model was not trained). In fact, the data might actually be contradictory in that two completely identical samples might have different labels, because of an error in data collection or because of insufficient precision in the representation of the feature values, thus a function that fully matches the training data might not exist at all. Thus, again, when exploring ML models configuring them for a safe and close to optimal performance of the systems that they model one needs to take into account that safely configuring the model does not mean at all that the modelled system itself is safe.

One way of mitigating this safety gap is to look for safe configurations of the NN−ReLU models that are *stable*. A safe configuration is stable if all configurations sufficiently close to it are also safe for all legal inputs. In other words, a stable satisfying assignment to configuration parameters is a Cartesian product of open or close intervals of configuration parameters within their respective legal ranges such that each assignment within the product is a satisfying assignment. In the industrial application where our research results have been applied, for most configuration parameters the radius is actually as large as 10% of the value of the variable in the configuration. This is because the sampling error from analog equipment can be dependent on the intended value itself. For some other configuration parameters, say representing clock ticks, the radius is defined through an absolute value, independently from the value of that variable in the configuration.

It has been shown recently that NNs and several other classification models are vulnerable to *adversarial examples* [30]. That is, these ML models misclassify examples that are only slightly different from correctly classified examples drawn from the data distribution. This is another reason why building stable safe configurations is important: we want the configuration to remain safe if the values of configuration parameters are perturbed, this being caused by a malicious adversary or the errors in sampling or sensing data from the equipment or environment. The roundoff errors in the software packages used in training NNs and other ML models are yet another source for the discrepancy between the intended models and the ones that we analyze formally. This list can be continued further.

Work [4] defines a *robustness* measure of an NN at an input vector as the maximal Chebyshev distance L_∞ to the nearest adversarial input vector and proposes an algorithm for estimating it. Work [16] defines a NN−ReLU as *δ-locally-robust* at an input vector if there is no adversarial data point within L_∞-distance smaller than δ, and reports that their Reluplex algorithm can verify whether a NN−ReLU is δ-locally-robust at a given input vector or is globally robust. Work [12] proposes an efficient way to generate adversarial data samples for the purpose of improving the accuracy of classification. For NN−ReLUs with a numeric output, a safety constraint applied to the output straightforwardly converts the model into a classifier of two classes SAT and UNSAT. We note that unlike the robustness, our notion of stability is defined with respect to the configuration parameters rather than the free inputs.

To reiterate, our aim in this work is to safely configure

real, complex systems, not NNs; the NNs are used to approximate the original systems. In this context, we would like to emphasize the following: real systems in many cases have multiple functionalities and depend on many variables but not all variables are equally important for all the properties of the outputs. Specifically in the CAD domain, often accurate models can be built using few variables only. See [24] for an example where only 10 and 30 features out of 10 000 available features were enough to build high quality models for a classification and a regression task, respectively, in the *Signal Integrity* domain where the results of this research have been applied. Our algorithms determine safe and stable regions for the NN approximations which are checked against the original system to see whether these safe regions are safe in the original system. We demonstrate that one can use SMT solvers to guide the NN model refinement not only based on spurious counter examples to the safety constraint on the output but also based on safe regions for a current NN approximation; the latter is a new paradigm in model refinement. Related abstraction refinement techniques such as usage of genetic algorithms, Bayesian optimization, or reinforcement learning are only heuristic methods, without any safety guarantees, and are not guided by a constraint solver.

Main contributions:

- The notion of Safe and Stable Configurations for systems represented by NNs and the corresponding SSC problem.
- The reflexively guarded $\exists^*\forall^*$ fragment (GEAR) and its connection to the SSC problem.
- A general satisfiability algorithm for GEAR, called GearSat and its variant GearSat$_\delta$ for the SSC problem.
- Proof that GearSat$_\delta$ is a δ-decision procedure for the SSC problem.
- Demonstration of the applicability of GearSat$_\delta$ to industrial configuration problems modelled using NNs in the CAD domain.

The rest of the paper is organized as follows. We start with preliminaries in Section II. In Section III we define the problem of configuration selection formally and define stable satisfying assignments for configuration parameters. In Section IV we introduce the GEAR fragment of $\exists^*\forall^*$ formulas capturing this problem in a general context and present a sound satisfiability algorithm GearSat. In Section V we introduce GearSat$_\delta$, an adaptation of this algorithm to normed domains as required in our application and prove that GearSat$_\delta$ is a δ-decision procedure for the SSC problem. Experimental results on industrial problems are reported in Section VI. The conclusions appear in Section VII.

II. Preliminaries

We consider systems which have continuous and discrete inputs and outputs. An input domain $\mathbb{D}$ is a Cartesian product of reals $\mathbb{R}$, integers $\mathbb{Z}$ and finite, non-empty sets with elements from $\mathbb{Z}$. Throughout this paper $\|\cdot\|$ denotes a fixed but arbitrary norm on $\mathbb{D}$.

A *real-valued system* defined on $\mathbb{D}$ can be represented as a function $f : \mathbb{D} \to \mathbb{R}$. A *configurable real-valued system* is a system which also has *configuration parameters* $f : \mathbb{D}_{par} \times \mathbb{D}_{in} \to \mathbb{R}$ where $\mathbb{D}_{par}$ is the domain for configuration parameters and $\mathbb{D}_{in}$ is the domain for inputs. We assume $\mathbb{D} \coloneqq \mathbb{D}_{par} \times \mathbb{D}_{in}$ is not empty and $\|\cdot\|$ is a norm on $\mathbb{D}$.

We assume that a system is given as a black-box function which can be evaluated on a collection of inputs and configuration parameters but its explicit representation is generally unknown. Given a finite collection D of data points from $\mathbb{D}$ we can build approximations of f using neural networks by training them on D.

A (feed-forward) neural network $\mathcal{N}$ consists of layers with inputs and outputs [29]. The input to the first layer is the input to $\mathcal{N}$ and the output of the last layer is the output of $\mathcal{N}$. The input of an intermediate layer is the output of the previous layer. Each layer is a composition of an affine transformation of its inputs with a non-linear activation function. One of the most commonly used activation functions is the rectified linear unit (ReLU), which is defined to be identity for all positive inputs and 0 for non-positive inputs. Even with such simple activation function neural networks can approximate all continuous functions [22].

One of the advantages of neural networks with the ReLU activation functions (NN−ReLU) is that they can be represented in a language amenable to SMT solvers. In particular, one can represent NN−ReLU either in the theory of linear arithmetic with conditionals or directly as a specialised decision procedure [16].

In this paper we are not concerned with particulars of representations of NN−ReLUs. We will consider a theory $\mathcal{T}_{\|\cdot\|}$ (in the SMT-LIB sense [3]) that can be used to specify NNs, and include:

- sorts for reals, integers, finite non-empty domains interpreted as subsets of integers, together with
- operations for linear arithmetic with real coefficients and variables of mixed real and integer sorts,
- usual arithmetic comparison operators $\{\geq, >, =\}$,
- the norm $\|\cdot\|$,
- Boolean operators and
- a collection of activation functions AF.

We will assume that there is a decision procedure for the quantifier-free fragment of $\mathcal{T}_{\|\cdot\|}$.

In our experiments (Section VI) we use only ReLU and linear activation functions (i.e., $AF = \{\texttt{ReLU}, \texttt{Lin}\}$) but our approach is applicable to arbitrary activation functions as long as there is a decision procedure for them. We also used Chebyshev norm $\|\cdot\|_\infty$ as this can be expressed using linear constraints. In these cases activation functions and the norm can be covered by standard SMT theories and we can use SMT solvers or mixed integer programming to solve the quantifier-free fragment of $\mathcal{T}_{\|\cdot\|}$. While our focus in this work is on NN models, we remark that the algorithms and decision procedures proposed in this work are also applicable to other ML models, including tree-based models such as random forest and polynomial models.

We will use p, q, x, y possibly with indices to denote variables and boldface $\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{x}$ will denote vectors of variables. When it is not essential we do not specify sorts of the variables. Given an assignment α, we use $[\![\cdot]\!]^\alpha$ to denote the interpretation of variables according to α, which is extended to interpretation of terms and formulas in the standard way.

We assume that NNs are expressible in $\mathcal{T}_{\|\cdot\|}$. In particular, with each neural network $\mathcal{N} : \mathbb{D}_{par} \times \mathbb{D}_{in} \to \mathbb{R}$ we associate a quantifier-free formula $\varphi_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{x}, y)$ in $\mathcal{T}_{\|\cdot\|}$ such that for every assignment α of variables, $\mathcal{N}([\![\boldsymbol{p}]\!]^\alpha, [\![\boldsymbol{x}]\!]^\alpha) = [\![y]\!]^\alpha$ if and only if $[\![\varphi_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{x}, y)]\!]^\alpha$ is true. Note that the special case of classification problems when $\mathcal{N} : \mathbb{D}_{par} \times \mathbb{D}_{in} \to \{0, \ldots, n\}$ is covered by our framework as well.

III. Safe and Stable Configurations

Consider a configurable system $f : \mathbb{D}_{par} \times \mathbb{D}_{in} \to \mathbb{R}$. We distinguish between parameters and inputs in order to be clear about their quantification, which is existential and universal, respectively. Let f be modelled by a neural network $\mathcal{N}$. Let $\varphi_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{x}, y)$ be a formula in $\mathcal{T}_{\|\cdot\|}$ defining $\mathcal{N}$ as described in Section II. A specification for the system is a formula $\varphi_{spec}(\boldsymbol{p}, \boldsymbol{x}, y)$ which includes constraints on paramenters, inputs and output. If the set of parameters is empty then a system is *safe* if the following formula holds:

$$\forall \boldsymbol{x} y(\varphi_{\mathcal{N}}(\boldsymbol{x}, y) \to \varphi_{spec}(\boldsymbol{x}, y)).$$

This notion is similar to the verification problem in [16], [28]. The main problem we consider in this paper is a different one: finding configuration parameters for the system that are safe and stable for all inputs, as defined below.

A *safe solution to the parameter configuration problem (or just a solution for short)* is an assignment α of parameters $\boldsymbol{p}$ such that the following formula holds:

$$\forall \boldsymbol{x} y(\varphi_{\mathcal{N}}([\![\boldsymbol{p}]\!]^\alpha, \boldsymbol{x}, y) \to \varphi_{spec}([\![\boldsymbol{p}]\!]^\alpha, \boldsymbol{x}, y)).$$

Finding solutions to parametrised systems corresponds to checking satisfiability of $\exists^*\forall^*$ formulas:

$$\exists \boldsymbol{p} \forall \boldsymbol{x} y(\varphi_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{x}, y) \to \varphi_{spec}(\boldsymbol{p}, \boldsymbol{x}, y)).$$

This is in contrast to safety properties where the problem can be formulated using just one type of quantifiers $\forall^*$.

Let $r \geq 0$ be a rational constant. A solution α is called *r-stable* if all parameter configurations which are r close to α are also solutions to the specification:

$$\forall \boldsymbol{q}(\|[\![\boldsymbol{p}]\!]^\alpha - \boldsymbol{q}\| \leq r \to \forall \boldsymbol{x} y(\varphi_{\mathcal{N}}(\boldsymbol{q}, \boldsymbol{x}, y) \to \varphi_{spec}(\boldsymbol{q}, \boldsymbol{x}, y))).$$

Similarly to above, finding r-stable solutions corresponds to checking satisfiability of $\exists^*\forall^*$ formulas.

$$\begin{aligned} &\exists \boldsymbol{p} \forall \boldsymbol{q}(\|\boldsymbol{p} - \boldsymbol{q}\| \leq r \to \\ &\qquad \forall \boldsymbol{x} y(\varphi_{\mathcal{N}}(\boldsymbol{q}, \boldsymbol{x}, y) \to \varphi_{spec}(\boldsymbol{q}, \boldsymbol{x}, y))). \end{aligned} \tag{1}$$

We call this the Safe and Stable Configuration problem (SSC).

Let us note that the stability condition connects existentially quantified parameters $\boldsymbol{p}$ with introduced universally quantified variables $\boldsymbol{q}$. In this case even when there are no inputs and only parameters the formula involves $\exists^*\forall^*$quantifier alternation (see also Remark 1).

In order to solve the SSC problem we first introduce a general GEAR fragment and a satisfiability algorithm for GEAR called GearSat. GearSat does not rely on properties of $\mathcal{T}_{\|\cdot\|}$ and is applicable to any theory without uninterpreted symbols and decidable quantifier-free fragment. Then we show that SSC can be expressed in a special fragment of GEAR called *normed GEAR*, for which we modify GearSat into a δ-complete decision procedure GearSat$_\delta$.

IV. The Reflexively guarded $\exists^*\forall^*$ fragment and GearSat

Algorithm 1 (EA-SAT-Basic) Solve $\exists \boldsymbol{p} \forall \boldsymbol{x} \varphi(\boldsymbol{p}, \boldsymbol{x})$ using a solver for the existential fragment.

procedure EA-Sat-Basic(φ)
  loop
    if $\varphi(\boldsymbol{p}, \boldsymbol{x})$ is unsat **then**
      return unsat
    end if
    $\alpha \leftarrow$ assignment of $\boldsymbol{p}, \boldsymbol{x}$ satisfying φ
    if $\neg\varphi([\![\boldsymbol{p}]\!]^\alpha, \boldsymbol{x})$ is unsat **then**
      return α restricted to $\boldsymbol{p}$
    end if
    $\varphi(\boldsymbol{p}', \boldsymbol{x}) \leftarrow \varphi(\boldsymbol{p}', \boldsymbol{x}) \wedge (\boldsymbol{p}' \neq [\![\boldsymbol{p}]\!]^\alpha)$ ▷ learn lemma
  end loop
end procedure

We start with a general Algorithm 1 for solving $\exists^*\forall^*$ formulas which is inspired by model-based quantifier instantiation procedures [11] and only requires a solver for the existential fragment.

Theorem 1. *Algorithm 1 is sound.*

The theorem follows from the observation that Algorithm 1 only generates lemmas which are implied by its input formula φ, making it a sound procedure.

The downside of Algorithm 1 is that it generates very weak lemmas excluding point-wise counter-examples. In particular, for infinite domains, Algorithm 1 does not terminate in general. To mitigate this, we propose a novel procedure (Algorithm 2) which generates more general lemmas which exclude large regions from the search space and facilitate termination. This procedure assumes that the quantifiers in $\exists^*\forall^*$ formulas are guarded as defined below.

Definition 1. A closed first-order $\exists^*\forall^*$ formula ξ of the form

$$\xi \equiv \exists \boldsymbol{p}[\eta(\boldsymbol{p}) \wedge \forall \boldsymbol{q}(\theta(\boldsymbol{p}, \boldsymbol{q}) \to \forall \boldsymbol{x}(\psi(\boldsymbol{q}, \boldsymbol{x})))] \tag{2}$$

is in the *reflexively guarded* $\exists^*\forall^*$ fragment, GEAR for short, iff η, θ and ψ are quantifier-free formulas and θ defines a reflexive relation. We say that $\boldsymbol{p}$ is guarded by $\eta(\boldsymbol{p})$ and $\boldsymbol{q}$ is guarded by $\theta(\boldsymbol{p}, \boldsymbol{q})$.

Algorithm 2 (GEARSAT) Solve reflexively guarded $\exists^*\forall^*$ formulas based on a solver for the existential fragment.

procedure GEARSAT(η, θ, ψ)
  $E(\boldsymbol{p}) \leftarrow \eta(\boldsymbol{p})$
  loop
    if $\psi(\boldsymbol{p}, \boldsymbol{x}) \wedge E(\boldsymbol{p})$ is unsat **then**
      return unsat
    end if
    $\alpha \leftarrow$ assignment of $\boldsymbol{p}, \boldsymbol{x}$ satisfying $\psi(\boldsymbol{p}, \boldsymbol{x}) \wedge E(\boldsymbol{p})$
    $\varphi(\boldsymbol{q}, \boldsymbol{x}) \leftarrow (\theta([\![\boldsymbol{p}]\!]^\alpha, \boldsymbol{q}) \rightarrow \psi(\boldsymbol{q}, \boldsymbol{x}))$
    if $\neg\varphi(\boldsymbol{x}, \boldsymbol{q})$ is unsat **then**
      return α restricted to $\boldsymbol{p}$
    end if
    $\beta \leftarrow$ assignment of $\boldsymbol{q}, \boldsymbol{x}$ satisfying $\neg\varphi(\boldsymbol{q}, \boldsymbol{x})$
    $E(\boldsymbol{p}) \leftarrow E(\boldsymbol{p}) \wedge \neg\theta(\boldsymbol{p}, [\![\boldsymbol{q}]\!]^\beta)$ ▷ learn guard lemma
  end loop
end procedure

In order to prove soundness of Algorithm 2, we require the guard θ to define a reflexive relation, i.e., $\forall\boldsymbol{q}(\theta(\boldsymbol{q}, \boldsymbol{q}))$ holds true. This can be motivated by the observation that θ connects the existentially and (a subset of the) universally quantified variables. In our application θ usually takes the form $\|\boldsymbol{p}-\boldsymbol{q}\| \leq r$, which is trivially reflexive, however, no properties besides reflexivity are required for soundness. We do not impose any restrictions on the guard η, it can be used to constrain the range of configuration parameters.

Theorem 2. *The Algorithm* GEARSAT *is sound for the GEAR fragment.*

Proof: Let ξ be a formula in the GEAR fragment of the form (2). Assume GEARSAT(η, θ, ψ) performs $N \geq 0$ iterations and terminates in iteration $N+1$ with result κ. By the construction in Algorithm 2, for each $n = 1, \ldots, N$ there are assignments α_n and β_n satisfying $A_n \coloneqq \eta(\boldsymbol{p}) \wedge \bigwedge_{i=1}^{n-1} \neg\theta(\boldsymbol{p}, [\![\boldsymbol{q}]\!]^{\beta_i}) \wedge \psi(\boldsymbol{p}, \boldsymbol{x})$ and $B_n \coloneqq \theta([\![\boldsymbol{p}]\!]^{\alpha_n}, \boldsymbol{q}) \wedge \neg\psi(\boldsymbol{q}, \boldsymbol{x})$, respectively. We show the cases for κ separately.

Consider $\kappa =$ unsat. If $N = 0$, then by construction $A_1 \equiv \eta(\boldsymbol{p}) \wedge \forall\boldsymbol{x}(\psi(\boldsymbol{p}, \boldsymbol{x}))$ is unsat. $\eta(\boldsymbol{p})$ is implied by ξ as is $\forall\boldsymbol{q}(\theta(\boldsymbol{p}, \boldsymbol{q}) \rightarrow \forall\boldsymbol{x}(\psi(\boldsymbol{q}, \boldsymbol{x})))$. Since θ defines a reflexive relation, ξ also implies $\forall\boldsymbol{x}(\psi(\boldsymbol{p}, \boldsymbol{x}))$. Therefore ξ is unsat. Otherwise $N > 0$, then $A_{N+1} \coloneqq A_N \wedge \neg\theta(\boldsymbol{p}, [\![\boldsymbol{q}]\!]^{\beta_N})$ is unsat. In order to derive a contradiction, assume there is $\boldsymbol{p}^*$ such that $\eta(\boldsymbol{p}^*) \wedge \forall\boldsymbol{q}(\theta(\boldsymbol{p}^*, \boldsymbol{q}) \rightarrow \forall\boldsymbol{x}(\psi(\boldsymbol{q}, \boldsymbol{x})))$ holds. By property of θ, so does $\theta(\boldsymbol{p}^*, \boldsymbol{p}^*)$, therefore, $\forall\boldsymbol{x}(\psi(\boldsymbol{p}^*, \boldsymbol{x}))$ is true. Since A_{N+1} is unsat and both $\eta(\boldsymbol{p}^*)$ and $\forall\boldsymbol{x}(\psi(\boldsymbol{p}^*, \boldsymbol{x}))$ are true, there is $n \leq N$ such that $\neg\theta(\boldsymbol{p}^*, [\![\boldsymbol{q}]\!]^{\beta_n})$ is false. Consequently, by assumption, $\forall\boldsymbol{x}(\psi([\![\boldsymbol{q}]\!]^{\beta_n}, \boldsymbol{x}))$ holds. However, β_n satisfies B_n and therefore also $\neg\psi(\boldsymbol{q}, \boldsymbol{x})$. A contradiction.

Consider $\kappa \neq$ unsat. Then the assignment α is computed which satisfies A_{N+1} and, in particular, $\eta(\boldsymbol{p})$. Thus, if $\theta([\![\boldsymbol{p}]\!]^\alpha, \boldsymbol{q}) \wedge \neg\psi(\boldsymbol{q}, \boldsymbol{x})$ is unsat, $\eta([\![\boldsymbol{p}]\!]^\alpha) \wedge \forall\boldsymbol{q}\boldsymbol{x}(\theta([\![\boldsymbol{p}]\!]^\alpha, \boldsymbol{q}) \rightarrow \psi(\boldsymbol{q}, \boldsymbol{x}))$ and therefore also $\xi([\![\boldsymbol{p}]\!]^\alpha)$ hold. ∎

Let us note that we can apply GEARSAT to general $\exists^*\forall^*$ formulas $\exists\boldsymbol{p}[\forall\boldsymbol{x}(\psi(\boldsymbol{p}, \boldsymbol{x}))]$ which are not explicitly guarded by first transforming them into a guarded form (2) where θ defines the identity relation and $\eta \equiv$ true. In this case GEARSAT performs the same steps as Algorithm 1.

In contrast to Algorithm 1, however, GEARSAT takes advantage of guards when they define large regions in parameter space. In this case, generated lemmas are negations of partial guard instantiations that exclude large regions from the search space around found counter-examples.

Let us note that GEARSAT terminates whenever the process of generating lemmas $E_n(\boldsymbol{p}) = \eta(\boldsymbol{p}) \wedge \bigwedge_{i=1}^{n} \neg\theta(\boldsymbol{p}, [\![\boldsymbol{q}]\!]^{\beta_i})$ is guaranteed to result in an unsatisfiable lemma after a finite number of steps. Note, that during the run of GEARSAT we generate strictly stronger lemmas, in particular $\exists\boldsymbol{p}(E_i(\boldsymbol{p}) \wedge \neg E_j(\boldsymbol{p}))$ holds for all $i < j$. From this it follows that GEARSAT is a decision procedure for finite domains and more generally for fragments where there are only finitely many non-equivalent lemmas of the form above.

In many applications including ours, solutions to the SSC problem are required to be stable on their (topological) neighbourhood. In this case the reflexive guard also enables *deciding* satisfiability even when the bounded domain itself is infinite. This statement is made precise in Theorem 4 for normed domains such as $\mathbb{D}_{par}$.

V. A δ-DECISION PROCEDURE FOR THE SAFE AND STABLE CONFIGURATION PROBLEM

The GEARSAT algorithm can be employed to find safe and r-stable solutions to the configuration problem as follows. As described in Section III the SSC problem can be represented using formulas of the form (1). Such formulas are a special case of the GEAR formulas (2) where the guard is $\theta_r \equiv \|\boldsymbol{p} - \boldsymbol{q}\| \leq r$, which we call *stability guard*, and $\psi(\boldsymbol{q}, \boldsymbol{x}, y) \equiv \varphi_{\mathcal{N}}(\boldsymbol{q}, \boldsymbol{x}, y) \rightarrow \varphi_{spec}(\boldsymbol{q}, \boldsymbol{x}, y)$. Then, safe and r-stable regions can be searched using Algorithm 2 applied to:

$$\varphi_r = \exists\boldsymbol{p}[\eta(\boldsymbol{p}) \wedge \forall\boldsymbol{q}(\theta_r(\boldsymbol{p}, \boldsymbol{q}) \rightarrow \forall\boldsymbol{x}y(\psi(\boldsymbol{q}, \boldsymbol{x}, y)))] \qquad (3)$$

We will call $\mathcal{T}_{\|\cdot\|}$ formulas of the form (3) *normed GEAR formulas*. They form a special case of reflexively guarded $\exists^*\forall^*$ formulas, since $\|\boldsymbol{p} - \boldsymbol{p}\| = 0$.

In many applications, including ours, we can tolerate to reject solutions if counter-examples are located within some tolerance of the safe region. To this end we introduce the notion of unsatisfiability under δ-*perturbation*, or δ-*unsatisfiability*.

Let θ_r be a stability guard as given above and $\delta > 0$, then the δ-perturbation of θ_r is $\theta_{r+\delta} = \|\boldsymbol{p} - \boldsymbol{q}\| \leq r + \delta$. Similarly, δ-perturbation $\varphi_{r+\delta}$ of φ_r, is obtained by replacing θ_r with $\theta_{r+\delta}$ in φ_r.

We say that φ_r is δ-*unsatisfiable* if δ-perturbation of φ_r is unsatisfiable. The δ-*decision problem for normed GEAR formulas* φ_r is defined as the problem of showing that either φ_r is satisfiable or showing that φ_r is δ-unsatisfiable. This can be seen as an adaptation of the notion of δ-decision from [10] to our setting.

We slightly modify the algorithm GEARSAT to GEARSAT$_\delta(\eta, r, \psi)$ to solve the δ-unsatisfiability problem for

normed GEAR formulas as follows. First, we strengthen the lemmas by δ to:

$$E(\boldsymbol{p}) \leftarrow E(\boldsymbol{p}) \wedge \|\boldsymbol{p} - [\![\boldsymbol{q}]\!]^{\beta}\| > r + \delta$$

Second, "return unsat" is modified to "return δ-unsat".

It is straightforward to generalise Theorem (2) to GEARSAT_δ. Next we show that GEARSAT_δ terminates when we consider bounded domains.

Theorem 3. *Let $\delta > 0$ and $r \geq 0$ be rational numbers. Consider a normed GEAR formula of the form* (3) *where η defines a bounded subset of $\mathbb{D}$. Then $\text{GEARSAT}_\delta(\eta, r, \psi)$ terminates.*

Proof: Towards a contradiction, assume GEARSAT_δ invoked on (η, r, ψ) does not terminate. Then there is no bound on the number of iterations since the individual steps are computable. Using the notations $A_n := \eta(\boldsymbol{p}) \wedge \bigwedge_{i=1}^{n-1} \neg\theta_{r+\delta}(\boldsymbol{p}, [\![\boldsymbol{q}]\!]^{\beta_i}) \wedge \psi(\boldsymbol{p}, \boldsymbol{x})$ and $B_n := \theta_r([\![\boldsymbol{p}]\!]^{\alpha_n}, \boldsymbol{q}) \wedge \neg\psi(\boldsymbol{q}, \boldsymbol{x})$ similar to those in the proof of Theorem 2, let $(\alpha_n)_n$ and $(\beta_n)_n$ be the sequences of assignments satisfying A_n and B_n, respectively, computed by $\text{GEARSAT}_\delta(\eta, r, \psi)$ in iteration $n \in \mathbb{N}$. Let $\boldsymbol{p}_n := [\![\boldsymbol{p}]\!]^{\alpha_n}$, $\boldsymbol{q}_n := [\![\boldsymbol{q}]\!]^{\beta_n}$ be elements of $\mathbb{D}_{par}$ for each $n \in \mathbb{N}$. We first show a lower bound on the distance between candidates $\boldsymbol{p}_n$. Let $k, n \in \mathbb{N}$ with $k < n$. The triangle inequality for $\|\cdot\|$ implies

$$\|\boldsymbol{p}_n - \boldsymbol{p}_k\| \geq \|\boldsymbol{p}_n - \boldsymbol{q}_k\| - \|\boldsymbol{p}_k - \boldsymbol{q}_k\|. \qquad (*)$$

Each assignment β_k satisfies B_k, in particular, $\theta_r(\boldsymbol{p}_k, \boldsymbol{q})$, i.e., $\|\boldsymbol{p}_k - \boldsymbol{q}_k\| \leq r$ holds. Additionally, α_n satisfies A_n, in particular $\neg\theta_{r+\delta}(\boldsymbol{p}, \boldsymbol{q}_k)$ from which we obtain $\|\boldsymbol{p}_n - \boldsymbol{q}_k\| > r + \delta$. These two facts together with $(*)$ imply $\|\boldsymbol{p}_n - \boldsymbol{p}_k\| > (r + \delta) - r = \delta$.

Thus, the pairwise distances between candidates $\boldsymbol{p}_n$ is at least $\delta > 0$. Since η defines a bounded set and is satisfied by α_n for each $n \in \mathbb{N}$, the number of candidates $\boldsymbol{p}_n$ with pairwise distance of at least δ to each other is also bounded by some $N \in \mathbb{N}$. Thus, the set defined by $\eta(\boldsymbol{p})$ is covered by $\{\boldsymbol{z} : \theta_\delta(\boldsymbol{z}, \boldsymbol{p}_n)\} \subseteq \{\boldsymbol{z} : \theta_{r+\delta}(\boldsymbol{z}, \boldsymbol{q}_n)\}$ for $n = 1, \ldots, N$ making A_{N+1} unsatisfiable and therefore GEARSAT_δ returns δ-unsat. A contradiction. ∎

Theorem (2) and Theorem (3) imply the following.

Theorem 4. *The algorithm GEARSAT_δ is a δ-decision procedure for normed GEAR formulas.*

A. Multiple solutions

We can use GEARSAT_δ to enumerate stable solutions in the following way. Maintain a conjunction η of quantifier-free formulas over free variables $\boldsymbol{p}$, initially true. Then, in a loop, first compute $\kappa = \text{GEARSAT}_\delta(\eta, \theta, \psi)$ and record the formula $E(\boldsymbol{p})$ which it constructs internally. Second, replace $\eta(\boldsymbol{p})$ by $E(\boldsymbol{p}) \wedge (\boldsymbol{p} \neq [\![\boldsymbol{p}]\!]^{\kappa})$ and repeat until $\kappa = \text{unsat}$. By Theorem 2, every $\kappa \neq \text{unsat}$ computed by this loop corresponds to a box around $[\![\boldsymbol{p}]\!]^{\kappa}$ of radius r with the property that $\mathcal{N}$ is safe for all parameters $\boldsymbol{p}$ from that box and for all unconstrained inputs $\boldsymbol{x}$. All κ are different, which is ensured by $(\boldsymbol{p} \neq [\![\boldsymbol{p}]\!]^{\kappa})$.

If, for instance, disjoint safe and r-stable regions are sought, this predicate can be adjusted to maintain a concrete distance between solutions which guarantees disjointness, i.e., $\|\boldsymbol{p} - [\![\boldsymbol{p}]\!]^{\kappa}\| > 2r$. Instead of η being empty initially, it can also be used to define a subset of the domain to be searched. This is presented in Algorithm 3. Let us note that in Algorithm 3 the lemmas are shared during the search for different solutions.

Algorithm 3 Enumerates r-stable pairwise disjoint solutions

```
function GEARREGIONS_δ(η, r, ψ)
    R ← ∅
    η_1(p) ← η(p)
    for i = 1, 2, ... do
        κ ← GEARSAT_δ(η_i, r, ψ)
        if κ = δ-unsat then
            break
        end if
        R ← R ∪ {[[p]]^κ}
        E ← GEARSAT_δ.E        ▷ lemmas from GEARSAT_δ
        η_{i+1}(p) ← E(p) ∧ ||p − [[p]]^κ|| > 2r
    end for
    return R
end function
```

B. Optimisation

In our application we want to find safe and stable configurations such that for all inputs the output of the neural network is greater than a specified threshold th. In this case the specification is of the form $\mathcal{N}(\boldsymbol{q}, \boldsymbol{x}) \geq th$ and ψ will be

$$\psi(\boldsymbol{q}, \boldsymbol{x}, y) \equiv (\varphi_{\mathcal{N}}(\boldsymbol{q}, \boldsymbol{x}, y) \rightarrow y \geq th).$$

Moreover we want to find configurations with high or close to optimal values of th.

For this we use GEARSAT_δ to find close to optimal solutions by incrementally increasing threshold th or by performing a binary search for close to optimal th. Similarly, for enumeration of solutions we can reuse lemmas generated by those calls to GEARSAT_δ that return satisfying assignments.

Remark 1. It is possible in GEAR formulas (2) (and (3) as a special case) to encode all universally quantified variables in $\forall\boldsymbol{x}(\psi(\boldsymbol{q}, \boldsymbol{x}))$ (which include all inputs and outputs) as parameters under stability conditions resulting in the following normalised form:

$$\exists\boldsymbol{p}[\eta(\boldsymbol{p}) \wedge \forall\boldsymbol{q}(\theta(\boldsymbol{p}, \boldsymbol{q}) \rightarrow \psi(\boldsymbol{q}))]$$

where $\boldsymbol{p}$ and $\boldsymbol{q}$ are of same shape, as follows. Let ξ, η, θ, ψ be as in Definition 1. For every variable x_i in $\boldsymbol{x}$ introduce an existentially quantified variable y_i and a universally quantified variable z_i. Next, define $\eta'(\boldsymbol{p}, \boldsymbol{y}) \equiv \eta(\boldsymbol{p})$ and $\theta'(\boldsymbol{p}, \boldsymbol{y}, \boldsymbol{q}, \boldsymbol{z}) \equiv \theta(\boldsymbol{p}, \boldsymbol{q})$. Then the formula

$$\exists\boldsymbol{p}\boldsymbol{y}[\eta'(\boldsymbol{p}, \boldsymbol{y}) \wedge \forall\boldsymbol{q}\boldsymbol{z}(\theta'(\boldsymbol{p}, \boldsymbol{y}, \boldsymbol{q}, \boldsymbol{z}) \rightarrow \psi(\boldsymbol{q}, \boldsymbol{z}))$$

has the same solutions as ξ and is in the above normalised form.

This transformation has the effect of eliminating the (universally quantified) input/output variables and replacing them with existentially quantified parameters under stability conditions which require the solution to hold over entire input/output domains. In this way we can uniformly treat inputs as parameters under stability conditions. We adopted this approach in our experiments.

VI. Experimental Evaluation

We evaluated our configuration selection algorithm on 10 training datasets collected in an Electrical Validation Lab at Intel. The output is an analog signal measuring the quality of a transmitter or a receiver of a channel to a peripheral device. Each channel is divided into eight bytes, and we treat each channel as an unordered categorical variable with eight levels. The integer variable in the data models clock ticks.

The datasets are freely available at http://www.cs.man.ac.uk/~korovink/fmcad2020: 5 transmitter (TX) datasets *s2_tx*, *m2_tx*, *h1_tx*, *h1_iter_tx*, *mu_tx* and 5 receiver (RX) counterparts *s2_rx*, *m2_rx*, *h1_rx*, *h1_iter_rx*, *mu_rx*. To avoid IP disclosure, the numeric features including the output are normalized to $[0, 1]$; the integer features are kept intact. We refer to [23], [24] for details on the design of closely related applications dealing with TX/RX/IO systems.

Our aim is to find safe and stable regions where the output o, normalized to $[0, 1]$, satisfies the constraint $o \geq th$ with as high $th \leq 1$ as possible in the grid ranging from 0.7 to 0.95 with an increment of 0.05. In addition, we aim at finding stable safe regions that are reusable across multiple bytes, with as high th as possible. We built `NN-ReLUs` using the `tensorflow` [1] and `Keras` [8] software packages. The different versions of *RX* and *TX* datasets have five to eight input features (not including the channel and bytes parameters). We use `NN-ReLUs` with two internal layers, 14 nodes in the first layer and 7 nodes in the second layer, which is in line with rule-of-thumb guidelines for selecting the number of internal nodes for a given number of inputs. As stability criterion for solutions we employed a radius of 10% of the value around a safe solution for the numerical variables and ± 5 clock ticks for the integer feature. These radii are measured in the Chebyshev-norm $\|\cdot\|_\infty$.

We implemented our GearSat algorithms using `Z3` [26] as a backend for solving the quantifier-free fragment of $\mathcal{T}_{\|\cdot\|}$, which can be encoded in `QF_LIRA` in the SMT-Lib format [3].

Let us first remark that although it is possible to directly encode the SSC problem as a quantified SMT formula without our algorithms, e.g., `Z3`-v4.8.8 fails to solve a single problem despite many state-of-the-art quantifier elimination procedures are integrated in `Z3`; the same holds also for `CVC4`-v1.7 [2] which is another top SMT solver. We included examples of quantified SMT encoding on the website with datasets. In our approach we only resort to quantifier free SMT calls to solve quantified normed GEAR formulas. We believe the main reason our algorithms perform well on these problems is due to strong lemmas that take advantage of the guarded form to exclude large regions from the search space.

RX

C:B	th	safe	lb-ce	lb-time	ub-ce	ub-time
0:0	0.9	100	54	319.05	–	–
0:1	0.85	100	69	700.42	0	54.25
0:2	0.9	29	2867	3034.49	–	–
0:3	0.85	100	251	512.31	0	111.00
0:4	0.9	100	128	830.72	–	–
0:5	0.85	100	82	627.68	0	254.81
0:6	0.85	100	121	680.75	0	123.89
0:7	0.85	41	2620	3409.79	0	102.07
1:0	0.8	100	762	606.90	134	290.40
1:1	0.8	1	188	264.75	0	61.46
1:2	0.9	100	369	700.23	–	–
1:3	0.8	100	3449	1328.61	2056	958.96
1:4	0.85	100	16	381.11	0	73.08
1:5	0.85	35	287	769.73	0	53.59
1:6	0.8	100	1088	980.34	0	68.65
1:7	0.9	100	84	405.49	–	–

TX

C:B	th	safe	lb-ce	lb-time	ub-ce	ub-time
0:0	0.9	100	156	131.59	–	–
0:1	0.9	51	1006	372.19	–	–
0:2	0.9	100	69	114.17	–	–
0:3	0.9	100	120	78.29	–	–
0:4	0.9	100	315	211.75	–	–
0:5	0.9	100	221	135.44	–	–
0:6	0.9	20	110	41.84	–	–
0:7	0.9	100	176	129.64	–	–
1:0	0.9	100	84	89.81	–	–
1:1	0.85	100	467	226.88	0	11.42
1:2	0.9	100	360	128.06	–	–
1:3	0.9	100	169	82.60	–	–
1:4	0.9	100	304	79.81	–	–
1:5	0.85	100	357	205.29	0	20.24
1:6	0.9	100	259	68.61	–	–
1:7	0.9	100	60	60.01	–	–

TABLE I
Benchmarks of finding up to 100 safe and stable regions for data set *s2*.

The results of computing stable safe regions along with their optimal thresholds in the grid of thresholds described above, for the receiver and transmitter datasets *s2_rx*, *s2_tx* and *h1_rx*, *h1_tx*, respectively, are shown in Tables I and III. The results are representative for all the datasets used in our experiments. For each combination of channel and byte values, a maximum number of regions was computed – up to a threshold of 100 regions. During the run, the algorithm generates candidate configuration parameters and checks for counter-examples. The system can be used for both finding safe and stable regions, and for checking that such regions do not exist for a given th. For the cases when there are no safe regions, the algorithm relaxes the safety constraint on the output by lowering the threshold value th.

For each combination of channel-byte pair (C:B), the tables give the best threshold (th), the number of safe regions found by the algorithm (safe), and the number of counter-examples (lb-ce) to the safety which are eliminated during the search, along with the computation time for the lower and upper bound of the threshold, respectively; these bounds are defined in the next paragraph. The stable safe regions in the tables have not only been constructed by Algorithm 2, but also by checking each region against samples from the training dataset: the

C 0	0	1	2	3	4	5	6	7
0	–	77	28	16	70	67	22	38
1	20	–	7	10	14	58	32	12
2	0	0	–	0	3	0	2	0
3	0	3	0	–	0	13	0	1
4	6	5	6	0	–	0	0	11
5	0	7	0	7	0	–	5	16
6	0	12	25	9	15	87	–	37
7	0	1	0	0	7	0	0	–

C 1	0	1	2	3	4	5	6	7
0	–	0	0	15	0	34	35	0
1	0	–	0	0	0	0	0	0
2	8	0	–	0	0	2	1	23
3	0	0	0	–	0	2	5	0
4	7	1	0	39	–	60	86	3
5	0	0	0	1	0	–	1	0
6	5	0	0	13	16	4	–	0
7	39	0	1	17	0	54	5	–

TABLE II

SHARED SAFE AND STABLE REGIONS ACROSS MULTIPLE BYTES PER CHANNEL IN DATA SET *s2_rx*: NUMBER OF REGIONS IN BYTE *column* IS SAFE WITH RESPECT TO BYTE *row*.

RX

C:B	th	safe	lb-ce	lb-time	ub-ce	ub-time
0:0	0.9	100	1584	26376.50	–	–
0:1	0.9	100	230	4780.70	–	–
0:2	0.9	100	723	9084.00	–	–
0:3	0.9	100	933	9596.69	–	–
0:4	0.9	100	195	4924.51	–	–
0:5	0.9	100	206	3511.69	–	–
0:6	0.85	100	525	5213.19	636	6213.53
0:7	0.8	100	44	2854.15	0	434.43
1:0	0.9	100	51	2887.82	–	–
1:1	0.85	100	81	1665.96	3	622.37
1:2	0.85	100	81	2758.16	33	651.10
1:3	0.85	100	149	3228.21	0	741.07
1:4	0.9	100	640	5713.27	–	–
1:5	0.85	100	274	3532.65	0	398.29
1:6	0.9	100	90	2691.84	–	–
1:7	0.9	100	336	4378.93	–	–

TX

C:B	th	safe	lb-ce	lb-time	ub-ce	ub-time
0:0	0.9	100	215	1597.45	–	–
0:1	0.9	100	341	1839.79	–	–
0:2	0.9	100	523	3045.20	–	–
0:3	0.9	100	186	1639.23	–	–
0:4	0.9	100	100	635.99	–	–
0:5	0.9	100	206	1544.68	–	–
0:6	0.9	100	157	873.79	–	–
0:7	0.9	76	186	694.96	–	–
1:0	0.9	100	80	607.82	–	–
1:1	0.9	10	269	843.37	–	–
1:2	0.9	100	245	1809.32	–	–
1:3	0.9	77	552	3206.61	–	–
1:4	0.9	100	151	657.32	–	–
1:5	0.85	100	152	681.89	0	42.05
1:6	0.9	100	42	376.40	–	–
1:7	0.9	100	726	5759.01	–	–

TABLE III

BENCHMARKS OF FINDING UP TO 100 SAFE AND STABLE REGIONS FOR DATA SET *h1*. TIMES ARE IN SECONDS, 'CE' IS THE NUMBER OF COUNTER-EXAMPLES, 'LB' AND 'UB' REFER TO THE PROOF OF LOWER AND UPPER BOUND ON THE THRESHOLD, RESPECTIVELY.

regions violated by the samples in the data were not considered safe even if the model output was safe on these samples. The algorithm has been run on the normalized form of the respective formulas as described in Remark 1.

If $th < 0.9$, then at least 2 searches for safe stable regions were performed, one with threshold th, which succeeded in finding some stable safe regions, and another with threshold $th + 0.05$, which did not find any and, in fact, did prove there are none above $th + 2 \cdot 0.05$. The factor 2 here comes from the heuristic used by the solver that the center of a candidate region should evaluate to $th + 0.05$. So we enumerated only regions with $center \geq th + 0.05$. If there are none, it proves the upper bound $th + 0.05$ on the safety threshold. If $th = 0.9$, no upper bound check with threshold 0.95 has been performed, since it could only prove the bound 1, which is clear by construction. Tables I and III also provide the number of counter-examples found during the proof and the computation time.

As can be seen from Algorithms 2 and 3, for a region proven safe two SMT calls were made and for each counter-example (lb-ce and ub-ce) up to two SMT calls were performed. In order to verify that the threshold th is minimal, one additional call per C:B combination was required. As can be seen from the tables, the total of these numbers in order to produce the results for each C:B range between 233 and 11 211 Z3 calls.

In addition, Table II shows the count of stable safe regions shared across multiple bytes of the same channel – from the safe regions reported in Table I. Note that this table is not symmetric because training samples falling in a shared stable safe region can violate the output constraint for some of the bytes (but not all the bytes) for which this region is safe.

As mentioned in the introduction, proving a safety constraint on an NN output does not mean the modelled system itself is safe. One reason for that is that it is very difficult to generate high-quality training samples to build accurate models when little is known about the behavior of the modelled system. It therefore takes a number of iterations to improve the training data and thereby improve the model. One can use the stable safe regions to generate new training samples within the safe regions in order to refine the current model. We have performed such *proof-based abstraction refinement* of the NN models on *h1_rx* and *h1_tx* datasets. We generated 100 random samples in each stable safe region and asked the Lab to measure the output. Interestingly, 790 out of 1600 stable safe regions for *h1_rx* remained safe in the sense that the system output still satisfies the output constraint; and 1106 out of the 1463 safe regions of *h1_tx* remained safe. This matched the user input that the *RX* model was much harder to analyze and configure safely.

VII. CONCLUSIONS AND FUTURE WORK

We have defined the problem of configuration selection for NN models in its general form and demonstrated the feasibility of our proposed algorithms on a real-life industrial application. Our work leverages the recent research on verifying inequality constraints on NN-ReLU output when the inputs are also

constrained with inequalities, which can be seen as an atomic black-box operation in our algorithms. In the current implementation we use the `Z3` solver for performing this operation in order to support real, integer and categorical variables. As immediate future work, we intend to integrate and evaluate other solvers that might be significantly faster for `NN-ReLU` with only real-valued inputs (LP) or real and integer valued inputs (MILP). In order to support `NNs` with transcendental activation functions (such as Sigmoid or Softmax), as well as transcendental constraints on safe and stable solutions, we are extending the implementation to work with solvers such as `ksmt` [5], dReal [10] and others. We will also integrate other machine learning models such as random forest and polynomial models which can be covered by our framework.

It is important to note that for many CAD applications, in `NN` models used to model complex systems, there rarely is a need for more than 5 to 20 input features; this is confirmed by our experience of using `NN` and other models for a wide range of CAD applications at Intel. Indeed, the state of the art is to apply advanced feature selection [13] techniques to the input features in the labeled dataset measured on the system, and to select a subset consisting of highly relevant and highly independent features that provide a high coverage of the variation that exists in the data. Thus computational complexity of the problem that we are dealing with is not preventing usage in real-life CAD applications. For applications in computer vision and related areas where very large `NN` models are required and the inputs are real variables, we believe that integration of fast decision procedures such as LP will help our algorithms to scale, especially when the number of configuration parameters is a relatively small fraction of the system's interface.

Besides selection of safe and stable regions, our algorithms addresses also the problem of selecting such regions where the performance of the output is close to optimal. As a future work, it would be interesting to adapt our algorithms to multi-objective optimisation problems where the validity of output constraints cannot be compromised for the benefit of optimisation and at the same time Pareto-optimal regions can be selected; besides the application domain discussed in this work, relevant examples include joint optimisation of power, performance and area, as well joint optimisation of voltage, frequency and temperature, without compromising safe operation. Currently multi-objective optimisation tasks are handled in our framework via reduction to single-objective optimisation using a weighted average over the optimisation objectives.

ACKNOWLEDGEMENT

This research was supported by a grant from Intel Corporation.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, Sh. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, Il. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zhengi. Large-Scale Machine Learning on Heterogeneous Systems, https://www.tensorflow.org/, 2015.

[2] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. Conference on Computer Aided Verification, CAV 2011.

[3] C. Barrett, P. Fontaine, C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). 2016, http://smtlib.cs.uiowa.edu/.

[4] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi. Measuring Neural Net Robustness with Constraints. Neural Information Processing Systems, NIPS 2016.

[5] F. Brauße, K. Korovin, M. V. Korovina, and N. Th. Müller. A CDCL-style calculus for solving non-linear constraints. Frontiers of Combining Systems, FroCoS 2019.

[6] R. Bunel, I. Turkaslan, P.H.S. Torr, P. Kohli, M. Pawan Kumar. A Unified View of Piecewise Linear Neural Network Verification. Neural Information Processing Systems, NIPS 2018.

[7] C.-H. Cheng, G. Nührenberg, C.-H. Huang, H. Ruess. Verification of Binarized Neural Networks via Inter-Neuron Factoring. arXiv:1710.03107, 2017.

[8] F. Chollet. Keras. https://github.com/fchollet/keras, 2015.

[9] S. Dutta, T. Kushner, S. Jha, S. Sankaranarayanan. Sherlock: A Tool for Verification of Deep Neural Networks, https://github.com/souradeep-111/sherlock

[10] S. Gao, J. Avigad, E. M. Clarke. δ-complete decision procedures for satisfiability over the reals. In Automated Reasoning - 6th International Joint Conference, IJCAR 2012.

[11] Y. Ge, L. de Moura. Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories, CAV 2009.

[12] I. J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, ICLR 2015.

[13] I. Guyon, A. Elisseeff. An Introduction to Variable and Feature Selection. Journal of Machine Learning Research 3, 2003.

[14] M. Emmer, Z. Khasidashvili, K. Korovin, A. Voronkov. Encoding Industrial Hardware Verification Problems into Effectively Propositional Logic, FMCAD 2010.

[15] M. Emmer, Z. Khasidashvili, K. Korovin, C. Sticksel, A. Voronkov. EPR-Based Bounded Model Checking at Word Level, IJCAR 2012.

[16] G. Katz, Cl. Barrett, D. Dill, K. Julian M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks , CAV 2017.

[17] Y. Kazak, C. Barrett, G. Katz, M. Schapira. Verifying Deep-RL-Driven Systems. Network Meets AI & ML, NetAI 2019.

[18] Z. Khasidashvili, M. Kinanah, A. Voronkov. Verifying Equivalence of Memories Using a First Order Logic Theorem Prover, FMCAD 2009.

[19] Z. Khasidashvili, K. Korovin, D. Tsarkov. EPR-based k-induction with Counterexample Guided Abstraction Refinement, GCAI 2015.

[20] K. Korovin. iProver–an instantiation-based theorem prover for first-order logic (system description), IJCAR 2008.

[21] S. Liang, R. Srikant. Why Deep Neural Networks For Function Approximation? ICLR 2017.

[22] Z. Lu, H. Pu, F. Wang, Z. Hu, L. Wang. The Expressive Power of Neural Networks: A View from the Width. Neural Information Processing Systems, 62316239, 2017.

[23] A. Manukovsky, Y. Juniman, Z. Khasidashvili. A Novel Method of Precision Channel Modeling for High Speed Serial 56Gb Interfaces, DesignCon'18, 2018.

[24] A. Manukovsky, Z. Khasidashvili, A.J. Norman, Y. Juniman, R. Bloch. Machine Learning Applications for Simulation and Modeling of 56 and 112 Gb SerDes Systems, DesignCon'19, 2019.

[25] G. F. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems, 2014.

[26] L. de Moura, N. Bjørner. Z3: An Efficient SMT Solver. TACAS 2008.

[27] J.A. Navarro-Perez, A. Voronkov. Encodings of Bounded LTL Model Checking in Effectively Propositional Logic, CADE 2007.

[28] L. Pulina, A. Tacchella. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. CAV 2010.

[29] D.E. Rumelhart, G.E. Hinton, R. J. Williams. Learning representations by back-propagating errors, Nature, vol. 323, 1986.

[30] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus. Intriguing properties of neural networks, 2014.