Learning with safety requirements: state of the art and open questions

Francesco Leofante[1], Luca Pulina[2], and Armando Tacchella[1]

[1] DIBRIS — Università di Genova — Via all'Opera Pia, 13; 16145 Genova; Italia
francesco.leofante@edu.unige.it – armando.tacchella@unige.it
[2] POLCOMING — Università di Sassari — Viale Mancini 5; 07100 Sassari; Italy
lpulina@uniss.it

Abstract. A standing challenge for the adoption of machine learning (ML) techniques in safety-related applications is that misbehaviors can be hardly ruled out with traditional analytical or probabilistic techniques. In previous contributions of ours, we have shown how to deal with safety requirements considering both Multi-Layer Perceptrons (MLPs) and Support Vector Machines (SVMs), two of the most effective and well-known ML techniques. The key to provide safety guarantees for MLPs and SVMs is to combine Satisfiability Modulo Theory (SMT) solvers with suitable abstraction techniques. The purpose of this paper is to provide an overview of problems and related solution techniques when it comes to guarantee that the input-output function of trained computational models will behave according to specifications. We also summarize experimental results showing what can be effectively assessed in practice using state-of-the-art SMT solvers. Our empirical results are the starting point to introduce some open questions that we believe should be considered in future research about learning with safety requirements.

1 Introduction

Machine learning (ML) techniques are adopted in a wide range of research and engineering domains. For instance, artificial agents that act in physical domains can be equipped with learning capabilities in order to acquire the dynamics of interest, thus saving developers from the burden of devising explicit models — see [1–3] for examples. However, in spite of some exceptions, ML techniques are confined to systems which comply to the lowest safety integrity levels, achievable with standard industrial best practices [4]. The main reason is the absence of effective safety assurance methods for systems using ML techniques, because traditional analytical and probabilistic methods can be ineffective in safety-related applications — see, e.g., [5]. In practice, guaranteeing safety amounts to controlling undesirable behaviors when the agent acts based on a model of the environment. In some cases of interest, a further challenge is related to the cost of sampling, i.e., acquiring new data for learning, so that the challenge of ensuring safety, must also be coupled with the paucity of samples.

In previous contributions of ours we have shown effective approaches to learning with safety requirements. In particular, in [6] we tackled the problem of ensuring that a trained Multi-Layer Perceptron (MLP) emits outputs which are guaranteed to stay within range of values considered to be safe. We called this *global safety* in [7], where we introduced two additional problems, namely *local safety* and *stability*. In the former case, we wish to guarantee that an input close to a training input sample, will elicit an output close to the corresponding training output sample. In the latter case, we wish to ensure that if the input is contained within some bounded region, also the output will be contained within a corresponding bounded set. Since effective learning with MLPs requires a large number of samples and their empirical tuning is often fraught with difficulties, in [8] we considered safety requirements in conjunction with kernel-bases methods. In order to reduce the number of samples we considered Active Learning (AL) — see, e.g., [9] — and we focused on Support Vector Regression (SVR) — see, e.g., [10] for a tutorial introduction.

In all the contributions above, the problem of providing safety guarantees is encoded to formulas that can be solved automatically by Satisfiability Modulo Theory (SMT) solvers [11]. Intuitively, verification of MLPs/SVRs via SMT involves abstraction of the learned model into a set of constraints expressed in linear arithmetic over real numbers. The abstraction process is guaranteed to be conservative, i.e., it provides an over approximation of the concrete MLP/SVR. In this way, when the SMT solver proves that the response of the abstract MLP/SVR cannot exceed a stated bound as long as the input values satisfy given preconditions, we can certify that the concrete MLP/SVR has the same property. On the other hand, if a counterexample is found, then it is either an artefact of the abstraction, or a true counterexample proving that the MLP/SVR is not satisfactory in terms of safety. The former case calls for a refinement of the abstraction, whereas the second may entail further training of the MLP/SVR.

The experimental assessment that we present considers SMT encodings obtained from different verification problems. In the case of MLPs, the encodings are related to control subsystems in the humanoids James [12] and iCub [13]. In the case of James, measuring external forces using a single force/torque sensor placed along the kinematic chain of the arm requires *compensation* of the manipulator dynamic, i.e., the contribution of internal forces must be subtracted from sensor readings [14]. MLP-based estimation of internal forces relies on angular positions and velocities of two shoulder and two elbow joints, and takes into account components from gravity, Coriolis forces, and manipulator inertia. In the case of iCub, we consider MLP-based extrapolation of forward kinematics of its arms. In particular, we consider angular positions of three shoulder, one elbow and three wrist joints as inputs, and end-effector positions as outputs. Finally, in the case of SVRs, we considered an artificial physical domain simulated with V-REP [15], wherewith we simulate the task of applying the right amount of force to send a ball into a goal in the presence of obstacles. We further assume that acquiring input samples is expensive, so that active learning instead of batch learning is required.

The remainder of this paper is organized as follows. Section 2 gives background notions on MLPs, SVRs, and SMT solvers. Section 3 summarizes the abstractions introduced in [6–8] and introduces global, local and stability requirements. The state of the art is presented in Section 4. Concluding remarks and open questions are presented in Section 5.

2 Preliminaries

2.1 Neural networks for regression

A *Multi-Layer Perceptron* (MLP) is a function $\nu : \mathbb{R}^n \to \mathbb{R}^m$ obtained by feeding n inputs through a network of *neurons*, i.e., computational units arranged in multiple layers ending with m outputs, where each neuron is characterized by an *activation function* $\sigma : \mathbb{R} \to \mathbb{R}$. We consider networks with only three cascaded layers, namely *input*, *hidden*, and *output* layer, so that there are always n neurons in the input layer, and m neurons in the output layer. Let $x = (x_1, \ldots, x_n)$ be the *input signal* to the network ν. We assume that input neurons compute the activation function $\sigma(r) = r$, i.e., their task is simply to pass forward the components of the input signal. Let h denote the number of hidden neurons, then the input of the j-th hidden neuron is

$$r_j = \sum_{i=1}^{n} a_{ji} x_i + b_j \qquad j = \{1, \ldots, h\} \tag{1}$$

where a_{ji} is the *weight* of the connection from the i-th neuron in the input layer ($1 \leq i \leq n$) to the j-th neuron in the hidden layer, and the constant b_j is the *bias* of the j-th neuron. We denote with $A = \{a_{11}, a_{12}, \ldots a_{1n}, \ldots, a_{h1}, a_{h2}, \ldots a_{hn}\}$ and with $B = \{b_1, \ldots, b_h\}$ the corresponding sets of weights. The activation function σ_h computed by hidden neurons is a non-constant, bounded and continuous function of its input. According to [16] this guarantees that, in principle, ν will be capable of approximating any continuous mapping $f : \mathbb{R}^n \to \mathbb{R}^m$ arbitrarily close. The input received by an output neuron is

$$s_k = \sum_{j=1}^{h} c_{kj} \sigma_h(r_j) + d_k \qquad k = \{1, \ldots, m\} \tag{2}$$

where c_{kj} denotes the weight of the connection from the j-th neuron in the hidden layer to the k-th neuron in the output layer, while d_k represents the bias of the k-th output neuron – as before, C and D denote the corresponding sets of weights. The *output signal* of the MLP is a vector $\nu(x) = (\sigma_o(s_1), \ldots, \sigma_o(s_m))$. The activation function of output neurons can be chosen as $\sigma_o = \sigma_h$, so that each output of ν is constrained to the range of σ_h, or as $\sigma_o(r) = r$, so that $\nu(x) = (s_1, \ldots, s_m)$.

The key aspect of MLPs is that the sets of parameters A, B, C, D are not manually determined but are computed using an iterative optimization procedure known as *training*. Let R be a *training set* comprised of t pairs $R = \{(x_1, y_1), \ldots (x_l, y_l)\}$ where for each $1 \leq i \leq l$, the vector x_i is some input signal (*pattern*) and y_i is the corresponding output signal (*label*). If we assume that R is generated by some unknown function $f : \mathbb{R}^n \to \mathbb{R}^m$, then we can see training as a way of extrapolating the unknown function f from the signals in the training set. The goal of training is to determine the parameters A, B, C, D which maximize similarity between $\nu(x)$ and y for each pair $(x, y) \in R$.

2.2 Kernel-based methods for regression

Support Vector Regression (SVR) is a supervised learning paradigm, whose mathematical foundations are derived from Support Vector Machine (SVM) [17–19]. Suppose we are given a set $R = \{(x_1, y_1) \dots (x_l, y_l)\} \subset \mathcal{X} \times \mathbb{R}$, where $\mathcal{X}$ is the input space with $\mathcal{X} = \mathbb{R}^d$. In the context of ε-SVR [20] for the linear case, our aim is to compute a function $\nu(x)$ which, in case of ε-SVRs, takes the form

$$\nu(x) = w \cdot x + b \text{ with } w \in \mathcal{X}, b \in \mathbb{R} \tag{3}$$

such that $\nu(x)$ deviates at most ε from the targets $y_i \in T$ with $\|w\|$ as small as possible. A solution to this problem is presented in [19], where slack variables ξ_i, ξ_i^* are introduced to handle otherwise unfeasible constraints. The formulation of the optimization problem stated above then becomes:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & y_i - w \cdot x_i - b \leq \varepsilon + \xi_i \\ & w \cdot x_i + b - y_i \leq \varepsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \end{aligned} \tag{4}$$

The constant $C > 0$ determines the trade-off between the "flatness" of f, i.e., how small is $\|w\|$, and how much we want to tolerate deviations greater than ε.

In order to solve the above stated optimization problem one can use the standard dualization method using Lagrange multipliers. The Lagrange function writes

$$\begin{aligned} L := \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) - \sum_{i=1}^{l}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ - \sum_{i=1}^{l}\alpha_i(\varepsilon + \xi_i - y_i + w \cdot x_i + b) - \sum_{i=1}^{l}\alpha_i^*(\varepsilon + \xi_i^* - y_i - w \cdot x_i - b) \end{aligned} \tag{5}$$

where $\eta_i, \eta_i^*, \alpha_i, \alpha_i^*$ are Lagrange multipliers. Solving the dual problem allows to rewrite (3) as

$$\nu(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)x_i \cdot x + b \tag{6}$$

The algorithm can be modified to handle non-linear cases. This could be done by mapping the input samples x_i into some feature space $\mathcal{F}$ by means of a map $\Phi : \mathcal{X} \to \mathcal{F}$. However this approach can easily become computationally unfeasible for high dimensionality of the input data. A better solution can be obtained by applying the so-called *kernel trick*. Observing eq. (6) it is possible to see that the SVR algorithm only depends on dot products between patterns x_i. Hence it suffices to know the function $K(x, x') = \Phi(x) \cdot \Phi(x')$ rather than Φ explicitly. By introducing this idea the optimization problem can be rewritten, leading to the non-linear version of (6):

$$\nu(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)K(x_i, x) + b \tag{7}$$

where $K(\cdot)$ is called *kernel function* as long as it fulfills some additional conditions — see, e.g., [10] for a full characterization of K.

2.3 Satisfiability Modulo Theory

A Constraint Satisfaction Problem (CSP) [21] is defined over a *constraint network*, i.e., a finite set of *variables*, each ranging over a given domain, and a finite set of *constraints*. Intuitively, a constraint represents a combination of values that a certain subset of variables is allowed to take. In this paper we are concerned with CSPs involving linear arithmetic constraints over real-valued variables. From a syntactical point of view, the constraint sets that we consider are built according to the following grammar:

$$\begin{aligned}
\textit{set} &\rightarrow \{\ \textit{constraint}\ \}^*\ \textit{constraint}\\
\textit{constraint} &\rightarrow (\{\ \textit{atom}\ \}^*\ \textit{atom})\\
\textit{atom} &\rightarrow \textit{bound} \mid \textit{equation}\\
\textit{bound} &\rightarrow \textit{value relop value}\\
\textit{relop} &\rightarrow < \mid \leq \mid = \mid \geq \mid >
\end{aligned}
\qquad
\begin{aligned}
\textit{value} &\rightarrow \textit{var} \mid \text{-}\ \textit{var} \mid \textit{const}\\
\textit{equation} &\rightarrow \textit{var} = \textit{value}\ \text{-}\ \textit{value}\\
&\mid\ \textit{var} = \textit{value} + \textit{value}\\
&\mid\ \textit{var} = \textit{const} \cdot \textit{value}
\end{aligned}$$

where *const* is a real value, and *var* is the name of a (real-valued) variable. Formally, this grammar defines a fragment of linear arithmetic over real numbers known as Quantifier-Free Linear Arithmetic over Reals (QF_LRA) [11].

From a semantical point of view, let X be a set of real-valued variables, and μ be an *assignment* of values to the variables in X, i.e., a partial function $\mu : X \rightarrow \mathbb{R}$. For all μ, we assume that $\mu(c) = c$ for every $c \in \mathbb{R}$, and that $\textit{relop}^\mu$, $+^\mu$, $-^\mu$, $\cdot^\mu$ denote the standard interpretations of relational and arithmetic operators over real numbers. Linear constraints are thus interpreted as follows:

- A constraint $(a_1 \ldots a_n)$ is true exactly when at least one of the atoms a_i with $i \in \{1, \ldots, n\}$ is true.
- Given $x, y \in X \cup \mathbb{R}$, an atom x *relop* y is true exactly when $\mu(x)$ $\textit{relop}^\mu$ $\mu(y)$ holds.
- Given $x \in X$, $y, z \in X \cup \mathbb{R}$, and $c \in \mathbb{R}$
  - $x = y \odot z$ with $\odot \in \{+, -\}$ is true exactly when $\mu(x) =^\mu \mu(y) \odot^\mu \mu(z)$,
  - $x = c \cdot y$ is true exactly when $\mu(x) =^\mu \mu(c) \cdot^\mu \mu(y)$,

Given an assignment, a constraint is thus a function that returns a Boolean value. A constraint is *satisfied* if it is true under a given assignment, and it is *violated* otherwise. A constraint network is *satisfiable* if it exists an assignment that satisfies all the constraints in the network, and *unsatisfiable* otherwise.

While different approaches are available to solve CSPs, this work is confined to Satisfiability Modulo Theories (SMT) [11]. SMT is the problem of deciding the satisfiability of a first-order formula with respect to some decidable theory $\mathcal{T}$. In particular, SMT generalizes the Boolean satisfiability problem (SAT) by adding background theories such as the theory of real numbers, the theory of integers, and the theories of data structures (*e.g.*, lists, arrays and bit vectors). The idea behind SMT is that the satisfiability of a constraint network can be decided in two steps. The first one is to convert each arithmetic constraint to a Boolean constraint, while the second one is to check the consistency of the assignment in the corresponding background theory. Checking that the Boolean assignment is feasible in the underlying mathematical theory can be performed by a specialized reasoning procedure (SMT solvers) – any procedure that computes feasibility of a set of linear inequalities in the case of QF_LRA. If the consistency check fails, then a new Boolean assignment is requested and the SMT solver goes on until either an arithmetically consistent Boolean assignment is found, or no more Boolean assignments can be found.

3 SMT-based abstraction and verification techniques

3.1 Abstraction for MLPs and SVRs

Research on learning with safety requirements has the objective of verifying trained MLPs/SVRs, i.e., guarantee that, given some input preconditions, the mapping ν respects stated output postconditions. In principle, this check can be automated in a suitable formal logic L by proving

$$\models_L \forall x \, (\pi(x) \rightarrow \varphi(\nu(x))) \tag{8}$$

where π and φ are formulas in L expressing preconditions on input signals and postconditions on output signals, respectively. Most often, activation functions for MLPs are non-linear and transcendental and so are kernels for SVRs, which makes (8) undecidable if L is chosen to be expressive enough to model such functions – see, e.g., [22]. In [6, 8] this problem is tackled introducing an *abstraction* framework. Given ν and properties π, φ, a corresponding abstract model $\tilde{\nu}$ and abstract properties $\tilde{\pi}, \tilde{\varphi}$ are defined so that

$$\models_{L'} \forall x' \, (\tilde{\pi}(x') \rightarrow \tilde{\varphi}(\tilde{\nu}(x'))) \tag{9}$$

is decidable, and (8) holds whenever (9) holds. If this is not the case, then there is some abstract input x' such that $\tilde{\pi}(x)$ is satisfied, but the abstract output $\tilde{\nu}(x')$ violates $\tilde{\varphi}$. There are two cases:

– If a concrete input x can be extracted from x' such that also $\nu(x)$ violates φ, then x' is a *realizable* counterexample, and ν fails to uphold (8).
– On the other hand, if no concrete input x can be extracted such that $\nu(x)$ violates φ, then x' is a *spurious* counterexample.

Spurious counterexamples arise because an abstract MLP/SVR $\tilde{\nu}$ is "more permissive" than its concrete counterpart ν. In this case, a *refinement* is needed, i.e., a new abstraction "tighter" than $\tilde{\nu}$ must be computed and checked. We call this abstraction-refinement loop *Counter-Example Triggered Abstraction Refinement* (CETAR) in [6]. Briefly stated, the main difference between CETAR and classical *Counter-Example Guided Abstraction Refinement* (CEGAR) is that, in the latter counterexamples are used as a basis to refine the abstraction, whereas in the former counterexamples are merely triggering the refinement, without being involved in computing the refinement. While there is no theoretical guarantee that the CETAR loop terminates, in practice it is expected that a network is declared safe or a realizable counterexample is found within a small number of refinement steps.

Details about the definition of $\tilde{\nu}$ for MLPs in such a way that L'=QF_LRA, as well as details of preconditions and postconditions to be checked, with related abstractions, are

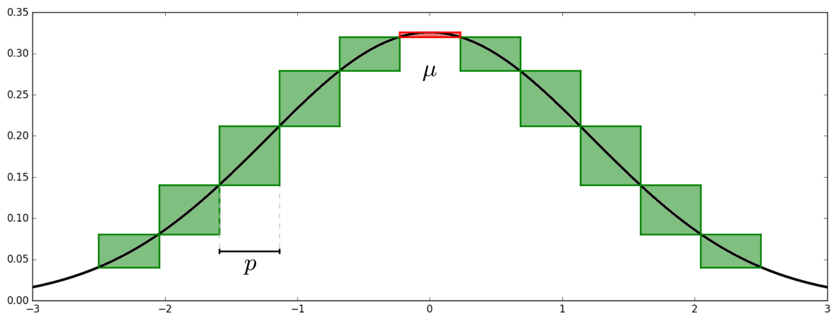

Fig. 1. Interval abstraction of a Gaussian kernel.

provided in [6, 7]. In the following, for the sake of explanation, we briefly summarize how to encode SVR verification in a corresponding QF_LRA problem as shown in [8]. In particular, our experimental analysis focuses on radial basis function (RBF) kernels

$$K(x_i, x) = e^{-\gamma\|x-x_i\|^2} \text{ with } \gamma = \frac{1}{2\sigma^2} \tag{10}$$

Given a SVR, we consider its input domain to be defined as $\mathcal{I} = D_1 \times \ldots \times D_n$, where $D_i = [a_i, b_i]$ with $a_i, b_i \in \mathbb{R}$, and the output domain to be defined as $\mathcal{O} = [c, d]$ with $c, d \in \mathbb{R}$. A concrete domain $D = [a, b]$ is abstracted to $[D] = \{[x, y] \mid a \leq x \leq y \leq b\}$, i.e., the set of intervals inside D, where $[x]$ is a generic element.

To abstract the RBF kernel K, we observe that (10) corresponds to a Gaussian distribution $\mathcal{G}(\mu, \sigma)$ with mean $\mu = x_i$ — the i-th support vector of ς — and variance $\sigma^2 = \frac{1}{2\gamma}$. Given a real-valued abstraction parameter p, let us consider the interval $[x, x+p]$. It is possible to show that the maximum and minimum of $\mathcal{G}(\mu, \sigma)$ restricted to $[x, x+p]$ lie in the external points of such interval, unless $\mu \in [x, x+p]$ in which case the maximum of $\mathcal{G}$ lies in $x = \mu$. The abstraction $\tilde{K}_p$ is complete once we consider two points x_0, x_1 with $x_0 < \mu < x_1$ defined as the points in which $\mathcal{G}(x_i) \ll G(\mu)$ with $i \in 0, 1$, where by "$a \ll b$" we mean that a is at least one order of magnitude smaller than b [3]. We define the *abstract kernel function for the i-th support vector* as

$$\tilde{K}_p([x, x+p]) = \begin{cases} [\min(\mathcal{G}(x), \mathcal{G}(x+p)), \mathcal{G}(\mu)] & \text{if } \mu \in [x, x+p] \subset [x_0, x_1] \\ [\min(\mathcal{G}(x), \mathcal{G}(x+p)), \max(\mathcal{G}(x), \mathcal{G}(x+p))] & \text{if } \mu \notin [x, x+p] \subset [x_0, x_1] \\ [0, \mathcal{G}(x_0)] & \text{otherwise} \end{cases} \tag{11}$$

According to eq. (11), p controls to what extent $\tilde{K}_p$ over-approximates K since large values of p correspond to coarse-grained abstractions and small values of p to fine-grained ones. A pictorial representation of eq. (11) is given in figure 1. The *abstract*

[3] Given a Gaussian distribution, 99.7% of its samples lie within $\pm 3\sigma$. In this case however, our RBF formulation does not include any normalization factor and stopping at $\pm 3\sigma$ would not be sufficient to include all relevant samples. The heuristics described in the text was therefore adopted to solve the problem.

SVR $\tilde{\varsigma}_p$ is defined rewriting equation (7) as

$$\tilde{\varsigma}_p = \sum_{i}^{l} (\alpha_i - \alpha_i^*) \tilde{K}_p(x_i, x) + b \tag{12}$$

where we abuse notation and use conventional arithmetic symbols to denote their interval arithmetic counterparts.

It is easy to see that every abstract SVR $\tilde{\varsigma}_p$ can be modeled in QF_LRA. Let us consider a concrete SVR $\varsigma : [0,1] \rightarrow (0,1)$, *i.e.*, a SVR trained on $T = \{(x_1, y_1) \dots (x_l, y_l)\}$ with $x_i \in [0,1] \subset \mathbb{R}$ and $y_i \in (0,1) \subset \mathbb{R}$. The abstract input domain is defined by:

$$x_i \geq 0 \qquad x_i \leq 1 \tag{13}$$

The coefficients and the intercept seen in equation (12) carry over to $\tilde{\varsigma}_p$ and are defined by constraints of the form

$$\alpha_i - \alpha_i* = \langle \alpha_i - \alpha_i* \rangle \qquad b = \langle b \rangle \tag{14}$$

where the notation $\langle \cdot \rangle$ represents the actual values obtained by training ς. To complete the encoding, we must provide constraints that correspond to (11) for all the support vectors of ς. Assuming $p = 0.5$, the encoding for the i-th kernel writes

```
**if** $(x \leq x_0)$
    **then** $(K_i \geq 0)$ $(K_i \leq \mathcal{G}(x_0))$
...
**if** $(0 < x)(x \leq 0.5)$
    **then** $(K_i \geq \min(\mathcal{G}(0), \mathcal{G}(0.5))$ $(K_i \leq \max(\mathcal{G}(0), \mathcal{G}(0.5))$
...
**if** $(x_1 < x)$
    **then** $(K_i \geq 0)$ $(K_i \leq \mathcal{G}(x_0))$
```

The expression "**if** $t_1 \dots t_m$ **then** $a_1 \dots a_n$" is an abbreviation for the set of constraints

$$\begin{array}{c} (\neg t_1 \dots \neg t_m a_1) \\ \dots \\ (\neg t_1 \dots \neg t_m a_n) \end{array} \tag{15}$$

where $t_1, \dots, t_m$ and $a_1, \dots a_n$ are atoms expressing bound on variables, and $\neg$ is Boolean negation (e.g., $\neg(x < y)$ is $(x \geq y)$). Finally, the output of the SVR is just

$$f(x) = \langle \alpha_1 - \alpha_1^* \rangle K_i(x_1, x) + \dots + \langle \alpha_n - \alpha_n^* \rangle K_n(x_n, x) + \langle b \rangle \tag{16}$$

with $x_1, \dots, x_n$ being the support vectors of ς

3.2 Problems in safety related learning

As mentioned previously, verification of MLPs and SVRs amounts to answer a logical query. In this subsection, we summarize the three different verification problems – namely, global safety, local safety, and stability – as introduced in [7], as well as the different ways of defining preconditions π and postconditions φ.

Global safety This property was considered in [6] for the first time, and requires that a trained model $\nu : \mathcal{I} \rightarrow \mathcal{O}$ with n inputs and m outputs fulfills the condition

$$\forall x \in \mathcal{I},\, \forall k \in \{1, \ldots, m\} : \nu_k(x) \in [l_k, h_k] \tag{17}$$

where $l_k, h_k \in E_k$ are constants chosen to define an interval wherein the k-th component of $\nu(x)$ is to range – we call them *safety thresholds* after [6]. It is easy to see how (17) can be expressed using the notation of (8) by defining

- $\pi(x)$ as a predicate which is true exactly when $x \in \mathcal{I}$, and
- $\varphi(y)$ as a predicate which is true exactly when $y \in \mathcal{B}$ where $\mathcal{B} = [l_1, h_1] \times \ldots \times [l_m, h_m]$.

We say that an MLP/SVR ν is *globally safe* if condition (8) holds when π and φ are defined as above. This notion of (global) safety is representative of all the cases in which an out-of-range response is unacceptable, such as, e.g., minimum and maximum reach of an industrial manipulator, lowest and highest percentage of a component in a mixture, and minimum and maximum dose of a drug that can be administered to a patient.

Local safety The need to check for global safety in MLPs can be mitigated using bounded activation functions in the output neurons to "squash" their response within an acceptable range, modulo rescaling. In principle, the same trick could be applied to SVRs as well, even if it is much less commonly adopted. A more stringent, yet necessary, requirement is thus represented by "local" safety. Informally speaking, we can say that an MLP/SVR ν trained on a dataset R of t patterns is "locally safe" whenever given an input pattern $x^* \neq x$ for all $(x, y) \in R$ it turns out that $\nu(x^*)$ is "close" to y_j as long as x^* is "close" to x_j and $(x_j, y_j) \in R$ for some $j \in \{1, \ldots, t\}$. Local safety cannot be guaranteed by design, because the range of acceptable values varies from point to point. Moreover, it is relevant in all those applications where the error of an MLP/SVR must never exceeds a given bound on yet-to-be-seen inputs, and where the response of an MLP must be stable with respect to small perturbations in its input in the neighborhood of "known" patterns.

Stability While local safety allows to analyze the response of a network even in the presence of bounded-output MLPs/SVRs, it suffers from two important limitations.

- It does not take into account the whole input domain, i.e., if a pattern falls out the union of the input polytopes, then nothing can be said about the quality of the response.
- It is tied to a specific training set, i.e., changing the set of patterns used to train the network can change the response of the safety check.

These two limitations are important in practice, in particular when the availability of training data is scarce so the chance of hitting "unseen" patterns is high, or when the training set is very sparse so the chance of having large regions of uncovered input space is high. One way to overcome these limitations is to keep the locality of the check, i.e., small input variations must correspond to small output variations, but drop

Family	Overall		Time (s)	Hardness		
	N	#		EA	ME	MH
ICUB_LOGI	720	504	68706.83	132	330	42
ICUB_RAMP	36	35	3980.89	15	20	–
ICUB_TANH	720	581	45618.63	257	296	28
JAMES_LOGI	720	489	84699.24	70	378	41
JAMES_RAMP	36	34	3101.20	10	23	1
JAMES_TANH	720	559	86560.52	131	371	57

Table 1. Encoding-centric view of the results. The table consists of seven columns where we report for each family their name in alphabetical order (column "Family"), the number of encodings included in the family, and the number of encodings solved (group "Overall", columns "N", "#", respectively), the CPU time taken to solve the encodings (column "Time"), the number of easy, medium and medium-hard encodings (group "Hardness", columns "EA", "ME", "MH").

the dependency from the training set, i.e., check the whole input space. Formally, given a trained model $\nu : \mathcal{I} \rightarrow \mathcal{O}$ we define *stability* as follows

$$\begin{aligned} \forall x_1, x_2 \in \mathcal{I} : ||x_1 - x_2|| \leq \delta \\ \rightarrow ||\nu(x_1) - \nu(x_2)^*|| \leq \epsilon \end{aligned} \tag{18}$$

where $\delta, \epsilon \in \mathbb{R}^+$ are two arbitrary constants which define the amount of variation that we allow in the input and the corresponding perturbation of the output. We observe that condition (18) requires non linear arithmetic to express the scalar product of the difference vectors. In other words, once fixed x_1, we have that x_2 must be contained in an hypersphere of radius δ centered in x_1 – and similarly for $\nu(x_1)$, $\nu(x_2)$ and ϵ. An overapproximation of such hypersphere, and thus a consistent abstraction of (18), is any convex polyhedra in which the hypersphere is contained. The degree of abstraction can be controlled by setting the number of facets of the polyhedra, and such an abstract condition can be expressed in QF_LRA.

As reported in [7] stability turns out to be the hardest condition to check for SMT solvers, so in the following we will focus on this one.

4 State of the art

4.1 Experiments with batch learning and MLPs

Considering the experimental results reported in [7], stability turned out to be the hardest condition to check for SMT solvers, and in Table 1 we report a summary of that experimental analysis. We consider both James and iCub case studies in the case of MLPs with different activation functions, namely logistic (logi), bounded ramp (ramp), and hyperbolic tangent (tanh) functions. In order to avoid confining the results to a single SMT solver, the number of encodings solved and the cumulative time taken for each family is computed considering the "SOTA solver", i.e., the ideal SMT solver that always fares the best time among all the solvers considered. An encoding is thus solved if at least one of the solvers solves it, and the time taken is the best among CPU times

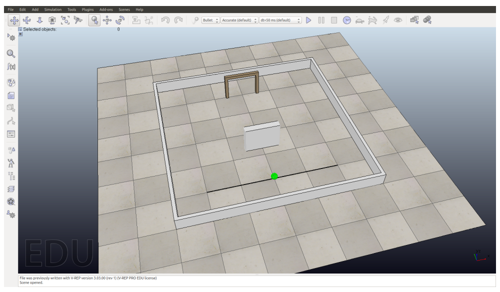

Fig. 2. Experimental setup in V-REP. In a field delimited by fences, a green ball is to be thrown from one side into the goal (brown door) on the opposite side. An obstacle (grey wall) may be placed inside the field in such a position that all or some straight shots will always be ineffective.

of the solvers that solved the encoding. The solvers involved in the experimentation were the participants to the QF_LRA track of the SMT Competition 2011 [23], namely CVC [24] (version 4 1.0rc.2026), MATHSAT [25] (version 5), SMTINTERPOL [26] (version 2.0pre), YICES [27] (v. 1.0.29, with smtlib2 parser), and Z3 [28] (v. 3.0). In Table 1, the encodings are classified according to their hardness with the following criteria: easy encodings are those solved by all the solvers, medium encodings are those non easy encodings that could still be solved by at least two solvers, medium-hard encodings are those solved by one reasoner only, and hard encodings are those that remained unsolved. Looking at the results, we observe that 2202 encodings out of 2952 were solved. With "only" 75% of the encodings solved, we confirm that this suite contains the most difficult formulas of the whole test set. In particular, 615 encodings are easy, 1418 medium, while 169 are medium-hard. For more details about the results, we refer the reader to [7].

4.2 Experiments with active learning and SVRs

Experimental Setup A pictorial representation of the case study we consider is shown in Figure 2. In this domain setup, a ball has to be thrown in such a way that it goes through a goal, possibly avoiding obstacles present in the environment. The learning agent interacting with the domain is allowed to control the force $f = (f_x, f_y)$ applied to the ball when it is thrown. Given that the coordinate origin is placed at the center of the wall opposing the goal — see Figure 2 — we chose $f_x \in [-20, 20]N$ and $f_y \in [10, 20]N$, respectively. The range for f_x is chosen in such a way that the whole field, including side walls, can be targeted, and the range for f_y is chosen so that the ball can arrive at the goal winning pavement's friction both in straight and kick shots. The collision between the side walls and the ball is completely elastic. In principle, the obstacle can be placed everywhere in the field, but we consider only three configurations, namely (a) no obstacle is present, (b) the obstacle occludes straight line trajectories — the case depicted in Figure 2 — and (c) the obstacle partially occludes the goal, so that not all straight shots may go through it. The domain of Figure 2 is implemented

Algorithm	Mean	STD	Median	IQR
Straight and Kick shots				
Random	97.6	13.68	100.0	–
QBC	88.27	24.28	100.0	–
DB	95.53	19.49	100.0	–
Straight shots				
Random	40.06	11.7364	40.0	5.0
QBC	15.05	3.7275	15.0	3.0
DB	14.75	0.8275	15.0	1.0
Kick shots				
Random	16.3	18.29	12.0	8.25
QBC	15.9	25.12	3.0	19.25
DB	9.28	13.51	7.0	5.0

Algorithm	Mean	STD	Median	IQR
Case a				
Random	97.6	13.68	100.0	–
QBC	88.27	24.28	100.0	–
DB	95.53	19.49	100.0	–
Case b				
Random	64.07	36.18	69.0	71.0
QBC	67.22	43.44	100.0	–
DB	63.63	39.49	100.0	–
Case c				
Random	91.94	23.26	100.0	–
QBC	85.46	24.67	100.0	–
DB	93.13	23.35	100.0	–

Table 2. Evaluation of the active learning algorithms. Left: statistics on the number of AL runs needed when no obstacle is present (case a), using different subsets of the feasible trajectories. “Mean” and “STD” are the sample mean and standard deviation, respectively; “Median” is the 50%-percentile, and IQR is the difference between 75%-percentile and 25%-percentile. When the median coincides with the maximum number of allowable runs, the IQR value cannot be computed (denoted with “–”). Right: Statistics on the number of AL runs needed for cases a, b and c,

using the V-REP simulator [15]. Concrete SVRs are trained using the Python library SCIKIT-LEARN, which in turn is based on LIBSVM [29]. Verification of abstract SVRs is carried out using Mathsat [30]. The Python library GPy [31] is used to train Gaussian process regression.

Results We train SVRs on a dataset of n samples built as $T = \{(f_i, y_i), \ldots, (f_n, y_n)\}$ where $f_i = (f_{xi}, f_{yi})$ is the force vector applied to the ball to throw it, and y_i is the observed distance from the center of the gate, which is also the target of the regression. Obviously, the regression problem could be extended, e.g., by including the initial position of the ball, but our focus here is on getting the simplest example that could lead to evaluation of different active learning paradigms and verification thereof. Preliminary tests were performed in an obstacle-free environment, using specific subsets of all the feasible trajectories. Three algorithms were compared, namely a random strategy for adding new samples to the learning set, the *query by committee* algorithm and the one proposed by Demir and Bruzzone [32] — listed in Table 2 as “Random”, “QBC” and “DB”, respectively. Each test episode consists in adding samples to the model, and it ends when either the Root Mean Squared Error (RMSE) is smaller than a threshold (here 0.2), or when at most 100 additional samples are asked for. Table 2 suggests that when the learning problem involves one single class of shots (e.g., only straight shots), the algorithms provide better results. Indeed, when both straight and kick shots are possible, all three algorithms require more than 100 additional samples in at least 50% of the runs in order to reach the required RMSE. We conducted further experiments, this time considering the three cases (a), (b), and (c) described before, and the full set of feasible trajectories in each case. The results obtained, shown in Table 2 (right), confirm the observation made on Table 2 (left). The best results are indeed obtained for case (b), wherein the obstacle is placed in such a way that only kick shots are feasible, thus turning the learning problem into a single-concept one.

5 Open questions

In this work we have summarized the state of the art in the topic of learning with safety requirements. The promise of this research is to make data-driven models fit enough to be deployed in safety-related context, e.g., adaptive agents operating in physical domains. However, several questions should be answered before the techniques surveyed in this paper can be pushed in practical applications with confidence in their effectiveness. Currently, we can consider the following issues:

– SMT solvers could deal with the majority of encodings in a reasonable amount of time; still, there might be room for improvement and scalability with respect to the size of the learning problems must be assessed.
– Safety requirements have been considered for batch learning with MLPs and active learning with SVRs; it would be worth investigating whether active learning with MLPs and batch learning with SVRs can give improved results in terms of learning and verification.
– There are learning techniques which are both very effective and natively provide (statistical) bounds on their prediction; Gaussian process regression (GPR) [33] is one such technique, with a fairly considerable record of success stories; it would be interesting to compare the level of confidence in the prediction given by GPR on one side, and the strict guarantees provided by MLPs/SVRs checked with SMT solvers on the other.
– Applications in which learning techniques ought to provide safety requirements are diverse, and include learning through reinforcement with continuous rather than discrete state-action-space representations — see [34] for a discussion. One important issue is thus the feasibility of techniques that we surveyed in such fields, wherein learning is part of a larger system devoted to program adaptive agents.

Our future works will focus on addressing the above open issues and whether the proposed methodologies can scale to real-world problems, possibly posing verification of learning models as a challenge problem for the SMT research community.

References

1. Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robotics and autonomous systems **57**(5) (2009) 469–483
2. Nguyen-Tuong, D., Peters, J.: Model learning for robot control: a survey. Cognitive processing **12**(4) (2011) 319–340
3. Kober, J., Peters, J.: Reinforcement learning in robotics: A survey. In: Reinforcement Learning. Springer (2012) 579–610
4. Smith, D., Simpson, K.: Functional Safety – A Straightforward Guide to applying IEC 61505 and Related Standards (2nd Edition). Elsevier (2004)
5. Kurd, Z., Kelly, T., Austin, J.: Developing artificial neural networks for safety critical systems. Neural Computing & Applications **16**(1) (2007) 11–19
6. Pulina, L., Tacchella, A.: An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In: 22nd International Conference on Computer Aided Verification (CAV 2010). Volume 6174 of Lecture Notes in Computer Science, Springer (2010) 243–257

7. Pulina, L., Tacchella, A.: Challenging smt solvers to verify neural networks. AI Communications **25**(2) (2012) 117–135
8. Leofante, F., Tacchella, A.: Learning in Physical Domains: Mating Safety Requirements and Costly Sampling. In: AI*IA 2016, Advances in Artificial Intelligence - 15th International Conference of the Italian Association for Artificial Intelligence, Genova, Italy, November 29 - December 1, 2016, Proceedings. (2016) To appear.
9. Settles, B.: Active learning literature survey. University of Wisconsin, Madison **52**(55-66) (2010) 11
10. Smola, A.J., Schölkopf, B.: A tutorial on support vector regression. Statistics and computing **14**(3) (2004) 199–222
11. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. Handbook of satisfiability **185** (2009) 825–885
12. Jamone, L., Metta, G., Nori, F., Sandini, G.: James: A humanoid robot acting over an unstructured world. In: Humanoid Robots, 2006 6th IEEE-RAS International Conference on, IEEE (2007) 143–150
13. Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., Von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., et al.: The icub humanoid robot: An open-systems platform for research in cognitive development. Neural Networks **23**(8) (2010) 1125–1134
14. Fumagalli, M., Gijsberts, A., Ivaldi, S., Jamone, L., Metta, G., Natale, L., Nori, F., Sandini, G.: Learning to Exploit Proximal Force Sensing: a Comparison Approach. From Motor Learning to Interaction Learning in Robots (2010) 149–167
15. Rohmer, E., Singh, S.P., Freese, M.: V-rep: A versatile and scalable robot simulation framework. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, IEEE (2013) 1321–1326
16. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Networks **4**(2) (1991) 251–257
17. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual workshop on Computational learning theory, ACM (1992) 144–152
18. Guyon, I., Boser, B., Vapnik, V.: Automatic capacity tuning of very large vc-dimension classifiers. Advances in neural information processing systems (1993) 147–147
19. Cortes, C., Vapnik, V.: Support-vector networks. Machine learning **20**(3) (1995) 273–297
20. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer-Verlag New York, Inc., New York, NY, USA (1995)
21. Mackworth, A.K.: Consistency in networks of relations. Artificial intelligence **8**(1) (1977) 99–118
22. Franzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. Journal on Satisfiability, Boolean Modeling and Computation **1** (2007) 209–236
23. Bruttomesso, R., Deters, M., , Griggio, A.: The 2011 smt competition (2011)
24. Barrett, C., Tinelli, C.: Cvc3. In: International Conference on Computer Aided Verification, Springer (2007) 298–302
25. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The mathsat5 smt solver. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer (2013) 93–107
26. Christ, J., Hoenicke, J., Nutz, A.: Smtinterpol: An interpolating smt solver. In: International SPIN Workshop on Model Checking of Software, Springer (2012) 248–254
27. Dutertre, B., De Moura, L.: A fast linear-arithmetic solver for dpll (t). In: International Conference on Computer Aided Verification, Springer (2006) 81–94
28. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer (2008) 337–340

29. Chang, C.C., Lin, C.J.: Libsvm: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST) **2**(3) (2011) 27
30. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT Solver. In Piterman, N., Smolka, S., eds.: Proceedings of TACAS. Volume 7795 of LNCS, Springer (2013)
31. GPy: GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy` (since 2012)
32. Demir, B., Bruzzone, L.: A multiple criteria active learning method for support vector regression. Pattern Recognition **47**(7) (2014) 2558–2567
33. Williams, C.K.I., Rasmussen, C.E.: Gaussian processes for regression. In: Advances in Neural Information Processing Systems 8, NIPS, Denver, CO, November 27-30, 1995. (1995) 514–520
34. Pathak, S., Pulina, L., Metta, G., Tacchella, A.: How to abstract intelligence?(if verification is in order). In: Proc. 2013 AAAI Fall Symp. How Should Intelligence Be Abstracted in AI Research. (2013)