A Computability Perspective on (Verified) Machine Learning

Tonicha Crook$^{[0000-0002-4882-9999]}$, Jay Morgan$^{[0000-0003-3719-362X]}$, Arno Pauly$^{[0000-0002-0173-3295]}$, and Markus Roggenbach$^{[0000-0002-3819-2787]}$

Department of Computer Science, Swansea University
Swansea, Wales, UK
t.m.crook15@outlook.com arno.m.pauly@gmail.com
m.roggenbach@swansea.ac.uk
Université de Toulon, Aix Marseille Univ, CNRS, LIS
Marseille, France
jay.morgan@univ-tln.fr

Abstract. In Computer Science there is a strong consensus that it is highly desirable to combine the versatility of Machine Learning (ML) with the assurances formal verification can provide. However, it is unclear what such 'verified ML' should look like.
This paper is the first to formalise the concepts of classifiers and learners in ML in terms of computable analysis. It provides results about which properties of classifiers and learners are computable. By doing this we establish a bridge between the continuous mathematics underpinning ML and the discrete setting of most of computer science.
We define the computational tasks underlying the newly suggested verified ML in a model-agnostic way, i.e., they work for all machine learning approaches including, e.g., random forests, support vector machines, and Neural Networks. We show that they are in principle computable.

Keywords: Machine Learning · adversarial examples · formal verification · computable analysis

1 Introduction

Machine Learning (ML) concerns the process of building both predictive and generative models through the use of optimisation procedures. The remarkable success of ML methods in various domains raises the question of how much trust one can put into the responses that an ML model provides. As ML models are also applied in critical domains, some form of verification seems essential (e.g. eloquently argued by Kwiatkowska [9]).

However, due to the widespread use of non-discrete mathematics in ML, traditional verification techniques are hard to apply to its artefacts. Furthermore, many ML applications lack specifications in the form of, say, an input/output relationship, on which 'classical' verification approaches are often based. A typical example of this would be an ML application that shall decide if a given picture depicts a cat. Lacking a specification, what kind of properties can be verified? We will take the view that, like in classical verification, it is useful to expand the range of properties beyond simple input/output relations.

By employing the toolset of computable analysis (the field concerned with computation on continuous data types), we are using the same continuous mathematics underpinning the theory of machine learning, and avoids any ad-hoc discretization.

We present an investigation into what kind of verification questions are answerable in principle about ML models – irrespective of the particular ML framework applied. We see these questions as basic building blocks for a future ML property specification language. Discretization, as far as it may be necessary for the sake of efficiency, can then be left to the implementation; without impacting correctness.

We use the language of computable analysis to formally define the computational questions we want to ask. We can prove that they are solvable in general (by exhibiting algorithms for them), while remaining independent of any concrete ML methodology. The semi-decision procedures in this paper are not meant for implementation. We are also not making any claims about computational complexity.

Our paper is organised as follows: in Section 2 we provide a gentle summary of our results. In Section 3, we provide definitions and key properties from computable analysis. Section 4 develops our theory with mathematical precision. Finally, Section 5 discusses related work.

2 A Gentle Summary of our Results

In this section, we provide a gentle introduction to our results. The technical details and proofs, which are using concepts of Computable Analysis, are provided later in the paper. The main results are of the nature that specific functions model aspects of interest in ML (verification), and are computable.

We first model classifiers, define elementary questions on them and explore when they are computable. Then we formalise the idea of adversarial examples for a classifier utilising metric spaces. We show that detecting adversarial examples or proving their absence is computable. Next, we consider the process of learning itself. One question of interest here is the robustness of the results of a learned classifier depending on changes of the training data. We study this in two settings within Subsection 2.3.

2.1 Classifiers

One basic notion of ML is that of a classifier. A classifier takes as an input some description of an object, say, in the form of a vector of real numbers, and outputs either a colour or does not give an answer. This makes classifiers a generalisation of semi-decision procedures.

Computable Analysis is developed as the theory of functions on the real numbers and other sets from analysis, which can be computed by machines. The first step in formalising a classifier in Computable Analysis is to discuss its domain and codomain. To this end Computable Analysis uses the notion of a represented space (to be formalised later in this paper, as are the other technical notions). The domain of a classifier can in general be an arbitrary represented space, while the codomain is defined as follows:

Definition 1. *The represented space $\mathbf{k}_\perp$ contains the elements $\{0, \ldots, k-1, \perp\}$, where $0, \ldots, k-1$ are discrete points and $\perp$ is an additional point specified by no information at all / represents 'no information'.*

Including the bottom element $\perp$ in our framework is essential to obtain a satisfactory theory. It can represent uncertainty. When we are modelling an ML-classifier that outputs probabilities attached to the colours, we could e.g. consider $n \in \mathbf{k}$ to be the answer if the assigned probability exceeds 0.5, and $\perp$ to be the answer if no individual colour exceeds 0.5.

A classifier has to be a computable function in order to be implementable. A key observation of Computable Analysis is that computable functions are by necessity continuous[1]. In fact, it turns out that the most suitable notion of function space in Computable Analysis is the space $\mathcal{C}(\mathbf{X}, \mathbf{Y})$ of continuous functions from $\mathbf{X}$ to $\mathbf{Y}$. This justifies the following definition:

Definition (Definition 8). *A* classifier *is a continuous function that takes some $x \in \mathbf{X}$ as input, and either outputs a colour $j \in \mathbf{k}$, or diverges (which is seen as outputting $\perp$). The collection of classifiers is the space $\mathcal{C}(\mathbf{X}, \mathbf{k}_\perp)$.*

Any concrete classifier we would care about will actually be computable. However, the definition includes also non-computable (but continuous) ones.

Example 1. A support vector machine produces separating hyperplanes, which act as classifier by returning one colour on one side of the hyperplane, another colour on the other side, and no answer for points on the hyperplane.

Given a classifier, what verification questions could we ask? We may want to confirm individual requirements or look at assertions regarding the behaviour of the classifier on an entire set or region A. These could be used for verification, where we desire to obtain a guarantee that the system is working correctly, or to identify potential errors. Concrete question include:

1. existsValue which answers true on input (n, A, f) iff $\exists x \in A \; f(x) = n$. Otherwise, there is no answer.

[1] Arguing informally, continuity means that sufficiently good approximations of the input specify approximations of the output to desired precision. Computability means that we can actually compute the desired approximations of the output from sufficiently good approximations of the input. The latter cannot be possible for discontinuous function.

Example 2. Consider a DDOS-attack detection system realized as a classifier f. The region A consists of data indicating an attack is happening, and the colour n means that the system concludes there is no attack. If existsValue(n, A, f) returns true, we have identified a false negative in f.

2. forallValue which answers true on input (n, A, f) iff $\forall x \in A\ f(x) = n$. Otherwise, there is no answer.

   Example 3. Continuing with example 2, here we may consider the property that for every data point in the region A the presence of the attack is successfully identified. If forallValue(n, A, f) returns true, we have verified that all points in the set A are classified as expected.

3. fixedValue, which on input (n, A, f) answers 1 iff $\forall x \in A\ f(x) = n$, and answer 0 iff $\exists x \in A\ f(x) \in \mathbf{k} \setminus \{n\}$. The answer $\perp$ is given if the classifier returns $\perp$ for at least one point in A, but does not return any colour except n on points from A.

   Example 4. Consider an automated stock trading system which makes decisions to buy, sell or hold a particular stock based on its technical indicators. Deciding to buy or sell is represented as a colour (as it is an active decision), while $\perp$ means to hold the current position. Given a region A of very positive technical indicator values, we may want to ideally be assured that the system will always buy, while the decision to sell would be a clear mistake. If fixedValue(buy$, A, f)$ returns 1, we know that the system meets the ideal requirement. If it returns 0, we have found a mistake. Answer $\perp$ means that the system falls short of its target without making a clear mistake.

4. constantValue which on input (A, f) answers 1 iff there is some $n \in \mathbf{k}$ such that fixedValue(n, A, f) answers 1, and which answers 0 iff fixedValue(n, A, f) answers 0 for all $n \in \mathbf{k}$.

   The question constantValue is a first approximation of how to deal with adversarial examples.

   Example 5. Assume we have reason to believe that all points in the region A are very similar, and should thus be classified in the same way by the classifier f. If constantValue(A, f) returns 1, we have the confirmation that this indeed happens. Obtaining the answer 0 suggests that a mistake might have happened, as two similar data points get assigned different colours. No answer (i.e. $\perp$) means that some points in A remain unclassified by f.

It remains to specify what regions A are considered and how they are represented. Two familiar notions from computable analysis are exactly what we need to make these questions computable, namely the compact sets $\mathcal{K}(\mathbf{X})$ and the overt sets $\mathcal{V}(\mathbf{X})$ (see Proposition 2). Finite sets are both compact and overt. This means that for any finite sample of data all four questions are computable. The far more interesting applications however concern infinite sets, e.g. all points belonging to a geometrically defined region, as they go beyond testing a classifier for a finite number of inputs.

2.2 Adversarial Examples

One specific verification task that has caught the attention of the ML community is to find *adversarial examples* [22,5,7] or to prevent them from occurring. One says that an adversarial example occurs when a 'small' change to the input results in an 'unreasonably large' change in the output (i.e. akin our fourth task above). For example, given a correctly classified image, small, even unnoticeable changes to the pixels in the image can vastly change the classification output.

Example 6. In particular image-based Deep Neural Networs (DNNs) can be easily fooled with precise pixel manipulation. The work in [23] uses a Gaussian mixture model to identify keypoints in images that describe the saliency map of DNN classifiers. Modifying these keypoints may then change the classification label made by said DNN. They explore their approach on 'traffic light challenges' (publicly available dashboard images of traffic lights with red/green annotations). In this challenge, they find modifying a single pixel is enough to change neural network classification.

If we want to discuss *small* changes in data, say, in an image, we need to assume a notion of distance. We will thus assume that our domain $\mathbf{X}$ comes equipped with a metric $d : \mathbf{X}\times\mathbf{X} \to \mathbb{R}_{\geq 0}$, specifically, that $(\mathbf{X}, d)$ is a computable metric space (which covers nearly any metric space considered in real analysis).

Detecting the presence or the absence of adversarial examples can be explored using the following function:

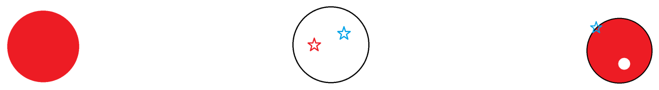
Definition 2. *Let $(\mathbf{X}, d)$ be a computable metric space, and $\mathcal{C}(\mathbf{X}, \mathbf{k}_\perp)$ the space of classifiers. The map* locallyConstant : $\mathbf{X} \times \mathbb{R}^+ \times \mathcal{C}(\mathbf{X}, \mathbf{k}_\perp) \to \mathbf{2}_\perp$ *returns* 1 *on input (x, ε, f) iff f returns the same colour $n \in \mathbf{k}$ for all $y \in \mathbf{X}$ with $d(x, y) \leq \varepsilon$. It returns 0 if there exists some $y \in \mathbf{X}$ with $d(x, y) < \varepsilon$ such that f returns distinct colours on x and y. It returns $\perp$, if some points ε-close to x remain unclassified by f (such as in the cases of the global-robustness property [10]), but no distinct colours appear; or if there is a distinct colour appearing at a distance of exactly ε to x.*

The map locallyConstant is illustrated in Figure 1. Consider the (closed) ε-ball around a point x. For a classifier f, locallyConstant outputs 1 if everything in the ball yields the same answer under f. The answer is 0 if there are two points (depicted as a red and a blue star in the figure) that yield distinct answers in the ball. The answer $\perp$ appears in two cases: If there are unclassified points inside the open ε-ball around x (the white region inside the ball on the right), or if another colour appears on the boundary of the ball, but not inside it (blue star).

We prove in Theorem 1 below that locallyConstant is computable under mild assumptions, namely that every closed ball $\overline{B}(x, \varepsilon)$ is compact (which implies that $(\mathbf{X}, d)$ is locally compact). The Euclidean space $\mathbb{R}^n$ is both a typical example for an effectively locally compact computable metric space where all closed balls are compact, and the predominant example relevant for ML.

To relate back to adversarial examples, locallyConstant tells us whether a classifier admits an adversarial example close to a given point x.

Fig. 1: Illustrating the map locallyConstant



(a) Output 1 (b) Output 0 (c) Output $\perp$

Example 7. In the context of fully-autonomous vehicles that use sensor-captured data as input for DNN models, [7] explains how lighting conditions and angles, as well as defects in sensor equipment themselves, yield realistic adversarial examples. Assume the metric to be chosen to model the impact of these issues on the sensor data. Then one could deploy locallyConstant on a fully-autonomous vehicle in order to detect if one can trust the classifier on the current input data (answer 1) or not (answer 0).

2.3 Learners

Up to now, we considered already trained ML procedures. Now we discuss the process of learning itself, and introduce the concept of a *learner*. A learner takes a sample of points with corresponding labels and outputs a trained model:

Definition 3. *A* learner *is a (computable) procedure that takes as an input a tuple* $((x_0, n_0), \dots, (x_\ell, n_\ell)) \in (\mathbf{X} \times \mathbf{k})^*$ *and outputs a classifier* $f \in \mathcal{C}(\mathbf{X}, \mathbf{k}_\perp)$*. The collection of all learners is the space* $\mathcal{C}((\mathbf{X} \times \mathbf{k})^*, \mathcal{C}(\mathbf{X}, \mathbf{k}_\perp))$*.*

Note that we not only consider classifiers to be continuous functions, but that also learning itself is assumed to be continuous. As discussed above, this is a consequence of demanding that learning is a computable process. Continuity here means in essence[2] that if the data sample is altered by changing some x_k to some very close x'_k instead, the resulting classifiers f and f' cannot assign distinct colours to the same point y (though they may still differ in the use of $\perp$).

Now we can ask how 'robust' the classifier we are learning with the training data actually is. This phenomenon has recently attracted attention in the ML literature under the term of underspecification [4]. Whether we view the phenomenon as robustness of learning or underspecification of the desired outcome is a matter of perspective: A failure of robustness is tied to the existence of (almost) equally good alternative classifiers.

Our first question on a classifier is: is it possible by adding one extra point to the training data to change the classification? In other words, can a small

[2] By uncurrying, we can move from a learner L to the function $\ell : (\mathbf{X} \times k)^* \times \mathbf{X} \to \mathbf{k}_\perp$ such that $L((x_i, n_i)_{i \leq j})(x) = \ell((x_i, n_i)_{i \leq j}, x)$. One of them is continuous iff the other is. Now we can see that if ℓ returns a colour, it returns the same colour on an open neighbourhood of both training sample and test point.

addition to the training data lead to a change in classification (this is not already guaranteed by continuity of learning; the size of training data is a discrete value).

1. robustPoint takes as input a point x, a learner and some training data. It answers 1 if every extension of the training data by one more sample point leads to x still receiving the same colour. It answers 0, if there exists an extension of the training data by a single sample point leading to x receiving a different colour. The case $\perp$ covers if x is unclassified for the original data or some of its extensions.
2. robustArea takes as input an area A, a learner and some training data. It yields 1 if every point in A is recognized as robust by robustPoint, it yields 0 if there exists a point $x \in A$ where robustPoint returns 0, and $\perp$ otherwise.

If $\mathbf{X}$ is computably compact and computably overt, and we take the regions A to be themselves compact and overt sets, then both operations are computable.

Allowing to add more than one point does not genuinely complicate the theory. However, it is useful to also ask the question where the additional points are added. Again, we consider a metric expressing the distance between the points added to the training set and the point whose classification we are interested in.

We will call training data *dense* at a point x, if adding a small number of additional data points sufficiently far from x does not change its classification under the learned classifier. It is *sparse* if the classification can be changed.

We will show that the operation SprsOrDns is computable (if the computable metric space $\mathbf{X}$ is computably compact). This operation has a number of parameters: a learner L, the number N of permitted additional data points, the distance ε, the training data $(x_i, n_i)_{i \leq \ell}$ and the point x of interest. It answers 0 iff $(x_i, n_i)_{i \leq \ell}$ is sparse at x, and answers 1 if $(x_i, n_i)_{i \leq \ell}$ is dense at x.

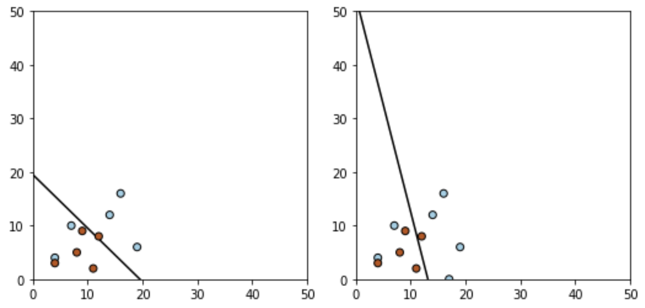


Fig. 2: Illustrating lack of robustness. Left: Original data, Right: Changed separating line due to one added data point (at (18,0)).

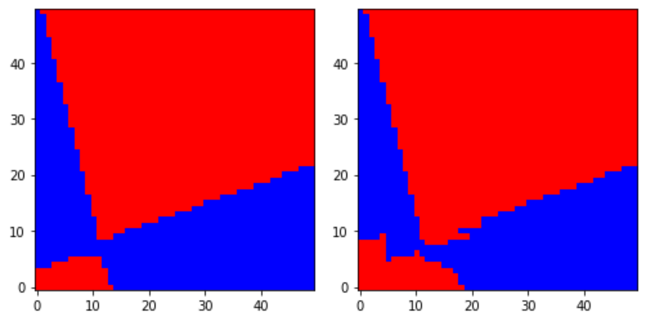


Fig. 3: Illustration of predicates concerning the original classifier. Left: Robustness (Red: Robustness, Blue: Not Robust), Right: Sparsity/Density (Red: Dense, Blue: Sparse) – note: $\perp$ is at the boundaries between the coloured areas.

Figures 2 and 3 illustrate the notions of robustness and sparsity/density on the same data set [3]. In Figure 2 we show a classification example: our algorithm utilises a Support Vector Machine to create a separating line between the grey and red dots. The left side shows the original data set, and the right side shows the data set after the addition of another grey dot. We point out that the separating line has moved significantly.

In the left image of Figure 3 one can see which areas of the input space can be affected in the classification due to adding one point to the training data. The algorithm adds the extra data point, calculates the separating line, and then checks whether each point in the graph is on the left or right of the separating line. It continues to do this for all possible addition data points in order to see what points of the graph never change sides of the line (robust) and which do. The blue area consists of all points which are not robust, i.e. can fall on either side of the separating line depending on the additional point. The red area consists of all robust points, i.e., they always fall on the same side of the separating line independent of the additional point.

The right image of Figure 3 then shows the notion of sparsity/density: As before, we consider whether a point could fall on either side of the separating line after the addition of another sample point. However, we only consider sample points further than 5 units away from the point under consideration. There can be points which are dense, but not robust (see the area directly above $(10, 0)$ and $(20, 0)$). This can be explained as follows: To obtain a very steep separating line, the extra data point needs to be placed in the area around $(15, 0)$. Thus, the steep separating lines contribute to the pattern at the top of the picture, but do not affect the red component at the bottom.

[3] Our overall algorithms are theoretical and not easily put into code. However, the individual concepts can be, in order to help the understanding of mathematical concepts such as robustness/sparsity and density.

Density implies robustness, but not the converse, as for the former we exclude additional training data too close to the point under discussion. It depends on the application which notion is more suitable. Lack of robustness under addition of a single data point indicates a too small set of training data. If we consider adding a more significant fraction of additional training data, robustness may be a too demanding notion. In this case, dense points are still those where only very similar counterexamples would challenge the response of the classifier.

3 Computing with real numbers and other non-discrete data types

The following summarises the formal definitions and key properties of the most important notions for our paper. It is taken near verbatim from [2]. A more comprehensive treatment is found in [16].

Definition 4. *A represented space is a pair $\mathbf{X} = (X, \delta_{\mathbf{X}})$ where X is a set and $\delta_{\mathbf{X}} :\subseteq \mathbb{N}^{\mathbb{N}} \to X$ is a partial surjection (the notation $:\subseteq$ denotes partial functions, $\mathbb{N}^{\mathbb{N}}$ is the space of infinite sequences of natural numbers). A function between represented spaces is a function between the underlying sets.*

For example, we will consider the real numbers $\mathbb{R}$ as a represented space $(\mathbb{R}, \rho)$ by fixing a standard enumeration $\nu : \mathbb{N} \to \mathbb{Q}$ of the rational numbers, and then letting $\rho(p) = x$ iff $\forall k \in \mathbb{N}\ |\nu(p(k)) - x| < 2^k$. In words, a name for a real number encodes a sequence of rational numbers converging to it with speed 2^{-k}. Likewise, the spaces $\mathbb{R}^n$ can be considered as represented spaces by encoding real vectors as limits of sequences of rational vectors converging with speed 2^{-k}. This generalises to the following:

Definition 5. *A computable metric space is a metric space (X, d) together with a dense sequence $(a_n)_{n\in\mathbb{N}}$ (generalizing the role of the rational numbers inside $\mathbb{R}$) which makes the map $(n, m) \mapsto d(a_n, a_m) : \mathbb{N}^2 \to \mathbb{R}$ computable. The induced representation of X is δ_d mapping $p \in \mathbb{N}^{\mathbb{N}}$ to $x \in \mathbf{X}$ iff $\forall n \in \mathbb{N}\ d(x, a_{p(n)}) < 2^{-n}$.*

An important facet of computable analysis is that equality is typically not decidable; in particular, it is not for the spaces $\mathbb{R}^n$. For these, inequality is semidecidable though (which makes them computably Hausdorff by definition). If equality is also semidecidable, a space is called computably discrete.

Definition 6. *For $f :\subseteq \mathbf{X} \to \mathbf{Y}$ and $F :\subseteq \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$, we call F a realizer of f (notation $F \vdash f$), iff $\delta_Y(F(p)) = f(\delta_X(p))$ for all $p \in \mathrm{dom}(f\delta_X)$. A map between represented spaces is called computable (continuous), iff it has a computable (continuous) realizer.*

Two represented spaces of particular importance are the integers $\mathbb{N}$ and the Sierpiński space $\mathbb{S}$. The represented space $\mathbb{N}$ has as underlying set $\mathbb{N}$ and the representation $\delta_{\mathbb{N}} : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ defined by $\delta_{\mathbb{N}}(p) = p(0)$, i.e., we take the first element of the sequence p. The Sierpiński space $\mathbb{S}$ has the underlying set $\{\top, \bot\}$ and the representation $\delta_{\mathbb{S}} : \mathbb{N}^{\mathbb{N}} \to \mathbb{S}$ with $\delta_{\mathbb{S}}(0^\omega) = \bot$ and $\delta_{\mathbb{S}}(p) = \top$ for $p \neq 0^\omega$.

Represented spaces have binary products, defined in the obvious way: The underlying set of $\mathbf{X} \times \mathbf{Y}$ is $X \times Y$, with the representation $\delta_{\mathbf{X}\times\mathbf{Y}}(\langle p, q\rangle) = (\delta_{\mathbf{X}}(p), \delta_{\mathbf{Y}}(q))$. Here $\langle\ ,\ \rangle : \mathbb{N}^{\mathbb{N}} \times \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ is the pairing function defined via $\langle p, q\rangle(2n) = p(n)$ and $\langle p, q\rangle(2n+1) = q(n)$.

A central reason why the category of represented spaces is such a convenient setting lies in the fact that it is cartesian closed. We have available a function space construction $\mathcal{C}(\cdot, \cdot)$, where the represented space $\mathcal{C}(\mathbf{X}, \mathbf{Y})$ has as underlying set the continuous functions from $\mathbf{X}$ to $\mathbf{Y}$, represented in such a way that the evaluation map $(f, x) : \mathcal{C}(\mathbf{X}, \mathbf{Y}) \times \mathbf{X} \to \mathbf{Y}$ becomes computable.

Having available the space $\mathbb{S}$ and the function space construction, we can introduce the spaces $\mathcal{O}(\mathbf{X})$ and $\mathcal{A}(\mathbf{X})$ of open and closed subsets respectively of a given represented space $\mathbf{X}$. For this, we identify an open subset U of $\mathbf{X}$ with its (continuous) characteristic function $\chi_U : \mathbf{X} \to \mathbb{S}$, and a closed subset with the characteristic function of the complement. As countable join (or) and binary meet (and) on $\mathbb{S}$ are computable, we can conclude that open sets are uniformly closed under countable unions, binary intersections, and preimages under continuous functions by merely using elementary arguments about function spaces.

Note that neither negation $\neg : \mathbb{S} \to \mathbb{S}$ (i.e. mapping $\top$ to $\bot$ and $\bot$ to $\top$) nor countable meet (and) $\bigwedge : \mathcal{C}(\mathbb{N}, \mathbb{S}) \to \mathbb{S}$ (i.e. mapping the constant sequence $(\top)_{n\in\mathbb{N}}$ to $\top$ and every other sequence to $\bot$) are continuous or computable.

We need two further hyperspaces, which both will be introduced as subspaces of $\mathcal{O}(\mathcal{O}(\mathbf{X}))$. The space $\mathcal{K}(\mathbf{X})$ of saturated compact sets identifies $A \subseteq \mathbf{X}$ with $\{U \in \mathcal{O}(\mathbf{X}) \mid A \subseteq U\} \in \mathcal{O}(\mathcal{O}(\mathbf{X}))$. Recall that a set is saturated, iff it is equal to the intersection of all open sets containing it (this makes the identification work). The saturation of A is denoted by $\uparrow A := \bigcap\{U \in \mathcal{O}(\mathbf{X}) \mid A \subseteq U\}$. Compactness of A corresponds to $\{U \in \mathcal{O}(\mathbf{X}) \mid A \subseteq U\}$ being open itself. The dual notion of compactness is *overtness*. We obtain the space $\mathcal{V}(\mathbf{X})$ of overt sets by identifying a closed set A with $\{U \in \mathcal{O}(\mathbf{X}) \mid A \cap U \neq \emptyset\} \in \mathcal{O}(\mathcal{O}(\mathbf{X}))$.

Aligned with the definition of the compact and overt subsets of a space, we can also define when a space itself is compact (respectively overt):

Definition 7. *A represented space $\mathbf{X}$ is (computably) compact, iff isFull* $: \mathcal{O}(\mathbf{X}) \to \mathbb{S}$ *mapping X to $\top$ and any other open set to $\bot$ is continuous (computable). Dually, it is (computably) overt, iff isNonEmpty* $: \mathcal{O}(\mathbf{X}) \to \mathbb{S}$ *mapping $\emptyset$ to $\bot$ and any non-empty open set to $\top$ is continuous (computable).*

The relevance of $\mathcal{K}(\mathbf{X})$ and $\mathcal{V}(\mathbf{X})$ is found in particular in the following characterisations, which show that compactness just makes universal quantification preserve open predicates, and dually, overtness makes existential quantification preserve open predicates.

Proposition 1 ([16, Proposition 40 & 42]). *The following are computable:*

1. *The map* $\exists : \mathcal{O}(\mathbf{X} \times \mathbf{Y}) \times \mathcal{V}(\mathbf{X}) \to \mathcal{O}(\mathbf{Y})$ *defined by*

$$\exists(R, A) = \{y \in Y \mid \exists x \in A\ (x, y) \in R\}.$$

2. The map $\forall : \mathcal{O}(\mathbf{X} \times \mathbf{Y}) \times \mathcal{K}(\mathbf{X}) \to \mathcal{O}(\mathbf{Y})$ *defined by*

$$\forall(R, A) = \{y \in Y \mid \forall x \in A\ (x, y) \in R\}.$$

The represented space $(\mathcal{V} \wedge \mathcal{K})(\mathbf{X})$ contains the sets which are both compact and overt, and codes them by providing the compact and the overt information simultaneously. Thus, both universal and existential quantification over elements of $(\mathcal{V} \wedge \mathcal{K})(\mathbf{X})$ preserve open predicates.

4 A Theory of Verified ML

Here we provide the mathematical counterpart to Section 2.

4.1 A Theory of Classifiers

As stated above, we consider classification tasks only. This means that a trained model will take as input some description of an object, and either outputs a class (which we take to be an integer from $\mathbf{k} = \{0, \dots, k-1\}$, $k > 0$), or it does not give an answer. Here, not giving an answer can happen by the algorithm failing to terminate, rather than by an explicit refusal to select a class. This is important to handle connected domains such as the reals, in light of the continuity of all computable functions. Formally, we are dealing with the represented space $\mathbf{k}_\perp$, which contains the elements $\{0, \dots, k-1, \perp\}$, where 0^ω is the only name for $\perp$, and any $0^m 1^\ell 0^\omega$ is a name for $\ell < k$, $m \in \mathbb{N}$.

Definition 8. *A* classifier *is a (computable/continuous) procedure that takes some* $x \in \mathbf{X}$ *as input, and either outputs a colour* $j \in \mathbf{k}$*, or diverges (which is seen as outputting* $\perp$*). The collection of classifiers is the space* $\mathcal{C}(\mathbf{X}, \mathbf{k}_\perp)$*.*

Example 8 (Expanding example 1). Consider the classifier we would obtain from Support Vector Machine [6]. The relevant space $\mathbf{X}$ will be $\mathbb{R}^n$ for some $n \in \mathbb{N}$. The classifier is described by a hyperplane P splitting $\mathbb{R}^n$ into two connected components C_0 and C_1. We have two colours, so the classifier is a map $p : \mathbb{R}^n \to \mathbf{2}_\perp$. If $x \in C_i$, then $p(x) = i$. If $x \in P$, then $p(x) = \perp$.

On the fundamental level, we need the *no-answer answer* $\perp$ as we will never be able to be certain that a numerical input is exactly on the separating hyperplane, even if we keep increasing the precision: equality on reals is not decidable.

Practically, computations might be performed using floating-point arithmetic, where equality is decidable. In this, the use of $\perp$ is still meaningful: If we keep track of the rounding errors encountered, we can use $\perp$ to denote that the errors have become too large to classify an input.

Example 9. Neural network classifiers compute a class score for every colour, which, when these class scores are normalised, share similar properties as a probability distribution. This translates into our framework by fixing a threshold $p \geq 0.5$, and then assigning a particular colour to an input iff its class score exceeds the threshold p. If no colour has a sufficiently high score, the output is $\bot$. As long as the function computing the class scores is computable, so is the classifier we obtain in this fashion. If our class scores can use arbitrary real numbers, we cannot assign a colour for the inputs leading to the exact threshold.

As motivated and discussed in Section 2, we show that we can compute the answers to the following verification questions:

Proposition 2. *The following maps are computable:*

1. existsValue : $\mathbf{k}\times\mathcal{V}(\mathbf{X})\times\mathcal{C}(\mathbf{X},\mathbf{k}_\bot) \to \mathbb{S}$*, which answers* true *on input* (n, A, f) *iff* $\exists x \in A\ f(x) = n$.
2. forallValue : $\mathbf{k}\times\mathcal{K}(\mathbf{X})\times\mathcal{C}(\mathbf{X},\mathbf{k}_\bot) \to \mathbb{S}$*, which answers* true *on input* (n, A, f) *iff* $\forall x \in A\ f(x) = n$.
3. fixedValue : $\mathbf{k} \times (\mathcal{V} \wedge \mathcal{K})(\mathbf{X}) \times \mathcal{C}(\mathbf{X},\mathbf{k}_\bot) \to \mathbf{2}_\bot$*, which on input* (n, A, f) *answers* 1 *iff* $\forall x \in A\ f(x) = n$*, and answer* 0 *iff* $\exists x \in A\ f(x) \in \mathbf{k} \setminus \{n\}$*, and* $\bot$ *otherwise.*
4. constantValue : $(\mathcal{V}\wedge\mathcal{K})(\mathbf{X})\times\mathcal{C}(\mathbf{X},\mathbf{k}_\bot) \to \mathbf{2}_\bot$*, which on input* (A, f) *answers* 1 *iff there is some* $n \in \mathbf{k}$ *such that* fixedValue(n, A, f) *answers* 1*, and which answers* 0 *iff* fixedValue(n, A, f) *answers* 0 *for all* $n \in \mathbf{k}$.

4.2 A Theory of Treating Adversarial Examples

One useful application of the map constantValue is using it on some *small* regions that we are interested in. In ML terms, it addresses the question if there are adversarial examples for a classifier in the vicinity of x. To characterise small regions, we would have available a metric, and then wish to use closed balls $\overline{B}(x, r)$ as inputs to constantValue.

To this end, we need to obtain closed balls $\overline{B}(x, r)$ as elements of $(\mathcal{V}\wedge\mathcal{K})(\mathbf{X})$. The property that for every $x \in \mathbf{X}$ we can find an $R > 0$ such that for every $r < R$ we can compute $\overline{B}(x, r) \in \mathcal{K}(\mathbf{X})$ is a characterization of effective local compactness of a computable metric space $\mathbf{X}$ [17]. We generally get $\mathrm{cl}B(x, r)$, the closure of the open ball, as elements of $\mathcal{V}(\mathbf{X})$. For all but countably many radii r we have that $\overline{B}(x, r) = \mathrm{cl}B(x, r)$, and we can effectively compute suitable radii within any interval [17].

Theorem 1. *Let* $\mathbf{X}$ *be an effectively locally compact computable metric space with metric* d *such that every closed ball is compact. The map* locallyConstant : $\mathbf{X} \times \mathbb{R}^+ \times \mathcal{C}(\mathbf{X},\mathbf{k}_\bot) \to \mathbf{2}_\bot$ *is computable, where* locallyConstant$(x, r, f) = 1$ *iff* $\forall y \in \overline{B}(x, r)\ f(x) = f(y) \neq \bot$*, and* locallyConstant$(x, r, f) = 0$ *iff* $\exists y_0, y_1 \in B(x, r)\ \bot \neq f(y_0) \neq f(y_1) \neq \bot$.

An adversarial example is the result of a small change or perturbation to the original input that results in a change of classification made by, say, a DNN. I.e. given the classifier f and an input x, an adversarial example is $f(x) \neq f(x+r)$ for $||r|| \leq \epsilon$ and $\epsilon > 0$. The question is: what do we call a 'small' perturbation, i.e., how does one choose the parameter r?

Example 10. Assume that we want to use our classifier to classify measurement results with some measurement errors. As an example, let us consider the use of ML techniques to separate LIGO sensor data indicating gravitational waves from terrestrial noise (e.g. [21]). If our measurements are only precise up to ε, then having an adversarial example for $r = \varepsilon$ tells us that we cannot trust the answers from our classifier. In the example, this could mean finding that the precise values our sensors show are classified as indicating a gravitational wave, but a negligible perturbation would lead to a 'noise'-classification.

We could use domain knowledge to select the radius r [13]. For example, in an image classification task, we could assert a priori that changing a few pixels only can never turn a picture of an elephant into a picture of a car. If we use Hamming distance as a metric on the pictures, stating what we mean with *a few pixels* gives us the value r such that any adversarial example demonstrates a fault in the classifier. Another example by [19] finds the upper and lower bounds of the input space via an optimisation procedure, following that DNNs are Lipschitz continuous functions and all values between these bounds are reachable.

So far it was the responsibility of the user to specify a numerical value for what a 'small' perturbation is in the definition of adversarial examples. As an alternative, we can try to compute the maximal value r such that on any scale smaller than r the point under consideration is not an adversarial example.

Corollary 1. *Let $\mathbf{X}$ be an effectively locally compact computable metric space with metric d such that all closed balls are compact. The map* $\mathrm{OptimalRadius} :\subseteq \mathbf{X} \times \mathcal{C}(\mathbf{X}, \mathbf{k}_\perp) \to \mathbb{R}$ *defined by* $(x, f) \in \mathrm{dom}(\mathrm{OptimalRadius})$ *iff* $f(x) \neq \perp$, $\exists y \perp \neq f(y) \neq f(x)$ *and* $\forall r, \varepsilon > 0\ \exists z \in B(x, r+\varepsilon) \setminus B(x, r)\ f(z) \neq \perp$*; and by*

$$\begin{aligned}\mathrm{OptimalRadius}(x, f) &= \sup \{r \in \mathbb{R} \mid \exists i \in \mathbf{k}\ \forall y \in \overline{B}(x, r)\ f(y) = i\} \\ &= \inf\ \{r \in \mathbb{R} \mid \exists y \in B(x, r) \perp \neq f(x) \neq f(y) \neq \perp\}\end{aligned}$$

is computable.

4.3 A Theory of Learners and their Robustness

Let us now consider the process of training the classifier. To keep matters simple, we will not adopt a dynamic view, but rather model this as a one-step process. We also only consider supervised learning, i.e., machine learning where the data set consists of labelled examples and the learning algorithm is learning a function that maps feature vectors to labels. Definition 3 formalised our conception of a learner as a map from finite sequences of labelled points to classifiers.

We do not prescribe any particular relationship between the training data and the behaviour of the resulting classifier. It could seem reasonable to ask that a learner L faithfully reproduces the training data, i.e. satisfies $L((x_i, n_i)_{i \leq \ell})(x_m) = n_m$. But such a criterion is, in general, impossible to satisfy. This is because our notion of training data does not rule out having multiple occurrences of the same sample point with different labels. It would also not match applications, as it often is desirable that a model can disregard parts of its training data as being plausibly faulty.

We can, however, ask whether a learner (e.g. CNN) when given non-contradictory training data will output a classifier faithfully reproducing it:

Proposition 3. *Let* $\mathbf{X}$ *be computably overt and computably Hausdorff. The operation*

$$\text{doesDeviate} : \mathcal{C}((\mathbf{X} \times \mathbf{k})^*, \mathcal{C}(\mathbf{X}, \mathbf{k}_\bot)) \to \mathbb{S}$$

returning true *on input* L *iff there is some input* $(x_i, n_i)_{i \leq \ell} \in (\mathbf{X} \times \mathbf{k})^*$ *with* $x_i \neq x_j$ *for* $i \neq j$*, and some* $m \leq \ell$ *such that* $L((x_i, n_i)_{i \leq \ell})(x_m) \in \mathbf{k} \setminus \{n_m\}$ *is computable.*

Robustness under additional training data Generally, our goal will not be so much to algorithmically verify properties of learners for arbitrary training data, but rather be interested in the behaviour of the learner on the given training data and hypothetical small additions to it. One question here would be to ask how robust a classifier is under small additions to the training data. A basic version of this would be:

Proposition 4. *Let* $\mathbf{X}$ *be computably compact and computably overt. The map*

$$\text{robustPoint} : \mathbf{X} \times (\mathbf{X} \times \mathbf{k})^* \times \mathcal{C}((\mathbf{X} \times \mathbf{k})^*, \mathcal{C}(\mathbf{X}, \mathbf{k}_\bot)) \to \mathbf{2}_\bot$$

answering 1 *on input* x, $(x_i, n_i)_{i \leq \ell}$ *and* L *iff*

$$\forall x_{\ell+1} \in \mathbf{X}\ \forall n_{\ell+1} \in \mathbf{k} \quad L((x_i, n_i)_{i \leq \ell})(x) = L((x_i, n_i)_{i \leq \ell+1})(x) \in \mathbf{k}$$

and answering 0 *iff*

$$\exists x_{\ell+1} \in \mathbf{X}\ \exists n_{\ell+1} \in \mathbf{k} \quad \bot \neq L((x_i, n_i)_{i \leq \ell})(x) \neq L((x_i, n_i)_{i \leq \ell+1})(x) \neq \bot$$

is computable.

We can lift robustPoint to ask about all points in a given region, or even in the entire space as a corollary:

Corollary 2. *Let* $\mathbf{X}$ *be computably compact and computably overt. The map*

$$\text{robustRegion} : (\mathcal{K} \wedge \mathcal{V})(\mathbf{X}) \times (\mathbf{X} \times \mathbf{k})^* \times \mathcal{C}((\mathbf{X} \times \mathbf{k})^*, \mathcal{C}(\mathbf{X}, \mathbf{k}_\bot)) \to \mathbf{2}_\bot$$

answering 1 *on input* A, $(x_i, n_i)_{i \leq \ell}$ *and* L *iff* robustPoint *answers* 1 *for every* $x \in A$ *together with* $(x_i, n_i)_{i \leq \ell}$ *and* L*, and which answer* 0 *iff there exists some* $x \in A$ *such that* robustPoint *answers* 0 *on input* x, $(x_i, n_i)_{i \leq \ell}$ *and* L*, and which answers* $\bot$ *otherwise, is computable.*

Sparsity of training data Allowing arbitrary additional training data as in the definition of robustness might not be too suitable – for example, if we add the relevant query point together with another label to the training data, it would not be particularly surprising if the new classifier follows the new data. If we bring in a metric structure, we can exclude new training data which is too close to the given point.

Definition 9. *Fix a learner $L : (\mathbf{X} \times \mathbf{k})^* \to \mathcal{C}(\mathbf{X}, \mathbf{k}_\perp)$, some $N \in \mathbb{N}$ and $\varepsilon > 0$. We say that $(x_i, n_i)_{i \leq \ell}$ is* sparse *at $x \in \mathbf{X}$, if there are $(y_i, m_i)_{i \leq j}$ and $(y'_i, m'_i)_{i \leq j'}$ such that $\ell + N \geq j, j' \geq \ell$, $y_i = y'_i = x_i$ and $m_i = m'_i = n_i$ for $i \leq \ell$, and $d(y_i, x), d(y'_i, x) > \varepsilon$ for $i > \ell$ satisfying $\perp \neq L((y_i, m_i)_{i \leq j})(x) \neq L((y'_i, m'_i)_{i \leq j'})(x) \neq \perp$.*

We say that $(x_i, n_i)_{i \leq \ell}$ is dense *at $x \in \mathbf{X}$ if for all $(y_i, m_i)_{i \leq j}$ and $(y'_i, m'_i)_{i \leq j'}$ such that $\ell + N \geq j, j' \geq \ell$, $y_i = y'_i = x_i$ and $m_i = m'_i = n_i$ for $i \leq \ell$, and $d(y_i, x), d(y'_i, x) \geq \varepsilon$ for $i > \ell$ it holds that $L((y_i, m_i)_{i \leq j})(x) = L((y'_i, m'_i)_{i \leq j'})(x) \neq \perp$.*

To put it in words: Training data is dense at a point whose label it determines, even if we add up to N additional points to the training data, which have to be at least ε away from that point. Conversely, at a sparse point, we can achieve different labels by such an augmentation of the training data. If we have chosen the parameters N and ε well, then we can conclude that based on the training data we can make reasonable assertions about the dense query points, but unless we have some additional external knowledge of the true distribution of labels, we cannot draw reliable conclusion about the sparse query points. We concede that it would make sense to include points under *sparse* where the classifiers will always output $\perp$ even if we enhance the training data, but this would destroy any hope of nice algorithmic properties.

Theorem 2. *Let $\mathbf{X}$ be a computably compact computable metric space. The operation*

$$\mathrm{SprsOrDns} : \mathcal{C}((\mathbf{X} \times \mathbf{k})^*, \mathcal{C}(\mathbf{X}, \mathbf{k}_\perp)) \times \mathbb{N} \times \mathbb{R}_+ \times (\mathbf{X} \times \mathbf{k})^* \times \mathbf{X} \to \mathbf{2}_\perp$$

answering 0 on input L, N, ε, $(x_i, n_i)_{i \leq \ell}$ and x iff $(x_i, n_i)_{i \leq \ell}$ is sparse at x, and answers 1 if $(x_i, n_i)_{i \leq \ell}$ is dense at x is computable.

5 Related Work

For Neural Networks already in 2010, Pulina and Tachella presented an approach for verifying linear arithmetic constraints on multiplayer perceptions by translating them into SAT-instances [18]. A decade later, a systematic review on testing and verification of neural networks already covered 91 articles [24]. A survey focused on verification of deep neural networks is [11]. The focus here is on the operation we call forallValue (Proposition 2) and its generalization beyond classification tasks. The computation is carried out by taking into account the specific structure of the network and the use of piecewise linear activation functions, which allows for the treatment of regions as rational polytopes. While taking these details into account enables the development of efficient algorithms, it is somewhat disappointing if using sigmoidal activation functions (as necessary for the final activation in a binary classification setup) instead requires one to modify even the theoretical framework behind the verification approach. (Such a modification has been carried out using Taylor models in place of polytopes, and the Taylor expansion of the sigmoidal activation function [8]). Our approach is model agnostic, in particular, independent of small details such as the choice of activation functions.

Again for Neural Networks, a more general approach to decidability of verification questions starts with the observation that as long as we are using piecewise linear activation functions and specifications definable in the theory of real closed fields (i.e. quantified formulas involving $+$, $\times$ and $\leq$), we obtain decidability (i.e. yes/no-answers, no need for $\bot$) for free. This follows the theory of real closed fields and is decidable (albeit with infeasible complexity). This was remarked e.g. in [8]. If we want to ask questions involving a particular data set, we need to be able to define the data set in the theory of real closed fields. This seems like an awkward requirement for experimental data. In contrast, our theory is compatible with data obtained through imprecise measurements [15]. Extending the approach based on real-closed fields to sigmoidal activation functions seems to require the truth of Schanuel's conjecture [12]. Again, our approach is model agnostic.

Recently using probably approximately correct (PAC) learning theory (for background [20]), a study into the intermediate setting where learners are required to be computable but not resource-bounded has been achieved by [1]. They have developed a notion of a computable learner similar to ours. They used key concepts of a computable enumerable representable (CER) hypothesis class, along with an empirical risk minimization (ERM) learner. This allowed them to find an ERM learner that is computable on every CER class that is PAC learnable in the realizable case. However, verification questions are not considered in [1].

6 Summary and Future work

We motivated and presented a number of questions that one might want to ask when verifying classifiers obtained by ML. These include elementary questions such as whether any point in a region gets assigned a particular colour, but also more advanced ones such as whether adversarial examples exist. Finally, we make a contribution to the phenomenon of underspecification by studying the robustness of learners. Using the framework of computable analysis we are capable of precisely formalizing these questions, and to prove them to be computable under reasonable (and necessary) assumptions.

Regarding the necessity of the assumptions, we point out that dropping conditions, or considering maps providing more information instead, will generally lead to non-computability. We leave the provision of counterexamples, as well as potentially a classification of *how* non-computable these maps are to future work. The notion of a *maximal partial algorithm* recently proposed by Neumann [14] also seems a promising approach to prove optimality of our results.

There is a trade-off between the robustness of a classifier and its 'accuracy'. It seems possible to develop a computable quantitative notion of robustness for our function locallyConstant, which could then be used as part of the training process in a learner. This could be a next step to adversarial robustness [3,5].

Rather than just asking questions about particular given classifiers or learners, we could start with a preconception regarding what classifier we would want to obtain for given training data. Natural algorithmic questions then are whether there is a learner in the first place that is guaranteed to meet our criteria for the classifiers, and whether we can compute such a learner from the criteria.

Our choice to consider classifiers as the sole entities to be learned in the present paper is meant to keep verification questions simple. Our framework allows for straight-forward extensions to any desired broader setting.

References

1. Ackerman, N., Asilis, J., Di, J., Freer, C., Tristan, J.B.: On the computable learning of continuous features. Presentation at CCA 2021
2. de Brecht, M., Pauly, A.: Noetherian Quasi-Polish spaces. In: 26th EACSL Annual Conference on Computer Science Logic (CSL 2017). LIPIcs, vol. 82, pp. 16:1–16:17 (2017)
3. Carlini, N., Wagner, D.: Towards Evaluating the Robustness of Neural Networks. In: IEEE Symposium on Security and Privacy. pp. 39–57 (2017)
4. D'Amour, A., Heller, K., Moldovan, D., Adlam, B., Alipanahi, B., Beutel, A., Chen, C., Deaton, J., Eisenstein, J., Hoffman, M.D., Hormozdiari, F., Houlsby, N., Hou, S., Jerfel, G., Karthikesalingam, A., Lucic, M., Ma, Y., McLean, C., Mincu, D., Mitani, A., Montanari, A., Nado, Z., Natarajan, V., Nielson, C., Osborne, T.F., Raman, R., Ramasamy, K., Sayres, R., Schrouff, J., Seneviratne, M., Sequeira, S., Suresh, H., Veitch, V., Vladymyrov, M., Wang, X., Webster, K., Yadlowsky, S., Yun, T., Zhai, X., Sculley, D.: Underspecification presents challenges for credibility in modern machine learning. Journal of Machine Learning Research **23**(226), 1–61 (2022), `http://jmlr.org/papers/v23/20-1335.html`
5. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
6. Hearst, M.A., Dumais, S.T., Osuna, E., Platt, J., Scholkopf, B.: Support vector machines. IEEE Intelligent Systems and their applications **13**(4), 18–28 (1998)
7. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety Verification of Deep Neural Networks. In: International Conference on Computer Aided Verification. pp. 3–29 (2017)
8. Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verifying the safety of autonomous systems with neural network controllers. ACM Trans. Embed. Comput. Syst. **20**(1) (2020). `https://doi.org/10.1145/3419742`
9. Kwiatkowska, M.Z.: Safety Verification for Deep Neural Networks with Provable Guarantees (Invited Paper). In: Fokkink, W., van Glabbeek, R. (eds.) 30th International Conference on Concurrency Theory (CONCUR 2019). Leibniz International Proceedings in Informatics (LIPIcs), vol. 140, pp. 1:1–1:5. Schloss

Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2019), http://drops.dagstuhl.de/opus/volltexte/2019/10903

10. Leino, K., Wang, Z., Fredrikson, M.: Globally-Robust Neural Networks. In: Proceedings of the 38th International Conference on Machine Learning. pp. 6212–6222. PMLR (Jul 2021)
11. Liu, C., Arnon, T., Lazarus, C., Strong, C.A., Barrett, C.W., Kochenderfer, M.J.: Algorithms for verifying deep neural networks. Found. Trends Optim. **4**(3-4), 244–404 (2021). https://doi.org/10.1561/2400000035
12. Macintyre, A., Wilkie, A.J.: On the decidability of the real exponential field. In: Odifreddi, P. (ed.) Kreiseliana. About and Around Georg Kreisel, pp. 441–467. A K Peters (1996)
13. Morgan, J., Paiement, A., Pauly, A., Seisenberger, M.: Adaptive neighbourhoods for the discovery of adversarial examples. arXiv preprint arXiv:2101.09108 (2021)
14. Neumann, E.: Decision problems for linear recurrences involving arbitrary real numbers. Logical Methods in Computer Science (2021), https://arxiv.org/abs/2008.00583
15. Pauly, A.: Representing measurement results. Journal of Universal Computer Science **15**(6), 1280–1300 (2009)
16. Pauly, A.: On the topological aspects of the theory of represented spaces. Computability **5**(2), 159–180 (2016). https://doi.org/10.3233/COM-150049
17. Pauly, A.: Effective local compactness and the hyperspace of located sets. arXiv preprint arXiv:1903.05490 (2019)
18. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Proceedings of the 22nd International Conference on Computer Aided Verification. p. 243–257. CAV'10, Springer-Verlag, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_24
19. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence. pp. 2651–2659 (7 2018)
20. Shalev-Shwartz, S., Ben-David, S.: Understanding machine learning: From theory to algorithms. Cambridge university press (2014)
21. Skliris, V., Norman, M.R.K., Sutton, P.J.: Real-time detection of unmodelled gravitational-wave transients using convolutional neural networks (2020). https://doi.org/10.48550/ARXIV.2009.14611
22. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing Properties of Neural Networks. arXiv preprint arXiv:1312.6199 (2013)
23. Wicker, M., Huang, X., Kwiatkowska, M.: Feature-guided black-box safety testing of deep neural networks. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 408–426. Springer (2018)
24. Zhang, J., Li, J.: Testing and verification of neural-network-based safety-critical control software: A systematic literature review. Information and Software Technology **123**, 106296 (2020), https://www.sciencedirect.com/science/article/pii/S0950584920300471