Journal of Fourier Analysis and Applications (2025) 31:35
https://doi.org/10.1007/s00041-025-10163-w

Computability of Classification and Deep Learning: From Theoretical Limits to Practical Feasibility Through Quantization

Holger Boche[1,2] · **Vit Fojtik**[3,4] · **Adalbert Fono**[3] · **Gitta Kutyniok**[3,4,5,6]

Received: 6 August 2024 / Revised: 24 February 2025 / Accepted: 18 April 2025

Abstract

The unwavering success of deep learning in the past decade led to the increasing prevalence of deep learning methods in various application fields. However, the downsides of deep learning, most prominently its lack of trustworthiness, may not be compatible with safety-critical or high-responsibility applications requiring stricter performance guarantees. Recently, several instances of deep learning applications have been shown to be subject to theoretical limitations of computability, undermining the feasibility of performance guarantees when employed on real-world computers. We extend the findings by studying computability in the deep learning framework from two perspectives: From an application viewpoint in the context of classification problems and a general limitation viewpoint in the context of training neural networks. In particular,

Dedicated to Prof. Dr. Karlheinz Gröchenig on the occasion of his 65th birthday.

Communicated by Hans G. Feichtinger.

✉ Holger Boche
boche@tum.de

Vit Fojtik
fojtik@math.lmu.de

Adalbert Fono
fono@math.lmu.de

Gitta Kutyniok
kutyniok@math.lmu.de

1 Technical University of Munich, Munich, Germany

2 BMBF Research Hub 6G-life, Munich, Germany

3 Ludwig-Maximilians-Universität München, Munich, Germany

4 Munich Center for Machine Learning (MCML), Munich, Germany

5 University of Tromsø, Tromsø, Norway

6 DRL-German Aerospace Center, Cologne, Germany

Published online: 29 May 2025

we show restrictions on the algorithmic solvability of classification problems that also render the algorithmic detection of failure in computations in a general setting infeasible. Subsequently, we prove algorithmic limitations in training deep neural networks even in cases where the underlying problem is well-behaved. Finally, we end with a positive observation, showing that in quantized versions of classification and deep network training, computability restrictions do not arise or can be overcome to a certain degree.

Keywords Computability · Deep learning · Classification · Approximation theory · Quantization · Decision problems

Mathematics Subject Classification 68T07 · 68T05 · 03D80 · 65D15

1 Introduction

With the advent of deep learning [58, 59, 65, 66] a new machine learning approach materialized that provides state-of-the-art results in various tasks. The performance of deep neural networks makes them the go-to strategy to tackle a multitude of problems in relevant applications such as image classification, speech recognition, and game intelligence, as well as more recent developments such as image and sound synthesis, chat tools, and protein structure prediction, to name a few [1, 36, 49, 53, 57, 76, 79]. Although a wide range of literature supports the power of deep learning, such as the well-known Universal Approximation Theorems [33, 41, 50], its universality and success still lack theoretical underpinning. Moreover, despite its impressive performance deep learning typically comes at the cost of black-box behavior and non-interpretability, as well as instability, non-robustness, and susceptibility to adversarial manipulation [2, 3, 5, 25, 45, 48, 56, 73, 77, 87].

In certain applications, the highlighted drawbacks, informally summarized by a lack of trustworthiness [20, 39], are tolerable or even avoidable by a human-in-the-loop approach [84]. However, increasing the autonomy of deep learning systems without impairing their trustworthiness poses a great challenge, especially in safety-critical or high-responsibility tasks—a prime example being autonomous driving [61, 68] and autonomous agents in general [81]. Moreover, regulators have begun to set forth guidelines, most prominently the European AI Act [42], to mitigate unforeseen and undesired effects of deep learning by stressing the importance of trustworthiness via principles such as transparency, accountability, and right to explanation [37, 38]. Therefore, it is crucial for forthcoming deep learning methods to tackle and alleviate the lack of trustworthiness. To that end, we first need to understand whether or to what degree trustworthiness can be realized, e.g., is it feasible to ask for 'hard' performance guarantees or verifiably correct results? We study this question from the viewpoint of algorithmic computations with correctness guarantees and analyze whether trustworthiness can be established from a mathematical perspective.

1.1 Algorithmic Computability

Computability theory aims to mathematically model computation and answer questions about the algorithmic solvability and complexity of given problems. The most commonly studied model of computation is the Turing machine [78], which is an idealized version of real-world digital hardware neglecting time and space constraints. We distinguish between two different modes of computations-problems on continuous and discrete domains. The former typically represents an idealized scenario whereas the latter treats a setting closer to the actual realization of digital hardware.

1.1.1 Computability on Continuous Domains

In recent years, there has been an increased interest in the computability of continuous problems, studying the capabilities of inherently discrete digital computers when employed in the real domain. For instance, complex real-world problems—a typical application scenario for deep learning techniques due to their increasing capabilities—may be represented by a model with continuous state and parameter space despite the eventual implementation on digital hardware. However, some results indicate a nonconformity between these two realms, and limitations of two types can be identified: By Type 1 failure of computability we refer to the situation where the problem in question cannot be algorithmically solved with performance guarantees, i.e., the computed output comes without error bounds specifying the distance to the true solution, leaving room for potential failure. This has been found in diverse settings including inverse problems, optimization, information and communication theory, and financial mathematics [7, 15, 17–19, 24, 60]. On the other hand, in Type 2 failure a computable solver (with performance guarantees) may exist, but we cannot algorithmically learn it from data. There have not been many results in this direction, but it has been shown to occur in the context of inverse problems [29] and simple neural networks [60].

1.1.2 Quantization

While modeling problems abstractly in a framework of real numbers certainly has benefits, in many applications access to real-valued data and parameters with unlimited precision is unrealistic since physical measurements generally guarantee only some bounded accuracy. For these reasons, problems can be described by discretized models. A typical approach in practice is to perform real-valued computation under quantization, i.e., associating continuous ranges with a discrete set of values [6, 46, 52, 75]. For instance, quantized deep learning has been a topic of interest, comparing its theoretical and practical capabilities to non-quantized deep learning [64, 85]. Naturally, the question arises of how these quantization techniques affect the properties of algorithmic computations, including the limitations of computability on continuous domains. Hereby, the key property of quantization techniques and the resulting models is their amenability to classical computing theory (based on exact binary representations).

1.2 Our Contributions

This paper aims to extend the theory of computability in deep learning and highlight the importance of ground truth descriptions in questions of computability by providing insights from two perspectives.

- We analyze the field of classification tasks, an instance of a classic application field for deep learning, from the computability viewpoint and highlight in Propositions 3.3 and 3.5 that the computability of a classifier is equivalent to the semi-decidability of its classes. Since only specific real sets are semi-decidable, classification on the real domain is typically not algorithmically solvable, i.e., Type 1 failure of computability arises. This emphasizes the importance of studying the computability of a specific problem independent of the employed solution strategy: If a tackled problem has computability restrictions, one can not expect to circumvent them with a deep learning approach.
- Furthermore, we study the computable realizability of training neural networks, asking whether a function representable by a neural network can be exactly learned from data. Regarded as a parametric model, one key issue in deep learning is identifying suitable parameters, i.e., finding an appropriate deep learning model, to solve a given task. This search process—the so-called learning or training—is typically based on data samples and constitutes the main obstacle in successfully employing deep learning. Theorem 3.7 shows that no general learning algorithm applicable to all (real-valued) networks exists implying that Type 2 failure is unavoidable in this scenario, i.e., providing general performance guarantees is not feasible for the derived solver.
- We also consider strategies for coping with the introduced failures. We show it is impossible to predict when an algorithmic approximation of a non-computable problem will fail in Proposition 4.1 and Corollary 4.3. On the other hand, Theorems 4.4 and 4.5 indicate that computability limitations in the context of learning can be avoided by relaxing exactness requirements on the learned network.
- Finally—and perhaps most importantly—we show in Theorems 4.8 and 4.9 and Proposition 4.11 that when considering quantized versions of the previous settings, issues of non-computability do not arise. Proposition 4.12 provides a word of caution, stating that the quantization function itself is non-computable—we cannot algorithmically determine which quantized values faithfully represent the original (real-valued) problem.

Non-computability should not be understood as undermining the power of deep learning, which has been consistently demonstrated. Rather it provides a different perspective on fundamental problems bringing us closer to trustworthy deep learning methods in major applications such as autonomous decision-making and critical infrastructure. In this theme, we also discuss connections to autonomous agents and their ability to ask for human intervention to avoid failure. In particular, our non-computability results indicate that theoretical correctness guarantees typically cannot be provided in digital hardware computations of continuous problems, including deep learning, and that automated detection of these limitations can not be algorithmically guaranteed. However, our positive findings demonstrate that in certain specific

settings—primarily, if the problem domain does not reside in a continuous but discrete domain—correctness guarantees and trustworthiness can be established for deep learning algorithms in the digital computing framework. Hence, the ground truth description of problems has a decisive influence on the feasible performance guarantees.

1.3 Related Work

'Hardness' results in neural network training have a long tradition going back to the 90 s [14, 80], where it was already shown that the training process can be NP-complete for certain architectures. The learning of existing neural networks has also been studied in [12], where difficulties in the form of an explosion of required sample size were shown, rather than algorithmic intractability. Similar results concerning the sample complexity were established in the framework of statistical query algorithms in [26]. The infeasibility of algorithmically learning an existing computable neural network (Type 2 failure) in a continuous setting has been shown for the specific context of inverse problems in [29] and classification problems in [8]. Furthermore, [60] showed that no algorithm can reach near-optimal training loss on all possible datasets for simple neural networks. Further properties of deep learning from the computability perspective concerning adversarial attacks, implicit regularization, hardness of approximation, and reasoning were studied in [8, 10, 43, 83].

Obtaining guarantees for successfully accomplishing a computation is not restricted to deep learning and training. The need to understand if an algorithmic method behaves in an intended manner to establish trustworthiness is a well-established concept. Classical automated verification tools range from testing, i.e., trying to falsify a system on specific instances, to model checking, i.e., aiming to verify the model of a system against its formal specification [28]. Testing is also the commonly employed approach in deep learning, however, it is by design unable to establish global guarantees and thereby increase trustworthiness. Therefore, interpretability methods, which shed light on the inner workings of deep learning by trying to explain the decision-making of neural networks, have been proposed as a strategy to establish more trustworthy methods [54, 69, 73]. The most rigorous approach relies on verifying the accuracy and correctness of deep learning methods without explicitly tracing internal computations [13, 55, 67, 86]. However, the findings in [9, 20] indicate that certifying the accuracy and robustness of deep learning in the computability framework is challenging if at all possible—similar to the classical model checking strategies.

Exit flags provide a middle ground (between testing and verification) by indicating (potentially) incorrect outputs of a computation given the input instance. However, it was already observed that for optimization problems correct exit flag routines can not exist in general [7]. In contrast, [32] analyzed the feasibility of verification methods in a classification setting (on real numbers) and came to a positive conclusion. The difference to our negative result in the same setting is explained by the underlying approach, i.e., in [32] the question is posed whether formal verification systems can be implemented given a computable classification setting. We contrast this view by highlighting that this ideal scenario may not always be achievable. Additionally, we

also want to mention a different perspective on classification problems in [62], where a measure describing the stability of a classification function is presented and studied.

1.4 Outline

In Sect. 2, we introduce the applied formalisms, including computability theory and neural networks as the main workhorse of deep learning. We present our main results concerning Type 1 and Type 2 failure of computability in Sect. 3. We conclude in Sect. 4 by studying strategies to cope with computability failures, whereby quantization is a main theme. The proofs of the theorems are provided in Appendix B.

2 Notation and Definitions

We first introduce some basic concepts and notation used in the following.

2.1 Computability of Real Functions

We begin by reviewing definitions from real-valued computability theory necessary for our analysis. For a more comprehensive overview, see, for instance, [4, 72, 82]. We also omit elementary topics of computability theory such as recursive functions and Turing machines. Here we refer the reader to [30].

Previous results in applied computability on the real domain introduce many different, although partly equivalent, versions of computation and computability. The general paradigm describing digital computation on real numbers introduced by Turing himself [78] is based on the transformation of sequences approximating real numbers with arbitrary precision. It provides the tools to study the capabilities and limitations of perfect digital computing.

Definition 2.1 We define the following notions of computability:

- A sequence of rational numbers $(q_k)_{k=1}^{\infty}$ in $\mathbb{Q}$ is *computable* if there exist recursive functions $a, b, s : \mathbb{N} \to \mathbb{N}$ such that

$$q_k = (-1)^{s(k)} \frac{a(k)}{b(k)}.$$

- A rational sequence $(q_k)_{k=1}^{\infty}$ *converges effectively* to $x \in \mathbb{R}$, if there exists a recursive function $e : \mathbb{N} \to \mathbb{N}$ such that for all $k_0 \in \mathbb{N}$ and all $k \geq e(k_0)$

$$|x - q_k| \leq \frac{1}{2^{k_0}}.$$

- A real number $x \in \mathbb{R}$ is *computable* if there exists a computable rational sequence $(q_k)_{k=1}^{\infty}$ converging effectively to x. Such a sequence is called a *representation* (or a *rapidly converging Cauchy name*) of x. We denote the set of all computable reals by $\mathbb{R}_c$.

- A real vector $\boldsymbol{x} \in \mathbb{R}^d$ is *computable* if all of its components are computable numbers, that is, $\boldsymbol{x} = (x_1, \ldots, x_d)^\top$ for $x_1, \ldots, x_d \in \mathbb{R}_c$ We denote the set of all computable real vectors by $\mathbb{R}_c^d$.
- A real sequence $(x_k)_{k=1}^\infty$ in $\mathbb{R}$ is *computable* if there exists a computable double-indexed rational sequence $(q_{k,\ell})_{k,\ell=1}^\infty$ such that, for some recursive function $e : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and all $k, \ell_0 \in \mathbb{N}$ and $\ell \geq e(k, \ell_0)$, we have

$$\left|x_k - q_{k,\ell}\right| \leq \frac{1}{2^{\ell_0}}.$$

Remark 2.2 All previous definitions can be extended to $\mathbb{R}^d$ and $\mathbb{R}_c^d$ with $d > 1$ by requiring that each (one-dimensional) component or component-wise sequence is computable, respectively.

Out of the various definitions of a computable real function (see [4, Appendix 2.9] for an overview) we introduce two. Borel–Turing computability can be seen as the standard intuitive notion of computation, i.e., an algorithm approximating a given function to any desired accuracy exists. On the other hand, Banach–Mazur computability is the weakest common definition of computability meaning that a function is not computable in any usual sense if it is not Banach-Mazur computable.

Definition 2.3 Given $D \subset \mathbb{R}^d$, a function $f : D \to \mathbb{R}_c^m$ is

- *Borel-Turing computable* if there exists a Turing machine M such that, for all $\boldsymbol{x} \in D \cap \mathbb{R}_c^d$ and all representations $(\boldsymbol{q}_k)_{k=1}^\infty$ of $\boldsymbol{x}$, the sequence $(M(\boldsymbol{q}_k))_{k=1}^\infty$ is a representation of $f(\boldsymbol{x})$;
- *Banach-Mazur computable* if for all computable sequences $(\boldsymbol{x}_k)_{n=1}^\infty$ in $D \cap \mathbb{R}_c^d$ the sequence $(f(\boldsymbol{x}_k))_{k=1}^\infty$ is also computable.

Remark 2.4 For a Borel–Turing computable function, there exists a Turing machine taking a sequence of increasingly precise approximations of the input and producing increasingly accurate approximations of the output, whereas, a Banach-Mazur computable function only guarantees that it preserves the computability of real sequences. However, it is well known that all Borel-Turing computable functions are also Banach–Mazur computable, and computable functions in either sense are continuous—that is, continuous on $\mathbb{R}_c$ with the inherited topology [4]. For simplicity, we often refer to "computable" functions rather than "Borel-Turing computable", as this is our framework's standard version of computability. We explicitly specify whenever we apply the notion of Banach-Mazur computability.

The main theme of this paper revolves around the failure of algorithmic computations studied from the perspective of computability. In particular, we ask in what circumstances failures arise, and under what additional conditions they potentially can be avoided. Thereby, we associate in the real domain the more intuitive term "algorithm" with "Borel–Turing computable function" and we distinguish two cases of algorithmic failure:

- We say that a problem suffers from *Type 1 failure of computability* if it has no computable solver, that is, for any algorithm there exists an instance of the problem to which the algorithm is not guaranteed to provide a correct solution.

- A problem is subject to *Type 2 failure of computability* if a solver cannot be algorithmically found based on data, that is, for any learning algorithm Γ there exists a problem instance s such that for any dataset $\mathcal{X}$ the output $\Gamma(\mathcal{X})$ of the algorithm is not guaranteed to be a correct solver of s.

Note that Type 1 as a special case of Type 2 failure is more fundamental since a (computable) solution cannot be learned from data if it does not exist. However, we show in Subsection 3.2 that instances of Type 2 free of Type 1 failure exist in the context of training neural networks.

2.2 Neural Networks

In this paper, we restrict our attention to feedforward neural networks. For additional background on (deep) neural networks theory—usually referred to as deep learning—we point to [44]. We characterize neural networks by their structure, coined architecture, and the associated sequence of their parameters, i.e., their weight matrices and bias vectors.

Definition 2.5 Given $L \in \mathbb{N}$, an *architecture* of depth L is a vector $S := (N_0, N_1, \dots, N_{L-1}, N_L) \in \mathbb{N}^{L+1}$. A *neural network* with architecture S is a sequence of pairs of *weight matrices* and *bias vectors* $((A_\ell, \boldsymbol{b}_\ell))_{\ell=1}^{L}$ such that $A_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $\boldsymbol{b}_\ell \in \mathbb{R}^{N_\ell}$ for all $\ell = 1, \dots, L$. We denote the set of neural networks with architecture S by $\mathcal{NN}(S)$ and the total number of parameters in the architecture by $N(S) := \sum_{\ell=1}^{L}(N_\ell N_{\ell-1} + N_\ell)$.

Remark 2.6 Typically, we consider $b_L := \mathbf{0}$ and denote the input dimension $N_0 := d$. Also, throughout this paper, we focus on the case $N_L = 1$ for simplicity of presentation, even though the results can be reformulated for the general case.

The architecture and the parameter then induce the network's input–output function, the so-called realization.

Definition 2.7 For $\Phi \in \mathcal{NN}(S)$, $D \subset \mathbb{R}^{N_0}$, and $\sigma : \mathbb{R} \to \mathbb{R}$ denote by $R_\sigma^D(\Phi) : D \to \mathbb{R}^{N_L}$ the *realization* of the neural network Φ with *activation* σ and domain D, that is,

$$R_\sigma^D(\Phi) := T_L \circ \sigma \circ \cdots \circ \sigma \circ T_1|_D,$$

where $\Phi = ((A_\ell, \boldsymbol{b}_\ell))_{\ell=1}^{L}$ and $T_\ell(\boldsymbol{x}) := A_\ell \boldsymbol{x} + b_\ell$, $\ell = 1, \dots, L$.

Parameters of neural networks are almost always the result of a learning algorithm. Let us briefly recapitulate the learning process since it pertains to our discussion on computability. A learning algorithm receives as input a dataset of sample pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^{n}$, which are usually sampled from some underlying target function f, that is, $f(\boldsymbol{x}_i) = \boldsymbol{y}_i$, and aims to find a function $\hat{f}$ approximating f. In deep learning, we typically take $\hat{f} := R_\sigma^D(\Phi)$ for some neural network Φ with a fixed architecture S. The algorithm initializes the network with typically random parameters, followed by

an iterative optimization on a loss function $\mathcal{L} : \mathcal{NN}(S) \times \left(\mathbb{R}^{N_0} \times \mathbb{R}^{N_L}\right)^n \to \mathbb{R}$. A popular choice is the mean square error

$$\mathcal{L}(\Phi, \boldsymbol{x}_1, \boldsymbol{y}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{y}_n) := \frac{1}{n} \sum_{i=1}^{n} \left\| R_\sigma^D(\Phi)(\boldsymbol{x}_i) - \boldsymbol{y}_i \right\|^2,$$

where $\|\cdot\|$ indicates the Euclidean norm throughout the paper. Note that there exists a homeomorphism between neural networks with architecture S and their parameter space $\mathbb{R}^{N(S)}$, that is, $\mathcal{NN}(S) \approx \mathbb{R}^{N(S)}$. Thus we view the learning algorithm as a computable function $\Gamma : \mathbb{R}_c^{n(N_0+N_L)} \to \mathbb{R}_c^{N(S)}$, which receives a set of samples $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^n$ and returns weights and biases of the optimized network. More precisely, given rational sequences representing the real data the algorithm produces a sequence representing weights and biases. Therefore, the parameters of the resulting neural network will be computable numbers. To distinguish between networks with real and computable parameters, we introduce the notation $\mathcal{NN}_c(S)$ for the set of neural networks with architecture S and parameters in $\mathbb{R}_c$. Again, we can establish a homeomorphism between neural networks $\mathcal{NN}_c(S)$ and their parameters $\mathbb{R}_c^{N(S)}$ with the inherited topology. An important property of any network in $\mathcal{NN}_c(S)$ is that its realization is a computable function provided that the applied activation function is computable.

Finally, for ease of presentation of our analysis concerning Type 2 failure of neural networks, we apply the following concise form to describe sampled data sets.

Definition 2.8 Given $n \in \mathbb{N}$, $D \subset \mathbb{R}^d$ and $f : D \to \mathbb{R}^m$, we denote by $\mathcal{D}_{f,D}^n$ the set of all *datasets* of size n generated from f on the input domain D, that is,

$$\mathcal{D}_{f,D}^n := \left\{ (\boldsymbol{x}_i, f(\boldsymbol{x}_i))_{i=1}^n \in (\mathbb{R}^d \times \mathbb{R}^m)^n \mid \boldsymbol{x}_i \in D, i = 1, \ldots, n \right\}.$$

For a neural network $\Phi \in \mathcal{NN}(S)$ with activation σ we denote for short $\mathcal{D}_{\Phi,D}^n := \mathcal{D}_{R_\sigma^D(\Phi)|_D,D}^n$.

3 Computability Limitations

Our first goal is to study and, in fact, establish Type 1 and Type 2 failure for general problem descriptions, namely in classification and learning. Subsequently, we will analyze approaches to cope and ideally lessen the derived failures without compromising the generality of the considered problems. The two settings are chosen because of their importance in deep learning: Classification is one of the main objectives tackled by deep learning, whereas learning is one key component of the method. Having reliable, flexible, and universal learning algorithms hugely benefits the applicability of deep learning in various fields. Thus, classification and learning are suitable choices to highlight the consequences of computability failures.

3.1 Type 1 Failure in Classification

A classification problem is modeled by a function

$$f : D \to \{1, \dots, C\}, \qquad D \subset \mathbb{R}^d, \quad C \in \mathbb{N},$$

that assigns each input $\boldsymbol{x} \in D$ a corresponding class $c \in \{1, \dots, C\}$. A typical example is image classification where the input domain D is for instance given by $D = [0, 255]^{h \times w}$ with $[0, 255]$ and $h, w \in \mathbb{N}$ encoding color and size (height and width) of an image, respectively. The range $[0, 255]$ may also be quantized to obtain a discrete input domain $\{0, \dots, 255\}^{h \times w}$—a setting studied in Subsection 4.2 to assess the impact of the input domain on computability properties.

As described in Subsection 2.2, the goal of deep learning is to learn a function $\hat{f} : D \cap \mathbb{R}_c^d \to \{1, \dots, C\}$ based on samples $(\boldsymbol{x}_i, f(\boldsymbol{x}_i))_{i=1}^n$ in $D \times \{1, \dots, C\}$ such that $\hat{f}$ is close to f with respect to a suitable metric. Hence, a crucial question is whether $\hat{f}$ can be obtained from an algorithmic computation given a specific closeness condition. Equivalently, this question can be expressed in terms of (semi-)decidability on the input domain of f in $\mathbb{R}^d$ if $\hat{f}$ is expected to exactly emulate f, i.e., $\hat{f} = f|_{\mathbb{R}_c^d}$.

Definition 3.1 A set $A \subset D \cap \mathbb{R}_c^d$ is

- *decidable* in D, if its indicator function $1_A : D \cap \mathbb{R}_c^d \to \mathbb{R}_c$ is computable;
- *semi-decidable* in D, if there exists a computable function $f : D' \to \mathbb{R}_c$, $D' \subset D \cap \mathbb{R}_c^d$, such that $A \subset D'$ and $f = 1_A|_{D'}$.

Remark 3.2 The notion of (semi-)decidability can be formulated in terms of algorithms as follows: A set $A \subset D \cap \mathbb{R}_c^d$

- is Borel–Turing decidable in D if there exists a Turing machine M correctly determines, after finitely many steps, whether an input $\boldsymbol{x} \in D \cap \mathbb{R}_c^d$ belongs to A or its complement $D \setminus A$.
- is semi-decidable in D if there exists a Turing machine M correctly identifies all $\boldsymbol{x} \in A$ in finite time but may run indefinitely for $D \setminus A$.

Recall that computable functions are necessarily continuous on $\mathbb{R}_c^d$. Since indicator functions are discontinuous on $\mathbb{R}_c^d$ (excluding the trivial cases $\mathbb{R}_c^d$ and $\emptyset$), only sets of the type $\mathbb{R}_c^d \cup B$, $B \subset \mathbb{R}^d \setminus \mathbb{R}_c^d$, are decidable in $\mathbb{R}^d$. Therefore, decidability in $\mathbb{R}^d$ is a very restrictive notion that typically will not be satisfied by a classifier f. Regarding semi-decidability, the following equivalence is immediate and also provides a necessary condition for learning a perfect emulator $\hat{f} = f|_{\mathbb{R}_c^d}$ since computability is a prerequisite for learnability.

Proposition 3.3 *For a domain $D \subset \mathbb{R}^d$ and a function $f : D \to \{1, \dots, C\}$, the restriction $f|_{\mathbb{R}_c^d}$ is computable if and only if each set $f^{-1}(i)$, $i = 1, \dots, C$, is semi-decidable in D.*

Remark 3.4 Note that semi-decidability in D of all sets $f^{-1}(i)$, $i = 1, \dots, C$, implies decidability in D of each of the sets. Therefore, for f to be computable, D has to have

a specific structure. In particular, if $D \cap \mathbb{R}_c^d$ is a connected set homeomorphic to $\mathbb{R}_c^d$, e.g., $D = (0, 1)^d$, then f is not computable unless it is constant on $D \cap \mathbb{R}_c^d$. However, in this case, the classification problem is itself trivial. Thus, a necessary condition for computability is that the sets $f^{-1}(i)$ are separated to a certain degree.

Example A simple example for a computable classification problem with disconnected input domain is given by $f : D \to \{1, 2\}$ with $f^{-1}(1) = (0, 1)$, $f^{-1}(2) = (2, 3)$, and $D = f^{-1}(1) \cup f^{-1}(2)$. Here, a simple check of whether a given input is smaller or larger than 1.5 is sufficient to determine the associated class of the input.

These observations can be summarized in the following conclusion, where $\operatorname{dist}(A, B) = \inf_{a \in A, b \in B} \|a - b\|$ denotes the distance between sets A and B.

Proposition 3.5 *If for a function $f : D \to \{1, \dots, C\}$ there exist distinct $i, j \in \{1, \dots, C\}$ with $\operatorname{dist}(f^{-1}(i), f^{-1}(j)) = 0$, then $f|_{\mathbb{R}_c^d}$ is not computable.*

Remark 3.6 The results in Proposition 3.3 and 3.5 imply that Type 1 computability failure is unavoidable in sufficiently general classification problems. Indeed, a requested property in many applications is identifying inputs not associated with the given classes. Instances that represent a new class, not determined beforehand, or erroneous/distorted instances that cannot be unequivocally assigned to the given classes may be part of the feasible input set. Formally, we can express this setting by a classifier

$$\hat{f}' : D' \to \{1, \dots, C + 1\}, \quad D \subset D' \subset \mathbb{R}^d,$$

where D' is connected, such that $\hat{f}'(\boldsymbol{x}) = f(\boldsymbol{x})$ for $\boldsymbol{x} \in D$ and $\hat{f}'(\boldsymbol{x}) = C + 1$ otherwise. However, we immediately observe that f' is not computable and the desired property cannot be achieved.

3.2 Type 2 Failure in Deep Learning

Going beyond the previous context of classification, our next step is to study whether algorithmic solvability can be expected in the absence of Type 1 failure. We explore this setting in the context of training in deep learning independently of a concrete application and show that algorithmic solvability still can not be attained in general. The following theorem states that for any learning algorithm, there exist functions representable by computable neural networks (i.e., not suffering from Type 1 failure) that the algorithm cannot learn from data. This implies that there is no universal algorithm for training neural networks based on data, even when a correct (computable) solving network exists, i.e., deep learning suffers from Type 2 failure of computability. More precisely, for any learning algorithm Γ, there exists a computable neural network Φ such that given any training data set generated from Φ, the algorithm Γ cannot reconstruct any neural network with the same realization as Φ.

Theorem 3.7 *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a Lipschitz continuous, but not affine linear activation function, such that $\sigma|_{\mathbb{R}_c}$ is Banach-Mazur computable. Consider an architecture $S = (d, N_1, \dots, N_{L-1}, 1)$ of depth $L \geq 2$ with $N_1 \geq 3$, and let $D \subset \mathbb{R}_c^d$ be bounded with a nonempty interior.*

For any $\varepsilon > 0$, $n \in \mathbb{N}$, and any Banach-Mazur computable function $\Gamma : (\mathbb{R}_c^d \times \mathbb{R}_c)^n \to \mathbb{R}_c^{N(S)}$ there exists $\Phi \in \mathcal{NN}_c(S)$ such that for all $\mathcal{X} \in \mathcal{D}_{\Phi,D}^n$ and all $\Phi' \in \mathcal{NN}_c(S)$ satisfying $R_\sigma^D(\Phi') = R_\sigma^D(\Phi)$, we have

$$\left\|\Gamma(\mathcal{X}) - \Phi'\right\|_2 > \varepsilon. \tag{1}$$

Remark 3.8 The assumption that σ is not affine linear excludes none of the commonly used activations such as ReLU, tanh, or sigmoid. Only functions of the type $\sigma(t) = at + b$ are not permitted. Moreover, the computability assumption concerning σ is not restrictive since it guarantees a computable realization, a prerequisite for subsequent algorithmic evaluation in usage.

As a direct consequence, we cannot reconstruct the original input–output function.

Corollary 3.9 *Under the assumptions of Theorem 3.7, there exists no Banach-Mazur computable function $\Gamma : (\mathbb{R}_c^d \times \mathbb{R}_c)^n \to \mathbb{R}_c^{N(S)}$ for $n \in \mathbb{N}$ such that for all $\Phi \in \mathcal{NN}_c(S)$ there exists a dataset $\mathcal{X} \in \mathcal{D}_{\Phi,D}^n$ satisfying*

$$R_\sigma^D(\Gamma(\mathcal{X})) = R_\sigma^D(\Phi).$$

Note that the lower bound on the distance in (1) in Theorem 3.7 applies to the parameter space, i.e., weight space, of neural networks. It is known that networks with weights far apart can still represent functions close together [71]. Despite the failure of exact reconstruction obtained in Corollary 3.9, a neural network with the same architecture, which approximates the desired realization to an arbitrary degree, might still exist. We will return to this observation in Subsection 4.1.3, where we consider strategies for coping with non-computability by relaxing the exactness condition.

4 Strategies for Failure Circumvention

Can we overcome Type 1 and Type 2 limitations described in the previous section? We analyze different approaches to either reformulate or relax the tackled problems thus making them less amenable to computability failures. In particular, we explore two strategies. First, we study the effect of incorporating a reasonable error mode (depending on a given task) in the computation. Subsequently, we investigate the impact of moving the problem from the real to a discrete space via quantization.

4.1 Error Control Strategies

The requirement of exact emulation $\hat{f} = f|_{\mathbb{R}_c^d}$ of a classification function f or exact reconstruction of neural networks as analyzed in Sect. 3 may be too strict. In a practical setting, errors may be unavoidable or even acceptable to a certain degree. In particular, approximation of $f|_{\mathbb{R}_c^d}$ via $\hat{f}$ or reconstructing an approximate network based on an appropriate metric is a simpler task than exact emulation or reconstruction, respectively. However, in both cases, we certainly would like to have guarantees either in

the form of a description of the inputs that lead to deviations from the ground truth or via worst/average case error bounds. Whether and to what degree such guarantees are achievable is the subject of the following analysis.

4.1.1 Computable Unpredictability of Correctness in Type 1 Failure

A key observation in classification was that Type 1 failure, i.e., the non-semi-decidability of the classes, is closely associated with the decision boundaries of the classes. Informally speaking, the semi-decidability of classes hinges on the ability to algorithmically describe the decision boundary so that inputs on the decision boundary can be properly classified. Therefore, identifying these critical inputs or indicating that the computation for a given input may be inaccurate would certainly be beneficial. Is it possible to implement this identification—the so-called exit flag—algorithmically?

To study the posed question we do not restrict ourselves to classification functions but consider a slightly more general framework. We formalize the problem for general real-valued functions $f : D \to \mathbb{R}$, $D \subset \mathbb{R}^d$. Assume we are given a computable function $\hat{f} : D_c \to \mathbb{R}_c$, $D_c = D \cap \mathbb{R}_c^d$, typically constructed by an algorithmic method to approximate f. Our aim is to algorithmically identify inputs $\boldsymbol{x} \in D_c$ such that $\hat{f}$ satisfies

$$\|f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\| < \varepsilon \qquad \text{for given } \varepsilon > 0.$$

In other words, we ask if there exists an algorithm, i.e., a computable function, $\Gamma_\varepsilon : D_c \to \mathbb{R}_c$ such that

$$\Gamma_\varepsilon(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \|f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\| < \varepsilon, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

One can further relax the complexity of the task by demanding that an algorithm Γ_ε^+ only identifies inputs $\boldsymbol{x}$ for which $\hat{f}(\boldsymbol{x})$ satisfies the ε-closeness condition in (2), but does not necessarily indicate when it does not hold. For instance, $\Gamma_\varepsilon^+(\boldsymbol{x})$ may either output zero or not stop the computation in finite time on the given input $\boldsymbol{x}$ if $\|f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\| \geq \varepsilon$. If f is a computable function, we can construct Γ_ε for any $\varepsilon > 0$. Hence, more interesting problems arise when f is non-computable and ε appropriately small. Otherwise, choosing ε large enough, certainly still entails the existence of Γ_ε if f is, for instance, a bounded function. By associating Γ_ε and Γ_ε^+ with classification functions, we can apply Proposition 3.3 to derive the following result.

Proposition 4.1 *Let $f : D \to \mathbb{R}$, $D \subset \mathbb{R}^d$, and assume that $\hat{f} : D_c \to \mathbb{R}_c$, $D_c = D \cap \mathbb{R}_c^d$, is a computable function. Define for $\varepsilon > 0$ the set*

$$D_\varepsilon^< := \left\{ \boldsymbol{x} \in D_c \;\middle|\; \|f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\| < \varepsilon \right\}$$

Then the following holds:

1. *The function* $\Gamma_\varepsilon : D_c \to \mathbb{R}_c$ *given by*

$$\Gamma_\varepsilon(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \|f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\| < \varepsilon, \\ 0, & \text{otherwise,} \end{cases}$$

is computable if and only if $D_\varepsilon^<$ *is decidable in* D.
2. *A computable function* $\Gamma_\varepsilon^+ : D_c' \to \mathbb{R}$, $D_c' \subset D_c$, *such that* $D_\varepsilon^< \subset D_c'$ *and* $\Gamma_\varepsilon^+ = \Gamma_\varepsilon|_{D_c'}$ *exists if and only if* $D_\varepsilon^<$ *is semi-decidable in* D.

Remark 4.2 Depending on the context, we might be more interested in finding the set of inputs where $\hat{f}$ fails rather than succeeds. This would lead us to the analogous observation that the semi-decidability of the set

$$D_\varepsilon^\geq = \left\{ \boldsymbol{x} \in D_c \;\middle|\; \|f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\| \geq \varepsilon \right\}$$

determines computability of $\Gamma_\varepsilon^- : D_c' \to \mathbb{R}$, $D_c' \subset D_c$, given by $\Gamma_\varepsilon^- = \Gamma_\varepsilon|_{D_c'}$ with $D_\varepsilon^\geq \subset D_c'$.

For a connected input domain, applying Proposition 3.5 yields the following result, which is a direct consequence of the already mentioned fact that only trivial subsets of $\mathbb{R}_c^d$ are decidable.

Corollary 4.3 *Under the conditions of Proposition 4.1, additionally assume that* D *is connected. Then an approximator* $\hat{f}$ *such that* $D_\varepsilon^<$ *is decidable exists if and only if* f *can be computably approximated with precision* ε, *that is, if there exists a computable function* $\tilde{f}$ *such that*

$$\|f - \tilde{f}\|_\infty < \varepsilon.$$

Example A similar statement does not hold if $D_\varepsilon^<$ is only assumed to be semi-decidable. Consider the sign function $\operatorname{sgn} : \mathbb{R} \to \mathbb{R}$, which is non-continuous on $\mathbb{R}_c$ and therefore non-computable, and take $\widehat{\operatorname{sgn}}(x) = \frac{2}{\pi}\arctan(x)$. For a given $\varepsilon > 0$ we can computably construct intervals $(-\infty, -x_0)$ and (x_0, ∞) where $|\operatorname{sgn}(x) - \widehat{\operatorname{sgn}}(x)| < \varepsilon$, i.e., $D_\varepsilon^< = (-\infty, -x_0) \cup (x_0, \infty)$ is semi-decidable. In fact, we can adjust the approximator to achieve the desired precision on a given interval $(x_0, \infty), x_0 > 0$. However, due to the discontinuity at 0, no computable function approximating sgn on the entire real line with precision $\varepsilon < 1$ exists.

These results also directly apply to the classification setting as a special case of the considered framework. By design, the classification setting even allows for stronger statements regarding the magnitude of the error ε. In particular, requiring precision of $\varepsilon < \frac{1}{2}$ for the approximator $\hat{f}$ of a classifier f mapping to $\{1, \ldots, C\}$ is equivalent to requiring exact emulation $\hat{f} = f|_{\mathbb{R}_c^d}$. However, this scenario was already covered in Subsection 3.1, where Type 1 failure was established. Hence, we can conclude that exit flag computations may be beneficial in certain situations but they are not appropriate to tackle Type 1 failure in classification.

Instead of analyzing individual inputs, one could seek global guarantees for the approximator $\hat{f}$ such as measuring the size of the failure set, i.e., estimating the likelihood of error on a given domain. While interesting, this approach has two key limitations for our purposes. First, there exists no established algorithmic framework supporting theoretical analysis in this setting (see Appendix A for potential concepts). Second and more important, it primarily provides a global quantitative measure of failure, whereas we focus on local guarantees for specific inputs. Thus, this method offers insights into the problem behavior but does not address individual input limitations.

4.1.2 Broader Reflections

In a broader context, related research questions have been raised in fields such as artificial general intelligence by Daniel Kahneman [40] or robotics by Pieter Abbeel [31]: Can autonomous systems recognize when they cannot correctly solve a task or instance and request human assistance instead of making erroneous decisions? In other words: 'Do they know when they don't know?' [27, 74].

The infeasibility of general exit flag computations in Proposition 4.1, as well as previous results on practically relevant physical problems in [16], can be viewed as evidence on limitations of artificial general intelligence in this sense. In particular, it provides a negative answer to the above question in certain scenarios. For example, a self-driving car with the option of human assistance cannot be guaranteed to ask for intervention in cases where computability limits its performance. The main conclusion should not be that autonomous systems are generally failure-prone since it is unclear to what degree non-computability limits practical performance. Rather, our results entail that strict technical trustworthiness demands [39] as well as legal requirements (interpreted in a strict technical sense) [20] can not be met. In other words, algorithmic verification of trustworthiness may not be able to provide sufficient guarantees so different, less strict reliability measures, such as statistical testing, must suffice.

It is worth pointing out that our results are limited to Turing complete models of computation, i.e., models equivalent to Turing machines, such as (idealized) digital computers. Theoretical models have been proposed that surpass Turing computability (so-called *hypercomputation*), able to process infinite-precision real numbers, which can in some settings overcome classic computability limitations and provide in principle trustworthiness guarantees [20, 21]. However, all currently commercially used computing devices are (at most) Turing complete. For a discussion of the realizability of hypercomputation, we refer to [34, 63], and references therein.

4.1.3 Problem Relaxation for Type 2 Failure

In contrast to classification, relaxing the exact reconstruction requirement yields learning benefits. The main advantage is that the solution set of networks connected to approximate reconstruction is noticeably larger enabling algorithmic approaches to perform the previously unattainable reconstruction task. Type 2 failure does not arise in our learning setting on the training data if an approximation error is permitted.

Theorem 4.4 *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be such that $\sigma|_{\mathbb{R}_c}$ is computable and let $S = (d, N_1, \dots, N_{L-1}, 1)$ be an architecture.*

Then, for any $\varepsilon > 0$ and $n \in \mathbb{N}$, there exists a computable function $\Gamma : (\mathbb{R}_c^d \times \mathbb{R}_c)^n \to \mathbb{R}_c^{N(S)}$ such that for all $\Phi \in \mathcal{NN}_c(S)$ and $\mathcal{X} \in \mathcal{D}^n_{\Phi, \mathbb{R}_c^d}$ we have

$$\left| R_\sigma^{\mathbb{R}_c^d} \left(\Gamma(\mathcal{X}) \right) (\boldsymbol{x}) - y \right| < \varepsilon \quad \text{for } (\boldsymbol{x}, y) \in \mathcal{X}. \tag{3}$$

Although Theorem 4.4 provides a positive computability result concerning learning it still has certain limitations:

- The algorithm constructed to prove Theorem 4.4 serves only for theoretical analysis. It is typically not efficiently translatable into a practically usable one in a generic problem setting. Thus, a relevant and open question is whether more applicable learning algorithms can be constructed with similar guarantees.
- The learning algorithm Γ derived from Theorem 4.4 presupposes a fixed accuracy parameter ε and dataset size n, i.e., for different choices of these parameters a separate learning algorithm needs to be constructed.
- The reconstruction guarantee only holds on the training data. However, given access to the training data $\mathcal{X}$, the posed task could be solved without constructing a neural network since one could explicitly implement an algorithm that on the input $(\boldsymbol{x}, y) \in \mathcal{X}$ returns y. Constructing a neural network with some prescribed realization on the training data is expected to yield a network that performs 'reasonably well' on, i.e., generalizes to, unseen data. The learning and evaluation, as given in (3), should ideally be performed on different data to ensure this hypothesis.

We cannot alleviate the first but resolve the second issue. Indeed, the proof of Theorem 4.4 implies that one could generalize the algorithm Γ by requiring it to take (computable) ε and n as additional inputs. In particular, the key step of the proof relies on an enumeration argument that can be extended to incorporate ε and n as well. Moreover, in a slightly different setting, one can connect the desired accuracy with the input dimension and sample complexity [12].

Hence, the final issue left to consider is the generalization ability. By imposing further conditions on the admissible networks, we can indeed ensure certain generalization capabilities of the networks.

Theorem 4.5 *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be such that $\sigma|_{\mathbb{R}_c}$ is computable and Lipschitz continuous. Fix $A_{max} \in \mathbb{N}$ and consider an architecture $S = (d, N_1, \dots, N_{L-1}, 1)$.*

Then, for any $\varepsilon > 0$ and $n \in \mathbb{N}$, there exist computable functions $\Gamma : (\mathbb{R}_c^d \times \mathbb{R}_c)^n \to \mathbb{R}_c^{N(S)}$ and $\Psi : \mathbb{R}_c^{N(S)} \to \mathbb{R}_c^+$ such that for all $\Phi = ((A_\ell, \boldsymbol{b}_\ell))_{\ell=1}^L \in \mathcal{NN}_c(S)$ with weights uniformly bounded by A_{max} and $\mathcal{X} \in \mathcal{D}^n_{\Phi, \mathbb{R}_c^d}$ we have

$$\left| R_\sigma^{\mathbb{R}_c^d} \left(\Gamma(\mathcal{X}) \right) (\boldsymbol{x}) - y \right| < \varepsilon \quad \text{for } (\boldsymbol{x}, y) \in \mathcal{X}^\Phi_{\Psi(\Gamma(\mathcal{X}))}, \tag{4}$$

where $\mathcal{X}_r^\Phi := \{(\boldsymbol{x}, \Phi(\boldsymbol{x})) \mid \boldsymbol{x} \in \bigcup_{(\boldsymbol{x}_i, y_i) \in \mathcal{X}} B_r(\boldsymbol{x}_i)\}$.

Remark 4.6 For specific classes of neural networks, a uniform lower bound $\delta > 0$ on $\Psi(\Phi)$ over the set of considered networks Φ can be established. Therefore, it is possible to provide approximate reconstruction guarantees for the entire input domain D if $\mathcal{X}^{\Phi}_{\Psi(\Gamma(\mathcal{X}))}$ covers D. For instance, given an equidistant data grid on D with width δ, an approximate reconstruction guarantee holds for the entire domain.

The imposed conditions on data and networks are necessary—without them, guarantees like (4) in Theorem 4.5 cannot be ensured. Besides, Theorems 4.4 and 4.5 assume the existence of networks that realize the ground truth, limiting the remaining task to algorithmically finding them. A natural extension is to consider training data from an arbitrary ground truth function $g : \mathbb{R}^d \to \mathbb{R}$, which may not be realizable by a given neural network architecture. For arbitrary training samples $(\boldsymbol{x}_i, g(\boldsymbol{x}_i))_{i=1}^n$, one must first confirm that a neural network with the given architecture can approximate g sufficiently well to employ Theorems 4.4 and 4.5. Given sufficient conditions on the data and the ground truth function class, computability guarantees remain feasible by considering the architecture's expressivity, which is a well-studied field [11]. An alternative approach is to seek an optimal network minimizing a loss function on the training data. However, in [60], it was shown that this leads to Type 2 failure of computability, even for simple networks. One possible strategy to circumvent this issue is to relax the optimality requirement, but this problem warrants further investigation.

4.1.4 Broader Reflections

Lastly, we want to highlight the key differences between neural network training and general classification tasks. Why do the problems entail different degrees of computability failures? The crucial observation is that our considered model of neural networks does not cover the typical (more general) neural network model applied in classification. A classifier $\hat{f}$ as considered in Subsection 3.1 is a discontinuous function, whereas a neural network, assuming continuous activation, possesses a continuous realization. In the context of neural network classification, a classifier $\hat{f} = \hat{f}_1 \circ R^D_\sigma(\Phi)$ is composed of a neural network Φ—the so-called feature map—and a (discontinuous) function $\hat{f}_1$ mapping from the features to the classes. The Type 1 failure of computability appears due to $\hat{f}_1$, with no a priori restriction on computability of Φ, i.e., Type 2 failure on this level is avoidable.

4.2 Quantization Strategies

Due to real-world constraints, a simplified and quantized real number model is typically employed to implement digital computations in practice. In quantization, real numbers are approximated by a discrete set of rationals. For instance, under fixed-point quantization, real numbers are replaced by rational numbers with a fixed number k of decimal places in some base system b, i.e., algorithms strictly operate on the set $b^{-k}\mathbb{Z}$. Thus, assuming fixed-point quantization we can restrict the previous analysis without loss of generality to classification problems on $\mathbb{Z}^d$ as well as neural networks with integer parameters and data. The crucial difference between integer computability and the previously considered real-valued framework is the feasibility of exact computations

so that approximative computations are not inherently necessary. The concept of exact algorithmic computations on integers is described by *recursive functions* (which the previously considered framework of Borel-Turing computable functions extends to the real domain); we refer to [30] for more details on recursive functions and classical computability on discrete sets.

Interestingly, quantizing the parameter does not lead to a critical degradation of expressive power in neural networks. In particular, in the limit, the capabilities of quantized and real networks align [22, 35, 47]. Nevertheless, in the context of the simplest quantization technique, namely fixed-point quantization, the computability limitations introduced in Sect. 3 are alleviated to a certain degree. In particular, we establish that both Type 2 and Type 1 failures of computability are mainly resolved in this setting.

4.2.1 Computability of Quantized Deep Learning

We show that under fixed-point quantization, the negation of Theorem 3.7 holds. That is, an algorithm exists that can re-learn the exact realization of neural networks of fixed architecture on the training data. To that end, we introduce the set of neural networks with integer parameters.

Definition 4.7 Given an architecture S, we denote by $\mathcal{NN}_{\mathbb{Z}}(S) \subset \mathcal{NN}_c(S)$ the set of neural networks with architecture S and parameters in $\mathbb{Z}$.

Now, we can formulate the exact statement about re-learning neural networks.

Theorem 4.8 *For any $\sigma : \mathbb{R} \to \mathbb{R}$, any architecture $S = (d, N_1, \dots, N_{L-1}, 1)$, and all $n \in \mathbb{N}$, there exists a recursive function $\Gamma : (\mathbb{Z}^d \times \mathbb{Z})^n \to \mathbb{Z}^{N(S)}$ such that for all $\Phi \in \mathcal{NN}_{\mathbb{Z}}(S)$ there exists a dataset $\mathcal{X} \in \mathcal{D}^n_{\Phi,\mathbb{Z}^d}$ with*

$$R_\sigma^{\mathbb{Z}^d}(\Gamma(\mathcal{X})) = R_\sigma^{\mathbb{Z}^d}(\Phi).$$

Despite its positive result, we can still raise two main limitations of Theorem 4.8. First, the theory-to-practice gap in learning algorithms remains an (open) issue as in the previous analysis. Second, Theorem 4.8 only guarantees the existence of a dataset enabling reconstruction. Can we improve the statement to ensure reconstruction for any dataset satisfying some (weak) conditions? Before answering the question we want to point out a well-known fact: One cannot expect an exact reconstruction of a neural network's realization on the entire input domain based on an arbitrary but finite set of data samples in general. On the one hand, networks with different architecture or parameters may realize the same function, on the other hand, networks, whose outputs agree on some inputs, may wildly diverge in their realization [71].

Theorem 4.9 *For any $\sigma : \mathbb{R} \to \mathbb{R}$, such that $\sigma|_{\mathbb{Z}}$ is a recursive function, any architecture $S = (d, N_1, \dots, N_{L-1}, 1)$, and all $n \in \mathbb{N}$, there exists a recursive function $\Gamma : (\mathbb{Z}^d \times \mathbb{Z})^n \to \mathbb{Z}^{N(S)}$ such that for all $\Phi \in \mathcal{NN}_{\mathbb{Z}}(S)$ and $\mathcal{X} \in \mathcal{D}^n_{\Phi,\mathbb{Z}^d}$ we have*

$$R_\sigma^{\mathbb{Z}^d}(\Gamma(\mathcal{X}))(\boldsymbol{x}) = R_\sigma^{\mathbb{Z}^d}(\Phi)(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in \mathcal{X}.$$

Remark 4.10 From a data-centric perspective, Theorems 4.8 and 4.9 describe edge cases, i.e., guarantees applicable to any test data in the former (at the cost of flexibility in the training data) and guarantees for any training data (at the cost of flexibility in the test data). Similar to Subsection 4.1.3, one can extend Theorem 4.9 by imposing regularity conditions on the considered networks or relaxing the exactness condition to provide generalization bounds. Furthermore, if quantization yields a finite input domain one can trivially control the data set sizes to ensure exact generalization on the considered domain.

4.2.2 Computability of Quantized Classification

Turning our attention to classification and Type 1 failure, we can distinguish between two cases. In many applications, classification is performed on a bounded domain D such as in image classification described in Subsection 3.1. Hence, the quantized version of the input domain is finite so any integer-valued function on the quantized domain is computable-one can encode the input–output pairs directly in an algorithm.

Proposition 4.11 *If $D \subset \mathbb{Z}^d$ is bounded, then $f : D \to \{1, \dots, C\}$ is recursive.*

4.2.3 Broader Reflections

In contrast to Proposition 4.11, unbounded sets typically do not appear in practical quantized classification problems since they correspond to working on an infinite domain. However, in such a scenario we cannot provide formal guarantees on the computability of classifiers. Similar to the real case, the task reduces to classical (semi-)decidability of (infinite) sets of integers, which is not algorithmically solvable in general [30]. Hence, in both real and quantized classification Type 1 failure may arise due to non-(semi)-decidable sets in the respective frameworks. Nevertheless, the occurrence of Type 1 failure appears to diverge in the frameworks. Although it is intricate to derive a formal proof to back this statement, informally it is motivated by the observation that non-semi-decidability is a more severe drawback in the real domain. For instance, we have seen that non-trivial sets on $\mathbb{R}_c^d$ are generally not decidable whereas such a strong claim is not valid for the integer domain.

We have shown that quantization circumvents or at least mitigates Type 1 and Type 2 failure. Does it imply that in a real-world setting, where we typically compute with digital computers in the quantized model, Type 1 and Type 2 failures do not arise? Not necessarily, it depends on the ground truth problem. If the ground truth is itself quantized, then it typically can be directly translated into the quantized model of a digital computer and in principle algorithmically solved. In contrast, a ground truth problem on a continuous domain must first be converted into an appropriate quantized problem that approximates the original problem. Therefore, a crucial question is whether the quantization process of a given problem can be carried out algorithmically without computability failure. In particular, it would be desirable to provide computable guarantees that the obtained approximation is close to the original. However, for a non-computable ground truth problem such an algorithmic verification contradicts its

non-computability, as the ground truth problem could then be algorithmically computed/approximated using the verifier. Thus, quantization itself is a non-computable task, which is a direct consequence of Proposition 4.1.

Proposition 4.12 *Let $f : \mathbb{R} \to \mathbb{R}$ such that $f|_{\mathbb{R}_c}$ is not computable and define for all $x \in \mathbb{R}_c$*

$$\hat{f}(x) := f\left(\lceil x - \tfrac{1}{2} \rceil\right).$$

Then, there exists $\varepsilon_0 > 0$ such that for all $\varepsilon \leq \varepsilon_0$ the function $\Gamma_\varepsilon : \mathbb{R}_c \to \mathbb{R}_c$ given by

$$\Gamma_\varepsilon(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \left| f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}) \right| < \varepsilon, \\ 0, & \text{otherwise} \end{cases}$$

is not computable.

Appendix A (Semi-)decidability of Real Sets

In this section, we provide further background on the (semi-)decidability of subsets of real numbers. The related definitions can be found in Subsection 2.1, in particular Definitions 2.1 and 2.3. For more details, we refer to [23, 51, 70, 82, 88].

First, note that feasible notions of computability exist beyond Borel–Turing and Banach–Mazur computability. A common approach is to relax the computability requirements on the input domain. The underlying idea is to separate the mapping from the input description leading to the following definition of computable function, which we call oracle computability to distinguish it from the previous notions.

Definition A.1 *(Oracle model)* For $\boldsymbol{x} \in \mathbb{R}^d$, a sequence $(\boldsymbol{q}_k)_{k=1}^\infty$ in $\mathbb{Q}^d$ such that

$$\left\| \boldsymbol{x} - \boldsymbol{q}_k \right\| \leq \frac{1}{2^k} \quad \text{for all } n \in \mathbb{N},$$

is called an *oracle representation* of $\boldsymbol{x}$. A function $f : D \to \mathbb{R}_c^m$, where $D \subset \mathbb{R}^d$, is *oracle computable* if there exists an Oracle Turing machine M such that for all $\boldsymbol{x} \in D$ and all oracle representations $(\boldsymbol{q}_k)_{k=1}^\infty$ of $\boldsymbol{x}$ the sequence $(M(\boldsymbol{q}_k))_{k=1}^\infty$ is a representation of $f(\boldsymbol{x})$.

Remark A.2 Note that, unlike Borel–Turing computability, the representing rational sequence $(\boldsymbol{q}_k)_{k=1}^\infty$ is not required to be computable. By the density of $\mathbb{Q}$, any real number has an oracle representation and, therefore, we can study computability on the whole real line. Intuitively, one can think of the sequences $(\boldsymbol{q}_k)_{k=1}^\infty$ being provided to the Turing machine by an oracle tape; for an introduction on Oracle Turing machines see citecomputabilitybook. This model is more general, but in typical practical applications, the presence of an oracle able to approximate any real number to arbitrary precision cannot be assumed.

The differences in the computability notion (and the respective input domains) in comparison with Borel–Turing computability also directly transfer to the (semi-)decidability of sets: Only trivial subsets of $\mathbb{R}^d$ are oracle decidable—whereas for Borel-Turing decidability the same statement holds for $\mathbb{R}_c^d$.

Definition A.3 A set $A \subset D \cap \mathbb{R}^d$ is

- *oracle decidable* in D, if its indicator function $1_A : D \to \mathbb{R}_c$ is oracle computable;
- *oracle semi-decidable* in D, if there exists an oracle computable function $f : D' \to \mathbb{R}_c$, $D' \subset D$, such that $A \subset D'$ and $f = 1_A|_{D'}$.

Intuitively, oracle decidability, as well as Borel-Turing decidability, is infeasible in general (except for the trivial cases) since there does not exist an algorithm that decides on arbitrary input $x \in \mathbb{R}$ (via representations) whether $x = 0$, $x > 0$ or $x < 0$—the crucial input is the edge case zero [72]. Hence, the best one can hope for is a notion of decidability 'up to equality': Instead of relying on the characterization of (semi-)decidability via characteristic functions, one can consider the (continuous and computable) distance function $d_A : \mathbb{R}^d \to \mathbb{R}$ of $A \subset \mathbb{R}^d$ defined by

$$d_A(\boldsymbol{x}) := dist(\boldsymbol{x}, A) = \inf_{\boldsymbol{a} \in A} \|\boldsymbol{x} - \boldsymbol{a}\| .$$

The distance function allows for a given $\boldsymbol{x} \in \mathbb{R}^d$ to compute how close $\boldsymbol{x}$ lies to A although, in general, we cannot determine whether $\boldsymbol{x} \in A$ or $\boldsymbol{x} \notin A$. Therefore, a decidability notion based on the distance function does not lead to the existence of algorithms deciding membership for a given set even though closed sets are uniquely determined by their distance function.

Nevertheless, for subsets of natural numbers, one can derive an interesting connection between classical decidability and the distance function [23, 88].

Proposition A.4 *A subset $A \subset \mathbb{N}$ is decidable (in the classical sense), if and only if A considered as a subset of the real numbers induces a (oracle) computable distance function.*

A similar statement also holds for semi-decidable sets on $\mathbb{N}$. To that end, we introduce a specific characterization of oracle semi-decidable sets [82].

Theorem A.5 *Let $V \subset \mathbb{R}^d$. The following are equivalent:*

(i) V is oracle semi-decidable.

(ii) V is recursively enumerable open, i.e., there exists a Turing machine that can enumerate centers $\boldsymbol{c}_k \in \mathbb{Q}^d$ and radii $r_k \in \mathbb{Q}_{>0}$ of open balls such that

$$V = \bigcup_{k \in \mathbb{N}} B(\boldsymbol{c}_k, r_k), \tag{A1}$$

i.e., there exist computable rational sequences $(\boldsymbol{c}_k)_{k=1}^{\infty}$, $(r_k)_{k=1}^{\infty}$ such that (A1) holds.

Remark A.6 Any oracle semi-decidable set has a specific structure, in particular, it is necessarily open.

The observed equivalence also carries over to Borel-Turing semi-decidability by adjusting the expression in (A1) to

$$V \cap \mathbb{R}_c = \bigcup_{n\in\mathbb{N}} B(\boldsymbol{c}_n, r_n) \cap \mathbb{R}_c. \tag{A2}$$

To highlight the differences note that one can show under certain assumptions that an interval $(a, b) \subset \mathbb{R}$ with non-computable endpoints $a, b \in \mathbb{R}$ is oracle semi-decidable and thus also Borel–Turing semi-decidable. Moreover, $[a, b]$ can be Borel–Turing semi-decidable as well for specific choices of non-computable a, b (since $[a, b]\cap\mathbb{R}_c = (a, b) \cap \mathbb{R}_c$), whereas $[a, b]$ is not oracle semi-decidable as a closed set.

Proposition A.7 ([23, 88]) *A set $A \subset \mathbb{N}$ is semi-decidable (in the classical sense), if and only if A considered as a subset of the real numbers is recursively enumerable open.*

Finally, we want to summarize and highlight the conclusions based on the introduced statements. Due to the extended input domain, oracle (semi-)decidability is the stronger condition than Borel-Turing (semi-)decidability. However, in both frameworks decidability is not a practical notion on the real numbers due to the inability to decide equality. Although semi-decidability is less restrictive, it is still rather impractical since only open sets are amenable to semi-decidability, e.g., closed sets or sets that are neither open nor closed do not fit the framework. Nevertheless, (semi-)decidability on real domains based on the distance function recovers the classical theory of (semi-)decidability on natural numbers indicating that the introduced definitions are indeed the right ones. The difference between the two domains is that non-(semi-)decidability does not arise due to the inability to test equality on natural numbers and is therefore much scarcer in this setting. In comparison, non-semi-decidability on the real domain may not necessarily be related to the inability to test inequality, however, due to this shortcoming non-semi-decidability is likely to occur if no further assumptions are posed on the considered sets.

Hence, one might urge for more suitable notions of (semi-)decidability on the real numbers, which circumvent equality comparisons. For instance, a reliable description of 'near-decidability' indicates whether an object is in a set or not up to some limited error. Simply relying on the distance function does not immediately entail the desired property. In contrast, recursive approximability related to a measure μ [70] describes the following setting: Given a parameter $n \in \mathbb{N}$, there exists an algorithm that correctly decides $A \subset \mathbb{R}^d$ except on some set $B \subset \mathbb{R}^d$ with $\mu(B) < 2^{-n}$, in which case the algorithm still halts but with possibly incorrect output. Hence, there is a trade-off between correctness and guaranteed termination of the computation in finite time. Thus, the approach measures the possible error via μ. Borel–Turing semi-decidability on the other hand ensures that an algorithm always provides the correct output once it finalizes the computation but it may not stop for certain inputs. In other words, it only indicates whether a point is near another point that is not in the considered set. However, this information cannot be used to deduce whether the considered point lies inside or outside the set.

Further pursuing notions related to recursive approximability is certainly valuable and might lead to further insights; in this work, we consider the extension of the classical (semi-)decidability definitions that lead to oracle/Borel–Turing (semi-)decidability. These definitions describe the existence of effective (semi-)decision programs, i.e., algorithms that necessarily compute correct outputs (or do not halt their computations). In this sense, the output of the algorithm can be unequivocally trusted. Moreover, the theory as well as the results in Subsection 3.1 can be extended to spaces with less structure than $\mathbb{R}^d$, e.g., to computable metric spaces with $\mathbb{R}^d$ being a special case thereof [23, 51].

Appendix B Proofs

B.1 Proof of Theorem 3.7

The proof of Theorem 3.7 is based on two results we present next. The key component of the first lemma lies behind many non-computability results, such as in [29] for the special case of inverse problems, but here we formulate a general version.

Lemma B.1 *Let Θ be a nonempty set, Λ a nonempty set of functions from Θ to $\mathbb{R}_c$, $\varepsilon > 0$, and $\Xi : \Theta \to \mathcal{P}(\mathbb{R}_c^m)$, where $m \in \mathbb{N}$ and $\mathcal{P}$ denotes the power set. Assume there exist sequences $(\iota_k^1)_{k=1}^\infty$, $(\iota_k^2)_{k=1}^\infty$ in Θ satisfying*

(i) $\left|f(\iota_k^1) - f(\iota_k^2)\right| \to 0$ *uniformly in* $f \in \Lambda$. *That is,*

$$\forall \delta > 0\ \exists k_0 \in \mathbb{N}\ \forall k \geq k_0\ \forall f \in \Lambda : \left|f(\iota_k^1) - f(\iota_k^2)\right| < \delta;$$

(ii) for all $k \in \mathbb{N}$, $dist(\Xi(\iota_k^1), \Xi(\iota_k^2)) > \varepsilon$.

Then, for all $n \in \mathbb{N}$ and all Banach-Mazur computable functions $\Gamma : \mathbb{R}_c^n \to \mathbb{R}_c^m$ there exists $\iota \in \Theta$ such that for all $(f_1, \ldots, f_n) \in \Lambda^n$:

$$dist(\Gamma(f_1(\iota), \ldots, f_n(\iota)), \Xi(\iota)) > \frac{\varepsilon}{3}.$$

Proof For contradiction assume that for some $n \in \mathbb{N}$ there exists a Banach–Mazur computable function $\Gamma : \mathbb{R}_c^n \to \mathbb{R}_c^m$ such that for all $\iota \in \Theta$ there exists $(f_1, \ldots, f_n) \in \Lambda^n$ with

$$\operatorname{dist}(\Gamma(f_1(\iota), \ldots, f_n(\iota)), \Xi(\iota)) \leq \frac{\varepsilon}{3}. \tag{B3}$$

Since Γ is Banach–Mazur computable, it is continuous on $\mathbb{R}_c^n$ [82], that is,

$$\forall \eta > 0\ \exists \delta > 0\ \forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}_c^n : \|\boldsymbol{x}_1 - \boldsymbol{x}_2\| < \delta \Rightarrow \|\Gamma(\boldsymbol{x}_1) - \Gamma(\boldsymbol{x}_2)\| < \eta.$$

Take $\eta = \frac{\varepsilon}{3}$. For the corresponding δ there exists by condition (i). some $k \in \mathbb{N}$ such that for all $f \in \Lambda$ we have $\left|f(\iota_k^1) - f(\iota_k^2)\right| < \frac{\delta}{n}$. This implies for all $(f_1, \ldots, f_n) \in \Lambda^n$

that

$$\begin{aligned}\left\|(f_1(\iota_k^1),\dots,f_n(\iota_k^1))-(f_1(\iota_k^2),\dots,f_n(\iota_k^2))\right\| &= \sqrt{\sum_{i=1}^n \left(f_i(\iota_k^1)-f_i(\iota_k^2)\right)^2} \\ &\le \sum_{i=1}^n \sqrt{\left(f_i(\iota_k^1)-f_i(\iota_k^2)\right)^2} = \sum_{i=1}^n \left|f_i(\iota_k^1)-f_i(\iota_k^2)\right| < \delta,\end{aligned}$$

and therefore

$$\left\|\Gamma(f_1(\iota_k^1),\dots,f_n(\iota_k^1))-\Gamma(f_1(\iota_k^2),\dots,f_n(\iota_k^2))\right\| < \frac{\varepsilon}{3}.$$

Together with (B3) we get

$$\begin{aligned}\operatorname{dist}\left(\Xi(\iota_k^1),\Xi(\iota_k^2)\right) \le\ & \operatorname{dist}\left(\Gamma(f_1(\iota_k^1),\dots,f_n(\iota_k^1)),\Xi(\iota_k^1)\right) + \\ & \left\|\Gamma(f_1(\iota_k^1),\dots,f_n(\iota_k^1))-\Gamma(f_1(\iota_k^2),\dots,f_n(\iota_k^2))\right\| + \\ & \operatorname{dist}\left(\Gamma(f_1(\iota_k^2),\dots,f_n(\iota_k^2)),\Xi(\iota_k^2)\right) \\ <\ & 3\frac{\varepsilon}{3}=\varepsilon,\end{aligned}$$

which contradicts condition (ii). □

The following is a reformulation of Theorem 4.2 from [71], stating that there exist functions representable by neural networks that are arbitrarily close in the supremum norm but can only be represented by networks with weights arbitrarily far apart. The norm $\|\cdot\|_{\text{scaling}}$ on the (parameter) space of neural networks is used in the mentioned theorem because it provides a bound on the Lipschitz constant of neural network realizations $\operatorname{Lip}(R_\sigma^D(\cdot))$, i.e., $\operatorname{Lip}(R_\sigma^D(\Phi)) \le C\,\|\Phi\|_{\text{scaling}}$ for some $C > 0$ and a network Φ, thus connecting the parameter space and the function space.

Definition B.2 For a neural network $\Phi = ((A_\ell, \boldsymbol{b}_\ell))_{\ell=1}^L$ set

$$\|\Phi\|_{\text{scaling}} := \max_{1\le\ell\le L} \|A_\ell\|_{\max} = \max_{1\le\ell\le L} \max_{i,j} |(A_\ell)_{i,j}|.$$

Lemma B.3 ([71, Theorem 4.2]) *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be Lipschitz continuous, but not affine linear. Let $S = (d, N_1, \dots, N_{L-1}, 1)$ be an architecture of depth $L \ge 2$ with $N_1 \ge 3$. Let $D \subset \mathbb{R}^d$ be bounded with a nonempty interior. Then there exist sequences $(\Phi_k)_{k=1}^\infty, (\mu_k)_{k=1}^\infty$ in $\mathcal{NN}(S)$ such that*

(i) $\|R_\sigma^D(\Phi_k) - R_\sigma^D(\mu_k)\|_\infty \to 0$,
(ii) for any $(\Phi_k')_{k=1}^\infty, (\mu_k')_{k=1}^\infty$ in $\mathcal{NN}(S)$ with $R_\sigma^D(\Phi_k') = R_\sigma^D(\Phi_k)$ and $R_\sigma^D(\mu_k') = R_\sigma^D(\mu_k)$ for all $k \in \mathbb{N}$, it holds that $\left\|\Phi_k' - \mu_k'\right\|_{\text{scaling}} \to \infty$.

Remark B.4 It can be shown that the divergence in point (ii) is uniform in the following sense:

$$\begin{aligned}&\forall \varepsilon > 0\ \exists k_0\ \forall k \geq k_0\\ &\forall \Phi'_k, \mu'_k \in \mathcal{NN}(S) \text{ such that } R^D_\sigma(\Phi'_k) = R^D_\sigma(\Phi_k),\ R^D_\sigma(\mu'_k) = R^D_\sigma(\mu_k):\\ &\qquad \left\|\Phi'_k - \mu'_k\right\|_{\text{scaling}} > \varepsilon.\end{aligned}$$

To see this, assume $R^D_\sigma(\mu_k) \equiv 0$, and for contradiction let there be a subsequence $(\Phi'_{k_\ell})_{\ell=1}^\infty$ in $\mathcal{NN}(S)$ with $R^D_\sigma(\Phi'_{k_\ell}) = R^D_\sigma(\Phi_{k_\ell})$ and $\left\|\Phi'_{k_\ell}\right\|_{\text{scaling}} \leq \varepsilon$ for some $\varepsilon > 0$. Then for some $C > 0$:

$$\text{Lip}(R^D_\sigma(\Phi_{k_\ell})) = \text{Lip}(R^D_\sigma(\Phi'_{k_\ell})) \leq C \left\|\Phi'_{k_\ell}\right\|_{\text{scaling}} \leq C\varepsilon,$$

which contradicts $\text{Lip}\left(R^D_\sigma(\Phi_{k_\ell})\right) \to \infty$ in condition (ii).

From the proof in [71] it can also be seen that for a computable σ at least one such pair of these sequences of neural networks lies in $\mathcal{NN}_c(S)$.

Proof of Theorem 3.7 Let $\Theta = \left\{R^D_\sigma(\Phi) \mid \Phi \in \mathcal{NN}_c(S)\right\}$. For $i \in \{1, \ldots, d\}$ and $\boldsymbol{x} \in D$ denote by $f^i_{\boldsymbol{x}} : \Theta \to \mathbb{R}_c$ the constant operator

$$f^i_{\boldsymbol{x}}(g) = x_i$$

and by $f_{(\boldsymbol{x})} : \Theta \to \mathbb{R}_c$ the operator

$$f_{(\boldsymbol{x})}(g) = g(\boldsymbol{x}).$$

Let $\Lambda = \left\{f^i_{\boldsymbol{x}} \mid \boldsymbol{x} \in D, i \in \{1, \ldots, d\}\right\} \cup \left\{f_{(\boldsymbol{x})} \mid \boldsymbol{x} \in D\right\}$ and define $\Xi : \Theta \to \mathcal{P}(\mathbb{R}_c^{N(S)})$ by

$$\Xi(g) = \left\{\Phi \mid R^D_\sigma(\Phi) = g\right\}.$$

By Lemma B.3 there exists a pair of sequences $(g_k)_{k=1}^\infty, (h_k)_{k=1}^\infty$ in Θ such that $\|g_k - h_k\|_\infty \to 0$. Therefore also $\left|f_{(\boldsymbol{x})}(g_k) - f_{(\boldsymbol{x})}(h_k)\right| \to 0$ uniformly in $\boldsymbol{x} \in D$. The same trivially holds for all $f^i_{\boldsymbol{x}}$, therefore condition (i). of Lemma B.1 is satisfied.

By Remark B.4, the sequences diverge uniformly in the scaling norm and therefore also in the Euclidean norm, meaning $\text{dist}(\Xi(g_k), \Xi(h_k)) \to \infty$ and, in particular, for any $\varepsilon > 0$ there exists k_0 such that $\text{dist}(\Xi(g_k), \Xi(h_k)) > 3\varepsilon$ for $k \geq k_0$. Hence, condition (ii). of Lemma B.1 holds with 3ε.

Together, by Lemma B.1 for all $n \in \mathbb{N}$ and all Banach-Mazur computable functions $\Gamma : (\mathbb{R}_c^d \times \mathbb{R}_c)^n \to \mathbb{R}_c^{N(S)}$ there exists $g \in \Theta$, such that for all $\left(f_1, \ldots, f_{n(d+1)}\right) \in \Lambda^{n(d+1)}$ we have

$$\text{dist}\left(\Gamma(f_1(g), \ldots, f_{n(d+1)}(g)), \Xi(g)\right) > \varepsilon.$$

However, by construction of Λ and Ξ, this entails that there exists $\Phi \in \Xi^{-1}(g)$, i.e., $\Phi \in \mathcal{NN}_c(S)$, such that for all $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n \in D$ and all $\Phi' \in \mathcal{NN}_c(S)$ with $R_\sigma^D(\Phi') = R_\sigma^D(\Phi) = g$ we have

$$\left\| \Gamma(\mathcal{X}) - \Phi' \right\|_2 > \varepsilon.$$

□

B.2 Proof of Theorem 4.4 and 4.5

The key component of the proofs in this section relies on enumerating rational neural networks, i.e., networks with rational parameters, and subsequently controlling the error induced thereby.

Proof of Theorem 4.4 First, enumerate the countable set of rational neural networks $\{\Phi_1, \Phi_2, \dots\}$ of the given architecture, in particular, we can associate $\mathbb{Q}^{N(S)}$ with $\{\Phi_1, \Phi_2, \dots\}$. For all $\hat{\Phi}, \Phi^* \in \mathcal{NN}_c(S)$, $\hat{\Phi}$ is a computable function so that

$$g_{\hat{\Phi}, \Phi^*}(\boldsymbol{x}) := \left| R_\sigma^{\mathbb{R}_c^d}(\hat{\Phi})(\boldsymbol{x}) - R_\sigma^{\mathbb{R}_c^d}(\Phi^*)(\boldsymbol{x}) \right|$$

is computable. Assume the training data $\mathcal{X} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ was generated by a neural network Φ. Next, we construct an algorithm that correctly recognizes whether $g_{\Phi_k, \Phi}(\boldsymbol{x}_i) < \varepsilon$ for all $i = 1, \dots, n$: Compute $g_{\Phi_k, \Phi}(\boldsymbol{x}_i)$ with precision (at least) $\frac{1}{2}\varepsilon$ and subsequently check whether the magnitude of the obtained (rational) number is smaller than $\frac{1}{2}\varepsilon$. If so, the algorithm returns Φ_k, if not, it continues by increasing $k \in \mathbb{N}$.

Moreover, by the density of rational networks, there exists a rational network Φ_{k_0} such that for all $i = 1, \dots, n$: $g_{\Phi_{k_0}, \Phi}(\boldsymbol{x}_i) < \frac{1}{2}\varepsilon$. Hence, the algorithm terminates not later than the k_0-th iteration, returning a correct answer. This characterizes a computable function Γ satisfying the claim. □

Extending the proof by incorporating the additional conditions yields Theorem 4.5.

Proof of Theorem 4.5 Fix some architecture S. First, observe that for arbitrary $\Phi, \hat{\Phi} \in \mathcal{NN}_c(S)$ and $\mathcal{X} \in \mathcal{D}^n_{\Phi, \mathbb{R}_c^d}$ the following holds:

$$\begin{aligned} |\Phi(\boldsymbol{x}) - \hat{\Phi}(\boldsymbol{x})| &\leq |\Phi(\boldsymbol{x}) - \Phi(\hat{\boldsymbol{x}})| + |\Phi(\hat{\boldsymbol{x}}) - \hat{\Phi}(\hat{\boldsymbol{x}})| + |\hat{\Phi}(\hat{\boldsymbol{x}}) - \hat{\Phi}(\boldsymbol{x})| \\ &\leq \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| (\mathrm{Lip}(R_\sigma^D(\Phi)) + \mathrm{Lip}(R_\sigma^D(\hat{\Phi}))) + |\Phi(\hat{\boldsymbol{x}}) - \hat{\Phi}(\hat{\boldsymbol{x}})|, \end{aligned}$$

where $\boldsymbol{x} \in D$ and $\hat{\boldsymbol{x}} = \mathrm{argmin}_{\{\boldsymbol{x}_i : (\boldsymbol{x}_i, y_i) \in \mathcal{X}\}} \|\boldsymbol{x} - \boldsymbol{x}_i\|$. Therefore, using Definition B.2 we get for arbitrary $r > 0$ and $(\boldsymbol{x}, y) \in \mathcal{X}_r^\Phi$

$$|y - \hat{\Phi}(\boldsymbol{x})| \leq r C(\sigma, S)(\|\Phi\|_{\text{scaling}} + \|\hat{\Phi}\|_{\text{scaling}}) + |\Phi(\hat{\boldsymbol{x}}) - \hat{\Phi}(\hat{\boldsymbol{x}})|,$$

where $C(\sigma, S) > 0$ is a computable constant depending on the architecture and the activation function. Hence, applying Theorem 4.4 shows that for all $\tilde{\varepsilon} > 0$ and $n \in \mathbb{N}$ there exists a computable function $\Gamma_{\tilde{\varepsilon}} : (\mathbb{R}_c^d \times \mathbb{R}_c)^n \to \mathbb{R}_c^{N(S)}$ such that for all $\Phi \in \mathcal{NN}_c(S)$ and $\mathcal{X} \in \mathcal{D}^n_{\Phi, \mathbb{R}_c^d}$ we have

$$\left| R_\sigma^{\mathbb{R}_c^d} \left(\Gamma_{\tilde{\varepsilon}}(\mathcal{X})\right)(\boldsymbol{x}) - y \right| < rC(\sigma, S)(\|\Phi\|_{\text{scaling}} + \|\Gamma_{\tilde{\varepsilon}}(\mathcal{X})\|_{\text{scaling}}) + \tilde{\varepsilon} \quad \text{for } (\boldsymbol{x}, y) \in \mathcal{X}_r^\Phi.$$

Restricting to input networks Φ with $\|\Phi\|_{\text{scaling}} \leq A_{\max}$ and setting

$$r^* = \frac{\varepsilon - \tilde{\varepsilon}}{C(\sigma, S)(A_{\max} + \|\Gamma_{\tilde{\varepsilon}}(\mathcal{X})\|_{\text{scaling}})} \quad \text{for some } \varepsilon > \tilde{\varepsilon}$$

gives

$$\left| R_\sigma^{\mathbb{R}_c^d} \left(\Gamma_{\tilde{\varepsilon}}(\mathcal{X})\right)(\boldsymbol{x}) - y \right| < \varepsilon \quad \text{for } (\boldsymbol{x}, y) \in \mathcal{X}_{r^*}^\Phi.$$

Finally, for given $\varepsilon > 0$ and $n \in \mathbb{N}$, set $\Gamma = \Gamma_{\frac{1}{2}\varepsilon}$ and define $\Psi : \mathbb{R}_c^{N(S)} \to \mathbb{R}_c$ by

$$\Psi(\Phi) = \frac{\varepsilon}{2C(\sigma, S)(A_{\max} + \|\Phi\|_{\text{scaling}})}.$$

Observing that Γ satisfies (4) and Ψ is a computable function (since $C(\sigma, S)$ and $\|\cdot\|_{\text{scaling}}$ are computable, and we may assume without loss of generality that ε and $A_{\max}$ are computable) gives the claim. □

B.3 Proof of Theorem 4.8 and 4.9

An enumeration argument similar to the ones in the previous proofs implies Theorem 4.8. In particular, the idea is to encode the target network as a single datapoint, which can be done recursively for integer vectors representing neural networks with integer parameters.

Proof of Theorem 4.8 Given an architecture $S = (d, N_1, \ldots, N_{L-1}, 1)$, $\mathcal{NN}_\mathbb{Z}(S)$ can be associated with $\mathbb{Z}^{N(S)}$, which in turn can be recursively encoded into $\mathbb{Z}^d$ by a recursively invertible function $g : \mathbb{Z}^{N(S)} \to \mathbb{Z}^d$ (see for instance [30] for details). Then, taking $\Gamma(\mathcal{X}) = g^{-1}(\boldsymbol{x}_1)$ with $\mathcal{X} = \{(\boldsymbol{x}_1, y_1) \ldots, (\boldsymbol{x}_n, y_n)\}$, a single datapoint of the form $\left(g(\Phi), R_\sigma^{\mathbb{Z}^d}(\Phi)(g(\Phi)), 0, \ldots\right) \in (\mathbb{Z}^d \times \mathbb{Z})^n$ can be used to reconstruct any neural network $\Phi \in \mathcal{NN}_\mathbb{Z}(S)$. Here we utilize the fact, that we can choose the dataset for each network specifically. □

By taking the training data more explicitly into account Theorem 4.9 follows.

Proof of Theorem 4.9 Given a dataset $\mathcal{X} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \in \mathcal{D}^n_{\Phi, \mathbb{Z}^d}$, enumerate the countable set of all neural networks $\{\Phi_1, \Phi_2, \ldots\}$ with a given architecture S, in

particular, we can associate $\mathbb{Z}^{N(S)}$ with $\{\Phi_1, \Phi_2, \dots\}$. For increasing $k \in \mathbb{N}$, check whether for all $i = 1, \dots, n$: $R_\sigma^{\mathbb{Z}^d}(\Phi_k)(\boldsymbol{x}_i) = y_i$. If so, return Φ_k, if not, continue.

If the data was generated using a neural network $\Phi = \Phi_{k_0}$, then the algorithm terminates at the latest in the k_0-th iteration, returning a correct answer. This characterizes a (partially) recursive function Γ satisfying the theorem. □

Funding Open Access funding enabled and organized by Projekt DEAL. Holger Boche acknowledges the financial support by the BMBF in the programme of "Souverän. Digital. Vernetzt", research HUB 6G-life, project identification number: 16KISK002, and the financial support by the BMBF Quantum Projects QUIET, Grant 16KISQ093, QD-CamNetz, Grant 16KISQ077, and QuaPhySI, Grant 16KIS1598K. H. Boche was also partially supported by the project "Next Generation AI Computing (gAIn)", funded by the Bavarian Ministry of Science and the Arts and the Saxon Ministry for Science, Culture, and Tourism. This work of V. Fojtik was supported by the the Munich Center for Machine Learning (MCML). This work of G. Kutyniok was supported in part by the Konrad Zuse School of Excellence in Reliable AI (DAAD), the Munich Center for Machine Learning (MCML) as well as the German Research Foundation under Grants DFG-SPP-2298, KU 1446/31-1 and KU 1446/32-1. Furthermore, G. Kutyniok acknowledges additional support by the project "Next Generation AI Computing (gAIn)", funded by the Bavarian Ministry of Science and the Arts and the Saxon Ministry for Science, Culture, and Tourism.

References

1. Achiam, J., Adler, S., Agarwal, S., et al.: GPT-4 Technical Report. arXiv:2303.08774 (2023)
2. Adcock, B., Dexter, N.: The gap between theory and practice in function approximation with deep neural networks. SIAM J. Math. Data Sci. **3**(2), 624–655 (2021). https://doi.org/10.1137/20M131309X
3. Antun, V., Renna, F., Poon, C., et al.: On instabilities of deep learning in image reconstruction and the potential costs of AI. Proc. Natl. Acad. Sci. **117**(48), 30088–30095 (2020). https://doi.org/10.1073/pnas.1907377117
4. Avigad, J., Brattka, V.: Computability and analysis: the legacy of Alan Turing, In: Lecture Notes in Logic, p. 1–47. Cambridge University Press, Cambridge, (2014)
5. Bach, S., Binder, A., Montavon, G., et al.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PlOS ONE (2015). https://doi.org/10.1371/journal.pone.0130140
6. Bar-Shalom, O., Weiss, A.J.: DOA estimation using one-bit quantized measurements. IEEE Trans. Aerosp. Electron. Syst. **38**(3), 868–884 (2002). https://doi.org/10.1109/TAES.2002.1039405
7. Bastounis, A., Hansen, A.C., Vlačić, V.: The extended Smale's 9th problem—on computational barriers and paradoxes in estimation, regularisation, computer-assisted proofs and learning. arXiv:2110.15734 (2021a)
8. Bastounis, A., Hansen, A.C., Vlačić, V.: The mathematics of adversarial attacks in AI—why deep learning is unstable despite the existence of stable neural networks. arxiv:2109.06098 (2021b)
9. Bastounis, A., Gorban, A.N., Hansen, A.C., et al.: The boundaries of verifiable accuracy, robustness, and generalisation in deep learning. In: Iliadis, L., Papaleonidas, A., Angelov, P., et al. (eds.) Artificial Neural Networks and Machine Learning-ICANN 2023, pp. 530–541. Springer, Cham (2023)
10. Bastounis, A., Campodonico, P., van der Schaar, M., et al.: On the consistent reasoning paradox of intelligence and optimal trust in AI: the power of 'I don't know'. arXiv:2408.02357 (2024)

11. Berner, J., Grohs, P., Kutyniok, G., et al.: The modern mathematics of deep learning. In: Mathematical Aspects of Deep Learning. Cambridge University Press, (2022a). https://doi.org/10.1017/9781009025096.002
12. Berner, J., Grohs, P., Voigtlaender, F.: Learning ReLU networks to high uniform accuracy is intractable. arXiv:2205.13531 (2022b)
13. Biondi, A., Nesti, F., Cicero, G., et al.: A safe, secure, and predictable software architecture for deep learning in safety-critical systems. IEEE Embed. Syst. Lett. **12**(3), 78–82 (2020). https://doi.org/10.1109/LES.2019.2953253
14. Blum, A.L., Rivest, R.L.: Training a 3-node neural network is NP-complete. Neural Netw. **5**(1), 117–127 (1992). https://doi.org/10.1016/S0893-6080(05)80010-3
15. Boche, H., Mönich, U.J.: Turing computability of Fourier transforms of bandlimited and discrete signals. IEEE Trans. Signal Process. **68**, 532–547 (2020). https://doi.org/10.1109/TSP.2020.2964204
16. Boche, H., Pohl, V.: On non-detectability of non-computability and the degree of non-computability of solutions of circuit and wave equations on digital computers. IEEE Trans. Inf. Theory **68**(8), 5561–5578 (2022). https://doi.org/10.1109/TIT.2022.3172837
17. Boche, H., Schaefer, R.F., Poor, H.V.: Denial-of-service attacks on communication systems: detectability and jammer knowledge. IEEE Trans. Signal Process. **68**, 3754–3768 (2020). https://doi.org/10.1109/TSP.2020.2993165
18. Boche, H., Fono, A., Kutyniok, G.: Limitations of deep learning for inverse problems on digital hardware. IEEE Trans. Inf. Theory **69**(12), 7887–7908 (2023). https://doi.org/10.1109/TIT.2023.3326879
19. Boche, H., Schaefer, R.F., Poor, H.V.: Algorithmic computability and approximability of capacity-achieving input distributions. IEEE Trans. Inf. Theory **69**(9), 5449–5462 (2023). https://doi.org/10.1109/TIT.2023.3278705
20. Boche, H., Fono, A., Kutyniok, G.: Mathematical algorithm design for deep learning under societal and judicial constraints: the algorithmic transparency requirement. arXiv:2401.10310 (2024)
21. Boche, H., Fono, A., Kutyniok, G.: Inverse problems are solvable on real number signal processing hardware. Appl. Comput. Harmon. Anal. (2025). https://doi.org/10.1016/j.acha.2024.101719
22. Bolcskei, H., Grohs, P., Kutyniok, G., et al.: Optimal approximation with sparsely connected deep neural networks. SIAM J. Math. Data Sci. **1**(1), 8–45 (2019). https://doi.org/10.1137/18M118709X
23. Brattka, V., Presser, G.: Computability on subsets of metric spaces. Theor. Comput. Sci. **305**(1), 43–76 (2003). https://doi.org/10.1016/S0304-3975(02)00693-X
24. Brattka, V., Ziegler, M.: Turing computability of (non-)linear optimization. In: Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG'01) (2001)
25. Chang, C.H., Creager, E., Goldenberg, A., et al.: Explaining image classifiers by counterfactual generation. arXiv:1807.08024 (2018)
26. Chen, S., Gollakota, A., Klivans, A., et al.: Hardness of noise-free learning for two-hidden-layer neural networks. Adv. Neural. Inf. Process. Syst. **35**, 10709–10724 (2022)
27. Cheng, Q., Sun, T., Liu, X., et al.: Can AI assistants know what they don't know? In: Proceedings of the 41st International Conference on Machine Learning. JMLR.org (2024)
28. Clarke, E., Grumberg, O., Peled, D., et al.: Model checking. In: The Cyber-Physical Systems Series. MIT Press (1999)
29. Colbrook, M.J., Antun, V., Hansen, A.C.: The difficulty of computing stable and accurate neural networks: on the barriers of deep learning and Smale's 18th problem. Proc. Natl. Acad. Sci. **119**(12), e2107151119 (2022). https://doi.org/10.1073/pnas.2107151119
30. Cooper, S.B.: Computability Theory. Chapman and Hall/CRC, New York (2017)
31. Covariant: AI Robotics for the Real World. https://www.youtube.com/watch?v=AAr99hQ64AI (2021)
32. Crook, T., Morgan, J., Pauly, A., et al.: A computability perspective on (verified) machine learning. In: Madeira, A., Martins, M.A. (eds.) Recent Trends in Algebraic Development Techniques, pp. 63–80. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-43345-0_3
33. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst. **2**(4), 303–314 (1989). https://doi.org/10.1007/BF02551274
34. Davis, M.: Why there is no such discipline as hypercomputation. Appl. Math. Comput. **178**(1), 4–7 (2006)
35. Elbrächter, D., Perekrestenko, D., Grohs, P., et al.: Deep neural network approximation theory. IEEE Trans. Inf. Theory **67**(5), 2581–2623 (2021). https://doi.org/10.1109/TIT.2021.3062161

36. Esser, P., Rombach, R., Ommer, B.: Taming transformers for high-resolution image synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, pp. 12873–12883, (2021). https://doi.org/10.1109/CVPR46437.2021.01268
37. European Commission: European Centre for Algorithmic Transparency. https://algorithmic-transparency.ec.europa.eu/about_en (2024a)
38. European Commission: Regulatory Framework Proposal on Artificial Intelligence. https://digital-strategy.ec.europa.eu/policies/regulatory-framework-ai (2024b)
39. Fettweis, G., Boche, H.: On 6G and trustworthiness. Commun. ACM **65**(4), 48–49 (2022). https://doi.org/10.1145/3512996
40. Fridman, L.: Daniel Kahneman: Thinking Fast and Slow, Deep Learning, and AI. https://lexfridman.com/daniel-kahneman/ (2020)
41. Funahashi, K.I.: On the approximate realization of continuous mappings by neural networks. Neural Netw. **2**(3), 183–192 (1989). https://doi.org/10.1016/0893-6080(89)90003-8
42. Future of Life Institute: EU Artificial Intelligence Act. https://artificialintelligenceact.eu/ (2024)
43. Gazdag, L.E., Hansen, A.C.: Generalised hardness of approximation and the SCI hierarchy—on determining the boundaries of training algorithms in AI. arxiv:2209.06715 (2023)
44. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
45. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv:1412.6572 (2014)
46. Gray, R.M., Neuhoff, D.L.: Quantization. IEEE Trans. Inf. Theory **44**(6), 2325–2383 (1998). https://doi.org/10.1109/18.720541
47. Haase, C.A., Hertrich, C., Loho, G.: Lower bounds on the depth of integral ReLU neural networks via lattice polytopes. arXiv:2302.12553 (2023)
48. He, Y., Meng, G., Chen, K., et al.: Towards security threats of deep learning systems: a survey. IEEE Trans. Softw. Eng. **48**(5), 1743–1770 (2020). https://doi.org/10.1109/TSE.2020.3034721
49. Hinton, G., Deng, L., Yu, D., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012). https://doi.org/10.1109/MSP.2012.2205597
50. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989). https://doi.org/10.1016/0893-6080(89)90020-8
51. Iljazović, Z., Kihara, T.: Computability of subsets of metric spaces. In: Brattka, V., Hertling, P. (eds.) Handbook of Computability and Complexity in Analysis, pp. 29–69. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-59234-9_2
52. Jacovitti, G., Neri, A.: Estimation of the autocorrelation function of complex Gaussian stationary processes by amplitude clipped signals. IEEE Trans. Inf. Theory **40**(1), 239–245 (1994). https://doi.org/10.1109/18.272490
53. Jumper, J., Evans, R., Pritzel, A., et al.: Highly accurate protein structure prediction with AlphaFold. Nature **596**(7873), 583–589 (2021). https://doi.org/10.1038/s41586-021-03819-2
54. Kästner, L., Crook, B.: Explaining AI through mechanistic interpretability. http://philsci-archive.pitt.edu/22747/ (2023)
55. Katz, G., Barrett, C., Dill, D.L., et al.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification, pp. 97–117. Springer, Cham (2017)
56. Kolek, S., Windesheim, R., Andrade-Loarca, H., et al.: Explaining image classifiers with multiscale directional image representation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, pp. 18600–18609, (2023). https://doi.org/10.1109/CVPR52729.2023.01784
57. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017). https://doi.org/10.1145/3065386
58. Le, Q., Miralles-Pechuán, L., Kulkarni, S., et al.: An overview of deep learning in industry. In: Data Analytics and AI. pp. 65–98, Auerbach Publications. (2020). https://doi.org/10.1201/9781003019855-5
59. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539
60. Lee, Y., Boche, H., Kutyniok, G.: Computability of optimizers. IEEE Trans. Inf. Theory **70**(4), 2967–2983 (2024). https://doi.org/10.1109/TIT.2023.3347071

61. Liu, L., Lu, S., Zhong, R., et al.: Computing systems for autonomous driving: state of the art and challenges. IEEE Internet Things J. **8**(8), 6469–6486 (2021). https://doi.org/10.1109/JIOT.2020.3043716
62. Liu, Z.N.D., Hansen, A.C.: Do stable neural networks exist for classification problems? A new view on stability in AI. arXiv:2401.07874 (2024)
63. Loff, B., Costa, J.F.: Five views of hypercomputation. Int. J. Unconv. Comput. **5**, 193–207 (2009)
64. Maly, J., Saab, R.: A simple approach for quantizing neural networks. Appl. Comput. Harmon. Anal. **66**, 138–150 (2023). https://doi.org/10.1016/j.acha.2023.04.004
65. Mathew, A., Amudha, P., Sivakumari, S.: Deep learning techniques: an overview. In: Hassanien, A.E., Bhatnagar, R., Darwish, A. (eds.) Advanced Machine Learning Technologies and Applications, pp. 599–608. Springer, Cham (2021)
66. Minar, M.R., Naher, J.: Recent advances in deep learning: an overview. arXiv:1807.08169 (2018)
67. Mirman, M., Hägele, A., Bielik, P., et al.: Robustness certification with generative models. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. Association for Computing Machinery, New York, NY, USA, pp. 1141–1154, (2021). https://doi.org/10.1145/3453483.3454100
68. Muhammad, K., Ullah, A., Lloret, J., et al.: Deep learning for safe autonomous driving: current challenges and future directions. IEEE Trans. Intell. Transp. Syst. **22**(7), 4316–4336 (2021). https://doi.org/10.1109/TITS.2020.3032227
69. Olah, C.: Mechanistic interpretability, variables, and the importance of interpretable bases. https://www.transformer-circuits.pub/2022/mech-interp-essay (2022)
70. Parker, M.W.: Three concepts of decidability for general subsets of uncountable spaces. Theor. Comput. Sci. **351**(1), 2–13 (2006). https://doi.org/10.1016/j.tcs.2005.09.052
71. Petersen, P., Raslan, M., Voigtlaender, F.: Topological properties of the set of functions generated by neural networks of fixed size. Found. Comput. Math. **21**(2), 375–444 (2021). https://doi.org/10.1007/s10208-020-09461-0
72. Pour-El, M.B., Richards, J.I.: Computability in analysis and physics. In: Perspectives in Logic, Cambridge University Press, Cambridge, (2017). https://doi.org/10.1017/9781316717325
73. Ras, G., Xie, N., van Gerven, M., et al.: Explainable deep learning: a field guide for the uninitiated. J. Artif. Int. Res. (2022). https://doi.org/10.1613/jair.1.13200
74. Ren, A.Z., Dixit, A., Bodrova, A., et al.: Robots that ask for help: uncertainty alignment for large language model planners. Proc. Mach. Learn. Res. **229** (2023)
75. Roth, K., Munir, J., Mezghani, A., et al.: Covariance based signal parameter estimation of coarse quantized signals. In: 2015 IEEE International Conference on Digital Signal Processing (DSP), IEEE, pp. 19–23, (2015). https://doi.org/10.1109/ICDSP.2015.7251323
76. Silver, D., Huang, A., Maddison, C.J., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016). https://doi.org/10.1038/nature16961
77. Tsipras, D., Santurkar, S., Engstrom, L., et al.: Robustness may be at odds with accuracy. arXiv:1805.12152 (2019)
78. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. **s2–42**(1), 230–265 (1936). https://doi.org/10.1112/plms/s2-42.1.230
79. Van Den Oord, A., Dieleman, S., Zen, H., et al.: Wavenet: a generative model for raw audio. In: Proceedings 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9) (2016)
80. Vu, V.: On the infeasibility of training neural networks with small mean-squared error. IEEE Trans. Inf. Theory **44**(7), 2892–2900 (1998). https://doi.org/10.1109/18.737520
81. Wang, L., Ma, C., Feng, X., et al.: A survey on large language model based autonomous agents. Front. Comput. Sci. (2024). https://doi.org/10.1007/s11704-024-40231-1
82. Weihrauch, K.: Computable Analysis: An Introduction. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-56999-9
83. Wind, J.S., Antun, V., Hansen, A.C.: Implicit regularization in AI meets generalized hardness of approximation in optimization—sharp results for diagonal linear networks. arxiv:2307.07410 (2023)
84. Wu, X., Xiao, L., Sun, Y., et al.: A survey of human-in-the-loop for machine learning. Future Gener. Comput. Syst. **135**, 364–381 (2022). https://doi.org/10.1016/j.future.2022.05.014
85. Yang, J., Shen, X., Xing, J., et al.: Quantization networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7308–7316 (2019)
86. Zhang, H., Chen, H., Xiao, C., et al.: Towards stable and efficient training of verifiably robust neural networks. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020 (2020)

87. Zhang, Y., Li, Y., Cui, L., et al.: Siren's song in the AI ocean: a survey on hallucination in large language models. arXiv:2309.01219 (2023)
88. Zhou, Q.: Computable real-valued functions on recursive open and closed subsets of Euclidean space. Math. Log. Q. **42**(1), 379–409 (1996). https://doi.org/10.1002/malq.19960420132