A Computability Approach to Legal Requirements of the EU AI Act

HOLGER BOCHE, Institute of Theoretical Information Technology, Technical University of Munich, Germany, CASA – Cyber Security in the Age of Large-Scale Adversaries– Exzellenzcluster, Ruhr-Universität Bochum, Germany, BMBF Research Hub 6G-life, Germany, Munich Quantum Valley (MQV), Germany, and Munich Center for Quantum Science and Technology (MCQST), Germany

VIT FOJTIK*, Mathematical Institute, Ludwig-Maximilians-Universität München, Germany and Munich Center for Machine Learning (MCML), Germany

ADALBERT FONO, Mathematical Institute, Ludwig-Maximilians-Universität München, Germany

GITTA KUTYNIOK, Mathematical Institute, Ludwig-Maximilians-Universität München, Germany, Munich Center for Machine Learning (MCML), Germany, Department of Physics and Technology, University of Tromsø, Norway, and Institute of Robotics and Mechatronics, DRL-German Aerospace Center, Germany

Over the past decade, deep learning's widespread success led to its increasing adoption in various fields, including high-stakes and legally sensitive domains such as autonomous systems, finance, and healthcare. To mitigate potential negative effects and increase accountability, regulators have proposed legal frameworks such as the European AI Act. The necessity of these regulations is underlined by the lack of trustworthiness in deep learning, describing the fact that comprehensible, safe, and reliable methods are missing. Consequently, legal requirements also propose measures to increase these aspects. We present a mathematical viewpoint based on computing theory to connect technical with legal requirements. In particular, it allows us to assess

*Corresponding author: fojtik@math.lmu.de

if a certain algorithmic implementation on a given hardware is in principle capable of meeting trustworthy demands and thereby legal requirements. Subsequently, we apply this mathematical framework to analyze two tasks closely associated with deep learning – namely, classification and learning. A key finding is that certain conditions on the problem description determine the theoretical realizability of trustworthiness in these scenarios.

CCS Concepts: • **Theory of computation** → **Machine learning theory**; **Computability**; • **Applied computing** → Law; • **Computing methodologies** → **Neural networks**.

Additional Key Words and Phrases: Computability, Deep Learning, Classification, Reliability, Quantization

ACM Reference Format:
Holger Boche, Vit Fojtik, Adalbert Fono, and Gitta Kutyniok. 2024. A Computability Approach to Legal Requirements of the EU AI Act. In *Proceedings of 4th ACM Symposium on Computer Science and Law (CS&Law 2025).* ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

We have witnessed groundbreaking advancements in various fields with the rise of deep learning technologies [55, 56, 59, 60]. 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 including image and sound synthesis, chat tools, and protein structure prediction, to name a few [1, 34, 47, 50, 54, 68, 71]. This underlines the expanding significance of the evolving deep learning models, making erroneous or incomprehensible decisions of these systems impactful on societal and individual levels. Consequently, regulators began to propose legal frameworks to mitigate unforeseen and unwanted effects of deep learning. Notable among these are the European AI Act [37] and the G7 Hiroshima Leaders Communique [41], which set forth guidelines and regulations that AI systems, including deep learning models, should satisfy. The European AI Act, in particular, lays out a clear legal framework that might serve as a blueprint for further regulatory proposals across the globe.

One might argue that such frameworks are crucial since, despite the impressive power [32, 40, 48] and performance of deep learning, it typically comes at the cost of downsides such as black-box behavior and non-interpretability, as well as instability, non-robustness, and susceptibility to adversarial manipulation [2, 3, 5, 25, 43, 46, 53, 65, 69, 78]. In certain applications, the highlighted drawbacks, informally summarized by a lack of trustworthiness [17, 38], are tolerable or even avoidable by a human-in-the-loop approach [75]. However, increasing the autonomy of deep learning systems without introducing potentially catastrophic failure modes poses a great technological challenge, especially in safety-critical or high-responsibility tasks such as autonomous driving.

To avoid these scenarios proposed legal regulations stress the importance of trustworthiness. In particular, they try to ensure trustworthiness via abstract principles such as transparency, accountability, and right to explanation, which describe requirements for traceability of AI-based decision-making as well as responsibilities and rights for the provider and user of said systems [35, 36]. Nevertheless, from a technical point of view, it is not clear how to satisfy and implement these demands. Ideally, deep learning systems would provide 'hard' performance guarantees, i.e., verifiably correct results, which by design substantiate trustworthiness. A less restrictive approach would only ask for deep learning models that recognize whenever they cannot correctly solve a given task or instance. Thus, erroneous responses would be avoided by enabling the system to stop and ask (a human) for help, thereby implementing its trustworthiness. Finally, by disclosing the factors and their interplay which leads to a decision, the trustworthiness of a deep learning system could be increased in foresight or assessed in hindsight. Now, the crucial question is to what degrees these approaches can be implemented in practice.

Different concepts exist to tackle this problem on various levels. In this paper, we take the viewpoint of computing theory, which aims to mathematically model computation and answer questions about the algorithmic solvability and complexity of given problems. The connection between computing theory and legal frameworks promoting trustworthiness is the following. Computations, e.g., prescribed by an algorithm such as deep learning, are performed on specific hardware platforms. Computing theory then provides exactly the mathematical framework to formalize the possible hardware platforms. Therefore, we can study if a certain implementation on a given hardware is in principle capable of meeting trustworthy demands and thereby legal requirements.

1.1 Impact

Next, we provide a high-level overview of our findings and their impact on legal requirements concerning trustworthiness in deep learning. The most commonly studied model of computation is the Turing machine [70], which is an idealized version of real-world digital computers neglecting time and space constraints. The feasibility and properties of the trustworthiness approaches from the previous paragraph have been studied by translating them into the computability framework. It turned out that under certain circumstances they are equivalent on digital hardware [17].

We further deepen this analysis by showing that it is infeasible in this computing model to establish trustworthiness (in the described sense) in deep learning in certain settings. This implies that theoretical trustworthiness guarantees cannot be provided for digital computations in this setting. Consequently, potential legal requirements such as transparency and right to explanation may be impossible to satisfy on digital hardware. We also highlight the importance of the ground truth description of the considered process in these results. Ground truth description refers to the mathematical representation of a problem and acts as a blueprint for a potential algorithm. A key finding is that the algorithmic implementation of decision processes for analog (physical) processes on digital hardware does not satisfy trustworthy requirements.

However, these results also need to be put into perspective. First, as already highlighted, the results concerning the impossibility of trustworthiness guarantees heavily rely on the ground truth descriptions of the given task. Certain representations of the ground truth problem avoid the impossibility result. Moreover, by taking into account the intended scope of a task, the ground truth description and the feasibility of trustworthiness may vary; further details are discussed in Section 3.2. Second, the applied notion of trustworthiness based on verifiable correctness via computability is rather strict. Other notions of trustworthiness are more amenable to practical implementations and choosing an appropriate one is highly relevant; for a more in-depth discussion of this topic, we refer to [17].

1.2 Outline

In Section 2, we introduce the applied formalisms, including neural networks, as the main workhorse of deep learning, and computing theory. Moreover, we provide a short overview of existing work at the intersection of computability and trustworthiness. Subsequently, we present our main results concerning failures of trustworthiness from the computability perspective in Section 3.1 and conclude by studying strategies to cope with the failures. The proofs of the statements are provided in Appendix D.

2 Preliminaries

2.1 Computability Theory

We first establish some basic concepts and notation used in the following. Turing machines provide the means to study digital computations [70]. 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. For instance, complex physical processes – a realistic 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. The distinction is nevertheless crucial since the underlying computability concepts depend on the input domain and the associated ground truth solution of the tackled problem.

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, i.e., continuous, domain. We only sketch the most important ideas and refer to Appendix A for a more thorough introduction. The key concepts in this framework are computable real functions; here we present two related definitions (see [4, Appendix 2.9] for an overview). 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. 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 2.1. Let $D \subset \mathbb{R}^d$. A function $f : D \to \mathbb{R}^m_c$ is

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

Remark 2.2. Informally, a representation refers to a computable (rapidly converging) Cauchy sequence and a sequence is computable if a representation (via computable Cauchy sequences) of the sequence exists (see Appendix A). The set of real numbers that possess a representation, the so-called computable real numbers, are denoted by $\mathbb{R}_c$. 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.

By formalizing problems in terms of functions, we can analyze the algorithmic solvability of the underlying problem. In particular, Borel-Turing computability (theoretically) guarantees verifiable correct results with error control and, thus, bridges the gap to various trustworthiness notions. A technical and detailed depiction of this link is presented in [17]. The key take-away is however that we may study computability as a prerequisite to trustworthiness. In this work, we proceed by analyzing two types of algorithmic failure, whereby we associate in the real domain the more intuitive term "algorithm" with "Borel-Turing computable function":

- 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 provides an incorrect 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 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 Section 3.1 that Type 2 free of Type 1 failure exists in the context of training neural networks. Here, the problem instance is an unknown function and the goal of the learning algorithm is to find a neural network that represents the sought function based on samples.

2.2 Neural Networks

From a mathematical perspective, a neural network is simply a structure with a certain architecture and an associated realization.

Definition 2.3. Let $L \in \mathbb{N}$. An *architecture* of depth L is a vector $S := (N_0, N_1, \ldots, N_{L-1}, N_L) \in \mathbb{N}^{L+1}$. 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 $\ell = 1, \ldots, L$. 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 2.4. 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 parameters then induce the network's input-output function, the so-called realization.

Definition 2.5. 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 $T_\ell(\boldsymbol{x}) := A_\ell \boldsymbol{x} + b_\ell$, $\ell = 1, \ldots, L$.

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, typically some version of stochastic gradient descent, receives as input a dataset of sample pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^{n}$, which are usually sampled from some underlying goal function f, that is, $f(\boldsymbol{x}_i) = \boldsymbol{y}_i$. The aim of learning is to find a neural network Φ optimizing some loss function $\mathcal{L} : \mathcal{NN}(S) \times \left(\mathbb{R}^{N_0} \times \mathbb{R}^{N_L}\right)^n \to \mathbb{R}$. Thus we can view the learning algorithm as a Borel-Turing computable function $\Gamma : \mathbb{R}_c^{n(N_0+N_L)} \to \mathbb{R}_c^{N(S)}$. 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 $\mathbb{R}_c$.

Definition 2.6. Given $n \in \mathbb{N}$, $f : \mathbb{R}^d \to \mathbb{R}^m$ and $D \subset \mathbb{R}^d$, we denote by $\mathcal{D}^n_{f,D}$ the set of all *datasets* of size n generated from f on the input domain D, that is,

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

For a neural network $\Phi \in NN(S)$ with activation σ we denote for short $\mathcal{D}^n_{\Phi,D} := \mathcal{D}^n_{R^D_\sigma(\Phi),D}$.

2.3 Previous Work

Non-computability has been a point of high interest in algorithmic computation since the results of Church [28] and Turing [70]. For continuous problems, the non-existence of a computable solver (Type 1 failure) has been shown in various applications including optimization, inverse problems, signal processing, and information theory [8, 13, 15, 16, 18–22, 24, 57].

In the context of neural network training, 'hardness' results have a long tradition going back to [12, 72], where it was shown that the training process can be NP-complete for certain architectures. The infeasibility of algorithmically learning an existing computable neural network (Type 2 failure) in a continuous setting has been shown for the specific case of inverse problems in [29] and classification problems in [9]. Furthermore, [57] showed that no algorithm can reach near-optimal training loss on all possible datasets even for elementary neural networks. Further properties of deep learning from the computability perspective concerning adversarial attacks, implicit regularization, and hardness of approximation were studied in [9, 42, 74]. A different context of learning an existing neural network has been studied in [10], 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].

Understanding the inner workings of deep learning and enabling the user to comprehend their decision-making points out a way to establish trustworthy methods. A key challenge is to increase the interpretability of deep learning algorithms, which is typically hindered due to their black-box behavior [51, 62, 65]. Another approach relies on verifying the accuracy and correctness of deep learning methods without explicitly tracing internal computations [11, 52, 61, 77]. However, the findings in [7, 17] indicate that certifying the accuracy and robustness of deep learning in the computability framework is challenging if at all possible in certain scenarios, which poses challenges for future applications. A potential direction to cope with this issue was considered in [14], where certain inverse problems that are not computable on digital computers were shown to be computable in a model of analog computation enabling implicit correctness guarantees in theory.

3 Main Results

The key findings in this paper extend the theory of computability in deep learning and highlight the importance of ground truth descriptions. As discussed in Section 2, we apply the notion of Borel-Turing computability to assess the attainability of trustworthiness as a surrogate for potential legal demands. In particular, by characterizing Type 1 and Type 2 failure of computability, we describe conditions under which trustworthiness cannot be achieved in certain scenarios. We establish Type 1 and Type 2 failure in two crucial aspects associated with deep learning. First, we analyze classification tasks, a major application field of deep learning, and show in Theorem 3.1 that Type 1 failure arises in classification on the real domain. Subsequently, we study the feasibility of training neural networks – having reliable, flexible, and universal learning algorithms is essential to the success of deep learning. Theorem 3.2 shows that no general learning algorithm applicable to all (real-valued) networks exists implying that Type 2 failure is unavoidable in this scenario.

Can we avoid or overcome these Type 1 and Type 2 limitations of computability? We analyze different approaches to either reformulate or relax the tackled problems thus making them less amenable to computability failures. First, the effect of incorporating a reasonable error mode in the computation is evaluated, but it turns out that this strategy does typically not alleviate Type 1 failure in the classification setting; we refer to Appendix C.1 for more details. Subsequently, we investigate the impact of moving the problem from the real to a discrete space via quantization. In quantization, real numbers are approximated by a discrete set of rationals since access to real-valued data and parameters with unlimited precision can not be expected in many applications [6, 44, 49, 67]. In perhaps the simplest quantization paradigm, 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 $b^{-k}\mathbb{Z}$. Naturally, the question arises of how these quantization techniques affect the properties of algorithmic computations, including the limitations of computability on continuous domains. For instance, quantized deep learning has been a topic of interest, comparing its theoretical and practical capabilities to non-quantized deep learning [58, 76]; in fact, the capabilities of quantized and real networks align in the limit [23, 33, 45].

By assuming fixed-point quantization we can restrict our analysis without loss of generality to classification problems on $\mathbb{Z}^d$ as well as neural networks with integer parameters and data. The concept of algorithmic computations on integers is described by *recursive functions* (which the 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. In Theorems 3.6 and 3.7 we show that when considering quantized versions of the previous settings, issues of non-computability do not arise. However, Theorem 3.9 provides a word of caution, stating that the act of quantization itself exhibits Type 1 failure – we cannot algorithmically determine which quantized values faithfully represent the original (real-valued) problem. Hence, the choice of the ground truth description can also influence the computability of a problem. How can we identify an appropriate choice? While there is no definite answer, we want to highlight the following approach. If we consider the underlying question in computing as "What can be (efficiently) automated?" [**?**], then the ground truth description confines the answer to the considered domain. In this sense, trustworthy algorithmic automation is not feasible if the task itself includes the quantization process.

3.1 Computability Limitations

A classification problem is modeled by a function

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

that assigns each input $\boldsymbol{x} \in D$ a corresponding class $c \in \{1, \ldots, C\}$. A typical example is image classification where the input domain D is for instance given by $D = [0, 256]^{h \times w}$ with $[0, 256]$ and $h, w \in \mathbb{N}$ encoding color and size (height and width) of an image, respectively. The range $[0, 256]$ may also be quantized so that a discrete set such as $\{0, \ldots, 256\}$ represents the input domain. We first explore the general case before returning to the quantized setting in Section 3.2.

Our first result states that a classification problem cannot be computable unless all its classes are strictly separated. This typically does not occur in practice, where the problem and the underlying classification task are associated with some continuous process. Therefore, performance guarantees cannot be provided for common classification problems, i.e., Type 1 failure of computability arises.

Theorem 3.1. *Let $f : D \to \{1, \ldots, C\}$ be a function such that there exists $i \neq j \in \{1, \ldots, C\}$ with $dist(f^{-1}(i), f^{-1}(j)) = 0$. Then, $f|_{\mathbb{R}^d_c}$ is not computable.*

For a proof and further details on the computability of classification problems, we refer to Appendix B.

Next, we focus on Type 2 failure of computability, i.e., situations where a computable approximator may exist but cannot be algorithmically found based on data. We explore this phenomenon in the context of deep learning, within a general framework by studying the learnability of functions that can be represented by a neural network from data, independently of the concrete application. This includes any instance of deep learning where Type 1 computability failure does not arise, going beyond the previous context of classification.

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, even when a 'correctly solving' network exists. Thus, deep learning suffers from Type 2 failure of computability.

Theorem 3.2. *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. Let $S = (d, N_1, \dots, N_{L-1}, 1)$ be an architecture of depth $L \geq 2$ with $N_1 \geq 3$. Let $D \subset \mathbb{R}_c^d$ be bounded with a nonempty interior.*

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

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

Remark 3.3. The assumption that σ is not affine linear excludes none of the commonly used activations such as ReLU, tanh, or sigmoid. 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 consequence, we cannot reconstruct the original input-output function.

Corollary 3.4. *Under the assumptions of Theorem 3.2, 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 $\mathcal{X} \in \mathcal{D}^n_{\Phi,D}$ satisfying*

$$R^D_\sigma(\Gamma(\mathcal{X})) = R^D_\sigma(\Phi).$$

3.2 Strategies for Failure Circumvention

First, we show that under fixed-point quantization, the negation of Theorem 3.2 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 3.5. Let S be an architecture. 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 3.6. *Let $S = (d, N_1, \dots, N_{L-1}, 1)$ be an architecture and let $\sigma : \mathbb{R} \to \mathbb{R}$.*

Then, for 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

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

Turning our attention to classification and Type 1 failure, we note that in most applications, classification is performed on a bounded domain D, such as in image classification described in Section 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.

Theorem 3.7. *Let $D \subset \mathbb{Z}^d$ and $f : D \to \{1, \dots, C\}$. If D is bounded, then f is recursive.*

Remark 3.8. 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.

While these results are positive, for them to apply, a ground truth problem on a continuous domain must first be converted into an appropriate quantized problem that approximates the original one. However, for a non-computable ground truth problem such an algorithmic verification contradicts its non-computability. Thus, quantization itself is a non-computable task and no algorithmic guarantees can be provided concerning the faithfulness of the quantized problem.

Theorem 3.9. *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

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

is not computable.

Acknowledgments

This work of H. Boche was supported in part by the German Federal Ministry of Education and Research (BMBF) within the national initiative on 6G Communication Systems through the Research Hub 6G-life under Grant 16KISK002.

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 (BMBF) 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 support from LMUexcellent, funded by the Federal Ministry of Education and Research (BMBF) and the Free State of Bavaria under the Excellence Strategy of the Federal Government and the Länder as well as by the Hightech Agenda Bavaria.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Ben Adcock and Nick Dexter. 2021. The gap between theory and practice in function approximation with deep neural networks. *SIAM Journal on Mathematics of Data Science* 3, 2 (2021), 624–655.

[3] Vegard Antun, Francesco Renna, Clarice Poon, Ben Adcock, and Anders C Hansen. 2020. On instabilities of deep learning in image reconstruction and the potential costs of AI. *Proceedings of the National Academy of Sciences* 117, 48 (2020), 30088–30095.

[4] Jeremy Avigad and Vasco Brattka. 2014. *Computability and analysis: the legacy of Alan Turing.* Cambridge University Press, Cambridge, 1–47.

[5] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one* 10, 7 (2015), e0130140.

[6] Ofer Bar-Shalom and Anthony J Weiss. 2002. DOA estimation using one-bit quantized measurements. *IEEE Trans. Aerospace Electron. Systems* 38, 3 (2002), 868–884.

[7] Alexander Bastounis, Alexander N. Gorban, Anders C. Hansen, Desmond J. Higham, Danil Prokhorov, Oliver Sutton, Ivan Y. Tyukin, and Qinghua Zhou. 2023. The Boundaries of Verifiable Accuracy, Robustness, and Generalisation in Deep Learning. In *Artificial Neural Networks and Machine Learning – ICANN 2023*, Lazaros Iliadis, Antonios Papaleonidas, Plamen Angelov, and Chrisina Jayne (Eds.). Springer Nature Switzerland, Cham, 530–541.

[8] Alexander Bastounis, Anders C Hansen, and Verner Vlačić. 2021. The extended Smale's 9th problem – On computational barriers and paradoxes in estimation, regularisation, computer-assisted proofs and learning. *arXiv:2110.15734* (2021).

[9] Alexander Bastounis, Anders C Hansen, and Verner Vlačić. 2021. The mathematics of adversarial attacks in AI – Why deep learning is unstable despite the existence of stable neural networks. *arXiv preprint arxiv:2109.06098* (2021).

[10] Julius Berner, Philipp Grohs, and Felix Voigtlaender. 2022. Learning ReLU networks to high uniform accuracy is intractable. *arXiv preprint arXiv:2205.13531* (2022).

[11] Alessandro Biondi, Federico Nesti, Giorgiomaria Cicero, Daniel Casini, and Giorgio Buttazzo. 2020. A Safe, Secure, and Predictable Software Architecture for Deep Learning in Safety-Critical Systems. *IEEE Embed. Syst. Lett.* 12, 3 (2020), 78–82.

[12] Avrim L. Blum and Ronald L. Rivest. 1992. Training a 3-node neural network is NP-complete. *Neural Networks* 5, 1 (1992), 117–127.

[13] Holger Boche and Christian Deppe. 2021. Computability of the zero-error capacity of noisy channels. In *2021 IEEE Information Theory Workshop (ITW).* 1–6.

[14] Holger Boche, Adalbert Fono, and Gitta Kutyniok. 2022. Inverse problems are solvable on real number signal processing hardware. *arXiv preprint arXiv:2204.02066* (2022).

[15] Holger Boche, Adalbert Fono, and Gitta Kutyniok. 2022. Non-computability of the pseudoinverse on digital computers. *arXiv preprint arXiv:2212.02940* (2022).

[16] Holger Boche, Adalbert Fono, and Gitta Kutyniok. 2023. Limitations of Deep Learning for Inverse Problems on Digital Hardware. *IEEE Transactions on Information Theory* 69, 12 (2023), 7887–7908.

[17] Holger Boche, Adalbert Fono, and Gitta Kutyniok. 2024. Mathematical Algorithm Design for Deep Learning under Societal and Judicial Constraints: The Algorithmic Transparency Requirement. *arXiv preprint arXiv:2401.10310* (2024).

[18] Holger Boche and Ullrich J. Mönich. 2020. Turing Computability of Fourier Transforms of Bandlimited and Discrete Signals. *IEEE Trans. Signal Process.* 68 (2020), 532–547.

[19] Holger Boche and Volker Pohl. 2020. On the Algorithmic Solvability of Spectral Factorization and Applications. *IEEE Trans. Inf. Theory* 66, 7 (2020), 4574–4592.

[20] Holger Boche, Rafael F. Schaefer, and H. Vincent Poor. 2020. Denial-of-Service Attacks on Communication Systems: Detectability and Jammer Knowledge. *IEEE Transactions on Signal Processing* 68 (2020), 3754–3768.

[21] Holger Boche, Rafael F. Schaefer, and H. Vincent Poor. 2020. Shannon Meets Turing: Non-Computability and Non-Approximability of the Finite State Channel Capacity. *Communications in Information and Systems* 20, 2 (2020), 81–116.

[22] Holger Boche, Rafael F. Schaefer, and H. Vincent Poor. 2023. Algorithmic Computability and Approximability of Capacity-Achieving Input Distributions. *IEEE Transactions on Information Theory* 69, 9 (2023), 5449–5462.

[23] Helmut Bolcskei, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. 2019. Optimal approximation with sparsely connected deep neural networks. *SIAM Journal on Mathematics of Data Science* 1, 1 (2019), 8–45.

[24] Vasco Brattka and Martin Ziegler. 2001. Turing computability of (non-)linear optimization. In *Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG'01).*

[25] Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. 2018. Explaining image classifiers by counterfactual generation. *arXiv preprint arXiv:1807.08024* (2018).

[26] Sitan Chen, Aravind Gollakota, Adam Klivans, and Raghu Meka. 2022. Hardness of noise-free learning for two-hidden-layer neural networks. *Advances in Neural Information Processing Systems* 35 (2022), 10709–10724.

[27] Qinyuan Cheng, Tianxiang Sun, Xiangyang Liu, Wenwei Zhang, Zhangyue Yin, Shimin Li, Linyang Li, Zhengfu He, Kai Chen, and Xipeng Qiu. 2024. Can AI Assistants Know What They Don't Know? *arXiv preprint arxiv:2401.13275* (2024).

[28] Alonzo Church. 1936. A note on the Entscheidungsproblem. *The journal of symbolic logic* 1, 1 (1936), 40–41.

[29] Matthew J Colbrook, Vegard Antun, and Anders C Hansen. 2022. The difficulty of computing stable and accurate neural networks: On the barriers of deep learning and Smale's 18th problem. *Proceedings of the National Academy of Sciences* 119, 12 (2022), e2107151119.

[30] S Barry Cooper. 2017. *Computability theory.* Chapman and Hall/CRC, New York.

[31] Covariant. 2021. AI Robotics for the Real World. https://www.youtube.com/watch?v=AAr99hQ64AI.

[32] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2, 4 (1989), 303–314.

[33] Dennis Elbrächter, Dmytro Perekrestenko, Philipp Grohs, and Helmut Bölcskei. 2021. Deep neural network approximation theory. *IEEE Transactions on Information Theory* 67, 5 (2021), 2581–2623.

[34] Patrick Esser, Robin Rombach, and Bjorn Ommer. 2021. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 12873–12883.

[35] European Commission. 2024. European centre for algorithmic transparency. https://algorithmic-transparency.ec.europa.eu/about_en.

[36] European Commission. 2024. Regulatory framework proposal on artificial intelligence. https://digital-strategy.ec.europa.eu/policies/regulatory-framework-ai.

[37] European Parliament. 2023. Artificial intelligence act. https://www.europarl.europa.eu/RegData/etudes/BRIE/2021/698792/EPRS_BRI(2021)698792_EN.pdf.

[38] G.P. Fettweis and Holger Boche. 2022. On 6G and Trustworthiness. *Commun. ACM* 65, 4 (2022), 48–49.

[39] Lex Fridman. 2020. Daniel Kahneman: Thinking Fast and Slow, Deep Learning, and AI. https://lexfridman.com/daniel-kahneman/.

[40] Ken-Ichi Funahashi. 1989. On the approximate realization of continuous mappings by neural networks. *Neural networks* 2, 3 (1989), 183–192.

[41] G7 Hiroshima Summit 2023. 2023. G7 Hiroshima Leaders' Communiqué. https://www.g7hiroshima.go.jp/documents/pdf/Leaders_Communique_01_en.pdf.

[42] Luca Eva Gazdag and Anders C. Hansen. 2023. Generalised hardness of approximation and the SCI hierarchy – On determining the boundaries of training algorithms in AI. *arXiv preprint arxiv:2209.06715* (2023).

[43] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[44] Robert M. Gray and David L. Neuhoff. 1998. Quantization. *IEEE transactions on information theory* 44, 6 (1998), 2325–2383.

[45] Christian Alexander Haase, Christoph Hertrich, and Georg Loho. 2023. Lower Bounds on the Depth of Integral ReLU Neural Networks via Lattice Polytopes. In *The Eleventh International Conference on Learning Representations*.

[46] Yingzhe He, Guozhu Meng, Kai Chen, Xingbo Hu, and Jinwen He. 2020. Towards security threats of deep learning systems: A survey. *IEEE Transactions on Software Engineering* 48, 5 (2020), 1743–1770.

[47] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine* 29, 6 (2012), 82–97.

[48] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[49] Giovanni Jacovitti and Alessandro Neri. 1994. Estimation of the autocorrelation function of complex Gaussian stationary processes by amplitude clipped signals. *IEEE transactions on information theory* 40, 1 (1994), 239–245.

[50] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (2021), 583–589.

[51] Lena Kästner and Barnaby Crook. 2023. Explaining AI through mechanistic interpretability. http://philsci-archive.pitt.edu/22747/

[52] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Computer Aided Verification*, Rupak Majumdar and Viktor Kunčak (Eds.). Springer International Publishing, Cham, 97–117.

[53] Stefan Kolek, Robert Windesheim, Hector Andrade-Loarca, Gitta Kutyniok, and Ron Levie. 2023. Explaining image classifiers with multiscale directional image representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18600–18609.

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[55] Quan Le, Luis Miralles-Pechuán, Shridhar Kulkarni, Jing Su, and Oisín Boydell. 2020. An overview of deep learning in industry. *Data analytics and AI* (2020), 65–98.

[56] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[57] Yunseok Lee, Holger Boche, and Gitta Kutyniok. 2024. Computability of Optimizers. *IEEE Transactions on Information Theory* 70, 4 (2024), 2967–2983.

[58] Johannes Maly and Rayan Saab. 2023. A simple approach for quantizing neural networks. *Applied and Computational Harmonic Analysis* 66 (2023), 138–150.

[59] Amitha Mathew, P Amudha, and S Sivakumari. 2021. Deep learning techniques: an overview. *Advanced Machine Learning Technologies and Applications: Proceedings of AMLTA 2020* (2021), 599–608.

[60] Matiur Rahman Minar and Jibon Naher. 2018. Recent advances in deep learning: An overview. *arXiv preprint arXiv:1807.08169* (2018).

[61] Matthew Mirman, Alexander Hägele, Pavol Bielik, Timon Gehr, and Martin Vechev. 2021. Robustness Certification with Generative Models. In *SIGPLAN PLDI 2021*. Association for Computing Machinery, New York, NY, USA, 1141–1154.

[62] Chris Olah. 2022. Mechanistic Interpretability, Variables, and the Importance of Interpretable Bases. https://www.transformer-circuits.pub/2022/mech-interp-essay.

[63] Philipp Petersen, Mones Raslan, and Felix Voigtlaender. 2021. Topological properties of the set of functions generated by neural networks of fixed size. *Foundations of computational mathematics* 21, 2 (2021), 375–444.

[64] Marian B. Pour-El and J. Ian Richards. 2017. *Computability in Analysis and Physics*. Cambridge University Press, Cambridge.

[65] Gabrielle Ras, Ning Xie, Marcel van Gerven, and Derek Doran. 2022. Explainable Deep Learning: A Field Guide for the Uninitiated. *J. Artif. Int. Res.* 73 (2022), 68 pages.

[66] Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. 2023. Robots That Ask For Help: Uncertainty Alignment for Large Language Model Planners. In *7th Annual Conference on Robot Learning*.

[67] Kilian Roth, Jawad Munir, Amine Mezghani, and Josef A Nossek. 2015. Covariance based signal parameter estimation of coarse quantized signals. In *2015 IEEE International Conference on Digital Signal Processing (DSP)*. IEEE, 19–23.

[68] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[69] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. 2019. Robustness May Be at Odds with Accuracy. In *7th International Conference on Learning Representations (ICLR)*.

[70] Alan Mathison Turing. 1936. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* s2-42, 1 (1936), 230–265.

[71] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu, et al. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* 12 (2016).

[72] V.H. Vu. 1998. On the infeasibility of training neural networks with small mean-squared error. *IEEE Transactions on Information Theory* 44, 7 (1998), 2892–2900.

[73] Klaus Weihrauch. 2012. *Computable analysis: An introduction*. Springer Science & Business Media, Berlin & Heidelberg.

[74] Johan S. Wind, Vegard Antun, and Anders C. Hansen. 2023. Implicit regularization in AI meets generalized hardness of approximation in optimization – Sharp results for diagonal linear networks. *arXiv preprint arxiv:2307.07410* (2023).

[75] Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He. 2022. A survey of human-in-the-loop for machine learning. *Future Gener. Comput. Syst.* 135 (2022), 364–381.

[76] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. 2019. Quantization networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 7308–7316.

[77] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane S. Boning, and Cho-Jui Hsieh. 2019. Towards Stable and Efficient Training of Verifiably Robust Neural Networks. In *ICLR 2020.*

[78] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren's song in the AI ocean: A survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219* (2023).

A 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, 64, 73]. 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. We follow standard definitions introduced by Turing [70].

Definition A.1. A sequence of rational numbers $(q_k)_{k=1}^\infty \subset \mathbb{Q}$ 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 $\mathbb{R}_c$.

Before defining computable real functions, we need to specify what it means for a real sequence to be computable.

Definition A.2. A real sequence $(x_k)_{k=1}^\infty \subset \mathbb{R}$ 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 A.3. 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.

B Computability of Classification

As described in Section 2.2, the goal of deep learning is to learn a function $\hat{f} : D \cap \mathbb{R}_c^d \to \{1, \ldots, C\}$ based on samples $(\boldsymbol{x}_i, f(\boldsymbol{x}_i))_{i=1}^n \subset D \times \{1, \ldots, C\}$ such that $\hat{f}$ is close to f with respect to a suitable measure. 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 B.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 B.2. The notion of (semi-)decidability can be explicitly expressed via algorithms in the following way. The set A is Borel-Turing decidable in D if there exists a Turing machine M taking as inputs representations of $\boldsymbol{x} \in D \cap \mathbb{R}_c^d$ which correctly identifies after finitely many iterations whether $\boldsymbol{x} \in A$ or $\boldsymbol{x} \in D \setminus A$. Similarly, A is semi-decidable in D if there exists a Turing machine M taking as inputs representations of $\boldsymbol{x} \in D$ which correctly identifies (after finitely many iterations) every input $\boldsymbol{x} \in A$, but which may run forever for $\boldsymbol{x} \in D \setminus A$.

Recall that computable functions are necessarily continuous on $\mathbb{R}_c^d$. However, indicator functions are discontinuous on $\mathbb{R}_c^d$ (excluding the trivial cases $\mathbb{R}_c^d$ and $\emptyset$). Thus, 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 due to the discrete image of f. In particular, the proposition provides a necessary condition for learning a perfect emulator $\hat{f} = f|_{\mathbb{R}_c^d}$ since computability is a prerequisite for learnability.

Proposition B.3. *Let $D \subset \mathbb{R}^d$ and consider $f : D \to \{1, \ldots, C\}$. Then, $f|_{\mathbb{R}_c^d}$ is computable if and only if each set $f^{-1}(i)$, $i = 1, \ldots, C$, is semi-decidable in D.*

Remark B.4. Note that if each set $f^{-1}(i)$, $i = 1, \ldots, C$, is semi-decidable in D, then also each set $f^{-1}(i)$ is decidable in D. 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. As a simple example for this type of problems consider $D = f^{-1}(1) \cup f^{-1}(2)$ with $C = 2$, $f^{-1}(1) = (0,1)$ and $f^{-1}(2) = (2,3)$. 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.

From these observations follows the statement of Theorem 3.1.

C Further Strategies for Failure Circumvention

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 Section 3.1 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 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.

C.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 – a so-called exit flag – algorithmically? In a bigger picture, related questions were already raised in different contexts such as general artificial intelligence by Daniel Kahneman [39] or robotics by Pieter Abbeel [31]: Can we expect algorithms (powering autonomous agents) to recognize whenever they cannot correctly solve a given task or instance enabling them to ask (a human) for help instead of executing an erroneous response? In other words: 'Do they know when they don't know?' [27, 66] We immediately observe that this problem depends on an exit flag computation in the computability framework. Hence, by studying exit flag computations, we can also theoretically assess the feasibility of automated help-seeking behavior of autonomous agents in certain scenarios.

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 (a Borel-Turing computable function) $\Gamma_\varepsilon : D_c \to \mathbb{R}_c$ such that

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

One can also 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 not computable. However, choosing ε large enough, certainly still entails the existence of Γ_ε if f is, for instance, a bounded function. Therefore, the relevant cases are associated with non-computable f and appropriately small ε. By associating Γ_ε and Γ_ε^+ with classification functions, we can apply Proposition B.3 to derive the following result.

Proposition C.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

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

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 C.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 $D_\varepsilon^\geq \subset D_c'$.

In the case of a connected input domain, the application of Proposition 3.1 yields the following result. Informally, it is a direct consequence of the already mentioned fact that only trivial subsets of $\mathbb{R}_c^d$ are decidable.

Corollary C.3. *Under the conditions of Proposition C.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*

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

Remark C.4. As the following example shows, a similar statement does not hold if $D_\varepsilon^<$ is only assumed to be semi-decidable. Consider the sign function $\mathrm{sgn} : \mathbb{R} \to \mathbb{R}$, which is non-continuous on $\mathbb{R}_c$ and therefore non-computable, and take $\widehat{\mathrm{sgn}}(x) = \frac{2}{\pi}\arctan(x)$. For a given $\varepsilon > 0$ we can computably construct intervals $(-\infty, -x_0)$ and (x_0, ∞) where $|\mathrm{sgn}(x) - \widehat{\mathrm{sgn}}(x)| < \varepsilon$, 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 B, where Type 1 failure was established. Hence, we can conclude that exit flag computations may be beneficial in certain situations but it is not appropriate to tackle Type 1 failure in classification.

Instead of analyzing individual inputs, one could also derive global guarantees for the approximator $\hat{f}$. For instance, one could determine the magnitude of the failure set for some appropriate measure, i.e., assess the likelihood of errors for some given input domain. Although this approach is interesting it has two main limitations for our intended goals. First, there does not exist an acknowledged algorithmic notion covering this framework that allows for theoretical studies. Second and more importantly, this strategy typically leads to a global quantitative measure of failure whereas we are interested in local guarantees for a given input. In essence, this framework does not tackle the limitations of individual inputs but provides further information on the entire input domain of the general problem.

Finally, we want to highlight that Proposition C.1 and Corollary C.3 imply that the algorithms envisioned by Kahneman and Abbeel can not be realized on digital hardware. For any algorithm Γ tackling a problem described by a non-computable function, there exist instances that Γ answers incorrectly and there is no algorithm Γ_{exit} that recognizes the failure for all erroneous instances. Hence, the answer to ‘Do they know when they don’t know?’ from the digital computing perspective is negative in certain scenarios.

D Proofs

D.1 Proof of Theorem 3.2

The proof of Theorem 3.2 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 D.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 $(\iota_k^1)_{k=1}^\infty, (\iota_k^2)_{k=1}^\infty \subset \Theta$ 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 $k \in \mathbb{N}$, $\mathrm{dist}(\Xi(\iota_k^1), \Xi(\iota_k^2)) > \varepsilon$.

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 $(f_1, \ldots, f_n) \in \Lambda^n$:

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

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

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

Since Γ is Banach-Mazur computable, it is continuous on $\mathbb{R}_c^n$ [73], 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

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

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 (3) 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 [63], 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 D.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 D.3 ([63, 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 $(\Phi_k)_{k=1}^\infty, (\mu_k)_{k=1}^\infty \subset \mathcal{NN}(S)$ such that*

(i) $\|R_\sigma^D(\Phi_k) - R_\sigma^D(\mu_k)\|_\infty \to 0$,

(ii) for any $(\Phi_k')_{k=1}^\infty, (\mu_k')_{k=1}^\infty \subset \mathcal{NN}(S)$ 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 $\left\|\Phi_k' - \mu_k'\right\|_{\mathit{scaling}} \to \infty$.

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

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

To see this, assume $R^D_\sigma(\mu_k) \equiv 0$, and for contradiction let there be a subsequence $(\Phi'_{k_\ell})_{\ell=1}^\infty \subset \mathcal{NN}(S)$ 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 $C > 0$:

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

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

From the proof in [63] 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 3.2. 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 $f^i_{\boldsymbol{x}} : \Theta \longrightarrow \mathbb{R}_c$ the constant operator

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

and by $f_{(\boldsymbol{x})} : \Theta \longrightarrow \mathbb{R}_c$ 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 $\Xi : \Theta \longrightarrow \mathcal{P}(\mathbb{R}_c^{N(S)})$ by

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

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

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

Together, by Lemma D.1 for all $n \in \mathbb{N}$ and all Banach-Mazur computable functions $\Gamma : (\mathbb{R}_c^d \times \mathbb{R}_c)^n \longrightarrow \mathbb{R}_c^{N(S)}$ 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

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

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 $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in D$ and all $\Phi' \in \mathcal{NN}_c(S)$ with $R^D_\sigma(\Phi') = R^D_\sigma(\Phi) = g$ we have

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

□

D.2 Proof of Theorem 3.6

An enumeration argument similar to the ones in the previous proofs implies Theorem 3.6. 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 3.6. Given an architecture $S = (d, N_1, \dots, N_{L-1}, 1)$, $\mathcal{NN}_{\mathbb{Z}}(S)$ can be associated with $\mathbb{Z}^{N(S)}$, which in turn can be recursively encoded into $\mathbb{Z}^d$ by a recursivelly 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 $\mathcal{X} = \{(\boldsymbol{x}_1, y_1) \dots, (\boldsymbol{x}_n, y_n)\}$, a single datapoint of the form $\big(g(\Phi), R_\sigma^{\mathbb{Z}^d}(\Phi)(g(\Phi)), 0, \dots\big) \in (\mathbb{Z}^d \times \mathbb{Z})^n$ 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. □