Received: 7 October 2022 | Accepted: 1 November 2022

DOI: 10.1002/pamm.202200174

Which neural networks can be computed by an algorithm? – Generalised hardness of approximation meets Deep Learning

Laura Thesing[1,*] and **Anders C. Hansen**[2]

[1] LMU Munich, Akademiestr. 7, 80799 Munich, Germany

[2] University of Cambridge, Wilberforce Rd, Cambridge CB3 0WA, United Kingdom

Classical hardness of approximation (HA) is the phenomenon that, assuming P $\neq$ NP, one can easily compute an ϵ -approximation to the solution of a discrete computational problem for $\epsilon > \epsilon_0 > 0$, but for $\epsilon < \epsilon_0$ – where ϵ_0 is the approximation threshold – it becomes intractable. Recently, a similar yet more general phenomenon has been documented in AI: Generalised hardness of approximation (GHA). This phenomenon includes the following occurrence: For any approximation threshold $\epsilon_1 > 0$, there are AI problems for which provably there exist stable neural networks (NNs) that solve the problem, but no algorithm can compute any NN that approximates the AI problem to ϵ_1-accuracy. Moreover, this issue is independent of the P vs NP question and thus is a rather different mathematical phenomenon than HA. GHA implies that the universal approximation theorem for NNs only provides a partial understanding of the power of NNs in AI. Thus, a classification theory describing which NNs can be computed by algorithms to particular accuracies is needed to fill this gap. We initiate such a theory by showing the correspondence between the functions that can be computed to ϵ-accuracy by an algorithm and those functions that can be approximated by NNs which can be computed to $\hat{\epsilon}$-accuracy by an algorithm. In particular, the approximation thresholds ϵ and $\hat{\epsilon}$ cannot differ by more than a factor of 12. This means that computing function approximations through NNs will be optimal – in the sense of best approximation accuracy achievable by an algorithm – up to a small constant, compared to any other computational technique.

1 Introduction

Deep learning (DL) with neural networks has risen to the state-of-the-art method in a wide range of applications from image classification [1], voice control [2], malware detection [3], to inverse problems [4–6]. Due to the immense success, these methods have reached our everyday life. However, at the same time, we are well aware of the limits of current AI technology, where inherent instabilities, AI generated hallucinations, unpredictable as well as biased behaviour of DL algorithms etc. are often observed, yet not well understood – and remedies are scarce. These issues are concerning, given the wide use of modern DL. Therefore, it is not surprising that after a long time of studying the wonders of DL, the studies of their limitations have gained more interest in the research community:

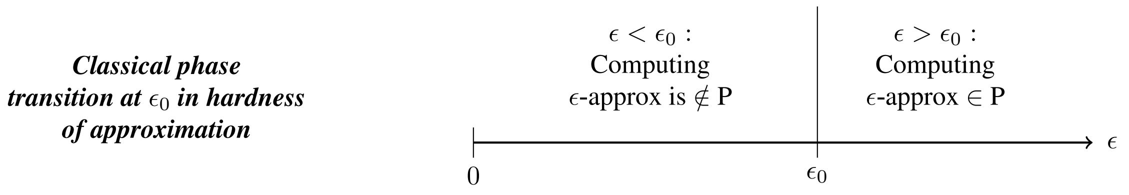
> *"2021 was the year in which the wonders of artificial intelligence stopped being a story [...] Many of this year's top articles grappled with the limits of deep learning (today's dominant strand of AI) and spotlighted researchers seeking new paths."*
>
> – '2021's Top Stories of AI', in IEEE Spectrum (Dec. -2021).

The instabilities, hallucinations, and unpredictable behaviour of AI methods, which can be observed in practice [7–12], are paradoxical and a real puzzle. Indeed, the universal approximation theorem [13] – that tell us that all continuous functions can be approximated arbitrarily well with a neural network – and its cousins [14–16] make it clear that the problem does not lie in the approximation power of neural networks (NNs). Nevertheless, instability and untrustworthiness of AI systems seem hard to cure. Recently, the difficulty of computing stable and accurate NNs have been studied in connection with a newly discovered phenomenon in computation: generalised hardness of approximation (GHA). This phenomenon shows how one can prove the existence of stable and accurate NNs, yet they can only be computed to a certain approximation threshold. Moreover, despite the existence of a stable optimal NN, any attempts of computing it below the approximation threshold will yield an unstable NN.

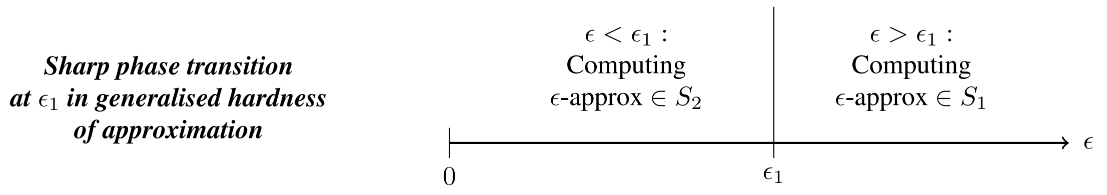
GHA is close in spirit to the long tradition of studying the limitations of computations that have been a core part of modern computer science, and the theory of hardness of approximation (HA) has been a mainstay in this program. HA [17, 18] is the phenomenon that computing an ϵ-approximation to a discrete computational problem can be in P (solvable in polynomial time) for $\epsilon > \epsilon_0$ but the problem becomes NP-hard when $\epsilon < \epsilon_0$, where ϵ_0 is the approximation threshold. In particular,

[*] Corresponding author: e-mail thesing@math.lmu.de

assuming P≠NP we have the following phase transition:


Classical phase transition at ϵ_0 in hardness of approximation
$\epsilon < \epsilon_0$: Computing ϵ-approx is $\notin$ P
$\epsilon > \epsilon_0$: Computing ϵ-approx $\in$ P
0
ϵ_0
ϵ

The phenomenon is a well established part of foundations and has been the subject of several Gödel and Nevanlinna Prizes. HA is almost exclusively associated with combinatorial computational problems, however – as was discovered in [19] and subsequently in [20, 21] and [22] – a generalised form of this phenomenon, namely GHA, can happen in other areas of the computational sciences regardless of the P vs NP question. GHA can be explained as follows: Given an $\epsilon > 0$, the ϵ-approximate computational problem is the problem of computing an approximation that is no more than ϵ away from the true solution – in some appropriate predefined metric. Suppose that we have a computational problem and two classes of approximate computational problems S_1 and S_2 with $S_1 \cap S_2 = \emptyset$ and $\epsilon_1 \geq \epsilon_2 > 0$. We say that the computational problem has an (S_1, S_2)-phase transition at (ϵ_1, ϵ_2) if we have the following: The approximate computational problem is in S_1 for $\epsilon > \epsilon_1$, and the approximate computational problem is in S_2 for $\epsilon < \epsilon_2$. If $\epsilon_1 = \epsilon_2$, we say that the phase transition is sharp and call ϵ_1 the approximation threshold. Schematically, the concept of generalised hardness of approximation with a sharp phase transition can be visualised as follows:


Sharp phase transition at ϵ_1 in generalised hardness of approximation
$\epsilon < \epsilon_1$: Computing ϵ-approx $\in S_2$
$\epsilon > \epsilon_1$: Computing ϵ-approx $\in S_1$
0
ϵ_1
ϵ

Recent works demonstrate how GHA happens in AI [20–22], in particular in DL in inverse problems and image classification. In particular, one can prove the existence of stable NNs for the task, yet there is a phase transition, and the NNs can only be computed to a specific approximation threshold. Hence, the universal approximation theorem gives an overly optimistic view of what deep learning can achieve, and thus we are left with the following crucial question:

Since there are neural networks which can be proven to exist – yet no algorithm can compute them to ϵ accuracy – for which problems can neural network approximations be computed, and to which accuracy?

We address this fundamental question and focus on the possibilities of neural networks and the potential of deep learning methods within the constraint that they must be computable to sufficient accuracy by a computer. The framework extends expressivity and approximation theory to include the algorithm to construct those approximations. The main theorem that we present in Theorem 4.1 – here in a simplified and reader-friendly way without mathematical technicalities – relates the computability of the function class to the approximation with neural networks that can be computed with algorithms.

Theorem 1.1 (Reader friendly light version of Theorem 4.1) *Let $\epsilon > 0$. A family of functions $\mathcal{F} \subset \{f \mid f : [0,1]^{d_1} \to \mathbb{R}^{d_2}\}$ can be approximated in the $\|\cdot\|_\infty$ norm with neural networks – that can be computed by an algorithm from point samples of the functions in $\mathcal{F}$ – to 12ϵ accuracy, if the functions in $\mathcal{F}$ are ϵ-computable from point samples, see Definition 2.3 (i.e. the functions can be computed to a precision ϵ – in the $\|\cdot\|_\infty$ norm – by an algorithm taking point samples of the functions as input).*

This means that computing function approximations through NNs will be optimal – in the sense of best approximation accuracy achievable by an algorithm – up to a small constant compared to any other computational technique. The small constant we obtain is 12. However, this is an upper bound, and it may very well be the case that this can be improved.

2 Theoretical preliminaries and generalised hardness of approximation

We take the word 'compute' literally, and thus the following quote explains the situation succinctly:

"But real number computations and algorithms, which work only in exact arithmetic can offer only limited understanding. Models that process approximate inputs and that permit round-off computations are called for."

— S. Smale (from the list of mathematical problems for the 21st century [23])

Indeed, since e.g. $\sqrt{\cdot}$ and $\exp(\cdot)$ do not have exact representations, inaccurate data input is a daily encounter. The relevance of such a model is further emphasised by the fact that even a rational number such as $1/3$ is stored approximately in base-2 when using floating-point arithmetic, a situation faced by most software. This model allows inaccurate input of arbitrary precision. We call it *the extended model*, see for example [24]. This extended model yields an extended version of Smale's 9th problem from the list of mathematical problems for the 21st century [23]). A very simplified summary of the results in [19] are as follows.

GHA and the extended Smale's 9th problem. *Consider the task of computing a minimiser of linear programs in the extended model, and choose any ℓ^p-norm to measure the error. Then, for any integer $K > 2$, there exists a class of feasible inputs Ω such that, simultaneously, we have the following.*

(i) No algorithm, even randomised, can produce K correct digits of the true solution for all inputs in Ω.

(ii) There does exist an algorithm that provides $K - 1$ correct digits for all inputs in Ω. However, any such algorithm will need an arbitrarily long time to achieve this.

(iii) The problem of producing $K - 2$ correct digits for all inputs in Ω is in P, i.e., can be solved in polynomial time in the number of variables n.

The above result was the first to document the GHA phenomenon in optimisation, where the above statements using integers K, $K - 1$, $K - 2$ can be viewed as 'quantised' phase transition thresholds. Recently, the GHA phenomenon has also been documented in AI [20–22].

The extended model is widespread [24–28]. However, the analysis often becomes technical compared to simpler models relying on exact input. The concept of computability of functions – in the extended model – from point samples is at the heart of modern AI. In particular, we want to compute approximations from point samples for a family of functions $\mathcal{F}$. The reasons are twofold. First, we want to have an algorithm not only for a single function f but for an entire family of functions $\mathcal{F}$. Second, the information that can be used to compute the approximation in supervised learning is only point samples of the function $f \in \mathcal{F}$. The algorithm can choose the accuracy of the point samples. These are only up to a certain accuracy as computers cannot process infinite strings representing the number. This kind of information is modelled with *recursive oracles*.

Definition 2.1 (Point Oracle) An oracle is a recursive mapping $\psi_x : \mathbb{N} \to \mathbb{Q}$ related to a value $x \in \mathbb{R}$ which outputs an approximation $\psi_x(n)$ to x with an accuracy of 2^{-n}.

The concept of an oracle allows us to give an algorithm information with arbitrary but finite precision. This relates to the fact that for some values, there does not exist a finite representation in the dyadic system. Hence, a computer that can only deal with finite inputs will never get this without inaccuracies. However, it is still essential that the algorithm can choose this accuracy.

Definition 2.2 (Sampling oracle) Let $f : [0,1]^{d_1} \to \mathbb{R}^{d_2}$. The *sampling oracle* ψ_f to f is a recursive mapping that takes as input a tuple (l, k) which determines the number of samples $N = (2^l)^{d_1}$ and the accuracy of each evaluation point, i.e. 2^{-k}. For the output we order the elements on the even grid with width 2^{-l} by π_l. Such that $\psi_f : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{N} \times \mathbb{Q}^{d_2}$ gives for the samples $\mathcal{X}_{l,d_1} = \left\{x_i = \pi_l(i), i = 1, \ldots, (2^l)^{d_1}\right\}$ a finite precision of each of the evaluation $f(x_i)$ in $\mathbb{Q}^{d_2}$ with a maximal error of 2^{-k}. The first component of the input and the output specifies which element of the sampling set is used. The second input of the input specifies the accuracy 2^{-k}, and the second part of the output is the approximation to the output at the sampling location specified by the first component.

The location of the sampling points is chosen on the dyadic grid because this allows a wide spread of the points and therefore a better error evaluation if we have information about the modulus of variation. Moreover, the sampling points have with this choice a finite dyadic representation and do not add extra error terms for the numerical analysis. Nevertheless, also other locations are possible to consider, and the choice of the location can give insight to the question at hand. We can now extend the definition of computability to computability from point samples, where we define explicitly where the information is taken from.

Definition 2.3 Let $\epsilon > 0$. A class of functions $\mathcal{F}$ with $f : [0,1]^{d_1} \mapsto \mathbb{R}^{d_2}$ is *ϵ-computable from point samples* if there exists an algorithm $\Gamma_{\mathcal{F}}$ which takes as input

1. a sampling oracle $\psi_f : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{Q}^d$ for $f \in \mathcal{F}$
2. a point oracle $\psi_x : \mathbb{N} \to \mathbb{Q}^{d_1}$ for the position $x \in \mathbb{R}^{d_1}$, where the function is evaluated,

and has the following properties: For any ψ_f : and ψ_x : the algorithm $\Gamma_{\mathcal{F}}$ outputs an approximation y to $f(x)$ such that $\|y - f(x)\|_\infty < \epsilon$. If this holds for all $\epsilon > 0$ we call $\mathcal{F}$ ∞-computable from point samples.

The definition is differs from classical theory in the following points: First, we consider a family of functions and not only a single function. This is because, for a single function, we could design a unique algorithm. However, as we do not know the functions, we are interested in a priori this would give us an advantage that is not existent in practice. The algorithm is supposed to work for all functions in the family. This is, for example, the case with the setting of neural networks. The optimisation problem and the set of possible neural networks, even though restricted to particular architectures for applications, should work for all functions we are using it for and not only for one specific problem. Second, the information from which the function is computed is specified explicitly, and no other information can be used. This is closely related to the previous point. The function should not only be computable in general, but also from the information we have at hand in practice. Here, also the number of necessary samples is of high interest. For the approximation with neural networks we need to have insights about how fast the functions are changing between two point locations. This can be measured with the modulus of variation. We use this measure instead of the modulus of continuity to allow for small jumps of the function.

Definition 2.4 (Modulus of variation) Let $f : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$. The *modulus of variation* of f is given by the function $\omega_f : \mathbb{R}_+ \to \mathbb{R}_+$ defined by

$$\omega_f(h) := \sup\{\|f(x) - f(y)\|_\infty : x, y \in \mathbb{R}^{d_1}, \|x - y\|_2 \leq h\}.$$

Remark 2.5 The term modulus of variation is more general than the concept of modulus of continuity, as we do not assume that $\omega(h) \to 0$ for $h \to 0$. In contrast, for a desired distance $\epsilon > 0$ in the output we will search for sufficiently small $h > 0$ such that $\omega_f(h) \leq \epsilon$. Therefore, there might also be cases where for some $h_0 > 0$ no difference can be observed for smaller $h < h_0$.

There is a key link between function families that are ϵ-computable from point samples and modulus of variation, as the following theorem establishes.

Theorem 2.6 *Let $\epsilon = 2^{-n} > 0$ with $n \in \mathbb{N}$. If a family of functions $\mathcal{F}$ is ϵ-computable from point samples and single-valued, every function $f \in \mathcal{F}$ has the modulus of variation ω_f bounded by 2^{-n+2}.*

3 Neural networks and approximations

For a straightforward discussion of NNs, it is sensible to separate the affine mappings, which are often denoted as weights and the realisation of the network as presented in [29]. We work with feedforward neural networks with ReLU activation.

Definition 3.1 ([29]) Let $d, L \in \mathbb{N}$. A neural network $\mathcal{A}$ with input dimension d and L layers is a sequence of matrix-vector tuples

$$\mathcal{A} = ((A_1, b_1), (A_2, b_2), \ldots, (A_L, b_L)),$$

where $D_0 = d$ and $D_1, \ldots, D_L \in \mathbb{N}$, and where each A_l is an $D_l \times D_{l-1}$ matrix, and $b_l \in \mathbb{R}^{N_l}$. If $\mathcal{A}$ is a neural network as above, and if $\sigma : \mathbb{R} \to \mathbb{R}$ is arbitrary, then we define the associated realization of $\mathcal{A}$ with activation function σ as the map $\Psi = R_\sigma(\mathcal{A}) : \mathbb{R}^d \to \mathbb{R}^{D_L}$ such that $\Psi(z) = R_\sigma(\mathcal{A})(z) = z_L$, where z_L results from the following scheme:

$$\begin{aligned} z_0 &:= z, \\ z_l &:= \sigma(A_l z_{l-1} + b_l), \text{ for } l = 1, \ldots, L-1, \\ z_L &:= A_L z_{L-1} + b_L, \end{aligned}$$

where σ acts component-wise, that is, $\sigma(y) = (\sigma(y_1), \ldots, \sigma(y_d))$ for $y = (y_1, \ldots, y_d) \in \mathbb{R}^d$. Here the activation function or non-linearity is most often the *rectified linear unit* (ReLU) function, i.e. $\sigma(z) = \max\{0, z\}$.

With this definition it is possible to define the algorithm Γ_W that gives the architectures and weights of a network and such represents the network representation. An important property for the analysis of the networks is their Lipschitz constant. The Lipschitz constant can be bounded for a feedforward network with ReLU activation. In [30] the analysis of the Lipschitz constant is studied. The authors derive the upper bound as follows.

Lemma 3.2 ([30]) *For a general neural network Ψ in one dimension we have that the Lipschitz constant $\mathcal{L}_\Psi$ is bounded by $\prod_{i=1}^{L} |A_{\Psi,i}|$.*

The computation of $\mathcal{L}_\Psi$ is NP-hard, and this bound is in general not sharp. A detailed discussion of this problem can be found in [30]. We now come to the main definition of this section which describes, when family of functions can be recursively approximated with neural networks.

Definition 3.3 Let $\epsilon > 0$. A family of functions $\mathcal{F}$ is ϵ-neural network-computable if there exists an algorithm Γ_W which, for any $f \in \mathcal{F}$ takes ψ_f as input and outputs a feed forward neural network Ψ_f with ReLU activation function (this is in form of a string describing the architecture and weights), such that for all $x \in [0,1]^{d_1}$ we have that $\|f(x) - \Psi_f(x)\| \leq \epsilon$.

The definition implies with the bound for the Lipschitz constant that for a constant $c > 1$ there exist $k_0 \in \mathbb{N}$ such that for all $k \geq k$, all $x \in [0,1]^{d_1}$ with any oracle ψ_x we get that $\|f(x) - \Psi_f(\psi_x(k))\| \leq c\epsilon$.

4 The main theorem – Are NNs optimal for computing function approximations?

In the main theorem, we relate the ϵ-computability from point samples from Definition 2.3 to the existence of a neural network approximation.

Theorem 4.1 *Let $\epsilon > 0$. A family of single-valued functions $\mathcal{F} \subset \left\{ f \,|\, f : [0,1]^{d_1} \to \mathbb{R}^{d_2} \right\}$ is ϵ-computable from point samples if it is $\epsilon - 2^{-b}$ neural network-computable for some $b \in \mathbb{N}$. Moreover, a family of single-valued functions $\mathcal{F}$ is 12ϵ-neural network-computable if it is ϵ-computable from point samples. Furthermore, suppose that $\omega : \mathbb{R}_+ \to \mathbb{R}_+$ with $\min_x \omega(x) \leq \epsilon$ and let $\mathcal{F}_\omega = \{f \in \mathcal{F} \,|\, \omega_f(x) \leq \omega(x)\, \text{for all}\ x \in \mathbb{R}_+\}$. Then we can bound the number of sufficient point samples (function evaluations) needed for ϵ-computability of elements in $\mathcal{F}_\omega$ by $N = \tilde{N}^{d_1}$ where $\omega(d_1^{1/2}/\tilde{N}) \leq \epsilon$ and the sufficient accuracy of the point samples is $\epsilon/(2\tilde{N}^{1/2})$.*

4.1 Implications, future work and connection to previous work

The main theorem states that computing function approximations through NNs will be optimal – in the sense of best approximation accuracy achievable by an algorithm – up to a small constant, compared to any other computational technique. The small constant we obtain is 12, and this leads to fundamental open problems:

- *Computing function approximations through NNs – Is it truly optimal?* Theorem 4.1 suggests that computing function approximations through NNs is optimal up to a small constant. However, is is truly optimal in the sense that this constant is equal to 1?
- *Determining the phase transitions in GHA in AI.* Theorem 4.1 demonstrates how NNs can be used to provide upper bounds on phase transitions in GHA in AI. In HA in computer science there is a rich tradition on determining the correct phase transitions – typically by matching upper and lower bounds. A similar theory is needed for GHA.
- *Sampling complexity.* We have seen from Theorem 4.1 that the number of sufficient samples is very large and suffers the curse of dimensionality. This means that if ω is not logarithmic, we might need an exponentially growing number of samples in relation to the desired accuracy. Hence, the number of sufficient samples for a stable approximation may be impossible in certain applications. This relates back to the point of choosing a sensible function class. Theorem 4.1 directly shows that a class of functions with a smaller Lipschitz constant would be easier to be approximated. Alternatively, there might also be other descriptions of function classes that reduce the number of sufficient samples – for example compositional functions, which allow reducing the inherent dimension [29].

Our results are connected to many areas and recent developments.

- *Generalised hardness of approximation.* The first discovery of the generalised hardness of approximation phenomenon was done by A. Bastounis et al. in [19]. Following this framework, the results [20] by M. Colbrook, V. Antun et al. demonstrated how generalised hardness of approximation happens in deep learning when NNs can be proven to exist and solve optimisation problems, yet there are phase transition depending on the accuracy and also the amount of data available, see also [21].
- *The mathematics of the Solvability Complexity Index (SCI) hierarchy.* Our work is part of the greater program on the mathematics behind the SCI hierarchy. The SCI framework was introduced in [31] and continued in the work by J. Ben-Artzi et al. [32], in the work by M. Colbrook et al. [33] as well as in the work by O. Nevanlinna [34,35] and co-authors. The SCI hierarchy is directly related to S. Smale's [36] program on the foundations of computational mathematics and the early work by C. McMullen [37].
- *Instability in AI.* Our results are intimately linked to the instability phenomenon in AI methods – which is widespread [7–12] – and our results add theoretical understandings to this vast research program. There are particular links to the work by B. Adcock et al. [38] and V. Antun et al. [11]. See also recent developments by D. Higham, I. Tyukin et al. [39].
- *Existence vs computability of NNs.* There is a substantial literature on existence results of NNs [16,29,40], see for example the review papers by A. Pinkus [15] and the work by R. DeVore, B. Hanin, and G. Petrova [14] and the references therein. However, as established in [20] by M. Colbrook, V. Antun et al., only a small subset of the NNs than can be proven to exist can be computed by algorithms. However, following the framework of A. Chambolle and T. Pock [41], the results in [20] demonstrate how – under specific assumptions – stable and accurate NNs can be computed.

Acknowledgements Open access funding enabled and organized by Projekt DEAL.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Advances in neural information processing systems **25**, 1097–1105 (2012).
[2] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, arXiv preprint arXiv:1810.04805 (2018).
[3] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, Drebin: Effective and explainable detection of android malware in your pocket., in: Ndss, (2014), pp. 23–26.
[4] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, IEEE Transactions on Image Processing **26**(9), 4509–4522 (2017).
[5] S. Arridge, P. Maass, O. Öktem, and C. B. Schönlieb, Acta Numerica **28**, 1–174 (2019).
[6] B. Adcock and A. C. Hansen, Compressive Imaging: Structure, Sampling, Learning (Cambridge University Press, 2021).
[7] D. Heaven et al., Nature **574**(7777), 163–166 (2019).
[8] S. G. Finlayson, J. D. Bowers, J. Ito, J. L. Zittrain, A. L. Beam, and I. S. Kohane, Science **363**(6433), 1287–1289 (2019).
[9] D. P. Hoffman, I. Slavitt, and C. A. Fitzpatrick, Nature Methods **18**(2), 131–132 (2021).
[10] C. Choi, IEEE Spectrum **September** (2021).
[11] V. Antun, F. Renna, C. Poon, B. Adcock, and A. C. Hansen, Proc. Natl. Acad. Sci. USA **117**(48), 30088–30095 (2020).
[12] N. M. Gottschling, V. Antun, B. Adcock, and A. C. Hansen, arXiv preprint arXiv:2001.01258 (2020).
[13] K. Hornik, M. Stinchcombe, and H. White, Neural networks **2**(5), 359–366 (1989).
[14] R. DeVore, B. Hanin, and G. Petrova, Acta Numer. **30**, 327–444 (2021).
[15] A. Pinkus, Acta Numer. **8**, 143–195 (1999).
[16] H. Bölcskei, P. Grohs, G. Kutyniok, and P. Petersen, SIAM Journal on Mathematics of Data Science **1**(1), 8–45 (2019).
[17] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, Journal of the ACM **43**(2), 268–292 (1996).
[18] S. Arora and B. Barak, Computational Complexity - A Modern Approach (Princeton University Press, 2009).
[19] A. Bastounis, A. C. Hansen, and V. Vlačić, arXiv:2110.15734 (2021).
[20] M. J. Colbrook, V. Antun, and A. C. Hansen, Proceedings of the National Academy of Sciences **119**(12), e2107151119 (2022).
[21] C. Choi, IEEE Spectrum **March** (2022).
[22] L. Gazdad and A. C. Hansen, arXiv preprint arXiv:2209.06715 (2022).
[23] S. Smale, Math. Intelligencer **20**, 7–15 (1998).
[24] K. Ko, Complexity Theory of Real Functions1991).
[25] M. Braverman and S. Cook, Notices of the American Mathematical Society **53**(3), 318–329 (2006).
[26] F. Cucker and S. Smale, Journal of the ACM **46**(1), 113–184 (1999).
[27] C. Fefferman and B. Klartag, Revista Matematica Iberoamericana **25**(1), 49 – 273 (2009).
[28] L. Lovasz, An Algorithmic Theory of Numbers, Graphs and Convexity, CBMS-NSF Regional Conference Series in Applied Mathematics (Society for Industrial and Applied Mathematics, 1987).
[29] P. Petersen and F. Voigtlaender, Neural Networks **108**, 296–330 (2018).
[30] A. Virmaux and K. Scaman, Lipschitz regularity of deep neural networks: analysis and efficient estimation, in: Advances in Neural Information Processing Systems, (2018), pp. 3835–3844.
[31] A. C. Hansen, J. Amer. Math. Soc. **24**(1), 81–124 (2011).
[32] J. Ben-Artzi, M. Marletta, and F. Rösler, Journal of the European Mathematical Society ((to appear)).
[33] M. Colbrook and A. C. Hansen, Journal of the European Mathematical Society ((to appear)).
[34] J. Ben-Artzi, A. C. Hansen, O. Nevanlinna, and M. Seidel, Comptes Rendus Mathematique **353**(10), 931 – 936 (2015).
[35] J. Ben-Artzi, M. J. Colbrook, A. C. Hansen, O. Nevanlinna, and M. Seidel, arXiv:1508.03280 (2020).
[36] S. Smale, Bull. Amer. Math. Soc. **4**(1), 1–36 (1981).
[37] P. Doyle and C. McMullen, Acta Math. **163**(3-4), 151–180 (1989).
[38] B. Adcock and N. Dexter, SIAM Journal on Mathematics of Data Science **3**(2), 624–655 (2021).
[39] I. Tyukin, D. Higham, and A. Gorban, On adversarial examples and stealth attacks in artificial intelligence systems, in: 2020 International Joint Conference on Neural Networks (IJCNN), (2020), pp. 1–6.
[40] D. Yarotsky, arXiv preprint arXiv:1802.03620 (2018).
[41] A. Chambolle, Journal of Mathematical Imaging and Vision **20**(1), 89–97 (2004).