Chapter 7

Computability of optimizers for AI and data science

Yunseok Lee[a], **Holger Boche**[b,c,d,e,f], **and Gitta Kutyniok**[a,g,h,i,*]

[a]*Ludwig-Maximilians-Universität München, Munich, Germany,* [b]*Technical University of Munich, Munich, Germany,* [c]*Ruhr University Bochum, Bochum, Germany,* [d]*Munich Center for Quantum Science and Technology, Munich, Germany,* [e]*Munich Quantum Valley, Munich, Germany,* [f]*BMBF Research Hub 6G-life, Munich, Germany,* [g]*Munich Center for Machine Learning, Munich, Germany,* [h]*University of Tromsø, Tromsø, Norway,* [i]*German Aerospace Center, Oberpfaffenhofen, Germany*

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

The fundamental question underlying all computing is 'What can be (efficiently) automated'?

–Computing as a discipline (Denning et al., 1989)

Contents

Handbook of Numerical Analysis, Volume 25, ISSN 1570-8659, https://doi.org/10.1016/bs.hna.2024.05.005

Abstract
Artificial Intelligence (AI) and Data Science stand as pivotal innovations that revolutionize methods and processes across a multitude of industries. In unison, they facilitate the management, storage, transmission, and analysis of vast data volumes. A significant portion of these challenges are articulated as optimization problems. The potency of AI and Data Science is deeply rooted in the successful resolution of these optimization problems, which are prevalent in areas such as machine learning model fine-tuning, operational research, and logistics.
However, it is crucial to acknowledge that solutions to these optimization problems do not always come with guarantees. This does not necessarily imply that research is lacking in this direction. Instead, it is often a manifestation of the constraints imposed by the nature of the digital hardware used for calculations.
Digital hardware is bound by physical limitations, including constraints on processing power, storage capacity, and time. A key limitation, however, is its inherent binary nature, which can handle only discrete values precisely and may struggle with mathematical functions on real variables. This chapter aims to summarize key results from the field of computability and highlight critical, yet lesser-known issues in optimization theory.

Keywords
Optimization, Artificial Intelligence, Computability, Turing machine, Digital computing, Information theory, Portfolio optimization, Cryptography

MSC Codes
65G50, 68Q05, 68T01, 49J99

1 Introduction

Artificial Intelligence (AI) is a branch of computer science that aims to create systems capable of performing tasks that would normally require human intelligence. These tasks include learning from data, processing natural language, recognizing patterns, and making decisions. Mathematically, AI often involves optimization problems. For example, in deep learning (a subset of AI), we often try to minimize a loss function that measures the difference between the model's predictions and the actual data. Often, an enormous amount of data is needed to achieve satisfactory results. The process of gathering, transmitting, and analyzing this data presents its own set of challenges, and is referred to as data science.

Data Science is an interdisciplinary field that uses scientific methods, processes, algorithms, and systems to extract knowledge and insights from structured and unstructured data. It involves various disciplines such as classical

statistics, data mining, AI, and database systems. AI plays a crucial role in data science since it provides the means to automatically identify patterns in data and make data-driven predictions or decisions. By using AI, data scientists can create models that learn from data, and these models can be used to predict future outcomes or discover underlying patterns. But data science is not just about analyzing data, but also about its effective management. This involves the communication, transfer, and safety-critical sharing, storing, and collecting of data. Many of these challenges, including those related to data management, can be formulated as optimization problems. For instance, when transferring data, we might want to minimize the time it takes or the bandwidth used. When storing data, we might want to minimize the storage space or the cost. Even privacy is commonly formulated as an optimization problem with constraints.

Optimization plays a pivotal role in a multitude of domains, including but not limited to artificial intelligence, data science, mathematics, natural sciences, and engineering. The primary objective in these fields is to either minimize or maximize a specific function, all while adhering to certain constraints. Furthermore, optimization finds extensive applications in diverse areas such as artificial intelligence, medical imaging, and communication. In these fields, it serves as a powerful tool to tackle complex and high-dimensional problems. However, most optimization algorithms are designed and executed on digital devices, such as computers, which can be mathematically modeled as Turing machines. This means that they have inherent limitations in their ability to represent and manipulate real numbers, which often require infinite precision. Therefore, most real numbers have to be approximated by finite representations, such as floating-point numbers, which introduce errors and uncertainties in the computation. These errors can affect the performance and reliability of optimization algorithms, depending on how they handle the approximation of inputs, that cannot be exactly represented on any digital device.

In this chapter, we examine central questions of AI, like deep learning, and of data science, like effective communication and security/cryptography. More precisely, we survey these issues in the specific context of optimization. However, most optimization algorithms only focus on finding the optimal value of a function that measures the quality of a solution. In many cases and applications, the optimizer, which is the solution that achieves the optimal value instead of the optimal value itself, is more important. Furthermore, we survey some of the challenges of finding the optimizer on digital hardware and show examples of optimizers that cannot be computed.

1.1 Overview

In Section 2, we delve into the core concepts of computability and examine various hardware models. Section 3 is dedicated to the introduction of neural networks and the underlying principles of deep learning. The discussion extends into Section 4, where we explore a range of computability and noncomputability

results, emphasizing the intractability of certain functions or properties. Section 5 establishes a general criterion for noncomputability. Subsequent sections provide an analysis of specific instances of noncomputable problems of practical relevance, encompassing areas such as neural networks, inverse problems, and information theory.

1.2 Numerical computations on digital hardware

Numerical computations are ubiquitous in modern science and engineering. One of the most remarkable achievements of the past decade has been the emergence of artificial intelligence and deep learning as powerful tools for solving various problems. However, these methods also face significant challenges and limitations that hinder their reliability, robustness, and applicability in critical domains such as medical imaging or autonomous driving. For instance, deep learning models are often unstable, untrustworthy, vulnerable to adversarial attacks, and lack theoretical guarantees. We hypothesize that one of the possible reasons for these difficulties is the inherent noncomputability of some optimization problems on digital hardware.

1.3 Computability and hardware

Algorithms are not just abstract entities, but also concrete implementations on particular hardware devices. The way numerical data is handled by the hardware, such as how π is represented and computed, can influence the behavior and results of an algorithm. To fully describe an algorithm, one needs to account for the details of how numbers are stored and processed on the hardware platform. We focus on the distinction between digital and analog hardware. Digital hardware makes use of discrete representation to store numbers, i.e., bits, while analog hardware can store and manipulate real quantities directly.

1.3.1 Turing machines

The most fundamental and widely-known hardware model in computer science is the notion of a (deterministic) Turing machine (Turing et al., 1936), which is an abstract device that can perform any computation that can be expressed by a finite set of instructions. A Turing machine operates on an unbounded tape that contains symbols from a finite alphabet, and it can manipulate the tape according to a transition function that depends on its current state and the symbol under the tape head. Turing machines capture the essential features of real-world computation and serve as a theoretical model for modern computers. Therefore, they are a useful tool for studying the computational capabilities and limitations of real machines. The finite alphabet of Turing machines—one might imagine them as bits—is a crucial aspect of their functioning. As a consequence, the number of states that Turing machines can attain is countable. Therefore they are unable to accurately represent uncountable sets, for example, the set of real numbers.

1.3.2 Blum-Shub-Smale machines

A Blum-Shub-Smale machine (Blum et al., 2000) is an analog extension of a Turing machine (Turing et al., 1936), which allows for storage and manipulation of real numbers. The stored numbers can be updated by applying functions from a fixed set of operations, in our case addition, subtraction, multiplication, division, and relations like "$<$", "$>$", and "$=$". Blum-Shub-Smale machines can also output real numbers as a result of their computations. By choosing different sets of operations, one can obtain different classes of Blum-Shub-Smale machines with different computational power and complexity. We refer the reader to Blum et al. (2000) for a comprehensive explanation of BSS machines.

1.3.3 Quantum computers

Quantum computers (Benioff, 1980) are a paradigm of computation that uses the principles of quantum mechanics to perform operations. Quantum computers use quantum bits, or qubits, as the basic unit of information, which can, unlike classical bits, exist in superpositions of two states, 0 and 1. Quantum computers manipulate qubits by applying quantum logic gates, which are operations that change the state of one or more qubits. By designing a sequence of quantum gates, one constructs a Hamiltonian, which describes the evolution of all qubits according to the Schrödinger equation. Through the clever choice of those quantum gates, one can perform calculations more efficiently than a Turing machine would be able to. Quantum computers also fall in the realm of analog hardware models, since the "storage" of qubits is performed physically and does not have to be truncated in its representation.

1.3.4 Neuromorphic computing

Inspired by the architecture and operation of biological cells, neuromorphic hardware (Mead, 1990) is a new approach to computing that uses analog components to mimic the behavior of neurons and synapses. Neuromorphic hardware differs from traditional digital hardware, which relies on binary logic and discrete states, by using chemical and electrical phenomena to process information. The potential benefit of neuromorphic hardware might be its higher energy efficiency. Neuromorphic hardware is a type of analog hardware since it encodes all its internal values through the physical quantities that control the system, such as electric current or voltage.

2 Basic notions

This chapter investigates optimization problems from the perspective of computability theory and effective analysis, which are fields of mathematics that study the boundaries and capabilities of computation and algorithmic approximation. Optimization lies at the heart of AI and data science. Ultimately, the extent to which these fields can be reliably applied hinges largely on our ability

to solve optimization problems reliably and effectively. The notation and concepts required are explained in this section.

2.1 Search for the optimal value

By optimization problems, we refer to the minimization or maximization of some functional $F : X \times Y \to \mathbb{R}$ over a solution space $X \subset \mathbb{R}^n$ and a parameter space $Y \subset \mathbb{R}^m$ for $n, m \in \mathbb{N}$, i.e.,

$$\min_{x \in X(y)} F(x, y) \quad \text{or} \quad \max_{x \in X(y)} F(x, y), \tag{1}$$

where $y \in Y$ is a parameter in the parameter space and $X(y) \subset X$ is a subset of the solution space, depending on y. From now on, and without loss of generality, we only consider maximization problems. This general form allows us to treat most problems from applications (Boyd and Vandenberghe, 2004).

Two main problem settings can be identified in this context: The first asks for the optimal value or an approximation of it, i.e.,

Optimal Value
Construct or approximate a function $\varphi : Y \to \mathbb{R}$ such that

$$\forall_{y \in Y} : \varphi(y) = \max_{x \in X(y)} F(x, y). \tag{2}$$

The second problem setting aims to find an optimizer, i.e.,

Optimizer
Construct or approximate a function $G : Y \to X$ such that

$$\forall_{y \in Y} : \varphi(y) = F(G(y), y) \ \wedge \ G(y) \in X(y), \tag{3}$$

where φ is the function defined by Eq. (2).

It is evident that constructing a function G yields a construction of a function φ. However, in general, the opposite direction does not hold. It is in this sense that finding φ is "easier" than finding G.

Optimization problems suffer the same curse as many other problems of wide interest, namely, there does in general not exist a closed-form solution for either φ or G. Therefore, solutions usually have to be approximated by numerical algorithms. For a wide variety of optimization problems, established algorithms that aim to approximate the optimal solution do exist. A classical class of approaches are iterative solvers, which construct a sequence of approximators. In some cases, it has been proven that this sequence does indeed converge to the optimizer (Arimoto, 1972; Cover, 1984). Depending on the application, either the function G or the function φ is of greater interest. Examples

for the former are portfolio optimization or compressed sensing and, for the latter, exemplary problems are computing the capacity of a channel or solving a deep learning problem. In practice, one often approximates G by an iterative scheme to obtain a sequence $G_n : Y \to X$ and, correspondingly, $\varphi_n : Y \to \mathbb{R}$ through $\varphi_n(y) := F(G_n(y), y)$. Depending on the applied algorithm, one might obtain one or a combination of the following guarantees:

- $\forall_{y \in Y} : \varphi_n(y) \to \varphi(y)$ with or without known convergence speed,
- $\forall_{y \in Y} : G_n(y) \to G(y)$ with or without known convergence speed.

Notice that in this case "known convergence speed" of a convergent Banach space sequence $a_n \to a$ refers to having an explicit description of a function $f : \mathbb{N} \to \mathbb{R}_+$ such that $\lim_{n \to \infty} f(n) = 0$ and $\|a_n - a\| \leq f(n)$ for all $n \in \mathbb{N}$.

Since finding φ is "easier" than finding G a significant amount of research in optimization has (successfully) focused on finding φ over finding G. There are numerous examples of iterative algorithms, which construct $\varphi_n \to \varphi$ with known convergence speed, as well as $G_n \to G$ without known convergence speed. A natural question arises if there is a way to bound the convergence speed of G_n since this would give a stop criterion, which ensures a small error of the computed optimizer to the true optimizer.

Usually, the existence of an optimizer is ensured through compactness of X or $X(y)$. Note that compactness only proves abstract existence, but does not provide a description or approximation of the optimizer itself. In this chapter, we use the following definition, if the optimization problem and its parameter space are evident,

$$Opt(y) := \left\{ x \in X(y) \middle| F(x, y) = \max_{x' \in X(y)} F(x', y) \right\}.$$

Using this notation, the objective of computing an optimizer can be interpreted as the process of identifying a function $G : Y \to X$ such that

$$\forall_{y \in Y} : G(y) \in Opt(y)$$

or an approximation of G, i.e., a function $G^* : Y \to X$ such that G and G^* are close. In our case, we define closeness by the supremum norm

$$\|G - G^*\|_\infty = \sup_{y \in Y} |G(y) - G^*(y)| < \alpha$$

for some $\alpha > 0$.

2.2 Computability

Computability theory studies the limitations and possibilities of various models of computation, in this case, deterministic Turing machines. One of the earliest and most influential results in this field was Turing's introduction of the first

steps in effective analysis and his proof of the undecidability of the Entscheidungsproblem. This proof showed that no algorithm can decide whether every Turing machine with a given input terminates after a finite amount of steps or not. Building on this foundation, the theory of effective analysis, also called computable analysis, emerged. This area focuses on real number functions and, in contrast to other computability areas, is not entirely discrete. We follow the definitions and notation from Pour-El and Richards (2017), which provides a comprehensive introduction to computability theory and its applications.

2.2.1 Turing machines

Deterministic Turing machines are a theoretical model of computation used to classify algorithmic tasks (Turing et al., 1936; Turing, 1938). They consist of a finite set of states, a finite alphabet of symbols, a tape divided into cells that can store one symbol each, a tape head that can read and write symbols on the tape, and a transition function that determines the next state, symbol, and head movement based on the current state and symbol. Deterministic Turing machines are digital in nature because they only allow a finite alphabet of discrete symbols, e.g., bits, and not analog because they cannot represent continuous values with infinite precision.

Definition 1. A function $f : \mathbb{N} \to \mathbb{N}$ is called **recursive** or **computable**, if there exists a Turing machine, which, given the input $x \in \mathbb{N}$, leaves $f(x)$ on its tape after termination. With slight abuse of notation, we equate a recursive function with its corresponding Turing machine.

2.2.2 Computable numbers and sequences

One of the main characteristics of Turing machines is the fact that they operate on a finite alphabet. This means that only natural numbers, and by extension, rational numbers, can be represented exactly by Turing machines. However, many real numbers, such as π or $\sqrt{2}$, are irrational and cannot be expressed as the ratio of two integers. A possible approach is to use sequences of rational numbers that converge to a real number. We say that a sequence of rational numbers is computable if there exists a Turing machine that outputs this sequence.

Definition 2. A sequence of rational numbers $(r_n)_{n\in\mathbb{N}}$ is **computable**, if there exist recursive functions $s, p, q : \mathbb{N} \to \mathbb{N}$ such that

$$\forall_{k\in\mathbb{N}} : r_k = (-1)^{s(k)} \frac{p(k)}{q(k)}.$$

Also, we adopt a notion of convergence, which is more natural for Turing machines.

Definition 3. A sequence of real numbers $(x_n)_{n\in\mathbb{N}}$ does **converge effectively** to a limit $x \in \mathbb{R}$, if

$$\forall_{n\in\mathbb{N}} : |x_n - x| \leq 2^{-n}.$$

Using these definitions, one can say that a real number is computable if it is the effective limit of a computable sequence of rational numbers. It turns out that most commonly used numbers such as algebraic numbers, π and e are computable. For instance, Chaitin's constant, which is defined as the probability that a randomly chosen Turing machine halts, is noncomputable because approximating it would violate the noncomputability of the Entscheidungsproblem.

Definition 4. A real number $x \in \mathbb{R}$ is called **computable**, if there exists a rational computable sequence $(q_n)_{n\in\mathbb{N}}$, such that

$$q_n \to x,$$

where the convergence is effective. The sequence $(q_n)_{n\in\mathbb{N}}$ is called a **representation** of x. We refer to the set of computable real numbers $\mathbb{R}_c$.

We can prove that the computable real numbers $\mathbb{R}_c$ form an algebraic field by showing that each field operation can be computed by a Turing machine, given the Turing machines that compute the operands. As we have also mentioned, most of the commonly used numbers in mathematics and science, such as rational numbers, algebraic numbers, and transcendental numbers like π, and e are computable. Therefore, the computable real numbers encompass a large and rich class of relevant numbers. This explains the practical usefulness of Turing machines for numerical analysis and optimization problems.

2.2.3 Computable continuous functions

A computable function is a function whose values can be calculated by a Turing machine. We already established discrete computable functions on natural numbers. We extend this notion to the continuous setting of functions on computable real numbers. For this, we begin with a formal definition of a computable function that is based on Turing's original paper. The idea is straightforward: Given any representation of an input, we can obtain a representation of the corresponding output.

Definition 5. Let $N, M \in \mathbb{N}$. A function $f : \mathbb{R}_c^N \to \mathbb{R}_c^M$ is **Borel-Turing computable**, if there exists a Turing machine, which transforms all representations $(r_n)_{n\in\mathbb{N}}$ of a vector $x \in \mathbb{R}_c^N$ to representations of $f(x)$.

A more general notion of computability can be introduced by defining the concept of computable real sequences. A real sequence is computable if there exists a computable double sequence of rational numbers that converges effectively to the real sequence:

Definition 6. A sequence of real numbers $(x_n)_{n\in\mathbb{N}}$ is **computable**, if there exists a rational computable double sequence $(r_{n,k})_{n,k\in\mathbb{N}}$ such that

$$\forall_{k,n\in\mathbb{N}} : |q_{n,k} - x_n| \leq 2^{-k}.$$

Based on this definition, we can generalize the concept of computable functions to a more technical level. A function is Banach-Mazur computable if it maps computable real sequences to computable real sequences.

Definition 7. Let $N, M \in \mathbb{N}$. A function $f : \mathbb{R}_c^N \to \mathbb{R}_c^M$ is **Banach-Mazur computable**, if for every computable real vector-valued sequence $(a_n)_{n\in\mathbb{N}}$, the sequence $(f(a_n))_{n\in\mathbb{N}} \subset \mathbb{R}^M$ is computable.

Mazur (1963) established that every Borel-Turing computable function is also Banach-Mazur computable, but the converse is not true in general.

Another important concept is the effective uniform continuity of a function. This definition is a computable version of the widely used notion of uniform continuity in calculus.

Definition 8. A function $f : [a, b] \to \mathbb{R}_c$ with $a, b \in \mathbb{R}_c$ is **effectively uniformly continuous** if there exists a recursive function $d : \mathbb{N} \to \mathbb{N}$ s.t.

$$\forall_{x,y\in[a,b]} : \forall_{N\in\mathbb{N}} : |x - y| \leq d(N)^{-1} \Rightarrow |f(x) - f(y)| \leq 2^{-N}.$$

The all-quantor ranges over every element in the interval, which includes noncomputable numbers as well. Therefore, any function that is effectively uniformly continuous on this interval must also satisfy the standard definition of uniform continuity.

2.2.4 Decidable sets

We next present the concept of decidable sets. A set is said to be decidable if, for any given element, there exists a Turing machine that can determine whether it belongs to the set or not. This notion is most widely used for subsets of natural numbers, which we will start with.

Definition 9. A set $A \subset \mathbb{N}$ is called **decidable**, if the function $\mathbb{1}_A : \mathbb{N} \to \mathbb{N}$, defined by

$$\mathbb{1}_A(n) \coloneqq \begin{cases} 1, & n \in A, \\ 0, & n \in A^c, \end{cases}$$

is recursive.

A relaxation of computable sets are semidecidable sets equipped with a partial decision procedure. This means that there exists a Turing machine, that can always confirm if a given element belongs to the set, but does not reject it if it is not an element. In other words, the algorithm will halt and output yes if the element is in the set, but does not terminate if the element is not in the set. Therefore, one can never be certain whether an element is outside the set or the algorithm is still computing.

Definition 10. A set $A \subset \mathbb{N}$ is called **semidecidable**, if there exists a Turing machine $\mathcal{TM}_A$ such that $\mathcal{TM}_A(n)$ outputs 1, if $n \in A$, and $\mathcal{TM}_A(n)$ does not terminate, if $n \in A^c$.

The concept of (semi-)decidability can be readily applied to subsets of real numbers, with a minor adjustment in the definition. Rather than testing if a given element is part of a set, we take for granted that the element is already contained in a bigger set, and we only have to determine if it is also included in a narrower subset of that set.

Definition 11. Given $B \subset \mathbb{R}_c^n$, a set $A \subset B$ is called **(semi-)decidable**, if there exists a Turing machine $\mathcal{TM}_A$ such that $\mathcal{TM}_A(x)$ outputs 1, if $x \in A$ and $\mathcal{TM}_A(x)$ outputs 0 (resp. does not terminate), if $x \in B \backslash A$. $\mathcal{TM}_A(x)$ can either output an arbitrary symbol or not terminate for $x \in B^c$.

One of the most famous noncomputable problems is the Entscheidungsproblem, which involves deciding whether a given Turing machine will ever stop on a given input or not. No algorithm can solve this problem for all Turing machines and inputs (Turing, 1938). A direct implication of this result is the existence of sets that are semidecidable but not decidable. This can be shown by identifying the set of Turing machines $\mathcal{T}$ and the set of possible inputs I with the set of natural numbers, which is feasible since both $\mathcal{T}$ and I are countably infinite sets. For this define

$$F := \{(t, i) \in \mathcal{T} \times I | t(i) \text{ terminates}\} \subset \mathcal{T} \times I \cong \mathbb{N},$$

the set of Turing machines and inputs, which do terminate. Then F is semidecidable using the Turing machine, which applies i on t for any $(t, i) \in \mathcal{T} \times I$ and outputs 1 if it terminates. By definition, this will output 1 if $(t, i) \in F$ and not terminate otherwise. But the noncomputability of the Entscheidungsproblem implies there is no Turing machine, which can decide if (i, t) is in F or not.

3 Deep learning as a key technique of artificial intelligence

Artificial intelligence is a tremendously successful field that aims to create programs which perform tasks that normally require human intelligence. A key component of AI is deep learning, which has achieved remarkable results in many challenging domains, by now already surpassing human performance in some cases. This section introduces the basic ideas of deep learning and discusses some of its drawbacks.

3.1 Essence of deep learning

Deep neural networks are composed of multiple layers of artificial neurons, which are mathematical functions consisting of adjustable linear-affine functions followed by a nonlinear activation function. By adjusting the parameters of each neuron's linear-affine function, the neural network is able to "learn" a function. This learning process is facilitated by employing a variant of gradient descent on a large data set.

Definition 12. A **(feed-forward) neural network** are functions $\Phi : \mathbb{R}_c^n \to \mathbb{R}_c^m$ of the form

$$\Phi(x) := (A_L \circ \rho \circ A_{L-1} \circ \ldots \circ \rho \circ A_1)(x)$$

where $L \in \mathbb{N}$ and

$$\forall_{l=1,\ldots,L} : A_l x = W_l x + b_l, \quad W_l \in \mathbb{R}_c^{n_l \times n_{l-1}}, b_l \in \mathbb{R}_c^{n_l}$$

with $n_0 = n, n_L = m$ and $\rho : \mathbb{R} \to \mathbb{R}$ is a (nonlinear) activation function, applied component-wise. The coefficients of W_l are called **weights** and b_l are called **biases**.

For a more general theory on neural networks, we refer to Berner et al. (2021).

The choice of activation functions plays an important role in the performance of deep learning models. Among the commonly used activation functions are ReLU, tanh, and sigmoid with the most popular being ReLU defined as $\rho(x) = \max(0, x)$. Therefore, in this chapter, we will focus on the analysis of neural networks using the ReLU activation functions. We define $\mathcal{NN}$ as the set of all neural networks. Neural networks are optimized using a large dataset $(x_i, y_i)_{i=1,\ldots,n} \subset \mathbb{R}_c^n \times \mathbb{R}_c$ by minimizing a loss function $\mathcal{L} : \mathcal{P}(\mathbb{R}_c^n \times \mathbb{R}_c) \times \mathcal{NN} \to \mathbb{R}_c$, which measures the "fit" of the neural network to the given data. For a simple regression task, a possible choice of loss function is the L^2-distance

$$\mathcal{L}((x_i, y_i)_{i=1,\ldots,n}, \Phi) = \frac{1}{n} \sum_{i=1}^{n} (\Phi(x_i) - y_i)^2.$$

Depending on the specific objective of the problem, such as classification or more complex forms of regression, other loss functions are commonly employed. Given a dataset and a loss function, the goal then is to find the optimal values of the weights and biases that minimize the loss. This is achieved by applying a variant of gradient descent, which is an iterative optimization algorithm. The basic idea of gradient descent is to update each parameter λ by subtracting a small fraction $\mu > 0$ of the gradient of the loss function with respect to that parameter

$$\lambda \leftarrow \lambda - \mu \frac{\partial \mathcal{L}}{\partial \lambda}((x_i, y_i)_{i=1,\ldots,n}, \Phi).$$

The efficient calculation of the gradient is done by backpropagation.

3.2 Drawbacks of deep learning

Despite its impressive achievements in various fields, such as computer vision, natural language processing, and speech recognition, deep learning also has serious drawbacks, especially when it comes to safety-critical applications like

medical imaging and autonomous driving. Some of the current main challenges of deep learning are:

1. *Intransparency:* Neural networks are often viewed as black boxes, meaning that their internal mechanisms and reasoning processes are not clear or understandable. This poses a problem for explaining and validating the results of deep learning models, as well as for detecting and fixing potential mistakes or biases.
2. *Theory-to-Practice gap:* Deep learning lacks rigorous theoretical foundations that can guarantee its performance and robustness. It is not well understood why deep learning works so well for some problems but not for others, or under what conditions it will fail or degrade.
3. *Instability:* Neural networks are susceptible to showing inconsistent behavior or poor generalization in some situations, even when they perform well on average. For instance, they are vulnerable to adversarial attacks, which are malicious inputs that are designed to fool or mislead the models by introducing subtle perturbations that are imperceptible to humans.

In addition to the well-known challenges of deep learning, mentioned above, there is another aspect that is sometimes overlooked: The computational limits of running neural networks on digital hardware. As we explained in Section 2, real numbers cannot be represented precisely by digital devices, which introduces errors and uncertainties in the computations. This issue is not inherent to neural networks themselves, but rather a consequence of the interplay of neural networks with digital hardware. Thus, these issues could be reduced or avoided on analog hardware.

4 Computability of optimal values and existence of computable optimizers

Depending on the dimension and regularity of the functional we can make different statements about the computability of the maximum, the maximizer, and if you can calculate the maximum and maximizer by computable means. We present a collection of results, encompassing both computable and noncomputable optimal values and optimizers. Our survey begins with the simplest one-dimensional scenario, before progressing to the more complex multidimensional cases.

4.1 One-dimensional optimization

For the one-dimensional case, consider a computable function $f : [a, b] \to \mathbb{R}_c$ that is effectively uniformly continuous on a computable interval. Then by Pour-El and Richards (2017, Theorem 7) $\max_{x\in[a,b]} f(x)$ is computable. Furthermore, the proof of the theorem provides an extra insight. It is not only the case that the maximum of every computable effectively uniformly continuous

function is computable, but also that there exists an algorithm for finding this maximum. We present the proof here for completeness:

Theorem 1. *Let $a, b \in \mathbb{R}_c$ with $a < b$ and define $\mathcal{F} := \{f : [a,b] \to \mathbb{R}_c |\ f$ Borel-Turing computable and effectively uniformly continuous$\}$. Then define $M : \mathcal{F} \to \mathbb{R}_c$ by*

$$M(f) = \max_{x \in [a,b]} f(x).$$

For a function $f \in \mathcal{F}$ define T_f as the Turing machine, which given a representation of $x \in \mathbb{R}_c$, outputs a representation of $f(x)$. Then there exists a Turing machine $T_{\max}$, which given a Turing machine T_f with $f \in \mathcal{F}$, outputs a representation of $M(f)$.

Remark. We write with slight abuse of notation,

$$T_{\max}(T_f) \text{ outputs a representation of } M(f).$$

Proof. Define the computable sequence

$$s_n = \max \left\{ f\left(a + \frac{j}{n}(b-a)\right) \middle| 1 \le j \le n \right\}.$$

By effective uniform continuity of f, there exists a recursive function $d : \mathbb{N} \to \mathbb{N}$, such that,

$$\forall_{x,y \in [a,b]} : \forall_{n \in N} : |x - y| \le d(n)^{-1} \Rightarrow |f(x) - f(y)| \le 2^{-n}. \tag{4}$$

Let $M \in \mathbb{N}$ with $M > b - a$ and $e(n) := M \cdot d(n)$. Next, we will show that

$$n \ge e(N) \Rightarrow |s_n - M(f)| \le 2^{-n}.$$

Given $n \ge e(N)$, this implies

$$\frac{b-a}{n} < \frac{M}{n} \le \frac{M}{e(N)} = \frac{1}{d(N)}.$$

Since each interval

$$I_j := \left[a + \frac{j}{n}(b-a), a + \frac{j+1}{n}(b-a) \right],\ j = 0, \ldots, n-1$$

has length $\frac{b-a}{n}$, we can deduce that

$$\forall_{j=1\ldots n-1} : \left| \max_{x \in I_j} f(x) - f\left(a + \frac{j}{n}(b-a)\right) \right| \le 2^{-n}.$$

This immediately implies

$$\forall_{n \ge e(N)} : |s_n - M(f)| \le 2^{-n},$$

hence (4) is proven. Since $(s_n)_{n\in\mathbb{N}}$ is a computable sequence, it is a representation of $M(f)$. Thus, one can use the Turing machine, which maps

$$f \mapsto (s_n)_{n\in\mathbb{N}}$$

to construct $M(f)$ for any $f \in \mathcal{F}$. □

This theorem is remarkable, as it guarantees the existence of a Turing machine that can calculate the maximal value for any computable and effectively uniform continuous function on a compact interval. However, when we focus on the maximizer instead, we encounter a surprising phenomenon. Specker showed in Specker (1959) that there does exist a function whose maximal value is computable, but whose maximizers are not. This circles back to the fact mentioned in Subsection 2.2.1 of finding a maximal value being "easier" to solve (see Problem (2)) than constructing a maximizer as in Problem (3). As shown, this is already observable in the one-dimensional case.

If a maximizer is at an isolated point though, the maximizer is always computable.

Theorem 2. *Let $a, b \in \mathbb{R}_c$ computable numbers with $a < b$ and $f : [a, b] \to \mathbb{R}_c$ be a computable and effectively uniformly continuous function with an isolated maximum,*

$$\exists_{\hat{x}\in[a,b]} : f(\hat{x}) = \max_{x\in[a,b]} f(x) \wedge \forall_{x\in[a,b]\setminus\{\hat{x}\}} : f(x) < f(\hat{x}).$$

Then $\hat{x} \in \mathbb{R}_c$ is computable.

Proof. Define

$$F_1(x) = \max_{\tau\in[a,x]} f(\tau).$$

Then F_1 is monotonously nondecreasing with $\forall_{x<\hat{x}} : F(x) < F(\hat{x})$ and $\forall_{x\geq\hat{x}} : F(x) = F(\hat{x})$. Also by Theorem 1, $F_1(x)$ is a computable and continuous function. Now for fixed $n \in \mathbb{N}$ and $k = 0, \ldots, 2^n$ consider the computable numbers

$$l_n^k := F_1\left(a + \frac{(b-a)k}{2^n}\right).$$

By comparing the $(n+2)$th element of the representations of l_n^k and $f(\hat{x}) - 2^{-n+1}$ one can compare l_n^k and $f(\hat{x}) - 2^{-n+1}$ up to an error of 2^{-n+1}, i.e., there exists a Turing machine which can check if

$$l_n^k > f(\hat{x}) - 2^{-n}$$

holds or not. Next, define the computable sequence

$$x_n := \min\left\{a + \frac{(b-a)k}{2^n} \middle| k \in \{0, \ldots, 2^n\} \wedge l_n^k > f(\hat{x}) - 2^{-n}\right\}.$$

Then x_n is a nondecreasing sequence with $x_n \to \hat{x}$.

Now by choosing a subsequence (n_k) one can construct a computable sequence $(x_{n_k})_{k\in\mathbb{N}}$ that converges to $\hat{x}$ effectively

$$\forall_{k\in\mathbb{N}} : |x_{n_k} - \hat{x}| < 2^{-k}.$$

So $\hat{x}$ is computable. □

For certain functions Theorem 2 implies the existence of a computable optimizer $\hat{x}$, which can be computed by a Turing machine $\mathcal{T}_{\hat{x}}$. For every $n \in \mathbb{N}$, this Turing machine outputs a rational number r_n with the property that

$$|\hat{x} - r_n| \leq 2^{-n}.$$

But this does not imply that there is a Turing machine that can generate $\mathcal{T}_{\hat{x}}$ given a Turing machine that defines the matching function. On the contrary, we will demonstrate later that this is not feasible in general. More examples of computable and noncomputable functions can be found in Pour-El and Richards (2017).

4.2 Computability of convex optimizers in higher dimensions

The previous sections focused on one-dimensional functions. However, many real-world applications require functions of more than one variable. Therefore, in this section, we extend the concepts and methods of optimization to multivariate functions and present the similarities and differences to the univariate case. Using the same proof as in 1 one can establish the multidimensional analogue of Theorem 1.

Theorem 3. *Let $I \subset \mathbb{R}_c^n$ be a computable rectangle and define $\mathcal{F} := \{f : I \to \mathbb{R}_c \mid f$ Borel-Turing computable and effectively uniformly continuous$\}$. Then define $M : \mathcal{F} \to \mathbb{R}_c$ by*

$$M(f) = \max_{x\in I} f(x).$$

For a function $f \in \mathcal{F}$ define T_f as the Turing machine, which given a representation of $x \in R_c^n$, outputs a representation of $f(x)$. Then there exists a Turing machine $T_{\max}$, which given a Turing machine T_f with $f \in \mathcal{F}$, outputs a representation of $M(f)$.

The following theorem can be considered a generalization of Theorem 2. It establishes the existence of a computable maximizer for convex functionals in the multidimensional case.

Theorem 4 (Wong, 1996). *Let $I \subset \mathbb{R}^n$ be a computable rectangle, and let $f : I \to \mathbb{R}$ be a convex, continuous, and Banach-Mazur computable. Then there exists a computable $x^* \in I \cap \mathbb{R}_c^n$, which minimizes f, i.e.,*

$$f(x^*) = \min_{x\in I} f(x).$$

We, again, remark that this theorem does not imply that the maximizer x^* can be calculated by computable means given f and I.

4.3 Example of multidimensional optimization in information theory

One way to generalize Theorem 4 is by relaxing the domain I to other geometries than computable rectangles or parameterizing I. For illustration, we present the well-known convex optimization problem of channel capacity from information theory.

Information theory plays a crucial role in AI. The sparse and compressed representation of data is not just useful in storing and transmitting data but also in designing efficient AI algorithms and neural network architectures. In this subsection, we consider the case of optimal transmission capacity of data through a discrete channel. The channel capacity is again described through a maximization problem. The goal is to maximize the mutual information between the input and output of a discrete memoryless channel, which measures the amount of information transmitted through the channel.

4.3.1 Communication model

In information theory, a point-to-point channel with one receiver and one transmitter is modeled by two discrete random variables X and Y over the probability spaces $\mathcal{X}$ and $\mathcal{Y}$. If we choose $\mathcal{X}$ and $\mathcal{Y}$ to be finite, we are describing a discrete memoryless channel (DMC). The channel itself is then given by a stochastic matrix $W \in \mathbb{R}^{m\times n}$, where $|\mathcal{X}| = n$ and $|\mathcal{Y}| = m$. X, Y and W are related by

$$W(x) = P(Y|X = x).$$

We define the mutual information of two discrete random variables X, Y over $\mathcal{X}$, $\mathcal{Y}$ as

$$I(X, Y) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} P_{(X,Y)}(x, y) \log\left(\frac{P_{(X,Y)}(x, y)}{P_X(x)P_Y(y)}\right),$$

where $P_{(X,Y)}$ is the probability mass function of (X, Y), and P_X and P_Y are the probability mass functions of X and Y. Now the capacity $C(W)$ of a DMC W is the maximal mutual information over all possible distributions over $\mathcal{X}$

$$C(W) := \max_{X\in\mathcal{P}(X)} I(X, Y).$$

In this case $\mathcal{P}(\mathcal{X})$ can be modeled as a convex set

$$\mathcal{P}(\mathcal{X}) = \left\{ (P_1, \dots P_n) \in \mathbb{R}^n_+ \,\middle|\, \sum_i P_i = 1 \right\}.$$

The capacity of a DMC is well established and goes back to Shannon (1948).

4.3.2 Blahut-Arimoto

The Blahut-Arimoto algorithm (Blahut, 1972; Arimoto, 1972) establishes a computable way to compute the capacity of any DMC. It works by iteratively approximating an estimate of the input distribution. So given a channel W, Blahut-Arimoto outputs a sequence of probability distributions $X_n \in \mathcal{P}(\mathcal{X})$. It is proven that the corresponding mutual information converges to the capacity

$$\lim_{n\to\infty} I(X_n, Y) = C(W).$$

Also, X_n do converge to a probability distribution X^*, which maximizes the mutual information

$$\lim_{n\to\infty} X_n = X^* \ \wedge \ I(X^*, Y) = C(W).$$

However, there is a significant difference in the effectiveness of the convergence of both quantities. On one hand, there is a stopping criterion for the capacity, which guarantees arbitrarily small error, i.e., for any $N \in N$ one can computably choose $n \in \mathbb{N}$, such that,

$$\|I(X_n, Y) - C(W)\| \leq 2^{-N}.$$

On the other hand, there is no computable stopping criterion known, that guarantees arbitrarily small approximation error for X^*. The question if such a stopping criterion exists has been answered lately in Boche et al. (2022) with a clear no. More precisely, the authors proved that no Banach-Mazur computable function exists at all, which maps all DMCs to an approximation of the optimal input distribution X^* with an arbitrarily small error. We refer to Theorem 9 for more details.

This result is unexpected, as both the set we optimize over and the objective function of the optimization problem are convex. Convexity is usually considered to be a desirable property that simplifies the analysis and solution of optimization problems. However, we show that there exist convex optimization problems that are not computable. On the contrary, we can use convexity to prove the existence of computable optimizers, even though finding those might not be computable. Define for the channel W the convex set of optimizers $\mathcal{P}_{opt}(W) \coloneqq \{X \in \mathcal{P}(\mathcal{X}) | I(X, Y) = C(W)\}$ where Y is determined through W and X.

Theorem 5 (Hinted at in Boche et al. (2022)). *Given a computable stochastic matrix W, there exists a $X^* \in \mathcal{P}_{opt}(W)$, s.t. X^* is computable, i.e., $X_i^* \in \mathbb{R}_c$ for all $i = 1, \ldots, n$.*

Proof. If $|\mathcal{P}_{opt}(W)| = 1$, we can apply Theorem 3 on every dimension separately, which finishes the proof.

If $|\mathcal{P}_{opt}(W)| > 1$, let $X^0, X^1 \in \mathcal{P}_{opt}(W)$ with $X^0 \neq X^1$. Since the mutual information is convex, $\mathcal{P}_{opt}(W)$ is convex, too. So for any $\lambda \in [0,1]$ the following distribution is also an optimizer $X^\lambda := (1-\lambda)X^0 + \lambda X^1 \in \mathcal{P}_{opt}(W)$. Now let $i_1 \in \{1,\dots,n\}$ be an index s.t. $X^0_{i_1} \neq X^1_{i_1}$. So $X^\lambda_{i_1} = (1-\lambda)X^0_{i_1} + \lambda X^1_{i_1} \in [\min(X^0_{i_1}, X^1_{i_1}), \max(X^0_{i_1}, X^1_{i_1})]$. Then there exists a $\lambda^* \in [0,1]$ s.t. $X^{\lambda^*}_{i_1} \in \mathbb{R}_c$ since $\mathbb{R}_c \subset \mathbb{R}$ is dense.

Now consider the set $\mathcal{P}^1_{opt}(W) := \{X \in \mathcal{P}_{opt}(W) | X_{i_1} = X^{\lambda^*}_{i_1}\}$ and the optimizer $X^*_1 := (X^0_1, \dots X^0_{i_1-1}, X^{\lambda^*}_{i_1}, X^0_{i_1+1}, \dots X^0_n) \in \mathcal{P}^1_{opt}(W)$. Now we can repeat the argument as before. If $|\mathcal{P}^1_{opt}(W)| = 1$, one can conclude that the only optimizer is computable by applying Theorem 3 on every dimension separately.

If $|\mathcal{P}^1_{opt}(W)| > 1$, we can repeat the procedure by fixing $X^1, X^2 \in \mathcal{P}^1_{opt}(W)$ and choosing an index $i_2 \in \{1,\dots,n\}$ with $i_1 \neq i_2$, s.t. $X^1_{i_2} \neq X^1_{i_2}$. Then we can construct an optimizer X^{λ^*}, s.t., $X^{\lambda^*}_{i_2} \in \mathbb{R}_c$. After repeating this process $k < n$ times, we end up with a set of optimizer $\mathcal{P}^k_{opt}(W)$ with just one computable element. □

We note that this proof is an adaptation of the proof of Theorem 4 and again does not provide any computable procedure to identify a computable optimizer, just its existence. As can be seen from our adaptation, the assumption in Theorem 4 of I of being a rectangle can be relaxed.

4.4 Other computable and noncomputable functions

Most commonly used functions are computable over any computable rectangle as domain without a singularity. This includes e^x, $\sin x$, $\cos x$, $\sqrt{x}$, $\log(x)$, and $\Gamma(x)$ as well as its compositions. To prove this it suffices to consider the approximation formulas of these functions commonly used in numerical mathematics and confirming they have a computable stopping criterion, which ensures arbitrarily small approximation error. As an example we can prove computability of $\sqrt{x}$ using Heron's formula (Schwarz and Köckler, 2013) for $x^* \in \mathbb{R}_c$

$$x_{n+1} := \frac{1}{2}\left(x_n + \frac{x_0}{x_n}\right)$$

with any suitable initial guess $x_0 \in \mathbb{R}_c$. It is well-known that the relative error

$$r_n := \frac{x_n}{x^*} - 1,$$

is always positive (assuming an appropriate initial guess) and decreases at least exponentially, i.e., $r_n \leq \frac{r_{n-1}}{2}$. Using this convergence result one can calculate for any $N \in \mathbb{N}$ an index $N_0 \in \mathbb{N}$ s.t.

$$\forall_{m > N_0} : |\sqrt{x^*} - x_m| \leq 2^{-N}.$$

Also, definite integrals of computable, uniformly effective continuous functions are computable. The derivative of a computable function is not computable in general, even if the derivative exists. However computable functions in C^2 do have a computable derivative. For a comprehensive theory in computable analysis, we refer the reader to Pour-El and Richards (2017) and Weihrauch (2000).

5 Finding the optimizer is not effectively solvable

This section begins by addressing a key question: Does there exist a Turing machine capable of outputting the optimizer when provided with a description of the functional and the domain? Previously, we focused on determining which optimal values and optimizers are computable. However, a subtly different yet arguably more important question is whether a computable optimizer can be discovered through computable methods. This section concentrates on results addressing this question.

5.1 General noncomputability theorem

In order to determine whether optimizers can be computed, we examine general optimization problems in multiple dimensions, as referenced in Section 2. If the solving function, denoted as G, which is defined in Problem (3), exhibits a certain type of computable discontinuity along with other technical properties, it becomes impossible to efficiently calculate the optimizer. Interestingly, this can occur even if the function F is both continuous and convex. More importantly, it has been established that not only is the calculation of the optimizer noncomputable, but also its approximation. This implies that there will always be at least one instance where any computable machine will fail to approximate the optimizer by a fixed positive constant.

Theorem 6 (Main Theorem (Lee et al., 2023)). *Let X, Y, $X(y)$ and F be as described in Section 2. Let $G : Y \to X$ such that, for all $y \in Y$, we have $G(y) \in Opt(y)$. Now let $Y_1, Y_2 \subset Y$, $y_1^* \in Y_1$, $y_2^* \in Y_2$ and $y_* \in Y$, and $\gamma : [-1, 1] \to Y$, a Turing computable, continuous path such that:*

- **(i)** $Y_1 \cap Y_2 = \varnothing$ *and* $G(Y_1) \cap G(Y_2) = \varnothing$,
- **(ii)** $\inf_{y_1 \in Y_1 y_2 \in Y_2} \|y_1 - y_2\| = 0$,
- **(iii)** $\inf_{y_1 \in Y_1 y_2 \in Y_2} \|G(y_1) - G(y_2)\| = \kappa > 0$,
- **(iv)** $\gamma(-1) = y_1^*$, $\gamma(1) = y_2^*$, $\gamma(t_0) = y_*$, *for some* $t_0 \in (-1, 1)$,
- **(v)** $\gamma([-1, t_0)) \subset Y_1$ *and* $\gamma((t_0, 1]) \subset Y_2$,
- **(vi)** $G(Y_1) \subset G(Y_1) \cup G(Y_2)$ *is decidable.*

Then G cannot be Borel-Turing computable. In fact, there does not even exist a Borel-Turing computable function, which can approximate G by up to an absolute error of $\alpha < \frac{\kappa}{2}$, i.e. there does not exist a Borel-Turing computable function $G^* : Y \to X$ such that $\|G - G^*\|_\infty \leq \alpha$. If we replace condition (vi) by

(vii) $Y_1 \subset Y_1 \cup Y_2$ *is decidable,*

then G can even not be Banach-Mazur computable. In fact, there does not even exist a Banach-Mazur computable function, which can approximate G by up to an absolute error of $\alpha < \frac{\kappa}{2}$.

This theorem is noteworthy as it suggests the impossibility of computing an optimizer, even when the optimizer is computable. This holds true even under favorable conditions, such as the convexity of the domain and the function. As we will discuss, this scenario occurs frequently in many practically relevant cases.

5.2 Noncomputability of neural networks and other optimizers

The previous theorem is a powerful tool that can be applied to various optimization problems. In this section, we demonstrate its effectiveness and generality by presenting some selected examples from different fields of study.

5.2.1 Neural networks

Smale's 1998 discussion on the limitations of AI (Smale et al., 1998) can be seen as a precursor to the specific case of neural networks, which can be viewed as an extension of these initial concepts (Colbrook et al., 2022). A remarkable example of noncomputability in neural networks is the impossibility of finding or approximating the optimal weights for a given loss function. Although other noncomputability results exist for neural networks, these typically focus on a particular application (Colbrook et al., 2022; Boche et al., 2023), or they diverge from the domain of effective analysis by examining general functions that are not tied to Turing machines (Grohs and Voigtlaender, 2023). For more on the works by Colbrook et al. (2022); Boche et al. (2023) we refer to Subsection 2.6.5. The noncomputability of neural networks were demonstrated in Lee et al. (2023) for a simple shallow neural network model with ReLU activation function without biases. Of course more sophisticated and deeper architectures exist, like convolutional neural networks, transformers, recursive neural networks, U-nets, and a whole zoo of combinations of these. Still, the most simple case of a simple shallow neural network is "complicated enough" to lead to noncomputability.

Theorem 7 (Neural Network (Lee et al., 2023)). *Given $d \geq 14$ and a data set $\mathcal{D} = \prod_{i=1}^{d}(x_i, y_i) \in (\mathbb{R}_c^3 \times \mathbb{R}_c)^d$, consider the minimization problem*

$$\min_{A \in \mathbb{R}_c^{3\times 3}} \sum_{i=1}^{d} (\|\rho(Ax_i)\|_1 - y_i)^2.$$

Let $G : (\mathbb{R}_c^3 \times \mathbb{R}_c)^d \to \mathbb{R}_c^{3\times 3}$ such that, for all $\mathcal{D} \in (\mathbb{R}_c^3 \times \mathbb{R}_c)^d$, we have: $G(\mathcal{D}) \in Opt(\mathcal{D})$. Then G is not Banach-Mazur computable.

All functions $G^* : (\mathbb{R}_c^3 \times \mathbb{R}_c)^d \to \mathbb{R}_c^{3\times 3}$ *satisfying*

$$\|G - G^*\|_\infty \leq \alpha < 4$$

are also not Banach-Mazur computable.

This theorem establishes that perfect loss-minimizing algorithms for neural networks cannot exist in the special case of a shallow neural network with the architecture described above. While this conclusion does not necessarily extend to wider and deeper neural networks, it is reasonable to expect similar results in more complex scenarios. Intuitively, as the number of parameters increases, finding loss-minimizing neural networks becomes more challenging. Consequently, we anticipate that analogous results hold true for general neural networks. It is essential to note that this theorem does not imply nonapproximability of neural networks when interpreted as functions; rather, it pertains specifically to the nonapproximability of their weights. Although some might view this limitation as a drawback of the theorem—since researchers are often more interested in the neural network's function itself rather than its precise weights—it serves as a caution against methods that directly manipulate or utilize trained neural network weights. Examples include techniques like Dropout (Hinton et al., 2012) and Layer-Wise Relevance Propagation (Montavon et al., 2019).

Furthermore, the nonapproximability of weights highlights the delicate nature of neural networks. It underscores the importance of ensuring that the corresponding neural network function indeed approximates the desired function accurately. Interestingly, if there were a computable method to recover all possible weight configurations from a given neural network function consistently, it would imply nonapproximability of the neural network function itself. We believe that additional noncomputability results for deep learning exist, and inherent issues such as instability (Gottschling et al., 2020) may represent fundamental challenges for neural networks on digital hardware—one that cannot be entirely overcome.

5.2.2 Financial mathematics - information theory

In portfolio optimization, the stock market can be modeled by a random vector $X \in \mathbb{R}_+^m$, where each component describes a separate stock. A portfolio $b \in \mathbb{R}_+^m$ is a vector such that $\sum_i b_i = 1$, which describes the allocation of the available funds. The vector $b^t X$ describes the evolution of the portfolio after one time step. Due to the multiplicative nature of investments, it is natural to maximize the convex functional (Cover, 1984; Latane, 1959)

$$\max_b \mathbb{E}[\log(b^t X)],$$

which is the expected return after one time step. A well-known approach to this optimization problem is an iterative algorithm found by (Cover, 1984). Cover's

algorithm uses similar ideas as the Blahut-Arimoto algorithm (Arimoto, 1972; Blahut, 1972) from information theory.

We present the result of Lee et al. (2023) that even though such an effective algorithm exists, finding a maximizing portfolio is noncomputable in general. This implies no approximation guarantee for optimal portfolios in Cover's algorithm—or any other algorithm—can be made.

We consider the case of a discrete random vector $X(\cdot) = \sum_{i=1}^{n}\sum_{j=1}^{m} p_{i,j}\delta_{x_{i,j}}(\cdot)e_j$, where $p_{i,j} > 0$ are probabilities, i.e., $\sum_{i,j} p_{i,j} = 1$, and $x_{i,j} \in \mathbb{R}_+$ are the possible outcomes. Also $e_j \in \mathbb{R}^m$ are the standard basis vectors and we assume $\forall_{j,i_1\neq i_2} : x_{i_1,j} \neq x_{i_2,j}$. We define this set of discrete random vectors as $\mathcal{D}_{n,m}$, which is identified with the Euclidean space $\mathbb{R}_c^{2nm}$ here by equating $p_{i,j}$ and $x_{i,j}$ to vector entries.

Theorem 8 (Log-Optimal Portfolio). *Let $X \in \mathcal{D}_{n,m}$. Define $W : \{b \in \mathbb{R}_+^m | \sum_i b_i = 1\} \to \mathbb{R}$ by*

$$W(b) := \mathbb{E}[\log(b^t X)],$$

and consider the corresponding maximization problem. For all $n, m \in \mathbb{N}_+$ with $m > 1$ define a function $G : \mathcal{D}_{n,m} \to \{b \in \mathbb{R}_+^m | \sum_i b_i = 1\}$ such that for all $y \in \mathcal{D}_{n,m}$, it holds $G(y) \in Opt(y)$. Then G is not Banach-Mazur computable.

All functions $G^* : \mathcal{D}_{n,m} \to \{b \in \mathbb{R}_+^m | \sum_i b_i = 1\}$ satisfying

$$\|G - G^*\|_\infty \leq \alpha < 1$$

are also not Banach-Mazur computable.

5.2.3 Optimal input distribution - information theory

In information theory, a point-to-point channel with one receiver and one transmitter is modeled by two discrete random variables X and Y over the probability spaces $\mathcal{X}$ and $\mathcal{Y}$. If we choose $\mathcal{X}$ and $\mathcal{Y}$ to be finite, we are describing a discrete memoryless channel (DMC). The channel itself is then given by a stochastic matrix $W \in \mathbb{R}^{m\times n}$, where $|\mathcal{X}| = n$ and $|\mathcal{Y}| = m$. X, Y and W are related by

$$W(x) = P(Y|X = x).$$

We define the mutual information of two discrete random variables X, Y over $\mathcal{X}$, $\mathcal{Y}$ as

$$I(X, Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P_{(X,Y)}(x, y)\log\left(\frac{P_{(X,Y)}(x, y)}{P_X(x)P_Y(y)}\right),$$

where $P_{(X,Y)}$ is the probability mass function of (X, Y), and P_X and P_Y are the probability mass functions of X and Y. Now the capacity $C(W)$ of a DMC W is the maximal mutual information over all possible distributions over $\mathcal{X}$

$$C(W) := \max_{X\in\mathcal{P}(\mathcal{X})} I(X, Y).$$

The capacity of a DMC is well established and goes back to Shannon (1948). Also, more recently, the capacity has been considered in more complicated settings (Boche et al., 2019b,a). Trying to find the capacity of a DMC is a classical optimization problem for which a well-known approach using an iterative algorithm with convergence guarantee exists (Arimoto, 1972; Blahut, 1972).

We will show that even though such an effective algorithm exists, it is still impossible to compute a maximizing distribution in general or give an approximation guarantee. This was first proven in Boche et al. (2022) and again proven as a special case of Theorem 6 in Boche et al. (2022).

Theorem 9 (Channel Capacity). *Let $\mathcal{X}$ and $\mathcal{Y}$ be finite sets, such that $|\mathcal{X}| = n \geq 3$ and $|\mathcal{Y}| = m \geq 2$ and W is a stochastic matrix. We define $\mathcal{P}(\mathcal{X}) := \{$random variables in $\mathcal{X}\}$ and $\mathcal{P}(\mathcal{Y}) := \{$random variables in $\mathcal{Y}\}$. Since discrete random vectors can be identified by their probabilities for each event, i.e., we uniquely describe $Y \in \mathcal{P}(\mathcal{Y})$ by the vector $(P(Y = y_1), \dots, P(Y = y_m)) \in \mathbb{R}_c^m$, with slight abuse of notation we equate those two objects by*

$$\mathcal{P}(\mathcal{Y}) \hat{=} \{(v_1, \dots, v_m) \in \mathbb{R}_c^m \,|\, \sum_{i=1}^{m} v_i = 1,\ \forall_i : v_i \geq 0\}$$

and analogously for $\mathcal{P}(\mathcal{X})$. Also, define $\mathcal{W}$ as the set of all stochastic matrices in $\mathbb{R}_c^{m \times n}$. Let $G : \mathcal{W} \to \mathcal{P}(\mathcal{X})$ be a function, such that, regarding the maximization problem

$$\max_{X \in \mathcal{P}(\mathcal{X})} I(X, Y),$$

for all $W \in \mathcal{W}$ we have $G(W) \in Opt(W)$. Then G is not Banach-Mazur computable.

All functions $G^* : \mathcal{W} \to \mathcal{P}(\mathcal{X})$ satisfying

$$\|G - G^*\|_\infty \leq \alpha < 1$$

are also not Banach-Mazur computable.

5.3 Wasserstein distance

The Wasserstein-1 distance, originally formulated by Kantorovich (1960) and Vaserstein (1969) to tackle optimal transport problems, is a metric defined on the set of real probability distributions with finite first moment, i.e.,

$$\mathcal{P}_1 := \left\{ \pi \text{ probability distributions} \,\middle|\, \inf_{c \in \mathbb{R}} \int |x - c| d\pi(x) < \infty \right\}.$$

One way to define the Wasserstein-1 metric $\mathbb{W}_1$ is by

$$\mathbb{W}_1(\pi_1, \pi_2) := \sup_{f \in Lip_1} \left| \mathbb{E}_{x \in \pi_1}[f(x)] - \mathbb{E}_{x \in \pi_2}[f(x)] \right|,$$

where $\pi_1, \pi_2 \in \mathcal{P}_1$ and

$$Lip_1 := \{f : \mathbb{R} \to \mathbb{R} | \forall_{x,y \in \mathbb{R}} : |f(x) - f(y)| \leq |x - y|\}.$$

This is the Kantorovich-Rubenstein duality formulation of the Wasserstein-1 distance. Recently this formulation of the Wasserstein distance came to particular interest in the context of Wasserstein-GANs (Arjovsky et al., 2017). The basic idea is to train a neural network, which is able to discriminate between the distribution of "nice" objects and the distribution of "adversarial" objects. This is done by maximizing over $\left|\mathbb{E}_{x\in\pi_1}[f(x)] - \mathbb{E}_{x\in\pi_2}[f(x)]\right|$, where f is the neural network to be trained and adding some regularizer to ensure that the Lipschitz constant is close to 1. We consider the following relaxed setting. First, we only consider probability distribution with computable density functions, supported in $[-\frac{1}{2}, \frac{1}{2}]$,

$$P([-1/2, 1/2]) := \Big\{ f : [-1/2, 1/2] \to \mathbb{R}_+ \Big| \int f = 1,\ f \text{ Borel-Turing computable} \Big\}.$$

Second, we restrict ourselves to a function space $\mathbb{F} \subset Lip_1$, which is made of Borel-Turing computable functions. The only additional assumption on $\mathbb{F}$ is:

$$\exists_{f_1,f_2\in\mathbb{F}} : \exists_{c_1,c_2\in\mathbb{R}} : \forall_{x\in[-1/2,1/2]} : f_1(x) = x + c_1 \text{ and } f_2(x) = |x| + c_2.$$

This assumption holds in the example case of normalized neural networks.

It was shown in Lee et al. (2023) that calculating such a Wasserstein maximizer, or even approximating it is not possible with a Turing machine in these settings.

Theorem 10 (Wasserstein distance). *We define*

$$\mathbb{W}_1'(\pi_1, \pi_2) := \sup_{f\in\mathbb{F}} \left|\mathbb{E}_{x\in\pi_1}[f(x)] - \mathbb{E}_{x\in\pi_2}[f(x)]\right|.$$

Let $p_1, p_2 \in P[-\frac{1}{2}, \frac{1}{2}]$ be two computable probability densities. Then the problem of finding a function $f \in \mathbb{F}$, such that $\mathbb{W}_1'(\pi_1, \pi_2) = \mathbb{E}_{x\in\pi_1}[f(x)] - \mathbb{E}_{x\in\pi_2}[f(x)]$ or $|\mathbb{W}_1'(\pi_1, \pi_2) - \mathbb{E}_{x\in\pi_1}[f(x)] + \mathbb{E}_{x\in\pi_2}[f(x)]| \leq \alpha < \frac{\sqrt{5}}{8\sqrt{3}}$ is not Banach-Mazur computable.

It might very well happen that such a maximizing function does not exist at all. In this case, finding such a function is trivially noncomputable. However, it was proven that finding such a maximizing function might not be computable even in the case a computable maximizing function does exist.

5.4 Lattice problem for cryptographic applications

The basic idea of encryption is to apply a function F to a message m together with a (secret) key k to obtain an encrypted message $F(k, m) = e$. Ideally, it is hard to recover m from e without knowledge of k and easy to do with knowledge of k. Usually, F is motivated by using problems that are hard or suspected to be hard for all Turing machines to solve (Diffie and Hellman, 2022; Rivest et al., 1978; Daemen and Rijmen, 2002). So complexity and computability questions on Turing machines are central for well working encryption schemes. We focus on the question of computability of one particular problem from the family of lattice problems.

Lattice problems have become a topic of interest with the rising feasibility of quantum computers. Since quantum computers are able to crack conventional encryptions efficiently (Hallgren, 2007; Shor, 1999), lattice problems are seen as a new viable source for encryptions. It is wildly believed, but not proven, that lattice problems are hard to solve not only for Turing machines (Blömer and Seifert, 1999) but also for quantum computers.

Different optimization problems are highly relevant candidates for post-quantum cryptography, among others (Regev, 2009) are the shortest vector problem, the shortest independent vector problem, the closest vector problem, and the short generator principal ideal problem. Since Regev's discoveries (Regev, 2009), tremendous efforts have been made to solve the mentioned problems (Eisenträger et al., 2014; Biasse and Song, 2016). We consider the shortest independent vectors problem (SIVP), for which a randomized algorithm with exponential runtime exists if the complexity of the input is bounded (Ajtai et al., 2001).

To formulate the SIVP, we first have to define lattices over a field. Commonly, finite fields such as $\mathbb{Z}/p\mathbb{Z}$, where $p \in \mathbb{N}$ is a large prime number, are considered for the message space. In the following, we consider lattices over the field of real computable numbers $\mathbb{R}_c$. As it was shown in Lee et al. (2023), the transition from large p to the "continuous" field $\mathbb{R}_c$ is problematic, and the corresponding optimization problem becomes noncomputable on Turing machines, while for finite fields there have been recent successes (Eisenträger et al., 2014; Biasse and Song, 2016). Given $n \in \mathbb{N}$ and a basis $B = \{b_1, \dots, b_n\} \subset \mathbb{R}_c^n$, we define the corresponding lattice as

$$\Lambda(B) := \left\{ \sum_{i=1}^{n} \lambda_i b_i \in \mathbb{R}_c^n | \forall_{i=1,\dots,n} : \lambda_i \in \mathbb{Z} \right\}.$$

Now define $\mathcal{B}(B)$ to be the set of bases in $\Lambda(B)$:

$$\mathcal{B}(B) = \{\beta \subset \Lambda(B) | \beta \text{ is basis of } \mathbb{R}_c^n\}.$$

The SIVP is described by the minimization problem

$$\min_{\beta \in \mathcal{B}(B)} \sum_{b \in \beta} \|b\|_2.$$

Theorem 11 (SIVP). *Let $n \in \mathbb{N}$ and define V as the set of all bases in $\mathbb{R}^n$. Let $G : V \to V$ such that regarding SIVP and all bases $B \in V$ we have $G(B) \in Opt(B)$. Then G is not Banach-Mazur computable.*

All functions $G^* : V \to V$ satisfying

$$\|G - G^*\|_\infty \leq \alpha < \frac{\sqrt{2}}{2}$$

are also not Banach-Mazur computable.

5.5 Inverse problems

Let $A : \mathbb{R}_c^n \to \mathbb{R}_c^m$, $x \in \mathbb{R}^n$, $e \in \mathbb{R}^m$, and $y = Ax + e$. Usually A is called the forward operator, x the ground truth image, e the noise, and y the measurement. An inverse problem is the task of recovering x only knowing A, y, and having insufficient knowledge about e. Common examples of A are the identity operator for denoising, the convolution operator for deblurring, the Radon transformation for computer-tomography, and the (subsampled) Fourier transformation for magnet-resonance-tomography.

There are different approaches to solving inverse problems. Three of the most popular ones are to formulate this as a minimization problem. In the spirit of signal processing theory sparse solutions for x are often preferred, i.e. solutions where $\|x\|_0$ is small. The idea of recovering sparse solutions is called sparse recovery. We introduce three popular sparse recovery approaches.

Define the Basis Pursuit (BP) problem as

$$\min_x \|x\|_{l^1} \; s.t. \; \|Ax - y\|_{l^2} \leq \epsilon.$$

For some $\lambda > 0$ define the Lasso problem as

$$\min_x \lambda \|x\|_{l^1} + \|Ax - y\|_{l^2}.$$

For some $\lambda > 0$ define the quadratic Lasso problem as

$$\min_x \lambda \|x\|_{l^1} + \|Ax - y\|_{l^2}^2.$$

These formulations are widely used in sparse recovery, but they have limitations that were exposed by the impossibility results in Boche et al. (2023).

Theorem 12 (Boche et al., 2023). *Fix the parameters $n \geq 2$ and $m < n$ as well as $\epsilon \in (0, \frac{1}{4}) \cap \mathbb{Q}$ and $\lambda \in (0, \frac{5}{4}) \cap \mathbb{Q}$ respectively. Consider the Basis Pursuit*

problem and let $K \geq C_{bp} > 0$ be sufficiently large. Let $G^* : \mathbb{C}^{m\times n} \times \mathbb{C}^m \to X$ be an arbitrary function with

$$\sup_{\substack{(A,y)\mathbb{C}^{m\times n}\times\mathbb{C}^m \\ \|A\|\leq K, \|y\|\leq 1}} \|G^* - G\|_{l_2} < \frac{1}{4}.$$

Then G^* is not Banach-Mazur computable.

Now consider the Lasso problem and let $K \geq C_l > 0$. Let $G^* : \mathbb{C}^{m\times n} \times \mathbb{C}^m \to X$ be an arbitrary function with

$$\sup_{\substack{(A,y)\mathbb{C}^{m\times n}\times\mathbb{C}^m \\ \|A\|\leq K, \|y\|\leq 1}} \|G^* - G\|_{l_2} < \frac{1}{8}.$$

Then G^* is not Banach-Mazur computable.

A different branch of noncomputability research in inverse problems was established in Colbrook et al. (2022), where a different notion of algorithm was used. The authors proved that for all three instances of sparse recovery, i.e., BP, Lasso, and quadratic Lasso, neural networks cannot achieve exact recovery. They proved the existence of training sets that allow neural networks to approximate the sparse solution up to $N - 1$ digits of precision, but not up to N digits, regardless of the network architecture or parameters.

Acknowledgments

The work of Yunseok Lee was supported by the German Research Foundation under Grant DFG-SPP-2298, KU 1446/32-1. The work of Holger Boche and Gitta Kutyniok was supported in part by the ONE Munich Strategy Forum (LMU Munich, TU Munich, and the Bavarian Ministery for Science and Art). The work of Holger Boche was also supported in part by the German Federal Ministry of Education and Research (BMBF) in the project Hardware Platforms and Computing Models for Neuromorphic Computing (NeuroCM) under Grant 16ME0442 and within the national initiative on 6G Communication Systems through the research hub 6G-life under Grant 16KISK002. The work of Gitta Kutyniok was also 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 and under Grant DFG-SFB/TR 109, Project C09 and the Federal Ministry of Education and Research under Grant MaGriDo.

References

Ajtai, Miklós, Kumar, Ravi, Sivakumar, Dandapani, 2001. A sieve algorithm for the shortest lattice vector problem. In: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, pp. 601–610.

Arimoto, S., 1972. An algorithm for computing the capacity of arbitrary discrete memoryless channels. IEEE Transactions on Information Theory 18 (1), 14–20. https://doi.org/10.1109/TIT.1972.1054753.

Arjovsky, Martin, Chintala, Soumith, Bottou, Léon, 2017. Wasserstein GAN. https://arxiv.org/abs/1701.07875. https://dx.doi.org/10.48550/ARXIV.1701.07875.

Benioff, Paul, 1980. The computer as a physical system: a microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. Journal of Statistical Physics 22 (5), 563–591.

Berner, Julius, et al., 2021. The modern mathematics of deep learning. https://doi.org/10.48550/ARXIV.2105.04026. https://arxiv.org/abs/2105.04026.

Biasse, Jean-François, Song, Fang, 2016. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM, pp. 893–902.

Blahut, R., 1972. Computation of channel capacity and rate-distortion functions. IEEE Transactions on Information Theory 18 (4), 460–473. https://doi.org/10.1109/TIT.1972.1054855.

Blömer, Johannes, Seifert, Jean-Pierre, 1999. On the complexity of computing short linearly independent vectors and short bases in a lattice. In: Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, pp. 711–720.

Blum, Lenore, Shub, Mike, Smale, Steve, 2000. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. In: The Collected Papers of Stephen Smale, vol. 3. World Scientific, pp. 1293–1338.

Boche, Holger, Fono, Adalbert, Kutyniok, Gitta, 2023. Limitations of deep learning for inverse problems on digital hardware. https://dx.doi.org/10.48550/ARXIV.2202.13490. http://arxiv.org/abs/1207.0580.

Boche, Holger, Schaefer, Rafael F., Poor, H. Vincent, 2019a. Coding for non-iid sources and channels: entropic approximations and a question of Ahlswede. In: 2019 IEEE Information Theory Workshop (ITW). IEEE, pp. 1–5.

Boche, Holger, Schaefer, Rafael F., Poor, H. Vincent, 2019b. Secure communication and identification systems—effective performance evaluation on Turing machines. IEEE Transactions on Information Forensics and Security 15, 1013–1025.

Boche, Holger, Schaefer, Rafael F., Poor, H. Vincent, 2022. Algorithmic computability and approximability of capacity-achieving input distributions. https://arxiv.org/abs/2202.12617. https://dx.doi.org/10.48550/ARXIV.2202.12617.

Boyd, Stephen, Vandenberghe, Lieven, 2004. Convex Optimization. Cambridge University Press.

Colbrook, Matthew J., Antun, Vegard, Hansen, Anders C., 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. https://doi.org/10.1073/pnas.2107151119.

Cover, T.M., 1984. An algorithm for maximizing expected log investment return. In: IEEE Transactions on Information Theory IT-30.2.

Daemen, Joan, Rijmen, Vincent, 2002. The Design of Rijndael, vol. 2. Springer.

Denning, Peter J., et al., 1989. Computing as a discipline. Communications of the ACM 32 (1), 9–23.

Diffie, Whitfield, Hellman, Martin E., 2022. New directions in cryptography. In: Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman, pp. 365–390.

Eisenträger, Kirsten, et al., 2014. A quantum algorithm for computing the unit group of an arbitrary degree number field. In: Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing, pp. 293–302.

Gottschling, Nina M., et al., 2020. The troublesome kernel: why deep learning for inverse problems is typically unstable. CoRR. arXiv:2001.01258. arXiv:2001.01258. http://arxiv.org/abs/2001.01258.

Grohs, Philipp, Voigtlaender, Felix, 2023. Proof of the theory-to-practice gap in deep learning via sampling complexity bounds for neural network approximation spaces. Foundations of Computational Mathematics, 1–59.

Hallgren, Sean, 2007. Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. Journal of the ACM 54 (1), 1–19.

Hinton, Geoffrey E., et al., 2012. Improving neural networks by preventing co-adaptation of feature detectors. CoRR. arXiv:1207.0580. arXiv:1207.0580. http://arxiv.org/abs/1207.0580.
Kantorovich, Leonid V., 1960. Mathematical methods of organizing and planning production. Management Science 6 (4), 366–422.
Latane, Henry Allen, 1959. Criteria for choice among risky ventures. Journal of Political Economy 67 (2), 144–155.
Lee, Yunseok, Boche, Holger, Kutyniok, Gitta, 2023. Computability of optimizers. arXiv:2301.06148 [math.OC].
Mazur, Stanisław, 1963. Computable analysis.
Mead, Carver, 1990. Neuromorphic electronic systems. Proceedings of the IEEE 78 (10), 1629–1636.
Montavon, Grégoire, et al., 2019. Layer-wise relevance propagation: an overview. In: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, pp. 193–209.
Pour-El, Marian B., Richards, J. Ian, 2017. Computability in analysis and physics. In: Perspectives in Logic. Cambridge University Press.
Regev, Oded, 2009. On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM 56 (6), 1–40.
Rivest, Ronald L., Shamir, Adi, Adleman, Leonard, 1978. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21 (2), 120–126.
Schwarz, Hans-Rudolf, Köckler, Norbert, 2013. Numerische mathematik. Springer-Verlag.
Shannon, Claude Elwood, 1948. A mathematical theory of communication. The Bell System Technical Journal 27 (3), 379–423.
Shor, Peter W., 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Review 41 (2), 303–332.
Smale, Steve, et al., 1998. Mathematical problems for the next century. The Mathematical Intelligencer 20 (2), 7–15.
Specker, E., 1959. Der Satz vom Maximum in der rekursiven Analysis. In: Heyting, A. (Ed.), Constructivity in Mathematics: Proceedings of the Colloquium Held at Amsterdam, 1957.
Turing, Alan Mathison, 1938. On computable numbers, with an application to the Entscheidungsproblem. A correction. Proceedings of the London Mathematical Society 2 (1), 544–546.
Turing, Alan Mathison, et al., 1936. On computable numbers, with an application to the Entscheidungsproblem. Journal of Mathematics 58 (345–363), 5.
Vaserstein, Leonid Nisonovich, 1969. Markov processes over denumerable products of spaces, describing large systems of automata. Problemy Peredači Informacii 5 (3), 64–72.
Weihrauch, Klaus, 2000. Computable Analysis: an Introduction. Springer Science & Business Media.
Wong, Kam-Chau, 1996. Computability of minimizers and separating hyperplanes. Mathematical Logic Quarterly 42 (1), 564–568.